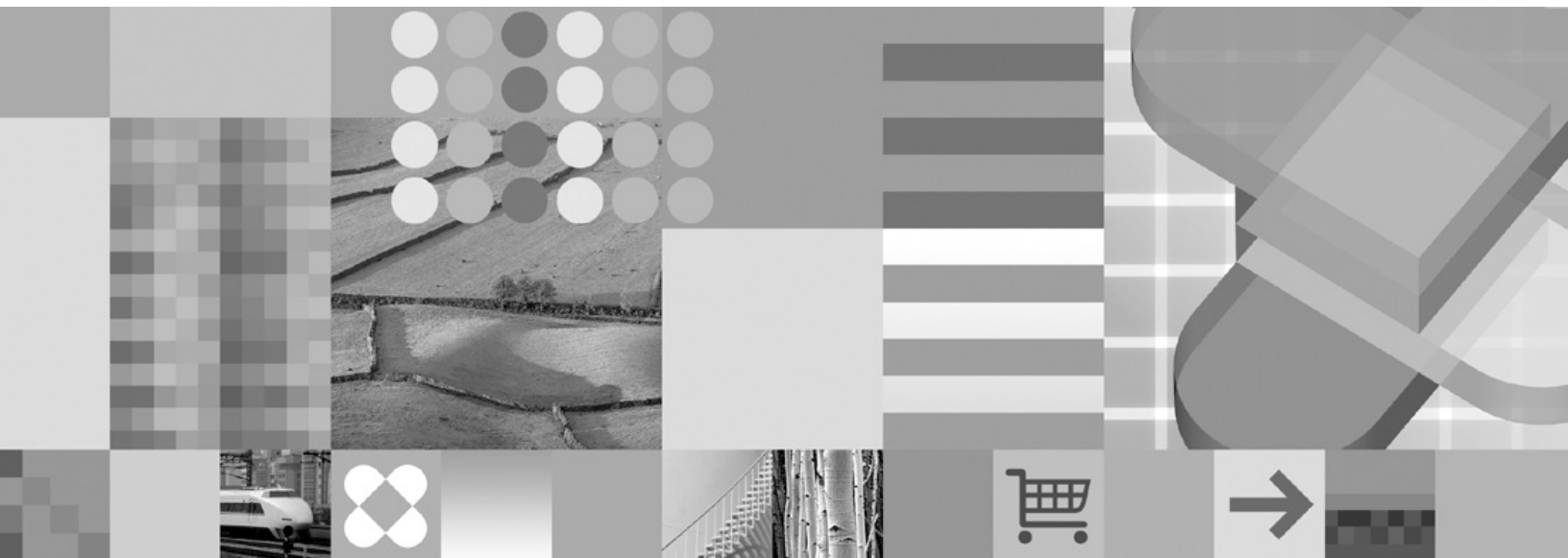




## Performance Monitoring and Tuning Guide





## Performance Monitoring and Tuning Guide

**Note**

Before using this information and the product it supports, be sure to read the general information under “Notices” at the end of this information.

**Fourth edition (February 2008)**

This edition applies to DB2 Version 9.1 for z/OS (DB2 V9.1 for z/OS), product number 5635-DB2, and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Specific changes are indicated by a vertical bar to the left of a change. A vertical bar to the left of a figure caption indicates that the figure has changed. Editorial changes that have no technical significance are not noted.

© Copyright International Business Machines Corporation 1982, 2008. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>About this information.</b>	<b>xiii</b>
DB2 Utilities Suite	xiii
Who should read this information	xiii
Terminology and citations	xiv
Accessibility features for DB2 Version 9.1 for z/OS	xiv
How to send your comments	xv
 <b>Part 1. Planning your performance strategy</b>	 <b>1</b>
 <b>Chapter 1. Managing performance in general.</b>	 <b>3</b>
 <b>Chapter 2. Setting reasonable performance objectives.</b>	 <b>5</b>
Defining your workloads	6
Sizing your workloads	6
Translating resource requirements into performance objectives.	7
Reviewing performance during external design.	8
Reviewing performance during internal design.	8
Reviewing performance during coding and testing	8
Reviewing performance after development	9
 <b>Chapter 3. Planning to review performance data</b>	 <b>11</b>
Typical review questions	11
Validating your performance objectives	13
 <b>Part 2. Designing your system for performance</b>	 <b>15</b>
 <b>Chapter 4. Managing CPU and I/O to improve response time and throughput</b>	 <b>17</b>
Controlling the number of I/O operations	17
Keeping access path statistics updated	17
Making buffer pools large enough for the workload.	18
Reducing the time required for I/O operations	18
Distributing data sets efficiently	18
Creating additional work file table spaces	20
Formatting early and speed-up formatting	21
Avoiding excessively small extents	22
 <b>Chapter 5. Controlling resource usage</b>	 <b>25</b>
Facilities for controlling resource usage	25
Prioritizing resources	26
Limiting resources for each job	26
Limiting resources for TSO sessions	27
Limiting resources for IMS and CICS.	27
Limiting resources for a stored procedure	27
Setting limits for system resource usage	27
Using reactive governing	28
Using predictive governing	28
Combining reactive and predictive governing	30
Limiting resource usage for plans or packages.	31
Limiting resource usage by client information for middleware servers.	32
Managing resource limit tables	33
Governing statements from a remote site	41
Calculating service unit values for resource limit tables.	42
Restricting bind operations	43

Restricting parallelism modes . . . . .	44
I The DB2 system monitor . . . . .	44
Reducing processor resource consumption . . . . .	45
Reusing threads for your high-volume transactions . . . . .	45
Minimizing the use of DB2 traces . . . . .	46
<b>Chapter 6. z/OS performance options for DB2 . . . . .</b>	<b>49</b>
Determining z/OS workload-management velocity goals . . . . .	49
How DB2 assigns I/O priorities . . . . .	50
<b>Chapter 7. Configuring storage for performance . . . . .</b>	<b>53</b>
Storage servers and channel subsystems . . . . .	53
Balancing the storage controller cache and buffer resources . . . . .	53
Improving the use of real and virtual storage . . . . .	53
Real storage . . . . .	55
Virtual storage . . . . .	56
Improving disk storage . . . . .	83
Selecting and configuring storage devices . . . . .	83
Using disk space effectively . . . . .	85
<b>Chapter 8. Improving DB2 log performance . . . . .</b>	<b>99</b>
Improving log write performance . . . . .	99
Types of log writes . . . . .	100
Improving log read performance . . . . .	101
Log statistics . . . . .	102
Calculating average log record size . . . . .	102
Improving log capacity . . . . .	103
Total capacity and the number of logs . . . . .	103
Choosing a checkpoint frequency. . . . .	103
Increasing the number of active log data sets. . . . .	104
Setting the size of active log data sets . . . . .	104
Controlling the amount of log data . . . . .	105
Controlling log size for utilities . . . . .	105
Controlling log size for SQL operations. . . . .	105
<b>Chapter 9. Monitoring and tuning stored procedures and user-defined functions . . . . .</b>	<b>109</b>
Controlling address space storage . . . . .	109
Assigning procedures and functions to WLM application environments . . . . .	110
<b>Chapter 10. Accounting for nested activities . . . . .</b>	<b>113</b>
<b>Chapter 11. Using materialized query tables to improve SQL performance . . . . .</b>	<b>115</b>
Configuring automatic query rewrite . . . . .	115
Materialized query tables and automatic query rewrite . . . . .	115
Enabling automatic query rewrite . . . . .	124
Creating a materialized query table . . . . .	126
Populating and maintaining materialized query tables. . . . .	130
Enabling a materialized query table for automatic query rewrite . . . . .	133
Recommendations for materialized query table and base table design . . . . .	133
Materialized query tables—examples shipped with DB2 . . . . .	134
<b>Chapter 12. Managing DB2 threads . . . . .</b>	<b>137</b>
Setting thread limits . . . . .	137
Allied thread allocation . . . . .	138
Step 1: Thread creation . . . . .	138
Step 2: Resource allocation . . . . .	139
Step 3: SQL statement execution . . . . .	139
Step 4: Commit and thread termination. . . . .	140
Variations on thread management . . . . .	141

Reusing threads . . . . .	141
Using distributed database access threads . . . . .	143
Setting thread limits for database access threads. . . . .	143
Using threads in INACTIVE MODE for DRDA-only connections . . . . .	144
Using threads with private-protocol connections. . . . .	146
Reusing threads for remote connections . . . . .	147
Using z/OS workload management to set performance objectives . . . . .	147
Classifying DDF threads. . . . .	147
Establishing performance periods for DDF threads . . . . .	149
Establishing performance objectives for DDF threads . . . . .	150
Setting CICS options for threads . . . . .	150
Setting IMS options for threads . . . . .	150
Setting TSO options for threads . . . . .	151
Setting DB2 QMF options for threads . . . . .	152

## **Chapter 13. Designing DB2 statistics for performance. . . . . 153**

Maintaining statistics in the catalog . . . . .	153
Statistics used for access path selection . . . . .	153
Setting default statistics for created temporary tables . . . . .	166
History statistics . . . . .	167
What other statistics provide index costs . . . . .	171
Modeling your production system . . . . .	172
Real-time statistics. . . . .	175
Setting up your system for real-time statistics . . . . .	175
Contents of the real-time statistics tables . . . . .	175
Operating with real-time statistics . . . . .	175

## **Part 3. Programming DB2 applications for performance. . . . . 233**

## **Chapter 14. Tuning your queries . . . . . 235**

Coding SQL statements as simply as possible . . . . .	235
Coding queries with aggregate functions efficiently. . . . .	235
Using non-column expressions efficiently . . . . .	236
Materialized query tables and query performance . . . . .	236
Encrypted data and query performance . . . . .	236
XML data and query performance . . . . .	237
Best practices for XML performance in DB2 . . . . .	237
Writing efficient predicates . . . . .	245
Ensuring that predicates are coded correctly . . . . .	246
Properties of predicates . . . . .	247
Predicates in the ON clause . . . . .	250
Using predicates efficiently. . . . .	251
When DB2 evaluates predicates . . . . .	251
Summary of predicate processing. . . . .	252
Examples of predicate properties . . . . .	258
Predicate filter factors . . . . .	260
Avoiding problems with correlated columns . . . . .	268
DB2 predicate manipulation . . . . .	272
Predicates with encrypted data . . . . .	278
Using host variables efficiently . . . . .	279
Changing the access path at run time . . . . .	279
Writing efficient subqueries. . . . .	283
Correlated and non-correlated subqueries . . . . .	284
When DB2 transforms a subquery into a join . . . . .	285
When DB2 Correlates and de-correlates subqueries. . . . .	287
Subquery tuning . . . . .	288
Using scrollable cursors efficiently . . . . .	289
Efficient queries for tables with data-partitioned secondary indexes . . . . .	290
Special techniques to influence access path selection . . . . .	293
Using package copies to alleviate performance regression . . . . .	293

Rewriting queries to influence access path selection . . . . .	296
Obtaining information about access paths . . . . .	299
Fetching a limited number of rows: FETCH FIRST <i>n</i> ROWS ONLY . . . . .	300
Minimizing overhead for retrieving few rows: OPTIMIZE FOR <i>n</i> ROWS . . . . .	301
Favoring index access . . . . .	303
Using the CARDINALITY clause to improve the performance of queries with user-defined table function references . . . . .	304
Reducing the number of matching columns . . . . .	305
Indexes for efficient star schema processing . . . . .	306
Rearranging the order of tables in a FROM clause . . . . .	307
Updating catalog statistics . . . . .	308
Using subsystem parameters to improve performance . . . . .	309
Giving optimization hints to DB2. . . . .	311
<b>Chapter 15. Programming for concurrency. . . . .</b>	<b>319</b>
Concurrency and locks defined . . . . .	319
Effects of DB2 locks . . . . .	320
Promoting basic concurrency . . . . .	323
Using system and subsystem options to promote concurrency . . . . .	323
Designing your databases for concurrency. . . . .	324
Programming your applications for concurrency. . . . .	325
Aspects of transaction locks . . . . .	329
Lock size . . . . .	329
The duration of a lock . . . . .	334
Lock modes . . . . .	334
The object of a lock . . . . .	336
How DB2 chooses lock types . . . . .	339
Options for tuning locks. . . . .	347
IRLM startup procedure options . . . . .	347
Setting installation options for wait times . . . . .	348
Bind options for locks . . . . .	353
Using other options to control locking . . . . .	372
Controlling DB2 locks for LOBs . . . . .	378
LOB locks . . . . .	378
Controlling the number of LOB locks . . . . .	380
Explicitly locking LOB tables . . . . .	380
Controlling lock size for LOB table spaces. . . . .	381
Controlling DB2 locks for XML data. . . . .	381
XML locks . . . . .	381
Controlling the number of XML locks . . . . .	384
Explicitly locking XML data . . . . .	384
Specifying the size of locks for XML data . . . . .	385
Claims and drains for concurrency control . . . . .	385
Objects that are subject to takeover . . . . .	385
Claims . . . . .	385
Drains. . . . .	386
How DB2 uses drain locks . . . . .	386
Utility locks on the catalog and directory . . . . .	387
Compatibility of utilities. . . . .	388
Concurrency during REORG . . . . .	389
Utility operations with nonpartitioned indexes . . . . .	389
<b>Chapter 16. Programming for parallel processing. . . . .</b>	<b>391</b>
Parallel processing. . . . .	391
Methods of parallel processing . . . . .	391
Partitioning for optimal parallel performance. . . . .	394
Determining if a query is I/O- or processor-intensive . . . . .	395
Determining the number of partitions for parallel processing . . . . .	395
Working with a table space that is already partitioned. . . . .	397
Making the partitions the same size . . . . .	397

Working with partitioned indexes . . . . .	398
Enabling parallel processing . . . . .	398
When parallelism is not used . . . . .	399

## **Chapter 17. Tuning and monitoring in a distributed environment . . . . . 401**

Remote access types: DRDA and private protocol . . . . .	401
--	-----

## **Chapter 18. Tuning distributed applications . . . . . 403**

Application and requesting systems . . . . .	403
BIND options for distributed applications . . . . .	403
SQL statement options for distributed applications . . . . .	403
Block fetching result sets . . . . .	404
Optimizing for very large results sets for DRDA. . . . .	409
Optimizing for small results sets for DRDA . . . . .	410
Data encryption security options . . . . .	411
Serving system . . . . .	411

## **Part 4. Monitoring DB2 for z/OS performance. . . . . 413**

## **Chapter 19. Planning for performance monitoring. . . . . 415**

Continuous performance monitoring . . . . .	415
Planning for periodic monitoring . . . . .	416
Detailed performance monitoring. . . . .	417
Exception performance monitoring . . . . .	417

## **Chapter 20. Using tools to monitor performance . . . . . 419**

Investigating SQL performance with EXPLAIN . . . . .	421
IBM Tivoli OMEGAMON XE . . . . .	422
Tivoli Decision Support for z/OS. . . . .	423
Response time reporting. . . . .	424
Using z/OS, CICS, and IMS tools . . . . .	426
Monitoring system resources . . . . .	426
Monitoring transaction manager throughput . . . . .	428

## **Chapter 21. Using DB2 Trace to monitor performance . . . . . 429**

Minimizing the effects of traces on DB2 performance . . . . .	429
Performance overhead from traces . . . . .	430
Types of traces . . . . .	430
Statistics trace . . . . .	431
Accounting trace . . . . .	431
Audit trace . . . . .	433
Performance trace . . . . .	433
Monitor trace . . . . .	433

## **Chapter 22. Recording SMF trace data . . . . . 435**

Activating SMF. . . . .	435
Allocating SMF buffers . . . . .	435
Reporting data in SMF . . . . .	436

## **Chapter 23. Recording GTF trace data. . . . . 437**

## **Chapter 24. Programming for the instrumentation facility interface (IFI) . . . . . 439**

Submitting DB2 commands through IFI . . . . .	439
Obtaining trace data through IFI . . . . .	440
Passing data to DB2 through IFI . . . . .	441
IFI functions. . . . .	441
Invoking IFI from your program . . . . .	442
Using IFI from stored procedures. . . . .	443

COMMAND: Syntax and usage with IFI . . . . .	443
Authorization for DB2 commands through IFI . . . . .	443
Syntax for DB2 commands through IFI . . . . .	443
Using READS requests through IFI . . . . .	445
Authorization for READS requests through IFI . . . . .	447
Syntax for READS requests through IFI . . . . .	447
Which qualifications are used for READS requests issued through IFI? . . . . .	458
Synchronous data and READS requests through IFI . . . . .	459
Using READS calls to monitor the dynamic statement cache. . . . .	461
Controlling collection of dynamic statement cache statistics with IFCID 0318 . . . . .	462
Using READA requests through IFI . . . . .	462
Authorization for READA requests through IFI . . . . .	463
Syntax for READA requests through IFI . . . . .	464
Asynchronous data and READA requests through IFI . . . . .	464
How DB2 processes READA requests through IFI . . . . .	465
Using WRITE requests through IFI . . . . .	465
Authorization for WRITE requests through IFI . . . . .	465
Syntax for WRITE requests through IFI. . . . .	466
Common communication areas for IFI calls . . . . .	466
Instrument facility communications area (IFCA). . . . .	466
Return area . . . . .	470
IFCID area . . . . .	471
Output area . . . . .	471
Using IFI in a data sharing group . . . . .	472
Data integrity and IFI . . . . .	473
Auditing data and IFI . . . . .	473
Locking considerations for IFI. . . . .	474
Recovery considerations for IFI . . . . .	474
Errors and IFI . . . . .	474
<b>  Chapter 25. Monitoring the use of IBM zIIPs . . . . .</b>	<b>477</b>
<b>  IBM System z9 Integrated Information Processor . . . . .</b>	<b>477</b>
<b>Chapter 26. Monitoring storage. . . . .</b>	<b>479</b>
Monitoring I/O activity of data sets. . . . .	479
Buffer Pool Analyzer . . . . .	479
Monitoring and tuning buffer pools using online commands . . . . .	479
Using OMEGAMON to monitor buffer pool statistics . . . . .	481
Monitoring work file data sets. . . . .	484
<b>Chapter 27. Monitoring of DB2 locking . . . . .</b>	<b>485</b>
Scenario for analyzing concurrency . . . . .	485
Scenario description . . . . .	485
OMEGAMON online locking conflict display. . . . .	490
Using the statistics and accounting traces to monitor locking . . . . .	490
Using EXPLAIN to tell which locks DB2 chooses . . . . .	491
<b>Chapter 28. Deadlock detection scenarios . . . . .</b>	<b>493</b>
Scenario 1: Two-way deadlock with two resources . . . . .	493
Scenario 2: Three-way deadlock with three resources . . . . .	495
<b>Chapter 29. Querying the catalog for statistics . . . . .</b>	<b>497</b>
<b>Chapter 30. Gathering monitor statistics and update statistics . . . . .</b>	<b>499</b>
<b>Chapter 31. When to reorganize indexes and table spaces . . . . .</b>	<b>501</b>
<b>Chapter 32. Monitoring SQL performance . . . . .</b>	<b>503</b>
Monitoring SQL performance with Optimization Service Center for DB2 for z/OS . . . . .	503

Tables that are used by optimization tools . . . . .	503
Using EXPLAIN to capture information about SQL statements.. . . .	557
Creating EXPLAIN tables . . . . .	558
Updating the format of an existing PLAN_TABLE . . . . .	565
Capturing EXPLAIN information. . . . .	567
Working with EXPLAIN table data . . . . .	572
Monitoring and optimizing queries with profile tables. . . . .	574
Profiles . . . . .	575
Profile tables . . . . .	576
Creating a profile . . . . .	588
Function keywords for SYSIBM.DSN_PROFILE_ATTRIBUTES . . . . .	589
Monitoring statements by using profile tables . . . . .	592
Limiting the number of statement reports from a monitor profile . . . . .	595
How DB2 resolves conflicting rows in the profile attributes table . . . . .	597
Obtaining snapshot information for monitored queries . . . . .	598
Subsystem parameters for optimization. . . . .	599
Modifying subsystem parameter values by using profile tables. . . . .	599
Checking for invalid plans and packages . . . . .	601
Monitoring parallel operations . . . . .	601
Using DISPLAY BUFFERPOOL . . . . .	602
Using DISPLAY THREAD . . . . .	602
Using DB2 trace . . . . .	603
Monitoring DB2 in a distributed environment . . . . .	604
The DISPLAY command. . . . .	604
Tracing distributed events . . . . .	605
Reporting server-elapsed time. . . . .	609
Monitoring distributed processing with RMF. . . . .	609
Duration of an enclave . . . . .	609
RMF records for enclaves . . . . .	610

---

## **Part 5. Analyzing performance data . . . . . 611**

### **Chapter 33. Looking at the entire system . . . . . 613**

### **Chapter 34. Beginning to look at DB2 . . . . . 615**

### **Chapter 35. A general approach to problem analysis in DB2 . . . . . 617**

### **Chapter 36. Interpreting DB2 trace output . . . . . 621**

The sections of the trace output . . . . .	621
SMF writer header section . . . . .	622
GTF writer header section . . . . .	623
Self-defining section . . . . .	628
Product section. . . . .	630
Trace field descriptions . . . . .	636

### **Chapter 37. Reading accounting reports from OMEGAMON . . . . . 637**

The accounting report (short format) . . . . .	637
The accounting report (long format). . . . .	638

### **Chapter 38. Interpreting records returned by IFI . . . . . 645**

Trace data record format . . . . .	645
Command record format . . . . .	647

### **Chapter 39. Interpreting data access . . . . . 649**

EXPLAIN tables . . . . .	649
PLAN_TABLE columns . . . . .	649
DSN_FUNCTION_TABLE columns . . . . .	658
DSN_STATEMENT_TABLE columns . . . . .	659

DSN_STATEMENT_CACHE_TABLE columns . . . . .	661
Single-table access . . . . .	664
Table space scan access (ACCESSTYPE='R' and PREFETCH='S') . . . . .	664
Aggregate function access (COLUMN_FN_EVAL) . . . . .	665
Index access (ACCESSTYPE is 'I', 'II', 'N', 'MX', or 'DX'). . . . .	666
Direct row access (PRIMARY_ACCESTYPE='D') . . . . .	678
Scans limited to certain partitions (PAGE_RANGE='Y') . . . . .	682
Parallel processing access (PARALLELISM_MODE='I', 'C', or 'X') . . . . .	683
Complex trigger WHEN clause access (QBLOCKTYPE='TRIGGR') . . . . .	683
Prefetch access . . . . .	684
Sort access . . . . .	689
Access to more than one table . . . . .	691
Join operations . . . . .	691
Cartesian join with small tables first . . . . .	694
Nested loop join (METHOD=1) . . . . .	694
When a MERGE statement is used (QBLOCK_TYPE ='MERGE') . . . . .	697
Merge scan join (METHOD=2) . . . . .	698
Hybrid join (METHOD=4) . . . . .	700
Star schema access . . . . .	701
Subquery access . . . . .	713
View and nested table expression access . . . . .	714
Merge processing . . . . .	715
Materialization . . . . .	716
Performance of merge versus materialization . . . . .	718
Using EXPLAIN to determine when materialization occurs . . . . .	718
Using EXPLAIN to determine UNION, INTERSECT, and EXCEPT activity and query rewrite . . . . .	720
Interpreting query parallelism . . . . .	722
A method for examining PLAN_TABLE columns for parallelism . . . . .	722
PLAN_TABLE examples showing parallelism . . . . .	723
 <b>Chapter 40. Estimating the cost of a SQL statement . . . . .</b>	<b>725</b>
Cost categories . . . . .	725
 <b>Part 6. Tuning DB2 for z/OS performance . . . . .</b>	<b>727</b>
 <b>Chapter 41. Providing cost information, for accessing user-defined table functions, to DB2 . . . . .</b>	<b>729</b>
 <b>Chapter 42. Improving index and table space access . . . . .</b>	<b>731</b>
How clustering affects access path selection . . . . .	731
When to reorganize indexes and table spaces . . . . .	733
Whether to rebind after gathering statistics . . . . .	736
 <b>Chapter 43. Updating the catalog . . . . .</b>	<b>739</b>
Correlations in the catalog . . . . .	739
Recommendation for COLCARD and FIRSTKEYCARD . . . . .	740
Recommendation for HIGH2KEY and LOW2KEY . . . . .	740
Statistics for distributions . . . . .	741
Recommendation for using the TIMESTAMP column . . . . .	741
 <b>Chapter 44. Tuning parallel processing . . . . .</b>	<b>743</b>
 <b>Chapter 45. Disabling query parallelism . . . . .</b>	<b>745</b>
 <b>Part 7. Appendixes . . . . .</b>	<b>747</b>
 <b>Information resources for DB2 for z/OS and related products . . . . .</b>	<b>749</b>

<b>How to obtain DB2 information.</b>	<b>755</b>
<b>How to use the DB2 library</b>	<b>759</b>
<b>Notices</b>	<b>763</b>
Programming Interface Information	765
General-use Programming Interface and Associated Guidance Information	765
Product-sensitive Programming Interface and Associated Guidance Information	765
Trademarks	765
<b>Glossary</b>	<b>767</b>
<b>Index</b>	<b>811</b>



---

## About this information

This information describes performance monitoring and tuning tasks for DB2® Version 9.1 for z/OS®.

Unless it is stated otherwise, this information assumes that DB2 is running in new-function mode (as opposed to compatibility mode or enabling-new-function mode).

---

## DB2 Utilities Suite

**Important:** In this version of DB2 for z/OS, the DB2 Utilities Suite is available as an optional product. You must separately order and purchase a license to such utilities, and discussion of those utility functions in this publication is not intended to otherwise imply that you have a license to them.

The DB2 Utilities Suite is designed to work with the DFSORT™ program, which you are licensed to use in support of the DB2 utilities even if you do not otherwise license DFSORT for general use. If your primary sort product is not DFSORT, consider the following informational APARs mandatory reading:

- II14047/II14213: USE OF DFSORT BY DB2 UTILITIES
- II13495: HOW DFSORT TAKES ADVANTAGE OF 64-BIT REAL ARCHITECTURE

These informational APARs are periodically updated.

### **Related information**

DB2 utilities packaging

---

## Who should read this information

This information is primarily intended for system and database administrators.

It assumes that the user is familiar with:

- The basic concepts and facilities of DB2
- Time Sharing Option (TSO) and Interactive System Productivity Facility (ISPF)
- The basic concepts of Structured Query Language (SQL)
- The basic concepts of Customer Information Control System (CICS®)
- The basic concepts of Information Management System (IMS™)
- How to define and allocate z/OS data sets using job control language (JCL).

Certain tasks require additional skills, such as knowledge of Transmission Control Protocol/Internet Protocol (TCP/IP) or Virtual Telecommunications Access Method (VTAM®) to set up communication between DB2 subsystems.

---

## Terminology and citations

In this information, DB2 Version 9.1 for z/OS is referred to as "DB2 for z/OS." In cases where the context makes the meaning clear, DB2 for z/OS is referred to as "DB2." When this information refers to titles of DB2 for z/OS books, a short title is used. (For example, "See *DB2 SQL Reference*" is a citation to *IBM® DB2 Version 9.1 for z/OS SQL Reference*.)

When referring to a DB2 product other than DB2 for z/OS, this information uses the product's full name to avoid ambiguity.

The following terms are used as indicated:

**DB2** Represents either the DB2 licensed program or a particular DB2 subsystem.

### OMEGAMON

Refers to any of the following products:

- IBM Tivoli® OMEGAMON® XE for DB2 Performance Expert on z/OS
- IBM Tivoli OMEGAMON XE for DB2 Performance Monitor on z/OS
- IBM DB2 Performance Expert for Multiplatforms and Workgroups
- IBM DB2 Buffer Pool Analyzer for z/OS

### C, C++, and C language

Represent the C or C++ programming language.

**CICS** Represents CICS Transaction Server for z/OS.

**IMS** Represents the IMS Database Manager or IMS Transaction Manager.

**MVS** Represents the MVS™ element of the z/OS operating system, which is equivalent to the Base Control Program (BCP) component of the z/OS operating system.

**RACF** Represents the functions that are provided by the RACF® component of the z/OS Security Server.

---

## Accessibility features for DB2 Version 9.1 for z/OS

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

### Accessibility features

The following list includes the major accessibility features in z/OS products, including DB2 Version 9.1 for z/OS. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers and screen magnifiers.
- Customization of display attributes such as color, contrast, and font size

**Tip:** The Information Management Software for z/OS Solutions Information Center (which includes information for DB2 Version 9.1 for z/OS) and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

### Keyboard navigation

You can access DB2 Version 9.1 for z/OS ISPF panel functions by using a keyboard or keyboard shortcut keys.

For information about navigating the DB2 Version 9.1 for z/OS ISPF panels using TSO/E or ISPF, refer to the *z/OS TSO/E Primer*, the *z/OS TSO/E User's Guide*, and the *z/OS ISPF User's Guide*. These guides describe how to navigate each interface, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

## **Related accessibility information**

Online documentation for DB2 Version 9.1 for z/OS is available in the Information Management Software for z/OS Solutions Information Center, which is available at the following Web site: <http://publib.boulder.ibm.com/infocenter/dzichelp>

## **IBM and accessibility**

See the *IBM Accessibility Center* at <http://www.ibm.com/able> for more information about the commitment that IBM has to accessibility.

---

## **How to send your comments**

Your feedback helps IBM to provide quality information. Please send any comments that you have about this book or other DB2 for z/OS documentation. You can use the following methods to provide comments:

- Send your comments by e-mail to [db2zinfo@us.ibm.com](mailto:db2zinfo@us.ibm.com) and include the name of the product, the version number of the product, and the number of the book. If you are commenting on specific text, please list the location of the text (for example, a chapter and section title or a help topic title).
- You can send comments from the Web. Visit the library Web site at:

[www.ibm.com/software/db2zos/library.html](http://www.ibm.com/software/db2zos/library.html)

This Web site has an online reader comment form that you can use to send comments.

- You can also send comments by using the feedback link at the footer of each page in the Information Management Software for z/OS Solutions Information Center at <http://publib.boulder.ibm.com/infocenter/db2zhhelp>.



---

## Part 1. Planning your performance strategy

The first step toward ensuring that DB2 meets the performance requirements of your solution is planning.

You can avoid some performance problems completely by planning for performance when you first design your system. As you begin to plan your performance, remember the following information.

- DB2 is only a part of your overall system. Any change to System z9™ hardware, disk subsystems, z/OS, IMS, CICS, TCP/IP, VTAM, the network, WebSphere®, or distributed application platforms (such as Windows®, UNIX®, or Linux®) that share your enterprise IT infrastructure can affect how DB2 and its applications run.
- The recommendations for managing performance are based on current knowledge of DB2 performance for “normal” circumstances and “typical” systems. Therefore, that this information provides the best, or most appropriate advice, for any specific site cannot be guaranteed. In particular, the advice on performance often approaches situations from a performance viewpoint only. Other factors of higher priority might make some of these performance recommendations inappropriate for your specific solution.
- The recommendations are general. Performance measurements are highly dependent on workload and system characteristics that are external to DB2.



---

## Chapter 1. Managing performance in general

The steps for managing DB2 are like those for any system.

Performance management is an iterative process.

To manage performance:

1. Establish your performance objectives when you plan your system.
2. Consider performance as you design and implement your system.
3. Plan how you will monitor performance and capture performance data.
4. Analyze performance reports to decide whether the objectives have been met.
5. If performance is thoroughly satisfactory, use one of the following options:
  - Monitor less, because monitoring itself uses resources.
  - Continue monitoring to generate a history of performance to compare with future results.
6. If performance has not been satisfactory, take the following actions:
  - a. Determine the major constraints in the system.
  - b. Decide where you can afford to make trade-offs and which resources can bear an additional load. Nearly all tuning involves trade-offs among system resources.
  - c. Tune your system by adjusting its characteristics to improve performance.
  - d. Continue to monitor the system.

Periodically, or after significant changes to your system or workload, reexamine your objectives, and refine your monitoring and tuning strategy accordingly.



---

## Chapter 2. Setting reasonable performance objectives

Performance objectives should be realistic, in line with your budget, understandable, and measurable.

How you define good performance for your DB2 subsystem depends on your particular data processing needs and their priority.

Common objectives include values for:

### **Acceptable response time**

A duration within which some percentage of all applications have completed.

### **Average throughput**

The total number of transactions or queries that complete within a given time.

### **System availability**

Which included mean time to failure and the durations of down time.

Objectives such as these define the workload for the system and determine the requirements for resources, such as processor speed, amount of storage, additional software, and so on. Often, however, available resources limit the maximum acceptable workload, which requires revising the objectives.

## **Service-level agreements**

Presumably, your users have a say in your performance objectives. A mutual agreement on acceptable performance, between the data processing and user groups in an organization, is often formalized and called a *service-level agreement*. Service-level agreements can include expectations of query response time, the workload throughput per day, hour, or minute, and windows provided for batch jobs (including utilities). These agreements list criteria for determining whether or not the system is performing adequately.

### **Example**

A service-level agreement might require that 90% of all response times sampled on a local network in the prime shift be under 2 seconds, or that the average response time not exceed 6 seconds even during peak periods. (For a network of remote terminals, consider substantially higher response times.)

### **Amount of processing**

Performance objectives must reflect not only elapsed time, but also the amount of processing expected. Consider whether to define your criteria in terms of the average, the ninetieth percentile, or even the worst-case response time. Your choice can depend on your site's audit controls and the nature of the workloads.

z/OS workload management (WLM) can manage to the performance objectives in the service-level agreement and provide performance reporting analysis. The terms used in the service-level agreement and the WLM service policy are similar.

---

## Defining your workloads

Before installing DB2, you should gather design data and evaluate your performance objectives with that information.

To define the workload of the system:

Determine the type of workload. For each type of workload, describe a preliminary workload profile that includes the following information:

- A definition of the workload type in terms of its function and its volume. You are likely to have many workloads that perform the same general function (for example, order entry through CICS or IMS transaction managers) and have an identifiable workload profile. Other workload types include SPUFI and DB2 QMF queries, transactions, utilities, and batch jobs.

For the volume of a workload that is already processed by DB2, use the summary of its volumes from the DB2 statistics trace.

- The relative priority of the type, including periods during which the priorities change.
- The resources that are required to do the work, including physical resources that are managed by the operating system (such as real storage, disk I/O, and terminal I/O) and logical resources managed by the subsystem (such as control blocks and buffers).

You should review and reevaluate your performance objectives at all phases of system development.

## Sizing your workloads

You can look at base estimates for transactions, query use, and batch processing to find ways to reduce the workload.

Changes in design in early stages, before contention with other programs, are likely to be the most effective. Later, you can compare the actual production profile against the base. Make an estimate even if these quantities are uncertain at this stage.

To establish your resource requirements:

Estimate resource requirements for the following items:

### Transactions

- Availability of transaction managers, such as IMS, CICS, or WebSphere
- Number of message pairs for each user function, either to and from a terminal or to and from a distributed application
- Network bandwidth and network device latencies
- Average and maximum number of concurrent users, either terminal operators or distributed application requesters
- Maximum rate of workloads per second, minute, hour, day, or week
- Number of disk I/O operations per user workload
- Average and maximum processor usage per workload type and total workload
- Size of tables
- Effects of objectives on operations and system programming

### **Query use**

- Time required to key in user data
- Online query processing load
- Limits to be set for the query environment or preformatted queries
- Size of tables
- Effects of objectives on operations and system programming

### **Batch processing**

- Batch windows for data reorganization, utilities, data definition activities, and BIND processing
- Batch processing load
- Length of batch window
- Number of records to process, data reorganization activity, use of utilities, and data definition activity
- Size of tables and complexity of the queries
- Effects of objectives on operations and system programming

## **Translating resource requirements into performance objectives**

You can convert your estimated resource requirements into performance objectives.

1. For each workload type, convert your estimated resource requirements into measurable performance objectives. Include the following factors when you consider your estimates:

### **System response time**

You cannot guarantee requested response times before any of the design has been done. Therefore, plan to review your performance targets when you design and implement the system.

Response times can vary for many reasons. Therefore, include acceptable tolerances in your descriptions of targets. Remember that distributed data processing adds overhead at both the local and remote locations.

Exclude from the targets any unusual applications that have exceptionally heavy requirements for processing or database access, or establish individual targets for those applications.

### **Network response time**

Responses in the processor are likely to be in microseconds, whereas responses in the network with appropriate facilities can be about a millisecond. This difference in response times means that an overloaded network can impact the delivery of server responses to user terminals or distributed applications regardless of the speed of the processor.

### **Disk response time**

I/O operations are generally responsible for much internal processing time. Consider all I/O operations that affect a workload.

### **Existing workload.**

Consider the effects of additional work on existing applications. In planning the capacity of the system, consider the total load on each major resource, not just the load for the new application.

### **Business factors**

When calculating performance estimates, concentrate on the expected

peak throughput rate. Allow for daily peaks (for example, after receipt of mail), weekly peaks (for example, a Monday peak after weekend mail), and seasonal peaks as appropriate to the business. Also allow for peaks of work after planned interruptions, such as preventive maintenance periods and public holidays. Remember that the availability of input data is one of the constraints on throughput.

2. Include statements about the throughput rates to be supported (including any peak periods) and the internal response time profiles to be achieved.
3. Make assumptions about I/O rates, paging rates, and workloads.

## **Reviewing performance during external design**

You should review performance during the external design phase for your system.

During the external design phase, you must:

1. Estimate the network, Web server, application server, processor, and disk subsystem workload.
2. Refine your estimates of logical disk accesses. Ignore physical accesses at this stage. One of the major difficulties is determining the number of I/Os per statement.

## **Reviewing performance during internal design**

You should review performance objectives during the internal design of your system.

During the internal design phase, you must:

1. Refine your estimated workload against the actual workload.
2. Refine disk access estimates against database design. After internal design, you can define physical data accesses for application-oriented processes and estimate buffer hit ratios.
3. Add the accesses for DB2 work file database, DB2 log, program library, and DB2 sorts.
4. Consider whether additional processor loads can cause a significant constraint.
5. Refine estimates of processor usage.
6. Estimate the internal response time as the sum of processor time and synchronous I/O time or as asynchronous I/O time, whichever is larger.
7. Prototype your DB2 system. Before committing resources to writing code, you can create a small database, update the statistics stored in the DB2 catalog tables, run SQL statements and examine the results.
8. Use DB2 estimation formulas to develop estimates for processor resource consumption and I/O costs for application processes that are high volume or complex.

## **Reviewing performance during coding and testing**

You should review your performance objectives during the coding and testing phase for your system.

During the coding and testing phases, you must:

1. Refine the internal design estimates of disk and processing resources.

2. Run the monitoring tools you have selected and check the results against your estimates. You might use a terminal network simulator such as TeleProcessing Network Simulator (TPNS) or other tools to test the system and simulate load conditions.

## Reviewing performance after development

When you are ready to test the complete system, review its performance in detail.

Take the following steps to complete your performance review:

1. Validate system performance and response times against your performance objectives.
2. Identify resources whose usage requires regular monitoring.
3. Incorporate the observed figures into future estimates:
  - a. Identify discrepancies from the estimated resource usage
  - b. Identify the cause of the discrepancies
  - c. Assign priorities to remedial actions
  - d. Identify resources that are consistently heavily used
  - e. Set up utilities to provide graphic representation of those resources
  - f. Project the processor usage against the planned future system growth to ensure that adequate capacity is available
  - g. Update the design document with the observed performance figures
  - h. Modify your procedures for making estimates according to what you have learned what you have learned

You need feedback from users and might have to solicit it. Establish reporting procedures, and teach your users how to use them. Consider logging incidents such as:

- System, line, and transaction or query failures
- System unavailable time
- Response times that are outside the specified limits
- Incidents that imply performance constraints, such as deadlocks, deadlock abends, and insufficient storage
- Situations, such as recoveries, that use additional system resources

You should log detailed information for such incidents.

- Time
- Date
- Location
- Duration
- Cause (if it can be determined)
- Action taken to resolve the problem



---

## Chapter 3. Planning to review performance data

When establishing requirements and planning to monitor performance, you should also plan how to review the results of monitoring.

You can inspect your performance data to determine whether performance has been satisfactory, to identify problems, and to evaluate the monitoring process.

- Plan to review the performance data systematically. Review daily data weekly and weekly data monthly; review data more often when reports raise specific questions that require investigation. Depending on your system, the weekly review might require about an hour, particularly after you have had some experience with the process and are able to locate quickly any items that require special attention. The monthly review might take half a day at first, less time later on. But when new applications are installed, workload volumes increased, or terminals added, allow more time for review.
- Review the data on a gross level, looking for problem areas. Review details only if a problem arises or if you need to verify measurements.
- When reviewing performance data, try to identify the basic pattern in the workload, and then identify variations of the pattern. After a certain period, discard most of the data you have collected, but keep a representative sample. For example, save the report from the last week of a month for three months; at the end of the year, discard all but the last week of each quarter. Similarly, keep a representative selection of daily and monthly figures. Because of the potential volume of data, consider using Tivoli Decision Support for z/OS, Application Monitor for z/OS, or a similar tool to track historical data in a manageable form.

---

### Typical review questions

You can use specific review questions to help guide your review of performance data.

Use the following questions as a basis for your own checklist. They are not limited strictly to performance items, but your historical data can provide most of their answers. If the performance data is for modeled workloads or changed workloads, the first question to ask for each category is, "What changed?"

#### How often was each transaction and SQL statement used?

- 1. Considering variations in the workload mix over time, are the monitoring times appropriate?
  2. Should monitoring be done more frequently during the day, week, or month to verify this?
  3. How many SELECT, INSERT, UPDATE, DELETE, PREPARE, DESCRIBE, DESCRIBE TABLE, PREPARE, OPEN, FETCH, and CLOSE statements are issued per second and per commit?
  4. How many IRLM and buffer pool requests are issued per second and per commit?

## How were processor and I/O resources used?

1. Has usage increased for functions that run at a higher priority than DB2 tasks? Examine CICS, IMS, z/OS, JES, TCP/IP, VTAM, WebSphere Application Server, WebSphere MQ (formerly called MQSeries®), other subsystems, and key applications.
2. Is the report of processor usage consistent with previous observations?
3. Are scheduled batch jobs able to run successfully?
4. Do any incident reports show that the first invocation of a function takes much longer than later ones? This increased time can happen when programs have to open data sets.
5. What is the CPU time and where is it accumulated? Separate CPU time into accounting TCB and SRB time, and distinguish non-nested, stored procedure, user-defined function, and trigger CPU times. Note the times for DB2 address spaces, DBM1, MSTR, IRLM, and DDF.
6. In a data sharing environment, how are the coupling facility (CF) lock, group buffer pool, and SCA structures performing? What is the peak CF CPU utilization?

## How much real storage was used, and how effective is storage?

1. Is the paging rate increasing? Adequate real storage is very important for DB2 performance.
2. What are the hit ratios for the buffer pools, the EDM pools (above and below), the EDM statement cache, the EDM DBD cache, the EDM skeleton pool, and the dynamic statement cache?

## To what degree was disk used?

Is the number of I/O requests increasing? DB2 records both physical and logical requests. The number of physical I/Os depend on the configuration of indexes, the data records per control interval, and the buffer allocations.

## To what extent were DB2 log resources used?

1. Is the log subject to undue contention from other data sets?

**Recommendation:** Do not put a recoverable (updated) resource and a log under the same RAID controller. If that controller fails, you lose both the resource and the log, and you are unable to perform forward recovery.

2. What is the I/O rate for requests and physical blocks on the log?
3. What is the logging rate for one log in MB per second?
4. How fast are the disk units that are used for logging?

## Do any figures indicate design, coding, or operational errors?

1. Are disk, I/O, log, or processor resources heavily used? If so, was that heavy use expected at design time? If not, can the heavy use be explained in terms of heavier use of workloads?
2. Is the heavy usage associated with a particular application? If so, is there evidence of planned growth or peak periods?
3. What are your needs for concurrent read/write and query activity?
4. Are there any disk, channel, or path problems?
5. Are there any abends or dumps?

### What are the effects of DB2 locks?

1. What are the incidents of deadlocks and timeouts?
2. What percentage of elapsed time is due to lock suspensions? How much lock or latch contention was encountered? Check the contention rate per second by class.
3. How effective is lock avoidance?

### Were there any bottlenecks?

1. Were any critical thresholds reached?
2. Are any resources approaching high utilization?

#### Related concepts

“Using z/OS, CICS, and IMS tools” on page 426

“Statistics trace” on page 431

“Accounting trace” on page 431

#### Related tasks

“Monitoring system resources” on page 426

Chapter 27, “Monitoring of DB2 locking,” on page 485

---

## Validating your performance objectives

After beginning to review and monitor performance, you need to find out if your objectives are reasonable.

You should consider questions about the objectives, such as:

- Are they achievable, given the available hardware?
- Are they based upon actual measurements of the workload?

When you measure performance against initial objectives and report the results to users:

1. Identify any systematic differences between the *internal response time*, the measured performance data, and the *external response time*, what the user sees.
2. If the measurements differ greatly from the estimates:
  - Revise response-time objectives for the application
  - Upgrade your system
  - plan a reduced application workload.

If, however, the differences between internal and external response times are not too large, you can begin monitoring and tuning the entire system.



---

## **Part 2. Designing your system for performance**

By considering performance as you design and configure your system, you can help to ensure better performance from DB2.



---

## Chapter 4. Managing CPU and I/O to improve response time and throughput

To ensure that DB2 meets your goals for response time and throughput, you should manage how your system uses processor resources and I/O processing.

---

### Controlling the number of I/O operations

You can improve the response time of your applications and queries by reducing the number of unnecessary I/O operations.

#### Keeping access path statistics updated

You can reduce the number of unnecessary I/O operations by using the RUNSTATS utility to keep statistics updated in the DB2 catalog.

The RUNSTATS utility collects statistics about DB2 objects. These statistics can be stored in the DB2 catalog and are used during the bind process to choose the path for accessing data. If you never use RUNSTATS and subsequently rebind your packages or plans, DB2 cannot have the information it needs to choose the most efficient access path. This can result in unnecessary I/O operations and excessive processor consumption.

To ensure that catalog statistics are updated:

- Run RUNSTATS at least once against each table and its associated indexes. How often you rerun the utility depends on how current you need the catalog data to be. If data characteristics of the table vary significantly over time, you should keep the catalog current with those changes. RUNSTATS is most beneficial for the following situations:
  - Table spaces that contain frequently accessed tables
  - Tables involved in a sort
  - Tables with many rows
  - Tables against which SELECT statements having many search arguments are performed
- For some tables, you cannot find a good time to run RUNSTATS. For example, you might use some tables for work that is in process. The tables might have only a few rows in the evening when it is convenient to run RUNSTATS, but they might have thousands or millions of rows in them during the day. For such tables, consider these possible approaches:
  - Set the statistics to a relatively high number and hope your estimates are appropriate.
  - Use volatile tables so that SQL operations choose index access whenever possible.

Whichever approach that you choose, monitor the tables because optimization is adversely affected by incorrect information.

#### Related concepts

Chapter 30, “Gathering monitor statistics and update statistics,” on page 499

## Making buffer pools large enough for the workload

You might improve the performance of I/O operations by increasing the size of your buffer pools.

You should make buffer pools as large as you can afford for the following reasons:

- Using larger buffer pools might mean fewer I/O operations and therefore faster access to your data.
- Using larger buffer pools can reduce I/O contention for the most frequently used tables and indexes.
- Using larger buffer pools can speed sorting by reducing I/O contention for work files.

However, many factors affect how you determine the number of buffer pools to have and how big they should be.

### Related tasks

“Determining size and number of buffer pools” on page 64

---

## Reducing the time required for I/O operations

You can use several methods to reduce the time required to perform individual I/O operations.

### Related concepts

“Parallel processing” on page 391

Chapter 6, “z/OS performance options for DB2,” on page 49

## Distributing data sets efficiently

Avoid I/O contention and increase throughput through the I/O subsystem by placing frequently used data sets on fast disk devices and by distributing I/O activity.

Distributing I/O activity is less important when you use disk devices with parallel access volumes (PAV) support and multiple allegiance support.

### Related tasks

“Reusing threads for your high-volume transactions” on page 45

## Putting frequently used data sets on fast devices

You can improve performance by assigning frequently used data sets to faster disk devices.

To make the best use of your disk devices:

- Assign the most frequently used data sets to the faster disk devices at your disposal.
- For partitioned table spaces, you might choose to have some partitions on faster devices than other partitions. Placing frequently used data sets on fast disk devices also improves performance for nonpartitioned table spaces.
- Consider partitioning any nonpartitioned table spaces that have excessive I/O contention at the data set level.

## Distributing the I/O

By distributing your data sets, you can prevent I/O requests from being queued in z/OS.

To distribute I/O operations:

- If you do not have parallel access volumes, allocate frequently used data sets or partitions across your available disk volumes so that I/O operations are distributed. Even with RAID devices, in which the data set is spread across the physical disks in an array, data should be accessed at the same time on separate logical volumes to reduce the chance of an I/O request being queued in z/OS.
- Consider isolating data sets that have characteristics that do not complement other data sets.

### **Partitioning schemes and data clustering for partitioned tablespaces:**

Depending on the type of operations that your applications emphasize, you have several options for distributing I/O.

If the partitions of your partitioned table spaces must be of relatively the same size (which can be a great benefit for query parallelism), consider using a ROWID column as all or part of the partitioning key.

For partitions that are of such unequal size that performance is negatively affected, alter the limit key values to set new partition boundaries and then reorganize the affected partitions to rebalance the data. Alternatively, you can use the REORG utility with the REBALANCE keyword to set new partition boundaries. REBALANCE causes DB2 to change the limit key values such that the rows in the range of partitions being reorganized are distributed across those partitions as evenly as possible.

If your performance objectives emphasize inserts, distribute the data in a manner that reduces the amount of clustered key values. Consider designing your database with randomized index keys design to remove clustering. You can also take advantage of the index page splitting by choosing the appropriate size for index pages. If the rate of inserts does not require you to spread out the inserts, consider creating or altering tables with the APPEND YES option.

In contrast, for performance objectives that emphasize read operations, your data clustering should reflect the sequence in which queries will be processed so that DB2 can use the sequential processing method of parallelism to reduce I/O and CPU time.

Partition data that will be subject to frequent update operations in a manner that provides plenty of free space, especially if new and updated rows might expand because they contain columns with varying-length data types or compressed data.

#### **Related concepts**

“Index splitting for sequential INSERT activity” on page 96

“Methods of parallel processing” on page 391

#### **Related tasks**

“Designing your databases for concurrency” on page 324

### **Increasing the number of data sets for an index:**

By increasing the number of data sets that are used for an index and spreading those data sets across the available I/O paths, you can reduce the physical contention on the index.

**GUIP** Using *data-partitioned secondary indexes*, or making the piece size of a nonpartitioned index smaller, increases the number of data sets that are used for the index.

A secondary index on a partitioned table space can be partitioned. When you partition an index, the partitioning scheme is that of the data in the underlying table, and each index partition has its own data set. Although data-partitioned secondary indexes promote partition independence and can provide performance advantages, they do not always improve query performance. Before using a data-partitioned secondary index, understand the advantages and disadvantages.

**GUIP**

## Creating additional work file table spaces

You can minimize I/O contention in certain situations by creating additional work file table spaces.

Because the rows of a declared global temporary table reside in a table space in a work file database, ensure that the work file database contains at least one 32-KB page size table space before you create a declared global temporary table. DB2 does not create an implicit table space for the declared global temporary table.

During the installation or migration process, you allocated table spaces for 4-KB and 32-KB buffering.

If your applications require any of the following objects or operations, allocate additional work file table spaces on separate disk volumes in a work file database (database DSNDB07 in a non data-sharing environment) to help minimize I/O contention:

- Created temporary tables
- Declared global temporary tables
- Non-correlated subqueries
- Materialized views
- Materialized nested table expressions
- Triggers with transition variables
- Large concurrent sorts or a single large sort
- Some merge, star, and outer joins

For a single query, the recommended number of work file disk volumes to have is one-fifth the maximum number of data partitions, with 5 as a minimum and 50 as a maximum. For concurrently running queries, multiply this value by the number of concurrent queries.

In a query parallelism environment, the number of work file disk volumes should be at least equal to the maximum number of parallel operations used for queries in the given workload.

You should place these volumes on different channel or control unit paths, and monitor the I/O activity for the work file table spaces, because you might need to further separate this work file activity to avoid contention. As the amount of work file activity increases, consider increasing the size of the buffer pool for work files

to support concurrent activities more efficiently. The general recommendation for the work file buffer pool is to increase the size to minimize the following buffer pool statistics:

- MERGE PASSES DEGRADED, which should be less than 1% of MERGE PASS REQUESTED
- WORKFILE REQUESTS REJECTED, which should be less than 1% of WORKFILE REQUEST ALL MERGE PASSES
- Synchronous read I/O, which should be less than 1% of pages read by prefetch
- Prefetch quantity of 4 or less, which should be near 8

To further improve performance, consider allocating more 32-KB data sets. DB2 uses 32 KB work file data sets when the total work file record size, including record overhead, exceeds 100 bytes, resulting in results in better performance, reduced work file buffer pool and disk space usage for work files, in some cases.

**GUIP** To create a new work file table space, *xyz*:

1. Define the required data sets using the VSAM DEFINE CLUSTER statement. You might want to use the definitions in the edited, installation job DSNTIJTM as a model.
2. Create the work file table space by entering the following SQL statement: (If you are using DB2-managed data sets, omit this step)

```
CREATE TABLESPACE xyz IN DSNDB07
    BUFFERPOOL BP7
    CLOSE NO
    USING VCAT DSN910;
```

**GUIP**

## Formatting early and speed-up formatting

You can speed up pre-formatting of data by allocating in cylinders, and by preformatting a table space before inserting data.

### Allocating space in cylinders or in large primary and secondary quantities

Specify your space allocation amounts to ensure allocation by CYLINDER. If you use record allocation for more than a cylinder, cylinder allocation is used. Cylinder allocation can reduce the time required to do SQL mass inserts and to perform LOGONLY recovery; it does not affect the time required to recover a table space from an image copy or to run the REBUILD utility.

When inserting records, DB2 pre-formats space within a page set as needed. The allocation amount, which is either CYLINDER or TRACK, determines the amount of space that is pre-formatted at any one time.

Because less space is pre-formatted at one time for the TRACK allocation amount, a mass insert can take longer when the allocation amount is TRACK than the same insert when the allocation amount is CYLINDER. However, smart secondary space allocation minimizes the difference between TRACK and CYLINDER.

The allocation amount is dependent on device type and the number of bytes you specify for PRIQTY and SECQTY when you define table spaces and indexes. The default SECQTY is 10% of the PRIQTY, or 3 times the page size, whichever is

larger. This default quantity is an efficient use of storage allocation. Choosing a SECQTY value that is too small in relation to the PRIQTY value results in track allocation.

For more information about how space allocation amounts are determined, see the description of the DEFINE CLUSTER command in *DFSMS/MVS™: Access Method Services for the Integrated Catalog*.

## Pre-formatting during LOAD or REORG

When DB2 pre-formatting delays impact the performance or execution time consistency of applications that do heavy insert processing, and if the table size can be predicted for a business processing cycle, consider using the PREFORMAT option of LOAD and REORG. If you preformat during LOAD or REORG, DB2 does not have to preformat new pages during execution. When the pre-formatted space is used and when DB2 has to extend the table space, normal data set extending and pre-formatting occurs.

Consider pre-formatting only if pre-formatting is causing a measurable delay with the insert processing or causing inconsistent elapsed times for insert applications.

**Recommendation:** Quantify® the results of pre-formatting in your environment by assessing the performance both before and after using pre-formatting.

## Avoiding excessively small extents

Data set extent size affects performance because excessively small extents can degrade performance during a sequential database scan.

### Example

Suppose that the sequential data transfer speed is 100 MB per second and that the extent size is 10 MB. The sequential scan must move to a new extent ten times per second.

**Recommendation:** Maintain extent sizes that are large enough to avoid excessively frequent extent moving during scans. Keep the extent size greater than 10 cylinders for large data sets.

### Maximum number of extents

An SMS-managed linear data set is limited to 123 extents on a volume and 7257 total extents on all volumes. A non-SMS-managed data set is limited to 123 extents on a volume and 251 total extents on all volumes. If a data set grows and extents are not monitored, jobs eventually fail due to these extent limitations.

**Recommendation:** Monitor the number of extents to avoid reaching the maximum number of extents on a volume and the maximum number of extents on all volumes.

### Specifying primary quantity for nonpartitioned indexes

Specifying sufficient primary and secondary allocations for frequently used data sets minimizes I/O time, because the data is not located at different places on the disks.

| Listing the catalog or VTOC occasionally to determine the number of secondary allocations that have been made for your more frequently used data sets can also be helpful. Alternatively, you can use IFCID 0258 in the statistics class 3 trace and real time statistics to monitor data set extensions. OMEGAMON monitors IFCID 0258.

**GUIP** To prevent wasted space for non-partitioned indexes, take one of the following actions:

- Let DB2 use the default primary quantity and calculate the secondary quantities. Do this by specifying 0 for the IXQTY subsystem parameter, and by omitting a PRIQTY and SECQTY value in the CREATE INDEX statement or ALTER INDEX statement. If a primary and secondary quantity were previously specified for an index, you can specify PRIQTY -1 and SECQTY -1 to change to the default primary quantity and calculated secondary quantity.
- If the MGEXTSZ subsystem parameter is set to NO, so that you control secondary space allocations, make sure that the value of  $\text{PRIQTY} + (N \times \text{SECQTY})$  is a value that evenly divides into PIECESIZE.

**GUIP**

**Related concepts**

“Real-time statistics” on page 175



---

## Chapter 5. Controlling resource usage

When system resources are shared among transactions, end user queries, Web requests, distributed application requests, and batch programs, you need to control how those resources are used, separate data, and set priorities carefully.

You might choose to emphasize resource use, performance, concurrency, or data security. Many of the things you currently do to improve response time or reduce processor consumption for a single DB2 subsystem also apply in the data sharing environment.

Choose the controls that best match your goals. For example, you might want to:

- Minimize resource usage
- Maximize throughput
- Maximize response time
- Ensure a certain level of service to some users
- Avoid conflicts between users

Consequently, Your goal might be to favor a certain class of users or to achieve the best overall system performance.

### **Related concepts**

Chapter 18, “Tuning distributed applications,” on page 403

Chapter 12, “Managing DB2 threads,” on page 137

---

## Facilities for controlling resource usage

DB2 includes a resource limit facility (governor), which helps control the use of system resources by DB2. Other facilities, such as z/OS workload management (WLM), and the DB2 QMF governor, complement the DB2 governor.

DB2 includes a resource limit facility (governor), which helps control the use of system resources by DB2. Other facilities, such as z/OS workload management (WLM), and the DB2 QMF governor, complement the DB2 governor. Because DB2 is integrated with the operating system and transaction subsystems, control definitions are used in most cases. This integration simplifies the task of controlling resources for the user.

Each of the objectives presented in the table below is matched with a control facility that you can use to achieve the objective.

*Table 1. Controlling the use of resources*

<b>Objective</b>	<b>Control facility</b>
Prioritizing resources	z/OS workload management
Limiting resources for each job	Time limit on job or step (through z/OS system settings or JCL)
Limiting resources for TSO sessions	Time limit for TSO logon
Limiting resources for IMS and CICS	IMS and CICS controls

Table 1. Controlling the use of resources (continued)

Objective	Control facility
Limiting resources for a stored procedure	ASUTIME column of SYSIBM.SYSROUTINES catalog table.
Limiting dynamic statement execution time	DB2 QMF governor and DB2 resource limit facility
Reducing locking contention	DB2 locking parameters, DISPLAY DB LOCKS, lock trace data, database design
Evaluating long-term resource usage	Accounting trace data, OMEGAMON reports
Predicting resource consumption	DB2 EXPLAIN statement, Visual Explain, predictive governing capability
Controlling use of parallelism	DB2 resource limit facility, SET CURRENT DEGREE statement

## Prioritizing resources

z/OS workload management (WLM) controls the execution of DB2 work based on the priorities that you set.

For more information about setting priorities on work, see *z/OS MVS Initialization and Tuning Guide*.

In CICS environments without the Open Transaction Environment (OTE) function, DB2 work and application work is performed in different tasks. DB2 work is managed at the subtask level. With CICS OTE, DB2 work and application work can be performed in the same task. You can manage the DB2 subtasks through various settings in the CICS resource definition online (RDO). Without OTE, some overhead is incurred for each task switch. Therefore, depending on the SQL activity, CICS OTE can improve performance significantly because of the reduction of needing to switch tasks.

In other environments such as batch and TSO, which typically have a single task requesting DB2 services, the task-level processor dispatching priority is irrelevant. Access to processor and I/O resources for synchronous portions of the request is governed solely by WLM.

## Limiting resources for each job

You can control the total amount of processor resources used for a job, instead of the amount used by a single query.

Because most of resource usage occurs within the standard job structure, you can control processor usage at the job level. Refer to the *z/OS MVS JCL User's Guide* for more information about setting resource limits. use this control.

To control the total amount of resources used for a job:

Change the TIME parameter for the job or step. The time limit applies even when DB2 is sorting the result rows. If the time limit is exceeded, the job step abends, and any uncommitted work is rolled back.

---

## Limiting resources for TSO sessions

You can control the amount of resources used for an entire TSO session.

Time limits can apply to either TSO sessions or to batch jobs. Your z/OS system programmer can provide a time parameter on the logon procedure or on a job statement in the logon pre-prompt exit. This time limit is for the session, rather than for an individual query or a single program. If you want to control the amount of resources used for an entire TSO session, rather than the amount used by a single query, use this control.

You can find more information about setting the resource limit for a TSO session in these manuals:

- *z/OS TSO/E Programming Guide*
- *z/OS TSO/E Customization*

---

## Limiting resources for IMS and CICS

You can use various IMS commands (including PROCLIM, or processing limit) to limit the resources used by a transaction, a class, a program, a database, or a subsystem.

For more information, see *IMS Command Reference*.

For a detailed description of performance factors in a CICS system, see *CICS Transaction Server for z/OS Performance Guide*.

---

## Limiting resources for a stored procedure

DB2 stored procedures are especially designed for high volume online transactions.

For information about controlling the amount of storage used by stored procedures address spaces, see “Controlling address space storage” on page 109.

To establish limits for stored procedures:

Take one of the following actions:

- Set a processor limit for each stored procedure, by updating the ASUTIME column of the SYSIBM.SYSROUTINES catalog table. This limit allows DB2 to cancel procedures that loop.
- Set a limit for the maximum number of times that a procedure can terminate abnormally, by specifying a value in the MAX ABEND COUNT field on installation panel DSNTIPX. This limit is a system limit that applies to all stored procedures and prevents a problem procedure from overwhelming the system with abend dump processing.
- Set a limit for the maximum number of times that a specific procedure can terminate abnormally, by specifying the STOP AFTER FAILURES option on the ALTER or CREATE PROCEDURE statement. This limit allows you to override the system limit specified in MAX ABEND COUNT and specify different limits for different procedures.

---

## Setting limits for system resource usage

The DB2 resource limit facility (governor) enables you to limit resource usage, bind operations, and parallelism modes in dynamic query processing.

You can use two modes of governing to limit the amount of system resources that a dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE query uses. These modes are called reactive and predictive. You can use either mode or both together.

You can use the resource limit facility to perform the following activities:

- Set warning and error thresholds for dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements. The resource limit facility can inform users (via your application programs) that a processing limit might be exceeded for a particular dynamic SQL statement. This is called *predictive governing*.
- Stop dynamic SQL statements (SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE ) that exceed the processor limit that you specified. This is sometimes called *reactive governing*.

The resource limit facility does not control static SQL statements regardless of whether they are executed locally or remotely, and no limits apply to primary or secondary authorization IDs with installation SYSADM or installation SYSOPR authority.

- Restrict bind and rebind activities to avoid performance impacts on production data.
- Restrict bind and rebind activities to avoid performance impacts on production data.

## Using reactive governing

You can use the DB2 resource limit facility to set limits for *reactive governing*, meaning that DB2 actually stops a dynamic query that overuses system resources.

When you specify reactive governing, the resource limit facility stops a currently running query that meets the conditions in a resource limit table row when the query uses more than the maximum amount of resources specified by the value of ASUTIME in that row. No limits apply to primary or secondary authorization IDs with installation SYSADM or installation SYSOPR authority. To specify reactive governing:

Specify either of the following values in the RLFFUNC column of the resource limit table:

- blank** Govern by plan name (RLST)
- '2'** Govern by package name (RLST)
- '8'** Govern by client information (RLMT)

Any statement that exceeds a limit that you set in the RLST terminates with a -905 SQLCODE and a corresponding '57014' SQLSTATE. You can establish a single limit for all users, different limits for individual users, or both. Limits do not apply to primary or secondary authorization IDs with installation SYSADM or installation SYSOPR authority. For queries that enter DB2 from a remote site, the local site limits are used.

## Using predictive governing

The DB2 provides a *predictive governing* capability that avoids wasting processing resources by giving you the ability to prevent a query from running when it

appears likely to exceed processing limits. In reactive governing, those resources are already used before the query is stopped.

The following figure provides an overview of how predictive governing works.

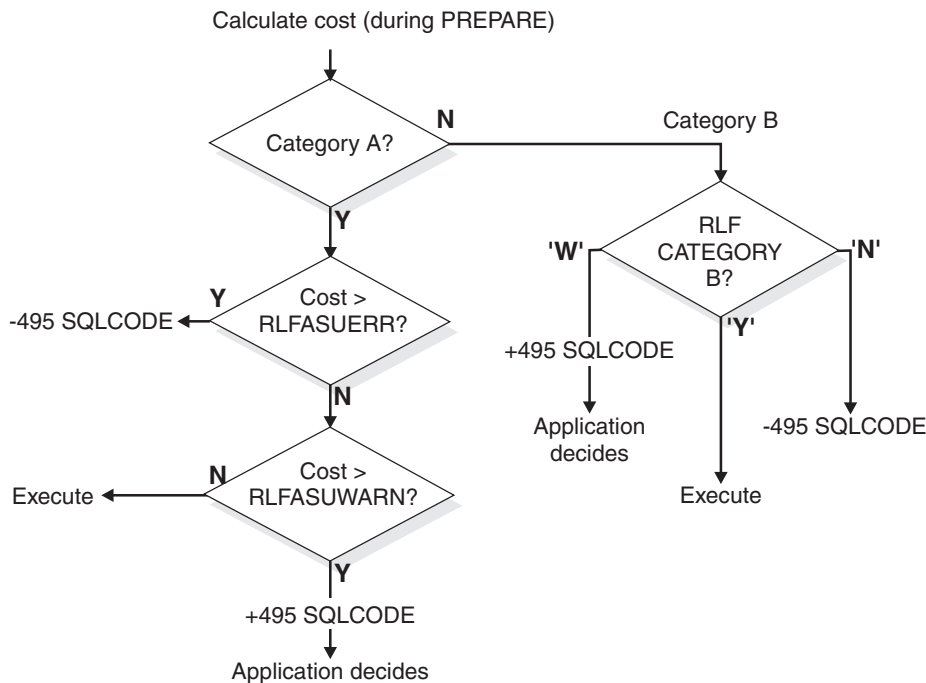


Figure 1. Processing for predictive governing

At prepare time for a dynamic SELECT, INSERT UPDATE, MERGE, or DELETE statement, DB2 searches the active resource limit table to determine if the processor cost estimate exceeds the error or warning threshold that you set in RLFASUWARN and RLFASUERR columns for that statement. DB2 compares the cost estimate for a statement to the thresholds that you set, and the following actions occur:

- If the cost estimate is in cost category A and the error threshold is exceeded, DB2 returns a -495 SQLCODE to the application, and the statement is not prepared or run.
- If the estimate is in cost category A and the warning threshold is exceeded, a +495 SQLCODE is returned at prepare time. The prepare is completed, and the application or user decides whether to run the statement.
- If the estimate is in cost category B, DB2 takes the action that you specify in the RLF\_CATEGORY\_B column; that is, it either prepares and executes the statement, does not prepare or execute the statement, or returns a warning SQLCODE, which lets the application decide what to do.
- If the estimate is in cost category B and the warning threshold is exceeded, a +495 SQLCODE is returned at prepare time. The prepare is completed, and the application or user decides whether to run the statement.

To specify predictive governing:

Specify either of the following values in the RLFFUNC column of a resource limit table:

'6' Govern by plan name (RLST)

'7' Govern by package name (RLST)

'9' Govern by client information (RLMT)

For more information about how to qualify rows in resource limit tables, see “How DB2 qualifies RLST rows” on page 32 and “How DB2 qualifies RLMT rows” on page 32

The following table is an RLST with two rows that use predictive governing.

Table 2. Predictive governing example

RLFFUNC	AUTHID	RLFCOLLN	RLFPKG	RLFASUWARN	RLFASUERR	RLF_CATEGORY_B
7	(blank)	COLL1	C1PKG1	900	1500	Y
7	(blank)	COLL2	C2PKG1	900	1500	W

The rows in the RLST for this example cause DB2 to act as follows for all dynamic INSERT, UPDATE, DELETE, SELECT, and MERGE statements in the packages that are listed in this table (C1PKG1 and C2PKG1):

- Statements in cost category A that are predicted to be less than 900 SUs are executed.
- Statements in cost category A that are predicted to be between 900 and 1500 SUs receive a +495 SQLCODE.
- Statements in cost category A that are predicted to be greater than 1500 SUs receive SQLCODE -495, and the statement is not executed.

## Combining reactive and predictive governing

You can govern a dynamic SQL statement both before and after the statement is executed.

For example, if the processing cost estimate is in cost category B and you decide to run the statement, you can use a resource limit table to terminate the statement after a certain amount of processor time, the same as it does today. To use both modes of governing, you need two rows in the resource limit table as shown in the following table.

Table 3. Combining reactive and predictive governing

RLFFUNC	AUTHID	PLANNAME	ASUTIME	RLFASUWARN	RLFASUERR	RLF_CATEGORY_B
6	USER1	PLANA	0	800	1000	W
(blank)	USER1	PLANA	1100	0	0	(blank)

The rows in the RLST for this example cause DB2 to act as follows for a dynamic SQL statement that runs under PLANA:

### Predictive mode

- If the statement is in COST\_CATEGORY A and the cost estimate is greater than 1000 SUs, USER1 receives SQLCODE -495 and the statement is not executed.
- If the statement is in COST\_CATEGORY A and the cost estimate is greater than 800 SUs but less than 1000 SUs, USER1 receives SQLCODE +495.
- If the statement is in COST\_CATEGORY B, USER1 receives SQLCODE +495.

### Reactive mode

In either of the following cases, a statement is limited to 1100 SUs:

- The cost estimate for a statement in COST\_CATEGORY A is less than 800 SUs
- The cost estimate for a COST\_CATEGORY A is greater than 800 and less than 1000 or is in COST\_CATEGORY B and the user chooses to execute the statement

## Limiting resource usage for plans or packages

You can limit the processor resources used by dynamic queries to improve performance.

A resource limit specification table (RLST) allows you to specify processor limits that are based on the collection, plan name or package name, authorization ID, and the location associated with the dynamic queries. You can specify reactive and predictive governing in an RLST.

Governing by plan name and package name are mutually exclusive:

- The resource limit facility governs the DBRMs in the MEMBER list specified on the BIND PLAN command. The RLFFUNC, RLFCOLLN, and RLFPKG columns must contain blanks. The following table shows an example of this:

Table 4. Qualifying rows by plan name

RLFFUNC	AUTHID	PLANNAME	LUNAME	ASUTIME
(blank)	JOE	PLANA	(blank)	(null)
(blank)	(blank)	WSPLAN	SAN_JOSE	15000
(blank)	(blank)	(blank)	PUBLIC	10000

The first row in the table above shows that when Joe runs PLANA at the local location, no limits restrict any dynamic statements in that plan. The second row shows that if anyone runs WSPLAN from SAN\_JOSE, the dynamic statements in that plan are restricted to 15 000 SUs each. The third row is entered as a cap for any unknown authorization IDs or plan names from any location in the network, including the local location. (An alternative would be to let the default values on installation panel DSNTIPR and DSNTIPO serve as caps.)

- Collection and package name

The resource limit facility governs the packages used during the execution of the SQL application program. PLANNAME must contain blank, and RLFFUNC must contain '2'. The following table is an example of this:

Table 5. Qualifying rows by collection or package name

RLFFUNC	AUTHID	RLFCOLLN	RLFPKG	LUNAME	ASUTIME
2	JOE	DSNESPSCS	(blank)	(blank)	40000
2	(blank)	(blank)	DSNESM68	PUBLIC	15000

The first row in the table above shows that when Joe runs any package in collection DSNESPSCS from the local location, dynamic statements are restricted to 40 000 SUs.

The second row indicates that if anyone from any location (including the local location) runs SPUIFI package DSNESM68, dynamic statements are limited to 15 000 SUs.

Populate a resource limit table with the appropriate values.

## How DB2 qualifies RLST rows

DB2 tries to find the closest match when determining limits for a particular dynamic statement.

DB2 uses the following search order:

1. Exact match
2. Authorization ID
3. Plan name, or collection name and package name
4. LU name
5. No row match

## When no row matches

If no row in the RLST matches the currently executing statement, DB2 uses the default value that is set on the RLST

The ACCESS ERROR field of installation panel DSNTIPO (for queries that originate locally) or DSNTIPR (for queries that originate remotely). This default value applies to reactive governing only.

For predictive governing, if no row matches, no predictive governing occurs.

## Limiting resource usage by client information for middleware servers

You can limit resource usage by client information to improve the performance of middleware servers that run dynamic queries.

Resource limit middleware tables (RLMTs) allow you to specify processor limits that are based on client information, such as the application name, end user ID, workstation ID, and IP address, associated with dynamic queries that run on middleware servers. By creating and populating an RLMT, you can limit queries reactively and predictively.

The following table shows examples of reactive and predictive client-based query limits in an RLMT:

Table 6. Qualifying rows for DNSRLMTxx

RLFFUNC	RLFEUAN	RLFEUID	RLFEUWN	RLFIP	ASUTIME
8	APP1	PAUL	(blank)	9.30.72.223	12000
9	(blank)	(blank)	WORKSTA10	(blank)	7000

The first row in the above table shows that when Paul runs the APP1 application from the 9.30.72.223 IP address, the resource limit facility limits dynamic queries run by APP1 to 12000 SUs each.

The second row shows that any queries that are requested from work station 10 are predictively limited to 7000 SUs each.

## How DB2 qualifies RLMT rows

DB2 tries to find the closest match when determining limits for a particular dynamic statement.

DB2 uses the following order of match the rows in an RLMT:

1. Exact match
2. Application name
3. User ID
4. Workstation name
5. IP address
6. No row match

### When no row matches

If no row in the RLMT matches the currently executing statement, DB2 uses the default set on the RLMT ACCESS ERROR field of installation panel DSNTIPO (for queries that originate locally) or DSNTIPR (for queries that originate remotely). This default applies to reactive governing only.

For predictive governing, if no row matches, no predictive governing occurs.

## Managing resource limit tables

You can use *resource limit tables* to specify resource usage limits for dynamic queries that run in DB2.

Resource limit tables allow you to limit the amount of processor resources, in service units, used by a dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE SQL statement. You can specify reactive or predictive governing, or use a combination of reactive and predictive limits in the same resource limit table.

When you install DB2, installation job DSNTIJSG creates a database, table space, table, and descending index for the resource limit specification. You can tailor those statements.

To create a new resource limit specification table, you must have sufficient authority to define objects in the DSNRLST database and to specify *authid*, which is the authorization ID specified on field RESOURCE AUTHID of installation panel DSNTIPP.

You can create two different types of resource limit tables, *resource limit specification tables* (RLST) and *resource limit middleware tables* (RLMT). You can create instances of either type of table, or instances of both, depending on your specific plan for limiting resources.

If you are a system administrator, you must determine how your location intends to use the resource limit facility and create several local procedures:

- For creating and maintaining your resource limit tables
- For establishing limits for any newly written applications
- For console operators, such as switching resource limit tables every day at a certain time

Resource limit tables can reside in any database; however, because a database has some special attributes while the resource limit facility is active, it is best to create resource limit tables in their own database. Only one resource limit table can be active at any particular time, however you might create several instances of either or both types of resource limit tables and use different instances at different times.

## Creating RLST resource limit tables

You can limit queries based on plan, collection, package, authorization ID, and location ID by creating an RLST named *authid.DSNRLSTxx* table.

To use this table, you must also create index named *authid.DSNARLxx* in the DSNRLST database. *xx* is any two-character alphanumeric value, and *authid* is specified when DB2 is installed. Because the two characters *xx* must be entered as part of the START command, they must be alphanumeric—no special or DBCS characters.

To create a DSNRLST resource limit specification table, use the following statement:

Issue the following statement:

```
CREATE TABLE authid.DSNRLSTxx
  (AUTHID  VARCHAR(128) NOT NULL WITH DEFAULT,
   PLANNAME CHAR(8) NOT NULL WITH DEFAULT,
   ASUTIME INTEGER,
   -----3-column format -----
   LUNAME  CHAR(8) NOT NULL WITH DEFAULT,
   -----4-column format -----
   RLFFUNC CHAR(1) NOT NULL WITH DEFAULT,
   RLFBIND CHAR(1) NOT NULL WITH DEFAULT,
   RLFCOLLN VARCHAR(128) NOT NULL WITH DEFAULT,
   RLFPKG  VARCHAR(128) NOT NULL WITH DEFAULT),
   -----8-column format -----
   RLFASUERR INTEGER,
   RLFASUWARN INTEGER,
   RLF_CATEGORY_B CHAR(1) NOT NULL WITH DEFAULT)
   -----11-column format -----
   IN DSNRLST.DSNRLSxx;
```

### RLST column descriptions:

Here is a complete description for all columns for the RLST type resource limit specification table.

In no case can all columns in a particular row be populated; the columns you must populate depend on what function is performed by that row (determined by the value in RLFFUNC) and how narrowly you want to qualify values. For example, you can qualify broadly by leaving the AUTHID column blank, which means that the row applies to all authorization IDs. Or, you can qualify very narrowly by specifying a different row for each authorization ID for which the function applies.

**Search order:** DB2 tries to find the most exact match when it determines which row to use for a particular function. The search order depends on which function is being requested (type of governing, bind operations, or parallelism restrictions). The search order is described under each of those functions.

### AUTHID

The resource specification limits apply to this primary authorization ID. A blank means that the limit specifications in this row apply to all authorization IDs for the location that is specified in LUNAME. No limits apply to primary or secondary authorization IDs with installation SYSADM or installation SYSOPR authority.

### PLANNAME

The resource specification limits apply to this plan. If you are specifying a function that applies to plans (RLFFUNC=blank or '6'), a blank means that

the limit specifications in this row apply to all plans for the location that is specified in LUNAME. Qualify by plan name only if the dynamic statement is issued from a DBRM bound in a plan, not a package; otherwise, DB2 does not find this row. If the RLFFUNC column contains a function for packages ('1,' '2,' or '7'), this column must be blank; if it is not blank, the row is ignored.

#### ASUTIME

The number of processor service units allowed for any single dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statement. Use this column for reactive governing.

Other possible values and their meanings are:

**null** No limit

#### 0 (zero) or a negative value

No dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements are permitted.

A relative metric is used so that the resource limit table values do not need to be modified when processors are changed. However, in some cases, DB2 workloads can differ from the measurement averages. In these cases, resource limit table value changes might be necessary. For information about how to calculate service units, see "Calculating service unit values for resource limit tables" on page 42.

#### LUNAME

The LU name of the location where the request originated. A blank value in this column represents the local location, *not* all locations. The value PUBLIC represents all of the DBMS locations in the network; these locations do not need to be DB2 subsystems. PUBLIC is the only value for TCP/IP connections.

#### RLFFUNC

Specifies how the row is used. The values that have an effect are:

**blank** The row reactively governs dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements by plan name.

**'1'** The row reactively governs bind operations.

**'2'** The row reactively governs dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements by package or collection name.

**'3'** The row disables query I/O parallelism.

**'4'** The row disables query CP parallelism.

**'5'** The row disables Sysplex query parallelism.

**'6'** The row predictively governs dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements by plan name.

**'7'** The row predictively governs dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements by package or collection name.

All other values are ignored.

#### RLFBIND

Shows whether bind operations are allowed. An 'N' implies that bind

operations are not allowed. Any other value means that bind operations are allowed. This column is used only if RLFFUNC is set to '1'.

#### **RLFCOLLN**

Specifies a package collection. A blank value in this column means that the row applies to all package collections from the location that is specified in LUNAME. Qualify by collection name only if the dynamic statement is issued from a package; otherwise DB2 does not find this row. If RLFFUNC=blank, '1', or '6', RLFCOLLN must be blank.

#### **RLFPKG**

Specifies a package name. A blank value in this column means that the row applies to all packages from the location that is specified in LUNAME. Qualify by package name only if the dynamic statement is issued from a package; otherwise DB2 does not find this row. If RLFFUNC=blank, '1', or '6', RLFPKG must be blank.

#### **RLFASUERR**

Used for predictive governing (RLFFUNC= '6' or '7'), and only for statements that are in cost category A. The error threshold number of system resource manager processor service units allowed for a single dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statement. If the predicted processor cost (in service units) is greater than the error threshold, an SQLCODE -495 is returned to the application.

Other possible values and their effects are:

**null** No error threshold

##### **0 (zero) or a negative value**

All dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements receive SQLCODE -495.

#### **RLFASUWARN**

Used for predictive governing (RELFFUNC= '6' or '7'), and only for statements that are in cost category A. The warning threshold number of processor service units that are allowed for a single dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statement. If the predicted processor cost (in service units) is greater than the warning threshold, an SQLCODE +495 is returned to the application.

Other possible values and their effects are:

**null** No warning threshold

##### **0 (zero) or a negative value**

All dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements receive SQLCODE +495.

**Important:** Make sure the value for RLFASUWARN is less than that for RLFASUERR. If the warning value is higher, the warning is never reported. The error takes precedence over the warning.

#### **RLF\_CATEGORY\_B**

Used for predictive governing (RLFFUNC='6' or '7'). Tells the governor the default action to take when the cost estimate for a given statement falls into cost category B, which means that the predicted cost is indeterminate and probably too low. You can tell if a statement is in cost category B by running EXPLAIN and checking the COST\_CATEGORY column of the DSN\_STATEMNT\_TABLE.

The acceptable values are:

- blank** By default, prepare and execute the SQL statement.
- Y** Prepare and execute the SQL statement.
- N** Do not prepare or execute the SQL statement. Return SQLCODE -495 to the application.
- W** Complete the prepare, return SQLCODE +495, and allow the application logic to decide whether to execute the SQL statement or not.

### Creating the DSNARL index:

A DSNARL index works in conjunction with an RLST resource limit specification table.

To create the index:

Issue the following statement:

```
CREATE UNIQUE INDEX authid.DSNARLxx
ON authid.DSNRLSTxx
(RLFFUNC, AUTHID DESC, PLANNAME DESC,
RLFCOLLN DESC, RLFPKG DESC, LUNAME DESC)
CLUSTER CLOSE NO;
```

The *xx* in the index name (DSNARL*xx*) must match the *xx* in the table name (DSNRLST*xx*), and it must be a descending index.

### Creating an RLMT resource limit table

You can use client information to govern dynamic statements that run on middleware servers by creating an *authid*.DSNRLMT*xx* table.

To use this table, you must also create an index named *authid*.DSNMRL*xx* in the DSNRLST database, where *xx* is any two-character alphanumeric value, and *authid* is specified when DB2 is installed. Because the two characters *xx* must be entered as part of the START command, they must be alphanumeric—no special or DBCS characters.

To create an RLMT resource limit table

Issue the following statement:

```
CREATE TABLE authid.DSNRLMTxx (RLFFUNC CHAR(1) NOT NULL WITH DEFAULT,
RLFEUAN CHAR(32) NOT NULL WITH DEFAULT,
RLFEUID CHAR(16) NOT NULL WITH DEFAULT,
FLFEUWN CHAR(18) NOT NULL WITH DEFAULT,
RLFIP CHAR(254) NOT NULL WITH DEFAULT)
ASUTIME INTEGER,
RLFASUERR INTEGER,
RLFASUWARN INTEGER,
RLF_CATEGORY_B CHAR(1) NOT NULL WITH DEFAULT)
IN DSNRLST.DSNRLSxx;
```

### RLMT column descriptions:

Here is a complete description for all columns in the RLMT.

The columns that you must populate depend on what function is performed by that row (determined by the value in RLFFUNC) and how narrowly you want to qualify values. For example, you can qualify broadly by leaving the RLFEUAN

column blank, which means that the row applies to all end user IDs. Or, you can qualify very narrowly by specifying a different row for each end user ID for which the function applies.

### Search order

DB2 tries to find the most exact match when it determines which row to use for a particular function. The search order depends on which function is being requested (reactive or predictive governing). The search order is described under each of those functions.

*Table 7. Columns of a RLMT resource limit specification table*

Column name	Description
RLFFUNC	Specifies how the row is used. The values that have an effect are: <ul style="list-style-type: none"> <li><b>'8'</b> The row reactively governs dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements by client information (RLEUID, RLFEUAN, RLFEUWN, and RLFIP).</li> <li><b>'9'</b> The row predictively governs dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements by client information (RLEUID, RLFEUAN, RLFEUWN, and RLFIP).</li> </ul> All other values are ignored.
RLFEUAN	Specifies an application name. A blank value in this column means that the row applies to all application names from the location specified in RLFIP.
RLEUID	Specifies an end user's user ID. A blank value means that the limit specifications in this row apply to every user ID for the location that is specified in RLFIP.
RLFEUWN	Specifies an end user's workstation name. A blank value in this column means that the row applies to all workstation names from the location that is specified in RLFIP.
RLFIP	The IP address of the location where the request originated. A blank value in this column represents all locations.
ASUTIME	The number of processor service units allowed for any single dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statement. Use this column for reactive governing. <p>Other possible values and their meanings are:</p> <ul style="list-style-type: none"> <li><b>null</b> No limit</li> <li><b>0 (zero) or a negative value</b> No dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements are permitted.</li> </ul> <p>A relative metric is used so that the resource limit table values do not need to be modified when processors are changed. However, in some cases, DB2 workloads can differ from the measurement averages. In these cases, resource limit table value changes might be necessary.</p>

Table 7. Columns of a RLMT resource limit specification table (continued)

Column name	Description
RLFASUERR	<p>Used for predictive governing (RLFFUNC= '9'), and only for statements that are in cost category A. The error threshold number of system resource manager processor service units allowed for a single dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statement. If the predicted processor cost (in service units) is greater than the error threshold, an SQLCODE -495 is returned to the application</p> <p>Other possible values and their effects are:</p> <p><b>null</b> No error threshold</p> <p><b>0 (zero) or a negative value</b> All dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements receive SQLCODE -495.</p>
RLFASUWARN	<p>Used for predictive governing (RELFFUNC= '9'), and only for statements that are in cost category A. The warning threshold number of processor service units that are allowed for a single dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statement. If the predicted processor cost (in service units) is greater than the warning threshold, an SQLCODE +495 is returned to the application.</p> <p>Other possible values and their effects are:</p> <p><b>null</b> No warning threshold</p> <p><b>0 (zero) or a negative value</b> All dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements receive SQLCODE +495.</p> <p><b>Important:</b> Make sure the value for RLFASUWARN is less than that for RLFASUERR. If the warning value is higher, the warning is never reported. The error takes precedence over the warning.</p>
RLF_CATEGORY_B	<p>Used for predictive governing (RLFFUNC='9'). Tells the governor the default action to take when the cost estimate for a given statement falls into cost category B, which means that the predicted cost is indeterminate and probably too low. You can tell if a statement is in cost category B by running EXPLAIN and checking the COST_CATEGORY column of the DSN_STATEMNT_TABLE.</p> <p>The acceptable values are:</p> <p><b>blank</b> By default, prepare and execute the SQL statement.</p> <p><b>Y</b> Prepare and execute the SQL statement.</p> <p><b>N</b> Do not prepare or execute the SQL statement. Return SQLCODE -495 to the application.</p> <p><b>W</b> Complete the prepare, return SQLCODE +495, and allow the application logic to decide whether to execute the SQL statement or not.</p>

### Creating the DSNMRL index:

A DSNMRL index works in conjunction with an RLMT resource limit specification table.

To create a DSNMRL index:

Issue the following SQL statement:

```
CREATE UNIQUE INDEX authid.DSNMRLxx
ON authid.DSNRLMTxx
(RLFFUNC, RFLEUAN DESC, RLFEUID DESC,
RLFEUWN DESC, RLFLIP DESC)
CLUSTER CLOSE NO;
```

The *xx* in the index name (DSNMRL*xx*) must match the *xx* in the table name (DSNRLMT*xx*) and it must be a descending index.

## Populating a resource limit table

You enable the resource limit facility by populating the resource limit tables with rows of data that describe the limits.

To insert, update, merge, or delete data in the resource limit table, you need only the usual table privileges on the resource limit table. No higher authority is required.

Use the SQL statements INSERT, UPDATE, MERGE, and DELETE to populate the resource limit specification table. The limit that exists when a job makes its first dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statement applies throughout the life of the job. If you update the resource limit specification table while a job is executing, that limit for that job does not change. Instead, the updates are effective for all new jobs and for those that have not issued their first dynamic SELECT, INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statement.

## Starting and stopping a resource limit table

You can create several resource limit tables and start and stop them as needed to support the type and amount of processing that occurs at different times.

At installation time, you can specify a default resource limit table to be used each time that DB2 is restarted.

If the governor is active and you restart it without stopping it, any jobs that are active continue to use their original limits, and all new jobs use the limits in the new table.

If you stop the governor while a job is executing, the job runs with no limit, but its processing time continues to accumulate. If you later restart the governor, the new limit takes effect for an active job only when the job passes one of several internal checkpoints. A typical dynamic statement, which builds a result table and fetches from it, passes those checkpoints at intervals that can range from moments to hours. As a result, your change to the governor might not stop an active job within the time you expect.

To activate any particular resource limit table:

- Use the DB2 command START RLIMIT ID=*xx* where *xx* is the two-character identifier that you specified on the name DSNRLST*xx* or DSNRLMT*xx*. This command gives you the flexibility to use a different resource limit tables at different times; however, only one pair of resource limit tables, one RLST and one RLMT, can be active at any given time.
- Use the DB2 command CANCEL THREAD to stop an active job that does not pick up the new limit when you restart the governor.

You can use different resource limit tables for the day shift and the evening shift, as shown in the following tables.

Table 8. Example of resource limit table for the day shift

AUTHID	PLANNAME	ASUTIME	LUNAME
BADUSER		0	LUDBD1
ROBYN		100000	LUDBD1
	PLANA	300000	LUDBD1
		50000	LUDBD1

Table 9. Example of RLST for the night shift. During the night shift AUTHID, ROBYN, and all PLANA users from LUDBD1 run without limit.

AUTHID	PLANNAME	ASUTIME	LUNAME
BADUSER		0	LUDBD1
ROBYN		NULL	LUDBD1
	PLANA	NULL	LUDBD1
		50000	LUDBD1

## Restricted activity on resource limit tables

While the resource limit facility (governor) is active, you cannot execute certain SQL statements on the resource limit table, or the table space and database that contain the resource limit table

The statements that are restricted while the resource limit facility is active are:

- DROP DATABASE
- DROP INDEX
- DROP TABLE
- DROP TABLESPACE
- RENAME TABLE

You cannot stop a database or table space that contains an active resource limit table; nor can you start the database or table space with ACCESS(UT).

## Governing statements from a remote site

Several important guidelines and restrictions apply when you use the resource limit facility in a distributed processing environment.

When you use the resource limit facility in conjunction with distributed processing:

Remember the following guidelines:

Option	Description
Dynamic statements from requesters that use DRDA <sup>®</sup> protocols	You can create an RLMT to govern by client information (RLFFUNC=8 and RLFFUNC=9) such as application name, user ID, workstation ID, and IP address. If you use an RLST, you must govern by package name (RLFFUNC=2 or RLFFUNC=7), which means that PLANNAME must be blank. Specify the originating system's LU name in the LUNAME column, or specify PUBLIC for all remote LUs. If you leave LUNAME blank, DB2 assumes that you mean the local location only and none of your incoming distributed requests qualify.
Dynamic statements from requesters that use DB2 private protocol	You must govern by plan name (RLFFUNC=(blank) or '6'), which means that RLFCOLLN and RLFPKG must be blank. Specify the LU name of the originating system in the LU column, or specify PUBLIC for all remote LUs. Again, a value other than PUBLIC takes precedence over PUBLIC. PLANNAME can be blank or the name of the plan created at the requester's location. RLFPKG and RLFCOLLN must be blank.
Dynamic statements coming from requesters that use TCP/IP	You cannot specify the LU name. You must use PUBLIC.
No row is present in the resource limit table to govern access from a remote location	The limit is the default set on the RLST ACCESS ERROR field of installation panel DSNTIPR.

## Calculating service unit values for resource limit tables

The processing time for a particular SQL statement varies according to the processor that executes it, but the number of service units required, a relative metric, remains roughly constant.

The resource limit facility samples processing time in a relative measurement called *service units*.

A relative metric is used so that you don't have to modify the resource limit table values when processors are changed. The number of service units consumed is not exact between different processors because the calculations for service units are dependent on performance averages measured before the release of a new processor. In some cases, DB2 workloads can differ from the measured averages. In these cases, resource limit table value changes might be necessary.

The easiest way to get SU times for the ASUTIME, RLFASUWARN, or RLFASUERR columns is to use the value in the PROCSU column of DSN\_STATEMNT\_TABLE as your starting point. You can also get the value from the IFCID 0022 record.

However, if you do not have statement table information, or if you have queries for which you have no information, you can use the following formula to calculate SU time:

$SU\ time = processor\ time \times service\ units\ per\ second\ value$

The value for service units per second depends on the processor model. You can find this value for your processor model in *z/OS MVS Initialization and Tuning Guide*, where SRM is discussed.

For example, if processor A is rated at 900 service units per second and you do not want any single dynamic SQL statement to use more than 10 seconds of processor time, you could set ASUTIME as follows:

$ASUTIME\ time = 10\ seconds \times 900\ service\ units/second = 9000\ service\ units$

Later, you could upgrade to processor B, which is rated at 1000 service units per second. If the value you set for ASUTIME remains the same (9000 service units), your dynamic SQL is only allowed 9 seconds for processing time but an equivalent number of processor service units:

$ASUTIME = 9\ seconds \times 1000\ service\ units/second = 9000\ service\ units$

As this example illustrates, after you establish an ASUTIME (or RLFASUWARN or RLFASUERR) for your current processor, you do not need to modify it when you change processors.

## Restricting bind operations

You can use the DB2 resource limit facility to set limits on the amount of resources used by bind operations.

To restrict bind operations:

Use an RLST, specify RLFFUNC='1' and qualify rows by authorization ID and LU name. The same precedence search order applies:

1. AUTHID and LUNAME match.
2. AUTHID matches.
3. LUNAME matches.

A value of PUBLIC for LUNAME applies to all authorization IDs at all locations, while a blank LUNAME governs bind operations for IDs at the local location only.

4. If no entry matches, or if your resource limit table cannot be read, the resource limit facility does not disable bind operations.

### Example

The table below is an example of an RLST that disables bind operations for all but three authorization IDs. Notice that BINDER from the local site is able to bind but that BINDER from San Francisco is not able to bind. Everyone else from all locations, including the local one, is disabled from doing binds.

*Table 10. Restricting bind operations*

RLFFUNC	AUTHID	LUNAME	RLFBIND
1	BINDGUY	PUBLIC	
1	NIGHTBND	PUBLIC	
1	(blank)	PUBLIC	N
1	BINDER	SANFRAN	N
1	BINDER	(blank)	

## Restricting parallelism modes

Resource limit tables allow you restrict and disable parallel processing for dynamic queries.

You might want to do this if, for example, one mode of parallelism for a particular query performs better or one mode for a particular query does not perform well.

To govern query parallelism:

Remember the following guidelines:

- The resource limit facility only governs query parallelism for dynamic queries when the value of the CURRENT DEGREE special register is 'ANY'.
- To disable all query parallelism for a dynamic query, you need rows in your resource limit table to cover all possible modes of parallelism for your system. You need one row with RLFFUNC='3' and one row with RLFFUNC='4', and, if Sysplex query parallelism is possible in your system, you must add an additional row containing RLFFUNC='5'. If parallelism is disabled for a query, the query runs sequentially.
- Qualifying by plan or by package are not separate functions as they are for predictive and reactive governing. If you want to qualify by plan name, the query must be executed from a plan or DB2 cannot find the row. If you want to qualify by package name, the query must be executed from a package or DB2 cannot find the row.

If you want to qualify rows by authorization ID only, you can leave all three columns blank: RLFCOLLN, RLFPKG, and RLFPLAN.

- If no entry can be found in your resource limit table that applies to parallelism, or if your resource limit table cannot be read, the resource limit facility does not disable query parallelism.

### Example

If the RLST in the following table is active, it causes the following effects:

- Disables I/O parallelism for all dynamic queries in IOHOG.
- Disables CP parallelism and Sysplex query parallelism for all dynamic queries in CPUHOG.

Table 11. Example RLST to govern query parallelism

RLFFUNC	AUTHID	LUNAME	RLFCOLLN	RLFPKG
3	(blank)	PUBLIC	blank	IOHOG
4	(blank)	PUBLIC	blank	CPUHOG
5	(blank)	PUBLIC	blank	CPUHOG

---

## The DB2 system monitor

The DB2 system monitor starts automatically and identifies problems related to CPU stalls and DBM1 below-the-bar storage.

The DB2 system monitor looks for CPU stalls that result in latch contention. When it detects a CPU stall, the DB2 system monitor attempts to clear the latch contention by temporarily boosting WLM priority. In addition, the system monitor issues the following messages:

- DSNV508I, to report when DBM1 storage that is below the 2-GB bar reaches critical storage thresholds
- A series of DSNV512I messages to identify the agents that consume the most storage.

Set the xxxxMSTR address space to SYSSTC dispatching priority in WLM to increase the effectiveness of the monitor.

---

## Reducing processor resource consumption

Many factors affect the amount of processor resources that DB2 consumes.

To reduce the consumption of processor resources by DB2:

- Cache authorizations for plans, packages, and routines, such as user-defined functions and stored procedures.
- Use an isolation level of cursor stability and CURRENTDATA(NO) to allow lock avoidance.
- Use the LOCKSIZE clause, when you create or alter a table space. Doing so avoids using excess granularity for locking, such as row-level locking.
- Reorganize indexes and table spaces to improve the performance of access paths. Reorganizing indexes and table spaces is also important after table definition changes, such as changing the data type of a column, so that the data is converted to its new definition. Otherwise, DB2 must track the data and apply the changes as the data is accessed.
- Ensure that your access paths are effective by , and by and .
  - Creating only necessary indexes
  - Updating statistics regularly
  - Rebinding as necessary

### Related concepts

Chapter 31, “When to reorganize indexes and table spaces,” on page 501

Chapter 30, “Gathering monitor statistics and update statistics,” on page 499

### Related tasks

“Specify the size of locks for a table space” on page 372

“Whether to rebind after gathering statistics” on page 736

“Choosing an ISOLATION option” on page 357

## Reusing threads for your high-volume transactions

For high-volume transactions, reusing threads can help performance significantly.

By reusing threads, you can reduce the amount processor resources that DB2 consumes.

To reuse threads:

- For IMS, process multiple input messages in one scheduling of the IMS processing program

- Set PROCLIM to a value greater than 1 and use class priority scheduling to share the cost of thread creation and termination among more than one transaction.
- Reuse threads with wait for input (WFI), or the IMS fast path and class scheduling.
- For CICS, enhance thread reuse through specifications for pool and entry threads in the CICS resource definition online (RDO). You can use Resource Recovery Services attachment facility to reuse threads.

## Minimizing the use of DB2 traces

By suppressing DB2 trace options, you might significantly reduce processing costs.

Using the DB2 trace facility, can consume a large amount of processing resources. Performance trace and global trace are especially resource intensive.

Suppressing other trace options, such as TSO, IRLM, z/OS, IMS, CICS, and other trace options, can also reduce overhead.

To minimize the amount of processor that is consumed by the DB2 trace facility:

Enable only the minimal trace and audit classes that you need. You should enable more detailed traces only when you encounter specific performance problems.

### Turning off global trace

You can reduce processor usage significantly by turning off global trace.

Global trace requires 2% to 100% additional processor utilization. If conditions permit at your site, the DB2 global trace should be turned off.

To turn off global trace:

- Specify NO for the field TRACE AUTO START on panel DSNTIPN at installation.
- If the global trace becomes needed for serviceability, use START TRACE command to start it.

### Accounting trace overhead

Enabling accounting class 2 provides additional information but also adds to the cost of processing.

Enabling accounting class 2 along with accounting classes 1 and 3 provides additional detail that relates directly to the accounting record IFCID 0003, and records thread level entry into and exit from DB2. Doing so allows you to separate DB2 times from application times.

Running accounting class 2 does add to the cost of processing. How much overhead occurs depends on how much SQL the application issues. Typically, an online transaction incurs an additional 2.5% when running with accounting class 2. A typical batch query application, which accesses DB2 more often, incurs about 10% overhead when running with accounting class 2. If most of your work is through CICS, you most likely do not need to run with class 2, because the class 1 and class 2 times are very close.

## Exception

If you use CICS Transaction Server for z/OS with the Open Transaction Environment (OTE), you should activate and run class 2. If you are concerned about high volume DB2 accounting records, for the DDF or DRDA threads and RRS attach threads, you can reduce the number of DB2 accounting records by using the DB2 system parameter ACCUMACC, which consolidates multiple accounting records into one.

## Audit trace overhead

The performance impact of auditing is directly dependent on the amount of audit data produced.

When the audit trace is active, the more tables that are audited and the more transactions that access them, the greater the performance impact. The overhead of audit trace is typically less than 5%.

When estimating the performance impact of the audit trace, consider the frequency of certain events. For example, security violations are not as frequent as table accesses. The frequency of utility runs is likely to be measured in executions per day. Alternatively, authorization changes can be numerous in a transaction environment.

## Performance trace overhead

You can significantly reduce processor overhead by turning on performance classes only to address specific performance problems.

The combined overhead of all performance classes runs from about 20% to 100%. The overhead for performance trace classes 1 through 3 is typically in the range of 5% to 30%. Consequently, you should turn on only the performance trace classes that are required to address a specific performance problem and qualify the trace as much as possible to limit the data gathered to only the data that you need. For example, you qualify the trace by the plan name and IFCID.

You can also qualify traces by any of the following attributes:

- Current or original authorization ID
- Connection name
- Correlation name
- Cached statement ID
- Location name
- Workstation ID
- Package name, collection, or program
- Transaction or application
- Role

## Statistics trace overhead

Because statistics trace information is written only periodically, CPU overhead is negligible.

Starting statistics traces class 1,3,4, (and 5 for data sharing environments) provides data at the system level. Statistics trace writes records at the statistics interval time that is specified by the STATIME parameter.



---

## Chapter 6. z/OS performance options for DB2

You can set z/OS performance options for DB2 using z/OS workload management.

With workload management (WLM), you define performance goals and assign a business importance to each goal. You define the goals for work in business terms, and the system decides how much resource, such as CPU and storage, should be given to the work to meet its goal.

WLM controls the dispatching priority based on the goals you supply. WLM raises or lowers the priority as needed to meet the specified goal. Thus, you do not need to fine-tune the exact priorities of every piece of work in the system and can focus instead on business objectives.

The three kinds of goals are:

### **Response time**

How quickly you want the work to be processed.

### **Execution velocity**

How fast the work should be run when ready, without being delayed for processor, storage, I/O access, and queue delay.

### **Discretionary**

A category for low priority work for which you define no performance goals.

Response times are appropriate goals for “end user” applications, such as DB2 QMF users running under the TSO address space goals, or users of CICS using the CICS work load goals. You can also set response time goals for distributed users, as described in “Using z/OS workload management to set performance objectives” on page 147.

For DB2 address spaces, velocity goals are more appropriate. A small amount of the work done in DB2 is counted toward this velocity goal. Most of the work done in DB2 applies to the end user goal.

For information about WLM and defining goals through the service definition, see *z/OS MVS Planning: Workload Management*.

---

## Determining z/OS workload-management velocity goals

With workload management (WLM), you define performance goals and assign a business importance to each goal.

### **Prerequisites:**

- You are managing CICS, IMS, or DDF according to WLM response-time goals
- You are set up to use WLM-established stored procedures address spaces

To set velocity goals:

- Use following service classes for velocity:
  - The default SYSSTC service class for:

- VTAM and TCP/IP address spaces
- IRLM address space (IRLMPROC)

**Important:** The *ssnmMSTR* address space should have SYSSTC priority, with the highest priority going to the DB2 system monitor task. The VTAM, TCP/IP, and IRLM address spaces must always have a higher dispatching priority than all of the DBMS address spaces, their attached address spaces, and their subordinate address spaces. Do not allow WLM to reduce the priority of VTAM, TCP/IP, or IRLM to or below that of the other DBMS address spaces.

- A high-velocity goal for a service class whose name you define, such as PRODREGN, for the following:

DB2(all address spaces, including *ssnmDBRM*, *ssnmMSTR*, and *ssnmDIST*, which should always be in the same address space.)

CICS (all region types)

IMS (all region types except BMPs)

The velocity goals for CICS and IMS regions are only important during startup or restart. After transactions begin running, WLM ignores the CICS or IMS velocity goals and assigns priorities based on the goals of the transactions that are running in the regions. A high velocity goal is good for ensuring that startups and restarts are performed as quickly as possible.

Similarly, when you set response time goals for DDF threads, the only work controlled by the DDF or stored procedure velocity goals are the DB2 service tasks (work performed for DB2 that cannot be attributed to a single user). The user work runs under separate goals for the enclave.

IMS BMPs can be treated along with other batch jobs or given a velocity goal, depending on what business and functional requirements you have at your site.

- Consider the following other workload management considerations:
  - IRLM must be eligible for the SYSSTC service class. To make IRLM eligible for SYSSTC, do not classify IRLM to one of your own service classes.
  - If you need to change a goal, change the velocity between 5 and 10%. Velocity goals do not translate directly to priority. Higher velocity tends to have higher priority, but this is not always the case.
  - WLM can assign I/O priority (based on I/O delays) separately from processor priority.  
For information about how read and write I/O priorities are determined, see “How DB2 assigns I/O priorities”.
  - z/OS workload management dynamically manages storage isolation to meet the goals you set.

## How DB2 assigns I/O priorities

DB2 informs z/OS about which address space’s priority is to be associated with a particular I/O request.

WLM handles the management of the request from after that. The table below describes to which enclave or address space DB2 associates I/O read requests.

*Table 12. How read I/O priority is determined*

Request type	Synchronous reads	Prefetch reads
Local	Application’s address space	Application’s address space

*Table 12. How read I/O priority is determined (continued)*

<b>Request type</b>	<b>Synchronous reads</b>	<b>Prefetch reads</b>
DDF or Sysplex query parallelism (assistant only)	Enclave priority	Enclave priority

The table below describes to which enclave or address space DB2 associates I/O write requests.

*Table 13. How write I/O priority is determined*

<b>Request type</b>	<b>Synchronous writes</b>	<b>Deferred writes</b>
Local	Application's address space	ssnmDBM1 address space
DDF	DDF address space	ssnmDBM1 address space



---

## Chapter 7. Configuring storage for performance

Increasing the I/O rate and decreasing the frequency of disk access to move data between real storage and storage devices is key to good performance.

To meet the diverse needs of application data, a range of storage options is available, each with different access speeds, capacities, and costs per megabyte.

This broad selection of storage alternatives supports requirements for enhanced performance and expanded online storage options, providing more options in terms of performance and price.

The levels in the DB2 storage hierarchy include real storage, storage controller cache, disk, and auxiliary storage.

---

### Storage servers and channel subsystems

Channels can be more of a bottleneck than any other component of the I/O subsystem with IBM TotalStorage® and other newer storage servers.

The degree of I/O parallelism that can be sustained efficiently is largely a function of the number of channels. In general, more channels mean better performance.

However, not all channels are alike. ESCON® channels, which used to be the predominant channel type, have a maximum instantaneous data transfer rate of approximately 17 MB per second. FICON® channels currently have a speed of 4 GB per second. FICON is the z/OS equivalent of Open Systems Fibre Channel Protocol (FCP). The FICON speed is bidirectional, theoretically allowing 4 GB per second to be sustained in both directions. Channel adaptors in the host processor and the storage server limit the actual speed. The FICON channels in the System z9 servers are faster than those in the prior processors, and they feature MIDAW improvements.

---

### Balancing the storage controller cache and buffer resources

The amount of cache to use for DB2 depends primarily on the relative importance of price and performance.

Having large memory resources for both DB2 buffers and storage controller cache is not often effective. If you decide to concentrate on the storage controller cache for performance gains, use the maximum available cache size. If the cache is substantially larger than the DB2 buffer pools, DB2 can make effective use of the cache to reduce I/O times for random I/O. For sequential I/O, the improvement from the cache provides is generally small.

---

### Improving the use of real and virtual storage

You can use several techniques to minimize the use of storage by DB2.

The amount of real storage often needs to be close to the amount of virtual storage.

To minimize the amount of storage that DB2 uses:

- Minimize storage needed for locks. You can save real storage by using the LOCKSIZE TABLESPACE option on the CREATE TABLESPACE statements for large tables, which affects concurrency. This option is most practical when concurrent read activity without a write intent, or a single write process, is used. You can use LOCKSIZE PAGE or LOCKSIZE ROW more efficiently when you commit your data more frequently or when you use cursor stability with CURRENTDATA NO. For more information on specifying LOCKSIZE TABLESPACE, see Chapter 27, “Monitoring of DB2 locking,” on page 485
- Improve the performance for sorting. The highest performance sort is the sort that is avoided. However, because some sorting cannot be avoided, make sorting as efficient as possible. For example, assign the buffer pool for your work file table spaces in database DSNDB07, which are used in sorting, to a buffer pool other than BP0, such as to BP07.
- Release unused thread storage. If you experience virtual storage constraints due to thread storage, consider having DB2 periodically free unused thread storage. To release unused storage threads, specify YES for the CONTRACT THREAD STG field on installation panel DSNTIPE. However, you should use this option only when your DB2 subsystem has many long-running threads and your virtual storage is constrained. To determine the level of thread storage on your subsystem, see IFCID 0225 or QW0225AL in statistics trace class 6.
- Provide for pooled threads. Distributed threads that are allowed to be pooled use less storage than inactive database access threads. On a per connection basis, pooled threads use even less storage than inactive database access threads.
- Ensure ECSA size is adequate. The extended common service area (ECSA) is a system area that DB2 shares with other programs. Shortage of ECSA at the system level leads to use of the common service area.  
DB2 places some load modules and data into the common service area. These modules require primary addressability to any address space, including the address space of the application. Some control blocks are obtained from common storage and require global addressability. For more information, see *DB2 Installation Guide*.
- Ensure EDM pool space is being used efficiently. Monitor your use of EDM pool storage using DB2 statistics and see “Controlling EDM storage size” on page 71
- Use less buffer pool storage. Using fewer and smaller buffer pools reduces the amount of real storage space DB2 requires. Buffer pool size can also affect the number of I/O operations performed; the smaller the buffer pool, the more I/O operations needed. Also, some SQL operations, such as joins, can create a result row that does not fit on a 4-KB page.
- Use the long-term page fix option for I/O intensive bufferpools. Use PGFIX(YES) for buffer pools with a high I/O rate, that is, a high number of pages read or written. For more information about this option see, see “Fixing a buffer pool in real storage” on page 69.
- Control maximum number of LE tokens. When a function is executed and needs to access storage used by Language Environment®, it obtains an LE token from the pool. Language Environment provides a common runtime environment for programming languages. A token is taken each time one of the following functions is executed:
  - Log functions (LOG, LN, LOG10)
  - Trigonometry functions (ACOS, ASIN, ATAN, ATANH, ATAN2, COS, COSH, SIN, SINH, TAN, and TANH)
  - EXP
  - POWER™

- RAND
- ADD\_MONTHS
- LAST\_DAY
- NEXT\_DAY
- ROUND\_TIMESTAMP
- TRUNC\_TIMESTAMP
- LOWER
- TRANSLATE
- UPPER

On completion of the call to LE, the token is returned to the pool. The MAXIMUM LE TOKENS (LEMAX) field on installation panel DSNTIP7 controls the maximum number of LE tokens that are active at any time. The LEMAX default value is 20 with a range of 0 to 50. If the value is zero, no tokens are available. If a large number of functions are executing at the same time, all the token might be used. Thus, if a statement needs a token and none is available, the statement is queued. If the statistics trace QLENTRDY is very large, indicating a delay for an application because an LE token is not immediately available, the LEMAX might be too small. If the statistics trace QLETIMEW for cumulative time spent is very large, the LEMAX might be too small. Increase the number of tokens for the MAXIMUM LE TOKENS field on installation panel DSNTIP7.

- Use non-compressed indexes. A compressed index might cause performance degradation for those applications that perform random updates to the index. Applications that only read data will decompress a collection of keys only once for multiple key retrievals. However, applications that perform random updates, might decompress the collection of keys once for every update.

#### **Related concepts**

“Overview of index access” on page 666

“Sort access” on page 689

“Using threads in INACTIVE MODE for DRDA-only connections” on page 144

“Making buffer pools large enough for the workload” on page 18

#### **Related tasks**

“Improving the performance of sort processing” on page 77

## **Real storage**

*Real storage* refers to the processor storage where program instructions reside while they are executing.

Real storage also refers to where data is held, for example, data in DB2 buffer pools that has not been paged out to auxiliary storage, the EDM pools, and the sort pool. To be used, data must either reside or be brought into processor storage or processor special registers. The maximum amount of real storage that one DB2 subsystem can use is the real storage of the processor, although other limitations might be encountered first.

The large capacity for buffers in real storage and the write avoidance and sequential access techniques allow applications to avoid a substantial amount of read and write I/O, combining single accesses into sequential access, so that the disk devices are used more effectively.

## Virtual storage

*Virtual storage* is auxiliary storage space that can be regarded as addressable storage because virtual addresses are mapped to real addresses.

### Tuning DB2 buffer, EDM, RID, and sort pools

Proper tuning of your buffer pools, EDM pools, RID pools, and sort pools can improve the response time and throughput for your applications and provide optimum resource utilization. Using data compression can also improve buffer-pool hit ratios and reduce table space I/O rates.

#### Tuning database buffer pools:

*Buffer pools* are areas of virtual storage that temporarily store pages of table spaces or indexes. When an application program accesses a row of a table, DB2 places the page that contains that row in a buffer.

If the requested data is already in a buffer, the application program does not have to wait for it to be retrieved from disk. Avoiding the need to retrieve data from disk results in faster performance.

If the row is changed, the data in the buffer must be written back to disk eventually. But that write operation might be delayed until DB2 takes a checkpoint, or until one of the related write thresholds is reached. (In a data sharing environment, however, the writing mechanism is somewhat different. .)

The data remains in the buffer until DB2 decides to use the space for another page. Until that time, the data can be read or changed without a disk I/O operation.

DB2 allows you to use up to 50 buffer pools that contain 4-KB buffers and up to 10 buffer pools each for 8-KB, 16-KB, and 32-KB buffers. You can set the size of each of those buffer pools separately during installation.

**Buffer Pool Analyzer:** You can use the Buffer Pool Analyzer for z/OS to get recommendations buffer pool allocation changes and to do “what if” analysis of your buffer pools.

To change the size and other characteristics of a buffer pool, or enable DB2 automatic buffer pool size management:

Use the ALTER BUFFERPOOL command. You can issue the ALTER BUFFERPOOL command at any time while DB2 is running

*Types of buffer pool pages:*

At any moment, a database buffer pool can have *in-use*, *updated*, and *available* pages.

#### In-use pages

Pages that are currently being read or updated. The data they contain is available for use by other applications.

#### Updated pages

Pages whose data has been changed but have not yet been written to disk.

#### Available pages

Pages that can be considered for new use, to be overwritten by an

incoming page of new data. Both in-use pages and updated pages are *unavailable* in this sense; they are not considered for new use.

#### *Read operations:*

DB2 uses *dynamic prefetch* to read data in almost all cases, but might use other read mechanisms in certain situations, including: *sequential prefetch*, *list sequential prefetch*, and *normal read*.

#### **Dynamic prefetch**

Dynamic prefetch can reduce paging and improve performance over sequential prefetch for access to data that is not on consecutive pages. Because dynamic prefetch uses sequential detection, it is more adaptable to dynamically changing access patterns than sequential prefetch. DB2 uses dynamic prefetch in almost all situations, the main exception being for table space scans. For a detailed description of dynamic prefetch, see “Dynamic prefetch (PREFETCH='D')” on page 685.

#### **Sequential prefetch**

Sequential prefetch is performed concurrently with other operations of the originating application program. It brings pages into the buffer pool before they are required and reads several pages with a single I/O operation.

Sequential prefetch can be used to read data pages, by table space scans or index scans with clustered data reference. It can also be used to read index pages in an index scan. Sequential prefetch allows CP and I/O operations to be overlapped.

See “Sequential prefetch (PREFETCH='S')” on page 686 for a complete description of sequential prefetch.

#### **List sequential prefetch**

List sequential prefetch is used to prefetch data pages that are not contiguous (such as through non-clustered indexes). List prefetch can also be used by incremental image copy. For a complete description of the mechanism, see “List prefetch (PREFETCH='L')” on page 686.

#### **Normal read**

Normal read is used when just one or a few consecutive pages are retrieved. The unit of transfer for a normal read is one page. DB2 rarely uses normal read.

#### *Write operations:*

Write operations are usually performed concurrently with user requests.

Updated pages are queued by data set until they are written when one of the following events occurs:

- A checkpoint is taken
- The percentage of updated pages in a buffer pool for a single data set exceeds a preset limit called the vertical deferred write threshold (VDWQT)
- The percentage of unavailable pages in a buffer pool exceeds a preset limit called the deferred write threshold (DWQT)

The following table lists how many pages DB2 can write in a single I/O operation.

*Table 14. Number of pages that DB2 can write in a single I/O operation*

Page size	Number of pages
4 KB	32
8 KB	16
16 KB	8
32 KB	4

The following table lists how many pages DB2 can write in a single utility I/O operation. If the number of buffers is large enough, DB2 can write twice as many pages for each I/O operation for a utility write.

*Table 15. The number of pages that DB2 can write for a single I/O operation for utility writes*

Page Size	Number of buffers	Number of pages
4 KB	BP > 80,000	128
	BP < 80,000	64
8 KB	BP > 40,000	64
	BP < 40,000	32
16 KB	BP > 20,000	32
	BP < 20,000	16
32 KB	BP > 10,000	16
	BP < 10,000	8

As with utility write operations, DB2 can write twice as many pages for each I/O in a LOB write operation. The following table shows the number of pages that DB2 can write for each I/O operation for a LOB write.

*Table 16. The number of pages that DB2 can write for in a single I/O operation for LOB writes*

Page Size	Number of buffers	Number of pages
4 KB	BP > 80,000	64
	BP < 80,000	32
8 KB	BP > 40,000	32
	BP < 40,000	16
16 KB	BP > 20,000	16
	BP < 20,000	8
32 KB	BP > 10,000	8
	BP < 10,000	4

*Assigning a table space or index to a buffer pool:*

How you assign data to buffer pools can have a significant impact on performance.

**Restriction:** You cannot use the ALTER statement to change the assignment of the catalog and the directory.

BP0 is the default buffer pool for sorting, but you can change that by assigning the work file table spaces to another buffer pool. BP0 has a default size of 20000 and a minimum size of 2000. As with any other buffer pool, use the ALTER BUFFERPOOL command to change the size of BP0.

To assign a table space or an index to a particular buffer pool:

Issue one of the following SQL statements and specify the BUFFERPOOL clause:

- CREATE TABLESPACE
- ALTER TABLESPACE
- CREATE INDEX
- ALTER INDEX

The buffer pool is actually allocated the first time a table space or index assigned to it is opened.

*Specifying a default buffer pool for user data:*

You can assign a buffer pool other than BP0 to user data and user indexes.

Choosing values other than the default value, BP0, for both user data and user indexes is a good idea, because BP0 must be used by the DB2 catalog and directory. BP0 is much more difficult to monitor and tune if user data and indexes also use that buffer pool.

In the DSNTIP1 installation panel, change the values in the following fields:

- DEFAULT BUFFER POOL FOR USER DATA
- DEFAULT BUFFER POOL FOR USER INDEXES

*Buffer pool thresholds:*

How DB2 uses of a buffer pool is governed by several preset values called thresholds.

**GUPI** Each *threshold* is a level of use which, when exceeded, causes DB2 to take some action. Certain thresholds might indicate a buffer pool shortage problem, while other thresholds merely report normal buffer management by DB2. The level of use is usually expressed as a percentage of the total size of the buffer pool. For example, the “immediate write threshold” of a buffer pool is set at 97.5%. When the percentage of unavailable pages in a buffer pool exceeds that value, DB2 writes pages to disk when updates are completed.

For very small buffer pools, of fewer than 1000 buffers, some of the thresholds might be lower to prevent “buffer pool full” conditions, but those thresholds are not described. **GUPI**

*Fixed thresholds:*

Some thresholds, such the immediate write threshold, you cannot change. You should monitor buffer pool usage and note how often those thresholds are reached.

**GUPI** If the fixed thresholds are reached too often, the remedy is to increase the size of the buffer pool, which you can do with the ALTER BUFFERPOOL

command. However, increasing the size can affect other buffer pools, depending on the total amount of real storage available for your buffers.

The fixed thresholds are more critical for performance than the variable thresholds. Generally, you want to set buffer pool sizes large enough to avoid reaching any of these thresholds, except occasionally.

Each of the fixed thresholds is expressed as a percentage of the buffer pool that might be occupied by unavailable pages.

From the highest value to the lowest value, the fixed thresholds are:

**Immediate write threshold (IWTH): 97.5%**

This threshold is checked whenever a page is to be updated. If the threshold has been exceeded, the updated page is written to disk as soon as the update completes. The write is synchronous with the SQL request; that is, the request waits until the write is completed. The two operations do not occur concurrently.

Reaching this threshold has a significant effect on processor usage and I/O resource consumption. For example, updating three rows per page in 10 sequential pages ordinarily requires one or two write operations. However, when IWTH has been exceeded, the updates require 30 synchronous writes.

Sometimes DB2 uses synchronous writes even when the IWTH has not been exceeded. For example, when more than two checkpoints pass without a page being written, DB2 uses synchronous writes. Situations such as these do not indicate a buffer shortage.

**Data management threshold (DMTH): 95%**

This threshold is checked before a page is read or updated. If the threshold is not exceeded, DB2 accesses the page in the buffer pool once for each page, no matter how many rows are retrieved or updated in that page. If the threshold is exceeded, DB2 accesses the page in the buffer pool once for each row that is retrieved or updated in that page.


**Recommendation:** Avoid reaching the DMTH because it has a significant effect on processor usage.

The DMTH is maintained for each individual buffer pool. When the DMTH is reached in one buffer pool, DB2 does not release pages from other buffer pools.

**Sequential prefetch threshold (SPTH): 90%**

This threshold is checked at two different times:

- Before scheduling a prefetch operation. If the threshold has been exceeded, the prefetch is not scheduled.
- During buffer allocation for an already-scheduled prefetch operation. If the threshold has been exceeded, the prefetch is canceled.

When the sequential prefetch threshold is reached, sequential prefetch is inhibited until more buffers become available. Operations that use sequential prefetch, such as those using large and frequent scans, are adversely affected. 

*Thresholds that you can change:*

You can change some thresholds directly by using the ALTER BUFFERPOOL command.

**GUIP** Changing a threshold in one buffer pool has no effect on any other buffer pool.

From highest to lowest default value, the variable thresholds are:

### **Sequential steal threshold (VPSEQT)**

This threshold is a percentage of the buffer pool that might be occupied by sequentially accessed pages. These pages can be in any state: updated, in-use, or available. Hence, any page might or might not count toward exceeding any other buffer pool threshold.

The default value for this threshold is 80%. You can change that to any value from 0% to 100% by using the VPSEQT option of the ALTER BUFFERPOOL command.

This threshold is checked before stealing a buffer for a sequentially accessed page instead of accessing the page in the buffer pool. If the threshold has been exceeded, DB2 tries to steal a buffer holding a sequentially accessed page rather than one holding a randomly accessed page.

Setting the threshold to 0% would prevent any sequential pages from taking up space in the buffer pool. In this case, prefetch is disabled, and any sequentially accessed pages are discarded as soon as they are released.

Setting the threshold to 100% allows sequential pages to monopolize the entire buffer pool.

### **Virtual buffer pool parallel sequential threshold (VPPSEQT)**

This threshold is a portion of the buffer pool that might be used to support parallel operations. It is measured as a percentage of the sequential steal threshold (VPSEQT). Setting VPPSEQT to zero disables parallel operation.

The default value for this threshold is 50% of the sequential steal threshold (VPSEQT). You can change that to any value from 0% to 100% by using the VPPSEQT option on the ALTER BUFFERPOOL command.

### **Virtual buffer pool assisting parallel sequential threshold (VPXPSEQT)**

This threshold is a portion of the buffer pool that might be used to assist with parallel operations initiated from another DB2 in the data sharing group. It is measured as a percentage of VPPSEQT. Setting VPXPSEQT to zero disallows this DB2 from assisting with Sysplex query parallelism at run time for queries that use this buffer pool.

The default value for this threshold is 0% of the parallel sequential threshold (VPPSEQT). You can change that to any value from 0% to 100% by using the VPXPSEQT option on the ALTER BUFFERPOOL command.

## **Deferred write threshold (DWQT)**

This threshold is a percentage of the buffer pool that might be occupied by unavailable pages, including both updated pages and in-use pages.

The default value for this threshold is 50%. You can change that to any value from 0% to 90% by using the DWQT option on the ALTER BUFFERPOOL command.

DB2 checks this threshold when an update to a page is completed. If the percentage of unavailable pages in the buffer pool exceeds the threshold, write operations are scheduled for enough data sets (at up to 128 pages per data set) to decrease the number of unavailable buffers to 10% below the threshold. For example, if the threshold is 50%, the number of unavailable buffers is reduced to 40%.

When the deferred write threshold is reached, the data sets with the oldest updated pages are written asynchronously. DB2 continues writing pages until the ratio goes below the threshold.

## **Vertical deferred write threshold (VDWQT)**

This threshold is similar to the deferred write threshold, but it applies to the number of updated pages for a single page set in the buffer pool. If the percentage or number of updated pages for the data set exceeds the threshold, writes are scheduled for that data set, up to 128 pages.

You can specify this threshold in one of two ways:

### **Percentage**

Percentage of the buffer pool that might be occupied by updated pages from a single page set. The default value for this threshold is 10%. You can change the percentage to any value from 0% to 90%.

### **Absolute number**

The total number of buffers in the buffer pools that might be occupied by updated pages from a single page set. You can specify the number of buffers from 0 to 9999. If you want to use the number of buffers as your threshold, you must set the percentage threshold to 0.

You can change the percent or number of buffers by using the VDWQT keyword on the ALTER BUFFERPOOL command.

Because any buffers that count toward VDWQT also count toward DWQT, setting the VDWQT percentage higher than DWQT has no effect: DWQT is reached first, write operations are scheduled, and VDWQT is never reached. Therefore, the ALTER BUFFERPOOL command does not allow you to set the VDWQT percentage to a value greater than DWQT. You can specify a number of buffers for VDWQT that is higher than DWQT, but again, with no effect.

This threshold is overridden by certain DB2 utilities, which use a constant limit of 64 pages rather than a percentage of the buffer pool size. LOAD, REORG, and RECOVER use a constant limit of 128 pages.

### **VDWQT set to 0:**

If you set VDWQT to zero, DB2 implicitly uses the smaller of 1% of the buffer pool (a specific number of pages), or the number determined by the buffer pool page size as shown in the following table, to avoid synchronous writes to disk.

*Table 17. Number of changed pages based on buffer pool size*

Buffer pool page size	Number of changed pages
4 KB	40
8 KB	24
16 KB	16
32 KB	12

#### GUPI

*Guidelines for setting buffer pool thresholds:*

How you set buffer pools depends on your workload and the type and size of data being cached. But always think about the entire system when making buffer pool tuning decisions.

#### GUPI

For additional help in tuning your buffer pools, try the Buffer Pool Analyzer for z/OS.

### Frequently re-referenced and updated pages

Suppose that you have a workload such as a branch table in a bank that contains a few hundred rows and is updated by every transaction. For such a workload, you want a high value for the deferred write and vertical deferred write threshold (90%). The result is that I/O is deferred until DB2 checkpoint and you have a lower I/O rate to disk.

However, if the set of pages updated exceeds the size of the buffer pool, setting both DWQT and VDWQT to 90% might cause the sequential prefetch threshold (and possibly the data management threshold and the immediate write threshold) to be reached frequently. You might need to set DWQT and VDWQT lower in that case.

### Rarely referenced pages

Suppose that you have a customer table in a bank that has millions of rows that are accessed randomly or are updated sequentially in batch.

In this case, lowering the DWQT or VDWQT thresholds (perhaps down to 0) can avoid a surge of write I/Os caused by DB2 checkpoint. Lowering those thresholds causes the write I/Os to be distributed more evenly over time. Secondly, this can improve performance for the storage controller cache by avoiding the problem of flooding the device at DB2 checkpoint.

### Query-only buffer pools

For a buffer pool that is used exclusively for sequential processing, setting VPSEQT to 100% is reasonable. If parallel query processing is a large part of the workload,

set VPPSEQT and, if applicable, VPXPSEQT, to a very high value.


### Mixed workloads

For a buffer pool used for both query and transaction processing, the value you set for VPSEQT should depend on the respective priority of the two types of processing. The higher you set VPSEQT, the better queries tend to perform, at the expense of transactions. If you are not sure what value to set for VPSEQT, use the default setting.

### Buffer pools that contain LOBs

Put LOB data in buffer pools that are not shared with other data. For both LOG YES and LOG NO LOBs, use a deferred write threshold (DWQT) of 0. LOBs specified with LOG NO have their changed pages written at commit time (*force-at-commit* processing). If you set DWQT to 0, those writes happen continuously in the background rather than in a large surge at commit. Dedicating a single buffer pool to LOB objects is especially efficient in data sharing environments.

LOBs defined with LOG YES can use deferred write, but by setting DWQT to 0, you can avoid massive writes at DB2 checkpoints.

Set group buffer pool cast out thresholds to a low value to reduce the need for a large group buffer pools for LOB objects. 

*Determining size and number of buffer pools:*

The size and the number of buffer pools that you use in your DB2 subsystem can significantly affect the performance of that subsystem.

*Enabling automatic buffer pool size management:*

You can reduce the amount of time that you spend monitoring and adjusting buffer pools by enabling the DB2 automatic buffer pool size management feature.

Automatic buffer pool management does not completely replace existing tools to configure, monitor, and tune buffer pool size. However, when you have initially sized your buffer pools, DB2 and WLM can "fine tune" the buffer pool size, based on long term trends and steady state growth. The DISPLAY BUFFERPOOL output includes an AUTOSIZE attribute. You can enable or disable automatic buffer pool management at the individual buffer pool level. Automatic buffer pool management is off by default.

To enable automatic buffer pool size management:

Issue an ALTER BUFFERPOOL command and specify the AUTOSIZE(YES) option. DB2 performs dynamic buffer pool size adjustments that are based on real-time workload monitoring.

For example, a noncritical DB2 subsystem can automatically reduce its buffer pool size. By doing so, it frees the storage so that it can be used by a mission-critical subsystem on the same LPAR, if important transactions fail to meet performance goals.

When you enable automatic buffer pool management, DB2 reports the buffer pool size and hit ratio for random reads to the z/OS Workload Manager (WLM) component. DB2 also automatically increases or decreases buffer pool size, as appropriate, by up to 25% of the originally allocated size.

#### *Choosing buffer pool sizes:*

Initially, you set the sizes (in number of pages) of your buffer pools on installation panels DSNTIP1 and DSNTIP2.

Because you can use the ALTER BUFFERPOOL command to modify the sizes of buffer pools, and enable automatic buffer pool management, choosing an exact size initially is not important.

#### *Buffer pool size guidelines:*

DB2 handles large buffer pools very efficiently. Searching in large buffer pools does not use any more of the processor's resources than searching in smaller pools.

If insufficient real storage exists to back the buffer pool storage, the resulting paging activity might cause performance degradation. If you see significant paging activity, increase the amount of real storage or decrease the size of the buffer pools.

**Important:** Insufficient storage causes paging, and in extreme situations, might cause the system to enter into wait state and require an IPL of the system.

#### **Advantages of large buffer pools**

In general, larger buffer pool sizes can:

- Result in a higher buffer pool hit ratio, which can reduce the number of I/O operations. Fewer I/O operations can reduce I/O contention, which can provide better response time and reduce the processor resource needed for I/O operations.
- Give an opportunity to achieve higher transaction rates with the same response time. For any given response time, the transaction rate depends greatly on buffer pool size.
- Prevent I/O contention for the most frequently used disks, particularly the catalog tables and frequently referenced user tables and indexes. In addition, a large buffer pool is beneficial when a DB2 sort is used during a query, because I/O contention on the disks containing the work file table spaces is reduced.

#### *The buffer pool hit ratio:*

*Buffer pool hit ratio* is a measure of how often a page access (a getpage) is satisfied without requiring an I/O operation.

**PSPI** You can help some of your applications and queries by making the buffer pools large enough to increase the buffer hit ratio.

Accounting reports, which are application related, show the hit ratio for specific applications. An accounting trace report shows the ratio for single threads. The OMEGAMON buffer pool statistics report shows the hit ratio for the subsystem as a whole. For example, the buffer-pool hit ratio is shown in field **A** in the following figure.

TOT4K READ OPERATIONS	QUANTITY	TOT4K WRITE OPERATIONS	QUANTITY
BPOOL HIT RATIO (%) <b>A</b>	73.12	BUFFER UPDATES	220.4K
GETPAGE REQUEST	1869.7K	PAGES WRITTEN	35169.00
GETPAGE REQUEST-SEQUENTIAL	1378.5K	BUFF.UPDATES/PAGES WRITTEN	6.27
GETPAGE REQUEST-RANDOM	491.2K	SYNCHRONOUS WRITES	3.00
SYNCHRONOUS READS <b>B</b>	54187.00	ASYNCHRONOUS WRITES	5084.00
SYNCHRON. READS-SEQUENTIAL	35994.00	PAGES WRITTEN PER WRITE I/O	5.78
SYNCHRON. READS-RANDOM	18193.00	HORIZ.DEF.WRITE THRESHOLD	2.00
GETPAGE PER SYN.READ-RANDOM	27.00	VERTI.DEF.WRITE THRESHOLD	0.00
SEQUENTIAL PREFETCH REQUEST	41800.00	DM THRESHOLD	0.00
SEQUENTIAL PREFETCH READS	14473.00	WRITE ENGINE NOT AVAILABLE	0.00
PAGES READ VIA SEQ.PREFETCH <b>C</b>	444.0K	PAGE-INS REQUIRED FOR WRITE	45.00
S.PRF.PAGES READ/S.PRF.READ	30.68		
LIST PREFETCH REQUESTS	9046.00		
LIST PREFETCH READS	2263.00		
PAGES READ VIA LST PREFETCH <b>D</b>	3046.00		
L.PRF.PAGES READ/L.PRF.READ	1.35		
DYNAMIC PREFETCH REQUESTED	6680.00		
DYNAMIC PREFETCH READS	142.00		
PAGES READ VIA DYN.PREFETCH <b>E</b>	1333.00		
D.PRF.PAGES READ/D.PRF.READ	9.39		
PREF.DISABLED-NO BUFFER	0.00		
PREF.DISABLED-NO READ ENG	0.00		
PAGE-INS REQUIRED FOR READ	460.4K		

Figure 2. OMEGAMON database buffer pool statistics (modified)

The buffer hit ratio uses the following formula to determine how many getpage operations did not require an I/O operation:

$$\text{Hit ratio} = (\text{getpages} - \text{pages\_read\_from\_disk}) / \text{getpages}$$

In the formula, *pages\_read\_from\_disk* is the sum of the following fields:

- Number of synchronous reads (**B**)
- Number of pages read via sequential prefetch (**C**)
- Number of pages read via list prefetch (**D**)
- Number of pages read via dynamic prefetch (**E**)

If you have 1000 getpages and 100 pages were read from disk, the equation would be as follows:

$$\text{Hit ratio} = (1000-100)/1000$$

The hit ratio in this case is 0.9.

## Highest and lowest hit ratios

### Highest hit ratio

The highest possible value for the hit ratio is 1.0, which is achieved when every page requested is always in the buffer pool. Reading index non-leaf pages tend to have a very high hit ratio since they are frequently re-referenced and thus tend to stay in the buffer pool.

### Lowest hit ratio

The lowest hit ratio occurs when the requested page is not in the buffer

pool; in this case, the hit ratio is 0 or less. A negative hit ratio means that prefetch has brought pages into the buffer pool that are not subsequently referenced. The pages are not referenced because either the query stops before it reaches the end of the table space or DB2 must take the pages away to make room for newer ones before the query can access them.

### A low hit ratio is not always bad

While it might seem desirable to make the buffer hit ratio as close to 1.0 as possible, do not automatically assume a low buffer-pool hit ratio is bad. The hit ratio is a relative value, based on the type of application. For example, an application that browses huge amounts of data using table space scans might very well have a buffer-pool hit ratio of 0. What you want to watch for is those cases where the hit ratio drops significantly for the same application. In those cases, it might be helpful to investigate further.

### Hit ratios for additional processes

The hit ratio measurement becomes less meaningful if the buffer pool is being used by additional processes, such as work files or utilities. Some utilities and SQL statements use a special type of getpage request that reserve an empty buffer without requiring that the page be read from disk.

A getpage is issued for each empty work file page without read I/O during sort input processing. The hit ratio can be calculated if the work files are isolated in their own buffer pools. If they are, then the number of getpages used for the hit ratio formula is divided in half as follows:

$$\text{Hit ratio} = ((\text{getpages} / 2) - \text{pages\_read\_from\_disk}) / (\text{getpages} / 2)$$



*Allocating buffer pool storage to avoid paging:*

DB2 limits the total amount of storage that is allocated for virtual buffer pools to approximately twice the amount of real storage. However, to avoid paging, it is strongly recommended that you set the total buffer pool size to less than the real storage that is available to DB2.

*Paging* occurs when the virtual storage requirements for a buffer pool exceed the real storage capacity for the z/OS image. In this case, the least recently used data pages in the buffer pool are migrated to auxiliary storage. Subsequent access to these pages results in a page fault and the page must be brought into real storage from auxiliary storage. Paging of buffer pool storage can negatively affect DB2 performance. The statistics for PAGE-INS REQUIRED FOR WRITE and PAGE-INS REQUIRED FOR READ in the are useful in determining if the buffer pool size setting is too large for available real storage.

If the amount of virtual storage that is allocated to buffer pools is more than twice the amount of real storage, you cannot increase the buffer pool size. DB2 allocates the minimum buffer pool storage for the BP0, BP8K0, BP16K0, and BP32K buffer pools as shown in the following table.

*Table 18. Buffer pool storage allocation for BP0, BP8K0, BP16K0, and BP32K*

Buffer pool page size	Minimum number of pages allocated
4 KB	2000

Table 18. Buffer pool storage allocation for BP0, BP8K0, BP16K0, and BP32K (continued)

Buffer pool page size	Minimum number of pages allocated
8 KB	1000
16 KB	500
32 KB	250

To avoid problems with paging:

Set the total buffer pool size to a value that is less than the amount of real storage that is available to DB2.

*Deciding whether to create additional buffer pools:*

You can assign all objects of each page size to the corresponding default buffer pool, or you can create additional buffer pools of each size according to your specific situation and requirements.

DB2 creates four default buffer pools, one for each of the four page sizes:

- BP0
- BP8K0
- BP16K0
- BP32K

*Using only default buffer pools:*

In certain situations, you should use the default buffer pools for each page size.

Choose only the default buffer pools for each page size when:

- Your system is already storage constrained.
- You have no one with the application knowledge that is necessary to do more specialized tuning.
- Your system is a test system.

*Using additional buffer pools:*

You might be able to improve performance by creating other buffers in addition to the default buffers.

Consider the following actions to take advantage of additional buffer pools:

1. Isolate data in separate buffer pools to favor certain applications, data, and indexes.
  - You can favor certain data and indexes by assigning more buffers. For example, you might improve the performance of large buffer pools by putting indexes into separate pools from data.
  - You can customize buffer pool tuning parameters to match the characteristics of the data. For example, you might want to put tables and indexes that are updated frequently into a buffer pool with different characteristics from those that are frequently accessed but infrequently updated.
2. Put work files into separate buffer pools. Doing so provides better performance for sort-intensive queries. Applications that use created temporary tables use work files for those tables. Keeping work files separate allows you to monitor temporary table activity more easily.

This process of segregating different activities and data into separate buffer pools has the advantage of providing good and relatively inexpensive performance diagnosis data from statistics and accounting traces.

*Choosing a page-stealing algorithm:*

When DB2 must take away a page in the buffer pool to make room for a newer page, this action is called *stealing* the page from the buffer pool.

DB2 usually uses a least-recently-used (LRU) algorithm for managing pages in storage. That is, it takes away older pages so that more recently used pages can remain in the buffer pool.

However, you can also choose to have DB2 use a first-in, first-out (FIFO) algorithm. With this simple algorithm, DB2 does not keep track of how often a page is referenced—the pages that are oldest are moved out, no matter how frequently they are referenced. This simple approach to page stealing results in a small decrease in the cost of doing a getpage operation, and it can reduce internal DB2 latch contention in environments that require very high concurrency.

To specify the page-stealing algorithm:

Issue an ALTER BUFFERPOOL COMMAND and specify the PGSTEAL option.

- In most cases, keep the default value, which is LRU.
- Use FIFO for buffer pools that have no I/O; that is, the table space or index remains in the buffer pool. Because all the pages are there, the additional cost of a more complicated page management algorithm is not required.
- Use the PGSTEAL option to keep objects that can benefit from the FIFO algorithm in different buffer pools from those that benefit from the LRU algorithm.

*Fixing a buffer pool in real storage:*

You can use the PGFIX keyword with the ALTER BUFFERPOOL command to fix a buffer pool in real storage for an extended period of time.

The PGFIX keyword has the following options:

**PGFIX(YES)**

The buffer pool is fixed in real storage for the long term. Page buffers are fixed when they are first used and remain fixed.

**PGFIX(NO)**

The buffer pool is not fixed in real storage for the long term. Page buffers are fixed and unfixed in real storage, allowing for paging to disk. PGFIX(NO) is the default option.

To prevent PGFIX(YES) buffer pools from exceeding the real storage capacity, DB2 uses an 80% threshold when allocating PGFIX(YES) buffer pools. If the threshold is exceeded, DB2 overrides the PGFIX(YES) option with PGFIX(NO).

To fix a buffer pool in real storage:

Issue an ALTER BUFFERPOOL command and specify PGFIX(YES). You should use PGFIX(YES) for buffer pools with a high I/O rate, those buffer pools with a high number of pages read or written. For buffer pools with zero I/O, such as some

read-only data or some indexes with a nearly 100% hit ratio, PGFIX(YES) is not recommended because it does not provide any performance advantage.

**Designing EDM storage space for performance:**

The environmental descriptor manager (EDM) pool contains active and skeleton application plans and packages. You can design them to avoid I/Os and speed processing for plans and packages.

You can design your EDM storage pools to avoid allocation I/O (a significant part of the total number of I/Os for a transaction), reduce the time that is required to check whether users who attempt to execute a plan are authorized to do so, and reduce the time that is required to prepare statements with the statement cache pool.

When pages are needed from the EDM storage pools, any pages that are available are allocated first. If the available pages do not provide enough space to satisfy the request, pages are “stolen” from an inactive SKCT, SKPT, DBD, or dynamic SQL skeleton. If enough space is still not available, an SQL error code is sent to the application program.

EDM storage pools that are too small cause the following problems.

- Increased I/O activity in DSNDB01.SCT02, DSNDB01.SPT01, and DSNDB01.DBD01
- Increased response times, due to loading the SKCTs, SKPTs, and DBDs
- Fewer threads used concurrently, due to a lack of storage

To ensure the best performance from EDM pools:

Design your EDM storage according to the following table.

*Table 19. Designing the EDM storage pools*

Design...	To contain...
EDM RDS below pool	Part of a plan (CT) or package (PT) that is below the 2-GB bar.
EDM RDS above pool	Part of a plan (CT) or package (PT) that is above the 2-GB bar.
EDM DBD pool	Database descriptors
EDM statement pool	The cached dynamic SQL statements
EDM skeleton pool	Skeleton copies of plans (SKCTs) and packages (SKPTs)

*EDM storage:*

The environmental descriptor manager, or EDM, pool contains active and skeleton application plans and packages.

EDM storage is composed of these five components, each of which is in a separate storage area:

**EDM RDS pool below**

A below-the-bar pool, which contains the part of the cursor tables (CTs) and package tables (PTs) that must be below the bar

### **EDM RDS pool above**

An above-the-bar pool that contains the part of the PTs and CTs that can be above the bar

### **EDM DBD pool**

An above-the-bar pool that contains database descriptors (DBDs)

### **EDM statement pool**

An above-the-bar pool that contains dynamic cached statements

### **EDM skeleton pool**

An above-the-bar pool that contains skeleton package tables (SKPTs) and skeleton cursor tables (SKCTs)

During the installation process, the DSNTINST CLIST calculates the sizes of the EDM pools (above and below), the EDM statement cache, the EDM DBD cache, and the EDM skeleton pool. You can check the calculated sizes on the DSNTIPC installation panel.

For data sharing, you might need to increase the EDM DBD cache storage estimate.

Because of an internal process that changes the size of plans initially bound in one release and then are rebound in a later release, you should carefully monitor the sizes of the EDM storage pools, and increase their sizes, if necessary.

#### *Controlling EDM storage size:*

You can take measures to control the size of your EDM storage.

To keep EDM storage, and especially the EDM pool, under control:

1. Use more packages. By using multiple packages you can increase the effectiveness of EDM pool storage management by having smaller objects in the pool.
2. Use RELEASE(COMMIT) when appropriate. Using the bind option RELEASE(COMMIT) for infrequently used packages and plans can cause objects to be removed from the EDM pool sooner.
3. Understand the impact of using DEGREE(ANY). A plan or package that is bound with DEGREE(ANY) can require 50 to 70% more storage for the CTs and PTs in the EDM pool than one bound with DEGREE(1). If you change a plan or package to DEGREE(ANY), check the change in the column AVGSIZE in SYSPLAN or SYSPACKAGE to determine the increase required.

#### *Measuring the efficiency of the EDM pool:*

You can use information in the DB2 statistics record to calculate the efficiency of the EDM pool, the EDM DBD cache, and the EDM statement cache.

To measure the efficiency of the EDM pool:

Gather the following ratios from the OMEGAMON statistics report:

- DBD HIT RATIO (%) **D**
- CT HIT RATIO (%) **E**
- PT HIT RATIO (%) **F**
- STMT HIT RATIO (%) **G**

These ratios for the EDM pool depend upon your location's work load. In most DB2 subsystems, a value of 80% or more is acceptable. This value means that at least 80% of the requests were satisfied without I/O.

The number of free pages is shown in FREE PAGES ( **B** ). For pools with stealable objects, if this value is more than 20% of The number of pages for the corresponding pools ( **A** ) during peak periods, the EDM pool size is probably too large for that type of pool. In this case, you can reduce its size without affecting the efficiency ratios significantly.

Below the bar and above the bar pools should have enough space for the maximum workload. The number of times that a failure occurred because a particular pool is shown in the values that are recorded in the FAILS DUE TO *type* POOL fields ( **C** ). You should increase the size of the corresponding pool if any of those fields contains a non-zero value.

### **EDM pools in the accounting report**

The DB2 statistics record provides information on the EDM pool, the EDM DBD cache, and the EDM statement cache. The following figure shows how OMEGAMON presents this information in the statistics report.

EDM POOL	QUANTITY
PAGES IN RDS POOL (BELOW)	<b>A</b> 32768.00
HELD BY CT	45.88
HELD BY PT	32722.12
FREE PAGES	<b>B</b> 8151.00
FAILS DUE TO POOL FULL	<b>C</b> 0.00
PAGES IN RDS POOL (ABOVE)	<b>A</b> 524.3K
HELD BY CT	0.00
HELD BY PT	0.00
FREE PAGES	<b>B</b> 524.3K
FAILS DUE TO RDS POOL FULL	<b>C</b> 0.00
PAGES IN DBD POOL (ABOVE)	<b>A</b> 262.1K
HELD BY DBD	67.00
FREE PAGES	<b>B</b> 262.1K
FAILS DUE TO RDSPool FULL	<b>C</b> 0.00
PAGES IN STMT POOL (ABOVE)	<b>A</b> 262.1K
HELD BY STATEMENTS	187.00
FREE PAGES	<b>B</b>
FAILS DUE TO STMT POOL FULL	<b>C</b> 0.00
PAGES IN SKEL POOL (ABOVE)	25600.00
HELD BY SKCTS	2.00
HELD BY SKPTS	86.00
FREE PAGES	25512.00
DBD REQUESTS	151.2K
DBD NOT FOUND	0.00
DBD HIT RATIO (%)	<b>D</b> 100.00
CT REQUESTS	0.00
CT NOT FOUND	0.00
CT HIT RATIO (%)	<b>E</b> N/C
PT REQUESTS	151.2K
PT NOT FOUND	0.00
PT HIT RATIO (%)	<b>F</b> 100.00
STMT REQUESTS	
STMT NOT FOUND	0.00
STMT HIT RATIO (%)	<b>G</b> 100.00
PKG SEARCH NOT FOUND	0.00
PKG SEARCH NOT FOUND INSERT	0.00
PKG SEARCH NOT FOUND DELETE	0.00
STATEMENTS IN GLOBAL CACHE	50.00
DYNAMIC SQL STMT	QUANTITY
PREPARE REQUESTS	8897.00
FULL PREPARES	0.00
SHORT PREPARES	9083.00
GLOBAL CACHE HIT RATIO (%)	100.00
IMPLICIT PREPARES	0.00
PREPARES AVOIDED	0.00
CACHE LIMIT EXCEEDED	0.00
PREP STMT PURGED	0.00
LOCAL CACHE HIT RATIO (%)	N/C

Figure 3. EDM storage usage in the OMEGAMON statistics report

Calculating the EDM statement cache hit ratio:

If you have caching turned on for dynamic SQL, the EDM storage statistics provide information that can help you determine how successful your applications are at finding statements in the cache.

PREPARE REQUESTS ( **A** ) records the number of requests to search the cache. FULL PREPARES ( **B** ) records the number of times that a statement was inserted into the cache, which can be interpreted as the number of times a statement was not found in the cache. To determine how often the dynamic statement was used from the cache, check the value in GLOBAL CACHE HIT RATIO ( **C** ).

To calculate the the EDM statement cache hit ratio:

Use the following formula:

$$(\text{PREPARE REQUESTS} - \text{FULL PREPARES}) / \text{PREPARE REQUESTS} = \text{hit ratio}$$

The following figure shows the dynamic SQL statements part of OMEGAMON statistics report.

DYNAMIC SQL STMT		QUANTITY	-----	-----
PREPARE REQUESTS	<b>A</b>	8897.00		
FULL PREPARES	<b>B</b>	0.00		
SHORT PREPARES		9083.00		
GLOBAL CACHE HIT RATIO (%)	<b>C</b>	100.00		

Figure 4. EDM storage usage for dynamic SQL statements in the OMEGAMON statistics report

*Controlling DBD size for large databases:*

When you design your databases, be aware that a very large number of objects in your database means a larger DBD for that database.

If a large number of create, alter, and drop operations are performed on objects in a database with a large DBD, DB2 might encounter more contention from the DBD among transactions that access different objects because storage is not automatically reclaimed in the DBD.

To control the size of DBDs for large databases:

- Monitor and manage DBDs to prevent them from becoming too large. Very large DBDs can reduce concurrency and degrade the performance of SQL operations that create or alter objects because of increased I/O and logging. DBDs that are created or altered in DB2 Version 6 or later do not need contiguous storage, but can use pieces of approximately 32 KB. Older DBDs require contiguous storage.
- When you create, alter, and drop objects in a database, use the MODIFY utility to reclaim storage in the DBD. Storage is not automatically reclaimed in a DBD for these operations.

*Balancing EDM space fragmentation and performance:*

You can control whether DB2 emphasizes space usage or performance for EDM pools of different sizes:

For smaller EDM pools, space utilization or fragmentation is normally more critical than for larger EDM pools. For larger EDM pools, performance is normally more critical. DB2 emphasizes performance and uses less optimum EDM storage allocation when the EDM pool size exceeds 40 MB.

To specify the search algorithm that DB2 uses:

Set the keyword EDMBFIT in the DSNTIJUZ job. The EDMBFIT keyword adjusts the search algorithm on systems with EDM pools that are larger than 40 MB.

Option	Description
Set EDMBFIT to NO	Which tells DB2 to use a first-fit algorithm. Doing so is especially important when a high class 24 latch (EDM LRU latch) contention exists. For example, be sure to set EDMBFIT to NO when class 24 latch contention exceeds 500 contentions per second.
Set EDMBFIT to YES	Which tells DB2 to use a better-fit algorithm. Do this when EDMPOOL full conditions occur for an EDM pool size that exceeds 40 MB.

### Increasing RID pool size:

The RID pool is used for all record identifier (RID) processing. You can improve the performance of transactions that use the RID pool by specifying a sufficient size for the RID POOL.

The RID pool is used for enforcing unique keys while updating multiple rows and for sorting RIDs during the following operations:

- List prefetch, including single index list prefetch
- Access via multiple indexes
- Hybrid joins

The RID Pool Processing section of the OMEGAMON accounting trace record contains information about whether a transaction used the RID pool.

To favor the selection and efficient completion of those access paths:

Increase the maximum RID pool size. However if the available RID pool storage is too small, the statement might revert to a table space scan.

The general formula for computing RID pool size is:

Number of concurrent RID processing activities ×  
average number of RIDs × 2 × 5 bytes per RID

- The default RID pool size is 8 MB. You can override this value on installation panel DSNTIPC.
- The RID pool, which all concurrent work shares, is limited to a maximum of 10 000 MB. The RID pool is created at system initialization, but no space is allocated until RID storage is needed. Space is allocated in 32-KB blocks as needed, until the maximum size that you specified on installation panel DSNTIPC is reached.
- A RID pool size of 0 disables those access paths. If you specify a RID pool size of 0, plans or packages that were previously bound with a non-zero RID pool size might experience significant performance degradation. Rebind any plans or packages that include SQL statements that use RID processing.

Whether your SQL statements that use RID processing complete efficiently or not depends on other concurrent work that uses the RID pool.

### Example

Three concurrent RID processing activities, with an average of 4000 RIDs each, would require 120 KB of storage, because:

$$3 \times 4000 \times 2 \times 5 = 120\text{KB}$$

### Controlling sort pool size and sort processing:

A sort operation is invoked when a cursor is opened for a SELECT statement that requires sorting.

The maximum size of the sort work area allocated for each concurrent sort user depends on the value that you specified for the SORT POOL SIZE field on installation panel DSNTIPC. The default value is 2 MB.

*Estimating the maximum size of the sort pool:*

You can change the maximum size of the sort pool by using the installation panels in UPDATE mode.

To determine a rough estimate for the maximum size:

Use the following formula.

$$32000 \times (16 + \text{sort key length} + \text{sort data length})$$

For sort key length and sort data length, use values that represent the maximum values for the queries you run. To determine these values, refer to the QW0096KL (key length) and QW0096DL (data length) fields in IFCID 0096, as mapped by macro DSNDQW01. You can also determine these values from an SQL activity trace.

### Example

If a column is in the ORDER BY clause that is not in the select clause, that column should be included in the sort data length and the sort key length as shown in the following example:

```
SELECT C1, C2, C3
FROM tablex
ORDER BY C1, C4;
```

If C1, C2, C3, and C4 are each 10 bytes in length, you could estimate the sort pool size as follows:

$$32000 \times (16 + 20 + (10 + 10 + 10 + 10)) = 2342000 \text{ bytes}$$

The values used in the example above include the items in the following table:

*Table 20. Values used in the sort pool size example*

Attribute	Value
Maximum number of sort nodes	32000
Size (in bytes) of each node	16
Sort key length (ORDER BY C1, C4)	20

Table 20. Values used in the sort pool size example (continued)

Attribute	Value
Sort data length (each column is 10 bytes in length)	10+10+10+10

*How sort work files are allocated:*

The work files that are used in sort are *logical work files*, which reside in work file table spaces in your work file database (which is DSNDB07 in a non data-sharing environment).

The sort begins with the input phase, when ordered sets of rows are written to work files. At the end of the input phase, when all the rows have been sorted and inserted into the work files, the work files are merged together, if necessary, into a single work file that contains the sorted data. The merge phase is skipped if only one work file exists at the end of the input phase. In some cases, intermediate merging might be needed if the maximum number of sort work files has been allocated.

DB2 uses the buffer pool when writing to the logical work file. Only the buffer pool size limits the number of work files that can be used for sorting.

A sort can complete in the buffer pool without I/Os. This ideal situation might be unlikely, especially if the amount of data being sorted is large. The sort row size is actually made up of the columns being sorted (the sort key length) and the columns that the user selects (the sort data length). Having a very large buffer pool for sort activity can help you to avoid disk I/Os.

When your application needs to sort data, the work files are allocated on a least recently used basis for a particular sort. To support large sorts, DB2 can allocate a single logical work file to several physical work file table spaces.

For example, if five logical work files are to be used in the sort, and the installation has three work file table spaces allocated, then the following table shows which work file table space would contain each logical work file.

Table 21. How work file table spaces are allocated for logical work files

Logical work file	Work file table space
1	1
2	2
3	3
4	1
5	2

*Improving the performance of sort processing:*

Many factors affect the performance of sort operations, but you can take measures to reduce I/O contention and minimize sort row size.

The following factors affect the performance of DB2 sort processing:

- Sort pool size
- I/O contention

- Sort row size
- Whether the data is already sorted

For any SQL statement that initiates sort activity, the OMEGAMON SQL activity reports provide information on the efficiency of the sort that is involved.

To minimize the performance impacts of sort processing:

- Increase the size of the sort pool. The larger the sort pool, the more efficient the sort is.
- Minimize I/O contention on the I/O paths to the physical work files, and make sure that physical work files are allocated on different I/O paths and packs to minimize I/O contention. Using disk devices with Parallel Access Volumes (PAV) support is another way to significantly minimize I/O contention. When I/Os occur in the merge phase of a sort, DB2 uses sequential prefetch to bring pages into the buffer pool with a prefetch quantity of eight pages. However, if the buffer pool is constrained, then DB2 uses a prefetch quantity of four pages or less, or disables prefetch entirely because of the unavailability of enough pages.

- Allocate additional physical work files in excess of the defaults, and put those work files in their own buffer pool.

Segregating work file activity enables you to better monitor and tune sort performance. It also allows DB2 to handle sorts more efficiently because these buffers are available only for sort without interference from other DB2 work.

- Increase the amount of available space for work files. Applications that use created temporary tables use work file space until a COMMIT or ROLLBACK occurs. (If a cursor is defined WITH HOLD, then the data is held past the COMMIT.) If sorts are happening concurrently with the temporary table's existence, then you probably need more space to handle the additional use of the work files.

Applications that require star join, materialized views, materialized nested table expressions, non-correlated subqueries or triggers also use work files.

- Write applications to sort only columns that need to be sorted because sorted rows appear twice in the sort row size. A smaller sort row size means that more rows can fit in the sort pool.
- Select VARCHAR columns only when they are required. Varying length columns are padded to their maximum length for sort row size.
- Set the buffer pool sequential steal threshold (VPSEQT) to 100% unless sparse index is used to access the work files. The default value, which is 80%, allows 20% of the buffers to go unused. A value of 99% prevents space map pages, which are randomly accessed, from being overwritten by massive prefetch.
- Increase the buffer pool deferred write threshold (DWQT) or data set deferred write threshold (VDWQT) values. If the DWQT or VDWQT are reached, writes are scheduled. For a large sort using many logical work files, this is difficult to avoid, even if a very large buffer pool is specified.

## Managing the opening and closing of data sets

Having the needed data sets open and available for use is important for the performance of transactions.

However, the number of open data sets affects the amount of available storage, and number of open data sets in read-write state affects restart time.

### Determining the maximum number of open data sets:

DB2 defers closing and de-allocating the table spaces or indexes until the number of open data sets reaches 99% of the value that you specified for DSMAX.

When DSMAX is reached, DB2 closes 300 data sets or 3% of the value of DSMAX, whichever number of data sets is fewer. Consequently, DSMAX controls not only the limit of open data sets, but also controls the number of data sets that are closed when that limit is reached.

*How DB2 determines DSMAX:*

DB2 uses a formula to calculate the initial value for DSMAX

Initially, DB2 calculates DSMAX according to the following formula.

- Let *concdb* be the number of concurrent databases specified on installation panel DSNTIPE.
- Let *tables* be the number of tables per database specified on installation panel DSNTIPD.
- Let *indexes* be the number of indexes per table. The installation CLIST sets this variable to 2.
- Let *tblspaces* be the number of table spaces per database specified on installation panel DSNTIPD.

DB2 calculates the number of open data sets with the following formula:

$$\text{concdb} \times \{(\text{tables} \times \text{indexes}) + \text{tblspaces}\}$$

*Modifying DSMAX:*

If you have many partitioned table spaces or LOB table spaces, you might need to increase DSMAX.

The formula used by DB2 does not take partitioned or LOB table spaces into account. Those table spaces can have many data sets. Do not forget to consider the data sets for nonpartitioned indexes defined on partitioned table spaces with many of partitions and multiple partitioned indexes. If those indexes are defined with a small PIECESIZE, there could be many data sets. You can modify DSMAX by updating field DSMAX - MAXIMUM OPEN DATA SETS on installation panel DSNTIPC.

DSMAX should be larger than the maximum number of data sets that are open and in use at one time. For the most accurate count of open data sets, refer to the OPEN/CLOSE ACTIVITY section of the OMEGAMON statistics report. Make sure the statistics trace was run at a peak period, so that you can obtain the most accurate maximum figure.

The best indicator of when to increase DSMAX is when the open and close activity of data sets is high, 1 per second as a general guideline. Refer to the OPEN/CLOSE value under the SER.TASK SWITCH section of the OMEGAMON accounting report. Consider increasing DSMAX when this value shows more than 1 event per second.

To calculate the total number of data sets (rather than the number that are open during peak periods), you can do the following:

1. To find the number of simple and segmented table spaces, use the following query. The calculation assumes that you have one data set for each simple,

segmented, and LOB table space. These catalog queries are included in DSNTESP in SDSNSAMP. You can use them as input to SPUFI.

```
SELECT CLOSERULE, COUNT(*)
  FROM SYSIBM.SYSTABLESPACE
 WHERE PARTITIONS = 0
 GROUP BY CLOSERULE;
```

2. To find the number of data sets for the partitioned table spaces, use the following query, which returns the number of partitioned table spaces and the total number of partitions.

```
SELECT CLOSERULE, COUNT(*), SUM(PARTITIONS)
  FROM SYSIBM.SYSTABLESPACE
 WHERE PARTITIONS > 0
 GROUP BY CLOSERULE;
```

Partitioned table spaces can require up to 4096 data sets for the data, and a corresponding number of data sets for each partitioned index.

3. To find the number of data sets required for each nonpartitioned index, use the following query.

```
SELECT CLOSERULE, COUNT(*)
  FROM SYSIBM.SYSINDEXES T1, SYSIBM.SYSINDEXPART T2
 WHERE T1.NAME = T2.IXNAME
 AND T1.CREATOR = T2.IXCREATOR
 AND T2.PARTITION = 0
 GROUP BY CLOSERULE;
```

The calculation assumes that you have only one data set for each nonpartitioned index. If you use pieces, adjust accordingly.

4. To find the number of data sets for the partitioned indexes, use the following query, which returns the number of index partitions.

```
SELECT CLOSERULE, COUNT(*)
  FROM SYSIBM.SYSINDEXES T1, SYSIBM.SYSINDEXPART T2
 WHERE T1.NAME = T2.IXNAME
 AND T1.CREATOR = T2.IXCREATOR
 AND T2.PARTITION > 0
 GROUP BY CLOSERULE;
```

You have one data set for each index partition.

5. To find the total number of data sets, add the numbers that result from the four queries. (For Query 2, use the sum of the partitions that was obtained.)

#### *Recommendations for DSMAX:*

As with many recommendations in DB2, you must weigh the cost of performance versus availability when choosing a value for DSMAX.

Consider the following factors:

- For best performance, you should leave enough margin in your specification of DSMAX so that frequently used data sets can remain open after they are no longer referenced. If data sets are opened and closed frequently, such as every few seconds, you can improve performance by increasing DSMAX.
- The number of open data sets on your subsystem that are in read/write state affects checkpoint costs and log volumes. To control how long data sets stay open in a read/write state, specify values for the RO SWITCH CHKPTS and RO SWITCH TIME fields of installation panel DSNTIPN. See “Switching to read-only for infrequently updated and infrequently accessed page sets” on page 82 for more information.
- Consider segmented table spaces to reduce the number of data sets.

To reduce open and close activity, you can try reducing the number of data sets by combining tables into segmented table spaces. This approach is most useful for development or end-user systems that include a lot of smaller tables that can be combined into single table spaces.

### Understanding the CLOSE YES and CLOSE NO options:

The CLOSE value for a table space or index affects the process of closing an object's data sets and how DB2 manages data set closing. *Page set* refers to a table space or index.

*The process of closing:*

DB2 dynamically manages page sets using two levels of page set closure—logical close and physical close.

#### Logical close

This occurs when the application has been deallocated from that page set. This is at either commit or deallocation time, depending on the RELEASE(COMMIT/DEALLOCATE) option of the BIND command, and is driven by the use count. When a page set is logically closed, the page set use count is decremented. When the page set use count is zero, the page set is considered not in use; this makes it a candidate for physical close.

#### Physical close

This happens when DB2 closes and deallocates the data sets for the page set.

*When the data sets are closed:*

The number of open data sets determines when DB2 must close data sets. When DB2 closes data sets, all data sets for a particular table space, index, or partition are closed.

The value you specify for CLOSE determines the order in which page sets that are not in use are closed. When the open data set count becomes greater than 99% of DSMAX, DB2 first closes page sets defined with CLOSE YES. The least recently used page sets are closed first.

If the number of open data sets cannot be limited by closing page sets or partitions defined with CLOSE YES, DB2 must close page sets or partitions defined with CLOSE NO. The least recently used CLOSE NO data sets are closed first.

Delaying the physical closure of page sets or partitions until necessary is called *deferred close*. Deferred closing of a page set or partition that is no longer being used means that another application or user can access the table space and employ the accompanying indexes without DB2 reopening the data sets. Thus, deferred closing of page sets or partitions can improve your applications' performance by avoiding I/O processing.

**Recommendation:** For a table space whose data is continually referenced, in most cases it does not matter whether it is defined with CLOSE YES or CLOSE NO; the data sets remain open. This is also true, but less so, for a table space whose data is not referenced for short periods of time; because DB2 uses deferred close to manage data sets, the data sets are likely to be open when they are used again.

You could find CLOSE NO appropriate for page sets that contain data you do not frequently use but is so performance-critical that you cannot afford the delay of opening the data sets.

If the number of open data sets is a concern, choose CLOSE YES for page sets with many partitions or data sets.

**Switching to read-only for infrequently updated and infrequently accessed page sets:**

For both CLOSE YES and CLOSE NO page sets, DB2 automatically converts infrequently updated page sets or partitions from read-write to read-only state according to the values you specify in the RO SWITCH CHKPTS and RO SWITCH TIME fields of installation panel DSNTIPL.

With data sharing, DB2 uses the CLOSE YES or CLOSE NO attribute of the table space or index space to determine whether to physically close infrequently accessed page sets that have had global buffer pool dependencies. Infrequently accessed CLOSE YES page sets are physically closed. Infrequently accessed CLOSE NO page sets remain open.

RO SWITCH CHKPTS is the number of consecutive DB2 checkpoints since a page set or partition was last updated; the default is 5. RO SWITCH TIME is the amount of elapsed time since a page set or partition was last updated; the default is 10 minutes. If either condition is met, the page set or partition is converted from read-write to read-only state.

**Updating SYSLGRNX:** For both CLOSE YES and CLOSE NO page sets, SYSLGRNX entries are updated when the page set is converted from read-write state to read-only state. When this conversion occurs for table spaces, the SYSLGRNX entry is closed and any updated pages are externalized to disk. For indexes defined as COPY NO, no SYSLGRNX entries occur, but the updated pages are externalized to disk.

**Performance benefits of read-only switching:** An infrequently used page set's conversion from read-write to read-only state results in the following performance benefits:

- Improved data recovery performance because SYSLGRNX entries are more precise, closer to the last update transaction commit point. As a result, the RECOVER utility has fewer log records to process.
- Minimized logging activities. Log records for page set open, checkpoint, and close operations are only written for updated page sets or partitions. Log records are not written for read-only page sets or partitions.

**Recommendations for RO SWITCH TIME and RO SWITCH CHKPTS:** In most cases, the default values are adequate. However, if you find that the amount of R/O switching is causing a performance problem for the updates to SYSLGRNX, consider increasing the value of RO SWITCH TIME, perhaps to 30 minutes.

Data on table spaces that are defined with the NOT LOGGED option is not protected by information on the log. You should externalize modifications to such data relatively quickly, without elongating the commit processing or the elapsed time. For table spaces that are defined with the NOT LOGGED option, consider setting RO SWITCH CHKPTS and RO SWITCH TIME to 1. This setting means that all read-write table spaces that are defined with the NOT LOGGED option that are not in use are converted to read-only when a DB2 checkpoint occurs. If a

| checkpoint has not occurred, all read-write table spaces that are not logged are  
| converted to read-only one minute after the commit of the last update. DB2 writes  
| the table space from the buffer pool to external media when it converts the table  
| space from read-write to read-only, externalizing any unprotected modifications to  
| the data.

---

## Improving disk storage

You can configure your storage devices and disk space to ensure better performance from DB2.

### Selecting and configuring storage devices

| Whether you use newer storage servers, such as the IBM TotalStorage DS8000™, or  
older storage devices types, such as RVA and 3390 effects the performance of your  
DB2 subsystem.

#### Selecting storage devices

Some storage device types are more optimal for certain types of applications.

When choosing storage devices types:

Consider the following hardware characteristics that affect performance.

- The size of the cache
- The number of channels and type of channels that are attached and online to a group of logical volumes
- The size of non-volatile storage (NVS), if deferred write performance is a problem
- Disk arrays
- Advanced features such as Parallel Access Volumes (PAV), Multiple Allegiance, and FlashCopy®

#### Storage servers

| An I/O subsystem typically consists of many storage disks, which are housed in  
*storage servers* such as the IBM TotalStorage DS8000.

Storage servers provide increased functionality and performance over that of “Just a Bunch of Disks” technology.

*Cache* is one of the additional functions. Cache acts as a secondary buffer as data is moved between real storage and disk. Storing the same data in processor storage and the cache is not useful. To be useful, the cache must be significantly larger than the buffers in real storage, store different data, or provide another performance advantage. TotalStorage and many other new storage servers use large caches and always pre-stage the data in the cache. You do not need to actively manage the cache in the newer storage servers as you must do with older storage device types.

| With IBM TotalStorage and other new storage servers, disk performance does not  
generally affect sequential I/O performance. The measure of disk speed in terms of  
RPM (revolutions per minute) is relevant only if the cache hit ratio is low and the  
I/O rate is very high. If the I/O rate per disk is proportional to the disk size, small  
disks perform better than large disks. Large disks are very efficient for storing  
infrequently accessed data. As with cache, spreading the data across more disks is  
always better.

### **Storage servers and advanced features:**

IBM TotalStorage offers many advanced features to further boost performance.

Other storage servers might offer similar functions.

#### *Parallel Access Volumes (PAV):*

The parallel access volumes feature allows multiple concurrent I/Os on a given device when the I/O requests originate from the same system.

Parallel access volumes (PAV) make storing multiple partitions on the same volume with almost no loss of performance possible. In older disk subsystems, if more than one partition is placed on the same volume (intentionally or otherwise), attempts to read the partitions result in contention, which shows up as I/O subsystem queue time. Without PAVs, poor placement of a single data set can almost double the elapsed time of a parallel query.

#### *Multiple allegiance:*

The multiple allegiance feature allows multiple active concurrent I/Os on a given device when the I/O requests originate from different systems.

Together, PAVs and multiple allegiance dramatically improve I/O performance for parallel work on the same volume by nearly eliminating I/O subsystem queue or PEND time and drastically lowering elapsed time for transactions and queries.

#### *FlashCopy:*

The FlashCopy feature provides for fast copying of full volumes.

After an initialization period is complete, the logical copy is considered complete but the physical movement of the data is deferred.

#### *Peer-to-Peer Remote Copy (PPRC):*

The PPRC and PPRC XD (Extended Distance) provides a faster method for recovering DB2 subsystems at a remote site in the event of a disaster at the local site.

### **Older storage device types:**

Unlike newer storage servers, older devices have much smaller caches.

The small caches require some user management. You can use DFSMS™ to provide dynamic management of the cache storage.

#### *Sort work files:*

Sort work files can have a large number of concurrent processes that can overload a storage controller with a small cache and thereby degrade system performance.

For example, one large sort could use 100 sequential files, needing 60 MB of storage. Unless the cache sizes are large, you might need to specify BYPASS on installation panel DSNTIPE or use DFSMS controls to prevent the use of the cache during sort processing. Separate units for sort work can give better performance.

## Using disk space effectively

How you allocate and manage data sets, compress your data, and design your indexes can effects the performance of DB2.

To use disk space more efficiently:

- Change your allocation of data sets to keep data sets within primary allocations.
- Manage them with the Hierarchical Storage Management functional component (DFSMSHsm™) of DFSMS.
- Compress your data.
- Choose a page size that gives you good disk use and I/O performance characteristics.
- Evaluate the need for and characteristics of your indexes.

To manage the use of disk, you can use RMF™ to monitor how your devices are used. Watch for usage rates that are higher than 30% to 35%, and for disk devices with high activity rates. Log devices can have more than 50% utilization without performance problems.

### Allocating and extending data sets

Primary and secondary allocation sizes are the main factors that affect the amount of disk space that DB2 uses.

In general, the primary allocation must be large enough to handle the storage needs that you anticipate. The secondary allocation must be large enough for your applications to continue operating until the data set is reorganized.

If the secondary allocation space is too small, the data set might have to be extended more times to satisfy those activities that need a large space.

IFCID 0258 allows you to monitor data set extension activities by providing information, such as the primary allocation quantity, maximum data set size, high allocated space before and after extension activity, number of extents before and after the extend, maximum volumes of a VSAM data set, and number of volumes before and after the extend. Access IFCID 0258 in Statistics Class 3 (SC03) through an IFI READA request.

### Planning the placement of DB2 data sets:

To improve performance, plan the placement of DB2 data sets carefully.

Concentrate mainly on data sets for system files (especially the active logs), for the DB2 catalog and directory, and for user data and indexes. The objective is to balance I/O activity between different volumes, control units, and channels. Doing so minimizes the I/O elapsed time and I/O queuing.

#### *Identifying crucial DB2 data sets:*

When you placing your data sets, you need to first consider data sets are crucial for DB2 to function properly.

To gather this information: If these reports are not available, consider these the most important data sets:

Use the I/O reports from the DB2 performance trace. If these reports are not available, consider the following data sets to be most important:

**For transactions**

- DSNDB01.SCT02 and its index
- DSNDB01.SPT01 and its index
- DSNDB01.DBD01
- DSNDB06.SYSPLAN table space and indexes on SYSPLANAUTH table
- DSNDB06.SYSPKAGE
- Active and archive logs
- Most frequently used user table spaces and indexes

**For queries**

- DSNDB01.DBD01
- DSNDB06.SYSPLAN table space and indexes on SYSPLANAUTH
- DSNDB06.SYSPKAGE
- DSNDB06.SYSDBASE table space and its indexes
- DSNDB06.SYSVIEWS table space and the index on SYSVTREE
- Work file table spaces
- DB2 QMF system table data sets
- Most frequently used user table spaces and indexes

These lists do not include other data sets that are less crucial to DB2 performance, such as those that contain program libraries, control blocks, and formats. Those types of data sets have their own design recommendations.

*Estimating concurrent I/O requests:*

The number of concurrent I/O requests is important when you calculate the number of data paths for your DB2 subsystem.

DB2 has a multi-tasking structure in which each user's request runs under a different task control block (TCB). In addition, the DB2 system itself has its own TCBs and SRBs for logging and database writes.

To estimate the maximum number of concurrent I/O requests when your system is loaded with data:

Use the following formula:

MAX USERS + 600 sequential prefetches + 600 asynchronous writes

*Changing catalog and directory size and location:*

You can change the size or location of your DB2 catalog or directory .

To change the size or location of DB2 catalog or directory data sets:

Choose one of the following actions:

- Run the RECOVER utility on the appropriate database
- Run the REORG utility on the appropriate table space

A hierarchy of recovery dependencies determines the order in which you should try to recover data sets.

**Related concepts**

Chapter 20, "Using tools to monitor performance," on page 419

## Formatting early and speed-up formatting

You can speed up pre-formatting of data by allocating in cylinders, and by preformatting a table space before inserting data.

## Allocating space in cylinders or in large primary and secondary quantities

Specify your space allocation amounts to ensure allocation by CYLINDER. If you use record allocation for more than a cylinder, cylinder allocation is used. Cylinder allocation can reduce the time required to do SQL mass inserts and to perform LOGONLY recovery; it does not affect the time required to recover a table space from an image copy or to run the REBUILD utility.

When inserting records, DB2 pre-formats space within a page set as needed. The allocation amount, which is either CYLINDER or TRACK, determines the amount of space that is pre-formatted at any one time.

Because less space is pre-formatted at one time for the TRACK allocation amount, a mass insert can take longer when the allocation amount is TRACK than the same insert when the allocation amount is CYLINDER. However, smart secondary space allocation minimizes the difference between TRACK and CYLINDER.

The allocation amount is dependent on device type and the number of bytes you specify for PRIQTY and SECQTY when you define table spaces and indexes. The default SECQTY is 10% of the PRIQTY, or 3 times the page size, whichever is larger. This default quantity is an efficient use of storage allocation. Choosing a SECQTY value that is too small in relation to the PRIQTY value results in track allocation.

For more information about how space allocation amounts are determined, see the description of the DEFINE CLUSTER command in *DFSMS/MVS: Access Method Services for the Integrated Catalog*.

## Pre-formatting during LOAD or REORG

When DB2 pre-formatting delays impact the performance or execution time consistency of applications that do heavy insert processing, and if the table size can be predicted for a business processing cycle, consider using the PREFORMAT option of LOAD and REORG. If you preformat during LOAD or REORG, DB2 does not have to preformat new pages during execution. When the pre-formatted space is used and when DB2 has to extend the table space, normal data set extending and pre-formatting occurs.

Consider pre-formatting only if pre-formatting is causing a measurable delay with the insert processing or causing inconsistent elapsed times for insert applications.

**Recommendation:** Quantify the results of pre-formatting in your environment by assessing the performance both before and after using pre-formatting.

## Avoiding excessively small extents

Data set extent size affects performance because excessively small extents can degrade performance during a sequential database scan.

## Example

Suppose that the sequential data transfer speed is 100 MB per second and that the extent size is 10 MB. The sequential scan must move to a new extent ten times per second.

**Recommendation:** Maintain extent sizes that are large enough to avoid excessively frequent extent moving during scans. Keep the extent size greater than 10 cylinders for large data sets.

## Maximum number of extents

An SMS-managed linear data set is limited to 123 extents on a volume and 7257 total extents on all volumes. A non-SMS-managed data set is limited to 123 extents on a volume and 251 total extents on all volumes. If a data set grows and extents are not monitored, jobs eventually fail due to these extent limitations.

**Recommendation:** Monitor the number of extents to avoid reaching the maximum number of extents on a volume and the maximum number of extents on all volumes.

## Specifying primary quantity for nonpartitioned indexes

Specifying sufficient primary and secondary allocations for frequently used data sets minimizes I/O time, because the data is not located at different places on the disks.

Listing the catalog or VTOC occasionally to determine the number of secondary allocations that have been made for your more frequently used data sets can also be helpful. Alternatively, you can use IFCID 0258 in the statistics class 3 trace and real time statistics to monitor data set extensions. OMEGAMON monitors IFCID 0258.

**GUIP** To prevent wasted space for non-partitioned indexes, take one of the following actions:

- Let DB2 use the default primary quantity and calculate the secondary quantities. Do this by specifying 0 for the IXQTY subsystem parameter, and by omitting a PRIQTY and SECQTY value in the CREATE INDEX statement or ALTER INDEX statement. If a primary and secondary quantity were previously specified for an index, you can specify PRIQTY -1 and SECQTY -1 to change to the default primary quantity and calculated secondary quantity.
- If the MGEXTSZ subsystem parameter is set to NO, so that you control secondary space allocations, make sure that the value of  $\text{PRIQTY} + (N \times \text{SECQTY})$  is a value that evenly divides into PIECESIZE.

**GUIP**

### Related concepts

“Real-time statistics” on page 175

## Reserving free space

By reserving free space, you can maintain the physical clustering of data and reduce the need to frequently reorganize table spaces and indexes. However, you do not want to allocate too much disk space.

**GUIP** You can use PCTFREE and FREEPAGE clauses to reserve free space in table spaces and indexes.

The decision of whether to specify the percentage of free space per page or the number of free pages depends on the type of SQL and the distribution of that activity across the table space or index.

When free space is needed, using PCTFREE is recommended rather than FREEPAGE in most situations.

- If update activity on compressed data, which often results in longer rows, is heavy or insert volume is heavy, use a PCTFREE value greater than the default.
- For concurrency, use MAXROWS or larger PCTFREE values for small tables and shared table spaces that use page locking. This reduces the number of rows per page, thus reducing the frequency that any given page is accessed.

In some situations, using FREEPAGE is recommended over PCTFREE:

- Use FREEPAGE rather than PCTFREE if MAXROWS is 1 or rows are larger than half a page because DB2 cannot insert a second row on a page.
- For the DB2 catalog table spaces and indexes, use the default values for PCTFREE. If additional free space is needed, use FREEPAGE.

The table spaces and indexes for the DB2 catalog can also be altered to modify FREEPAGE and PCTFREE. These options are not applicable for LOB table spaces.

**GUIP**

#### **When to reserve free space:**

By reserving free space, you can maintain the physical clustering of data and reduce the need to frequently reorganize table spaces and indexes.

However, you do not want to allocate too much disk space. When deciding whether to allocate free space, consider the data and each index separately and assess the insert and update activity on the data and indexes.

When you specify a sufficient amount of free space, advantages during normal processing include:

- Better clustering of rows (giving faster access)
- Fewer overflows
- Less frequent reorganizations needed
- Less information locked by a page lock
- Fewer index page splits

Disadvantages of specifying free space include:

- More disk space occupied
- Less information transferred per I/O
- More pages to scan
- Possibly more index levels
- Less efficient use of buffer pools and storage controller cache

## When free space is not needed

You do not need to specify free for tables and indexes in the following situations:

- The object is read-only.  
If you do not plan to insert or update data in a table, no free space is needed for either the table or its indexes.
- The object is not read-only, but inserts are at the end, and updates that lengthen varying-length columns are few.

For example, if inserts are in ascending order by key of the clustering index or are caused by LOAD RESUME SHRLEVEL NONE and update activity is only on fixed-length columns with non-compressed data, the free space for both the table and clustering index should be zero.

Generally, free space is beneficial for a non-clustering index because inserts are usually random. However, if the non-clustering index contains a column with a timestamp value that causes the inserts into the index to be in sequence, the free space should be zero.

## Specifying free space on pages:

The PCTFREE clause specifies what percentage of each page in a table space or index is left free when loading or reorganizing the data.

**GUPI** DB2 uses the free space later when you insert or update your data. When no free space is available, DB2 holds your additional data on another page. When several records are physically located out of sequence, performance suffers. If you have previously used a large PCTFREE value to force one row per page, you should specify a MAXROWS clause instead.

To specify the percentage of free space on a page:

- To determine the amount of free space currently on a page, run the RUNSTATS utility, and examine the PERCACTIVE column of SYSIBM.SYSTABLEPART.
- Specify the value of PCTFREE for the table space or index. For example, the default value for PCTFREE is 5, meaning that 5% of every page is kept free when you load or reorganize the data. You can include the PCTFREE clause for a number of statements.
  - ALTER INDEX
  - ALTER TABLESPACE
  - CREATE INDEX
  - CREATE TABLESPACE

The default value of PCTFREE for indexes is 10. The maximum amount of space that is left free in index nonleaf pages is 10%, even if you specify a value higher than 10 for PCTFREE.

- If you previously used a large PCTFREE value to force one row per page, specify a MAXROWS clause with a value of 1 on the CREATE or ALTER TABLESPACE statement instead. The MAXROWS clause has the advantage of maintaining the free space even when new data is inserted. **GUPI**

## Specifying single-row pages:

If you have previously used a large PCTFREE value to force one row per page, you should specify a MAXROWS clause instead.

The MAXROWS clause has the advantage of maintaining the free space even when new data is inserted.

To specify single-row pages:

Include a MAXROWS clause with a value of 1 on the CREATE or ALTER TABLESPACE statement.

If more free space is needed, use FREEPAGE rather than PCTFREE if MAXROWS is 1.

### Specifying the ratio of free pages:

The FREEPAGE clause specifies how often DB2 leaves a full page of free space when loading data or when reorganizing data or indexes.

DB2 uses the free space that you allocated later when you insert or update your data.

For example, if you specify 10 for value of the FREEPAGE clause, DB2 leaves every 10th page free.

Specify the value of FREEPAGE for the tablespace or index. You can include the FREEPAGE clause for the following statements:

- ALTER INDEX
- ALTER TABLESPACE
- CREATE INDEX
- CREATE TABLESPACE

The maximum value you can specify for FREEPAGE is 255; however, in a segmented table space, the maximum value is 1 less than the number of pages specified for SEGSIZE.

## Compressing your data

You can reduce the amount of disk space that is needed by compressing your data.

You can use the DSN1COMP utility to determine how well compression of your data will work. Data in a LOB table space or a table space that is defined in the work file database (the table space for declared temporary tables) cannot be compressed.

When you compress data, bit strings that occur frequently are replaced by shorter strings. Information about the mapping of bit strings to their replacements is stored in a *compression dictionary*. Computer processing is required to compress data before it is stored and to decompress the data when it is retrieved from storage. In many cases, using the COMPRESS clause can significantly reduce the amount of disk space needed to store data, but the compression ratio that you achieve depends on the characteristics of your data.

With compressed data, you might see some of the following performance benefits, depending on the SQL work load and the amount of compression:

- Higher buffer pool hit ratios
- Fewer I/Os
- Fewer getpage operations

To compress data:

1. Specify COMPRESS YES in the appropriate SQL statement:
  - CREATE TABLESPACE
  - ALTER TABLESPACE
2. Run the LOAD or REORG utility.

### **Deciding whether to compress data:**

You should consider many factors before you decide whether to compress data.

Consider these factors before compressing data:

#### **Data row size**

DB2 compresses the data of one record at a time. (The prefix of the record is not compressed.) As row lengths become shorter, compression yields diminishing returns because 8 bytes of overhead are required to store each record in a data page. On the other hand, when row lengths are very long, compression of the data portion of the row might yield little or no reduction in data set size because DB2 rows cannot span data pages. In the case of very long rows, using a larger page size can enhance the benefits of compression, especially if the data is accessed primarily in a sequential mode.

If compressing the record produces a result that is no shorter than the original, DB2 does not compress the record.

#### **Table space size**

Compression can work very well for large table spaces. With small table spaces, the size of the compression dictionary (64 KB) can offset the space savings that compression provides.

#### **Processing costs**

Decompressing a row of data costs significantly less than compressing that same row.

The data access path that DB2 uses affects the processor cost for data compression. In general, the relative overhead of compression is higher for table space scans and is less costlier for index access.

#### **I/O costs**

When rows are accessed sequentially, fewer I/Os might be required to access data that is stored in a compressed table space. However, the reduced I/O resource consumption is traded for extra processor cost for decoding the data.

- If random I/O is necessary to access the data, the number of I/Os does not decrease significantly, unless the associated buffer pool is larger than the table and the other applications require little concurrent buffer pool usage.
- Some types of data compress better than others. Data that contains hexadecimal characters or strings that occur with high frequency compresses quite well, while data that contains random byte frequencies might not compress at all. For example, textual and decimal data tends to compress well because certain byte strings occur frequently.

## Data patterns

The frequency of patterns in the data determines the compression savings. Data with many repeated strings (such as state and city names or numbers with sequences of zeros) results in good compression savings.

## Table space design

Each table space or partition that contains compressed data has a compression dictionary, which is built by using the LOAD utility with the REPLACE or RESUME NO options or the REORG TABLESPACE utility without the KEEPDICTIONARY option for either utility.

A dictionary is also built for LOAD RESUME YES without KEEPDICTIONARY if the table space has no rows. The dictionary contains a fixed number of entries, usually 4096, and resides with the data. The dictionary content is based on the data at the time it was built, and does not change unless the dictionary is rebuilt or recovered, or compression is disabled with ALTER TABLESPACE.

If you use LOAD to build the compression dictionary, the first  $n$  rows loaded in the table space determine the contents of the dictionary. The value of  $n$  is determined by how much your data can be compressed.

If you have a table space with more than one table and the data used to build the dictionary comes from only one or a few of those tables, the data compression might not be optimal for the remaining tables. Therefore, put a table you want to compress into a table space by itself, or into a table space that only contains tables with similar kinds of data.

REORG uses a sampling technique to build the dictionary. This technique uses the first  $n$  rows from the table space and then continues to sample rows for the remainder of the UNLOAD phase. In most cases, this sampling technique produces a better dictionary than does LOAD, and using REORG might produce better results for table spaces that contain tables with dissimilar kinds of data.

## Existing exit routines

An exit routine is executed before compressing or after decompressing, so you can use DB2 data compression with your existing exit routines. However, do not use DB2 data compression in conjunction with DSN8HUFF. (DSN8HUFF is a sample edit routine that compresses data using the Huffman algorithm, which is provided with DB2. This adds little additional compression at the cost of significant extra CPU processing.

## Logging effects

If a data row is compressed, all data that is logged because of SQL changes to that data is compressed. Thus, you can expect less logging for insertions and deletions; the amount of logging for updates varies. Applications that are sensitive to log-related resources can experience some benefit with compressed data.

External routines that read the DB2 log cannot interpret compressed data without access to the compression dictionary that was in effect when the data was compressed. However, using IFCID 306, you can cause DB2 to write log records of compressed data in decompressed format. You can retrieve those decompressed records by using the IFI function READS.

## **Distributed data**

DB2 decompresses data before transmitting it to VTAM.

### **Increasing free space for compressed data:**

You can provide free space to avoid the potential problem of more getpage and lock requests for compressed data.

In some cases, using compressed data results in an increase in the number of getpages, lock requests, and synchronous read I/Os. Sometimes, updated compressed rows cannot fit in the home page, and they must be stored in the overflow page. This can cause additional getpage and lock requests. If a page contains compressed fixed-length rows with no free space, an updated row probably has to be stored in the overflow page.

To avoid the potential problem of more getpage and lock requests:

Add more free space within the page. Start with 10% additional free space and adjust further, as needed. If, for example, 10% free space was used without compression, start with 20% free space with compression for most cases. This recommendation is especially important for data that is heavily updated.

### **Determining the effectiveness of compression:**

Before compressing data, you can use the DSN1COMP stand-alone utility to estimate how well it can be compressed.

After data is compressed, you can use compression reports and catalog statistics to determine how effectively it was compressed.

To find the effectiveness of data compression:

- Use the DSN1COMP stand-alone utility to find out how much space can be saved and how much processing the compression of your data requires. Run DSN1COMP on a data set that contains a table space, a table space partition, or an image copy. DSN1COMP generates a report of compression statistics but does not compress the data.
- Examine the compression reports after you use REORG or LOAD to build the compression dictionary and compress the data. Both utilities issue a report message (DSNU234I or DSNU244I). The report message gives information about how well the data is compressed and how much space is saved. (REORG with the KEEPDICTIONARY option does not produce the report.)
- Query catalog tables to find information about data compression
  - PAGESAVE column of the SYSIBM.SYSTABLEPART tells you the percentage of pages that are saved by compressing the data.
  - PCTROWCOMP columns of SYSIBM.SYSTABLES and SYSIBM.SYSTABSTATS tells you the percentage of the rows that were compressed in the table or partition the last time RUNSTATS was run. Use the RUNSTATS utility to update these catalog columns.

## **Evaluating your indexes**

In many cases, you might be able to eliminate indexes that are no longer necessary or change the characteristics of an index to reduce disk usage.

Dropping unneeded indexes also improves performance because of savings in index maintenance.

### **Eliminating unnecessary partitioning indexes:**

A partitioning index requires as many data sets as the partitioned table space.

In *table-controlled partitioning*, partitioning key and limit keys are specified in the CREATE TABLESPACE statement and not the CREATE INDEX statement, which eliminates the need for any partitioning index data sets.

To eliminate the disk space that is required for the partitioning index data sets:

1. Drop partitioning indexes that are used solely to partition the data, and not to access it.
2. Convert to table-controlled partitioning.

### **Dropping indexes that were created to avoid sorts:**

Indexes that are defined only to avoid a sort for queries with an ORDER BY clause are unnecessary if DB2 can perform a backward scan of another index to avoid the sort.

In earlier versions of DB2, you might have created ascending and descending versions of the same index for the sole purpose of avoiding a sort operation.

To recover the space that is used by these indexes:

Drop indexes that were created to avoid sorts.

For example, consider the following query:

```
SELECT C1, C2, C3 FROM T
WHERE C1 > 1
ORDER BY C1 DESC;
```

Having an ascending index on C1 would not have prevented a sort to order the data. To avoid the sort, you needed a descending index on C1. DB2 can scan an index either forwards or backwards, which can eliminate the need to have indexes with the same columns but with different ascending and descending characteristics.

For DB2 to be able to scan an index backwards, the index must be defined on the same columns as the ORDER BY and the ordering must be exactly opposite of what is requested in the ORDER BY. For example, if an index is defined as C1 DESC, C2 ASC, DB2 can use:

- A forward scan of the index for ORDER BY C1 DESC, C2 ASC
- A backward scan of the index for ORDER BY C1 ASC, C2 DESC

However, DB2 does need to sort for either of the following ORDER BY clauses:

- ORDER BY C1 ASC, C2 ASC
- ORDER BY C1 DESC, C2 DESC

### **Using non-padded indexes:**

You can save disk space by using non-padded indexes instead of padded indexes.

When you define an index as NOT PADDED, the varying-length columns in the index are not padded to their maximum length. If the index contains at least one varying-length column, the length information is stored with the key. Consequently, the amount of savings depends on the number of varying-length columns in the index and the actual length of the columns in those indexes versus their maximum lengths.

To use index padding efficiently:

As a general rule, use non-padded indexes only if the average amount that is saved is greater than about 18 bytes per column. For example, assume that you have an index key that is defined on a VARCHAR(128) column and the actual length of the key is 8 bytes. An index that is defined as NOT PADDED would require approximately 9 times less storage than an index that is defined as PADDED, as shown by the following calculation:

$$(128 + 4) / (8 + 2 + 4) = 9$$

### **Compressing indexes:**

You can reduce the amount of space that an index takes up on disk by compressing the index.

However, keep in mind that index compression is heavily data-dependent, and some indexes might contain data that will not yield significant space savings. Compressed indexes might also use more real and virtual storage than non-compressed indexes. The amount of additional real and virtual storage required depends on the compression ratio used for the compressed keys, the amount of free space, and the amount of space used by the key map.

To reduce the size of the index on disk:

- Use the DSN1COMP utility on existing indexes to get an indication of the appropriate page size for new indexes. You can choose 8K and 16K bufferpool page sizes for the index. Choosing a 16K buffer pool instead of a 8K buffer pool accommodates a potentially higher compression ratio, but also increases the potential to use more storage. Estimates for index space savings from the DSN1COMP utility, whether on the true index data or some similar index data, will not be exact.
- Specify the COMPRESS YES option when you issue an ALTER INDEX or CREATE INDEX statement.

### **Index splitting for sequential INSERT activity:**

DB2 detects sequential inserts and splits index pages asymmetrically to improve space usage and reduce split processing.

You can further improve performance by choosing the appropriate page size for index pages.

When all the entries in a leaf page are consumed during inserts, DB2 allocates a new page and moves some entries from the old page to the new page. DB2 detects when a series of inserts adds keys in ascending or descending sequential order.

| When such a pattern is detected, DB2 splits the index pages asymmetrically, by  
| placing more or fewer keys on the newly allocated page. In this way, DB2 allocates  
| page space more efficiently and reduces the frequency of split processing  
| operations.

| Traditionally, DB2 split index pages by moving approximately half the entries to  
| the new page. According to that logic, when sequential inserts added keys in  
| ascending order, the freed space in the old index page was never used. This meant  
| that an index used only half of the allocated page space. Page-splitting also  
| occurred more frequently because the index would fill the available half of each  
| newly allocated page very quickly.

| Larger index page sizes can be beneficial in cases where a frequent index split  
| results from heavy inserts. The frequency of index splitting can be determined  
| from LEAFNEAR, LEAFFAR, and NLEAF in SYSINDEXES and SYSINDEXPART  
| catalog tables, latch 70 (and latch class 6 in statistics) contention in data sharing,  
| and latch 254 contention in non data sharing from performance trace.

| A smaller index page size can be beneficial for achieving higher buffer pool hit  
| ratios in random read-intensive applications.



---

## Chapter 8. Improving DB2 log performance

By understanding the day-to-day activity on the log, you can more effectively pinpoint when problems occur and better understand how to tune for best performance.

DB2 logs changes made to data, and other significant events, as they occur. The characteristics of your workload have a direct effect on log write performance. Long-running tasks that commit infrequently incur a lot more data to write at commit than a typical transaction. These tasks can cause subsystem impact because of the excess storage consumption, locking contention, and resources that are consumed for a rollback.

Do not forget to consider the cost of reading the log as well. The cost of reading the log directly affects how long a restart or a recovery takes because DB2 must read the log data before applying the log records back to the table space.

---

### Improving log write performance

To improve log write performance:

- Choose a large size for OUTPUT BUFFER size. The OUTPUT BUFFER field of installation panel DSN TIPL lets you specify the size of the output buffer used for writing active log data sets. The maximum size of this buffer (OUTBUFF) is 400,000 KB. Choose as large a size as your system can tolerate to decrease the number of forced I/O operations that occur because there are no more buffers. A large size can also reduce the number of wait conditions.
- Choose fast devices for log data sets. The devices that are assigned to the active log data sets must be fast. In environments with high levels of write activity, high-capacity storage systems, such as the IBM TotalStorage DS8000 series, are recommended to avoid logging bottlenecks.
- Avoid device contention. Place the copy of the bootstrap data set and, if using dual active logging, the copy of the active log data sets, on volumes that are accessible on a path different than that of their primary counterparts.
- Preformat new active log data sets. Whenever you allocate new active log data sets, preformat them using the DSNJLOGF utility. This action avoids the overhead of preformatting the log, which normally occurs at unpredictable times.
- Stripe active log data sets. The active logs can be striped using DFSMS. *Striping* is a technique to improve the performance of data sets that are processed sequentially. Striping is achieved by splitting the data set into segments or stripes and spreading those stripes across multiple volumes. Striping can improve the maximum throughput log capacity and is most effective when many changes to log records occur between commits. Striping is useful if you have a high I/O rate for the logs. Striping is needed more with ESCON channels than with the faster FICON channels.
- Stripe archive log data sets on disk. If writes to the archive do not complete as fast as writes to the active log, transactions might slow down while waiting for the active log to be emptied. Second, when applying log records from the archive log, striping helps DB2 read the archive data sets faster.

## Types of log writes

Log writes are divided into two categories: asynchronous and synchronous.

### Asynchronous writes

Asynchronous writes are the most common. These asynchronous writes occur when data is updated. Before, and after, image records are usually moved to the log output buffer, and control is returned to the application. However, if no log buffer is available, the application must wait for one to become available.

### Synchronous writes

Synchronous writes usually occur at commit time when an application has updated data. This write is called 'forcing' the log because the application must wait for DB2 to force the log buffers to disk before control is returned to the application. If the log data set is not busy, all log buffers are written to disk. If the log data set is busy, the requests are queued until it is freed.

### Writing to two logs

Dual logging is shown in the figure below.

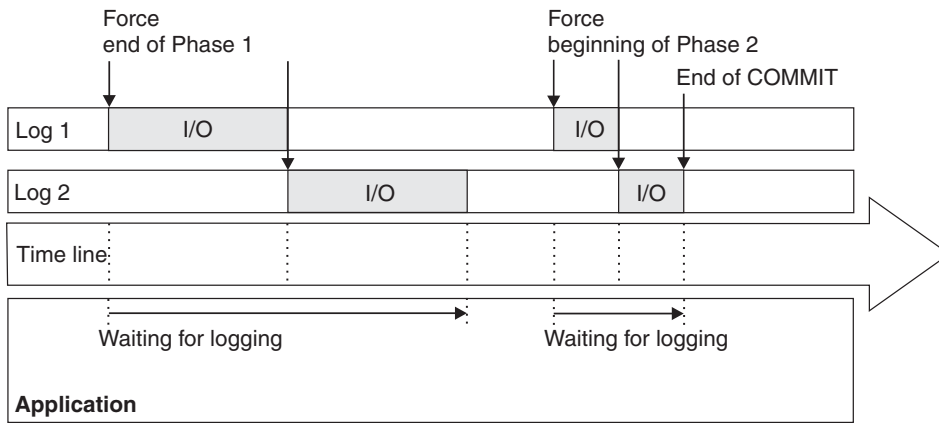


Figure 5. Dual logging during two-phase commit

If you use dual logging (recommended for availability), the write to the first log sometimes must complete before the write to the second log begins. The first time a log control interval is written to disk, the write I/Os to the log data sets are performed in parallel. However, if the same 4-KB log control interval is again written to disk, the write I/Os to the log data sets must be done serially to prevent any possibility of losing log data in case of I/O errors on both copies simultaneously.

### Two-phase commit log writes

Because they use two-phase commit, applications that use the CICS, IMS, and RRS attachment facilities force writes to the log twice. The first write forces all the log records of changes to be written (if they have not been written previously because of the write threshold being reached). The second write writes a log record that takes the unit of recovery into an in-commit state.

---

## Improving log read performance

The performance impact of log reads is evident during a rollback, restart, and database recovery.

DB2 must read from the log and apply changes to the data on disk. Every process that requests a log read has an input buffer dedicated to that process. DB2 searches for log records in the following order:

1. Output buffer
2. Active log data set
3. Archive log data set

If the log records are in the output buffer, DB2 reads the records directly from that buffer. If the log records are in the active or archive log, DB2 moves those log records into the input buffer used by the reading process (such as a recovery job or a rollback).

DB2 reads the log records faster from the active log than from the archive log. Access to archived information can be delayed for a considerable length of time if a unit is unavailable or if a volume mount is required (for example, a tape mount).

To improve log read performance:

- Archive to disk. If the archive log data set resides on disk, it can be shared by many log readers. In contrast, an archive on tape cannot be shared among log readers. Although it is always best to avoid reading archives altogether, if a process must read the archive, that process is serialized with anyone else who must read the archive tape volume. For example, every rollback that accesses the archive log must wait for any previous rollback work that accesses the same archive tape volume to complete. If you do not have enough space to maintain the archive data sets on disk, consider using DFHSM to write the archive data sets to tape. This method has a disadvantage in that HSM must read the archive data set from disk in order to write it to tape, but the recall process is improved for a number of reasons. You can pre-stage the recalls of archive data sets in parallel (to striped data sets), and when the data sets are recalled, parallel readers can proceed.
- Avoid device contention on the log data sets by placing your active log data sets on different volumes and I/O paths to avoid I/O contention in periods of high concurrent log read activity. When multiple concurrent readers access the active log, DB2 can ease contention by assigning some readers to a second copy of the log. Therefore, for performance and error recovery, use dual logging and place the active log data sets on a number of different volumes and I/O paths. Whenever possible, put data sets within a copy or within different copies on different volumes and I/O paths. Ensure that no data sets for the first copy of the log are on the same volume as data sets for the second copy of the log.
- Stripe active log data sets. The active logs can be striped using DFSMS. *Striping* is a technique to improve the performance of data sets that are processed sequentially. Striping is achieved by splitting the data set into segments or stripes and spreading those stripes across multiple volumes. Striping can improve the maximum throughput log capacity and is most effective when many changes to log records occur between commits. Striping is useful if you have a high I/O rate for the logs. Striping is needed more with ESCON channels than with the faster FICON channels.

---

## Log statistics

You can gather statistics for logging activities from the OMEGAMON statistics report.

A non-zero value for **A** in the following example indicates that your output buffer is too small. Ensure that the size you choose is backed up by real storage. A non-zero value for **B** is an indicator that your output buffer is too large for the amount of available real storage.

LOG ACTIVITY	QUANTITY	/SECOND	/THREAD	/COMMIT
READS SATISFIED-OUTPUT BUFF	211.00	0.12	N/C	0.00
READS SATISFIED-OUTP.BUF(%)	100.00			
READS SATISFIED-ACTIVE LOG	0.00	0.00	N/C	0.00
READS SATISFIED-ACTV.LOG(%)	0.00			
READS SATISFIED-ARCHIVE LOG	0.00	0.00	N/C	0.00
READS SATISFIED-ARCH.LOG(%)	0.00			
TAPE VOLUME CONTENTION WAIT	0.00	0.00	N/C	0.00
READ DELAYED-UNAVAIL.RESOUR	0.00	0.00	N/C	0.00
ARCHIVE LOG READ ALLOCATION	0.00	0.00	N/C	0.00
ARCHIVE LOG WRITE ALLOCAT.	0.00	0.00	N/C	0.00
CONTR.INTERV.OFFLOADED-ARCH	0.00	0.00	N/C	0.00
LOOK-AHEAD MOUNT ATTEMPTED	0.00	0.00	N/C	0.00
LOOK-AHEAD MOUNT SUCCESSFUL	0.00	0.00	N/C	0.00
UNAVAILABLE OUTPUT LOG BUFF <b>A</b>	0.00	0.00	N/C	0.00
OUTPUT LOG BUFFER PAGED IN <b>B</b>	8.00	0.00	N/C	0.10
LOG RECORDS CREATED <b>C</b>	234.00	0.13	N/C	2.85
LOG CI CREATED <b>D</b>	9.00	0.01	N/C	0.11
LOG WRITE I/O REQ (COPY1&2)	96.00	0.05	N/C	1.17
LOG CI WRITTEN (COPY1&2)	96.00	0.05	N/C	1.17
LOG RATE FOR 1 LOG (MB/Sec)	N/A	0.00	N/A	N/A
LOG WRITE SUSPENDED	39.00	0.02	N/C	0.48

Figure 6. Log statistics in the OMEGAMON statistics report

### Calculating average log record size

One way to determine how much log volume you need is to consider the average size of a log record.

As a general estimate, you can start with 200 bytes as the average size. To increase the accuracy of this estimate, get the real size of the log records that are written.

To calculate the average size of log records that are written:

1. Collect the following values from the statistics report:

**LOG RECORDS CREATED**

The number of log records created ( **C** )

**LOG CI CREATED**

The number of control intervals created in the active log counter ( **D** )

2. Use the following formula:

$$\mathbf{D} \times 4 \text{ KB} / \mathbf{C} = \text{avg size of log record}$$

---

## Improving log capacity

The capacity that you specify for the active log affects DB2 performance significantly.

If you specify a capacity that is too small, DB2 might need to access data in the archive log during rollback, restart, and recovery. Accessing an archive takes a considerable amount of time.

The following subsystem parameters affect the capacity of the active log. In each case, increasing the value that you specify for the parameter increases the capacity of the active log.

### **NUMBER OF LOGS field on installation panel DSNTIPL**

Controls the number of active log data sets that you create.

### **ARCHIVE LOG FREQ field on installation panel DSNTIPL**

Where you provide an estimate of how often active log data sets are copied to the archive log.

### **UPDATE RATE field on installation panel DSNTIPL**

Where you provide an estimate of how many database changes (inserts, update, and deletes) you expect per hour.

### **The DB2 installation CLIST**

Uses UPDATE RATE and ARCHIVE LOG FREQ to calculate the data set size of each active log data set.

### **CHECKPOINT FREQ field on installation panel DSNTIPN**

Specifies the number of log records that DB2 writes between checkpoints or the number of minutes between checkpoints.

## Total capacity and the number of logs

You need to have sufficient capacity in the active log to avoid reading the archives, and you need to consider how that total capacity should be divided.

Those requirements can make the configuration of your active log data sets a challenge. Having too many or too few active log data sets has ramifications. This information is summarized in the following table.

*Table 22. The effects of installation options on log data sets.* You can modify the size of the data sets in installation job DSNTIJIN

Value for ARCHIVE LOG FREQ	Value for NUMBER OF LOGS	Result
Low	High	Many small data sets. Can cause operational problems when archiving to tape. Checkpoints occur too frequently.
High	Low	Few large data sets. Can result in a shortage of active log data sets.

## Choosing a checkpoint frequency

If log data sets are too small, checkpoints occur too frequently, and database writes are not efficient.

At least one checkpoint is taken each time DB2 switches to a new active log data set. As a guideline, you should provide enough active log space for at least 10 checkpoint intervals.

To specify the checkpoint interval:

- Specify the CHECKPOINT FREQ subsystem parameter. You can change CHECKPOINT FREQ dynamically with the SET LOG or SET SYSPARM command.

You can specify the interval in terms of the number of log records that are written between checkpoints or the number of minutes between checkpoints.

- Avoid taking more than one checkpoint per minute by raising the CHECKPOINT FREQ value so that the checkpoint interval becomes at least one minute during peak periods. In general, the recommended checkpoint frequency is between 2 and 5 minutes.

## Increasing the number of active log data sets

You can use the change log inventory utility (DSNJU003) to add more active log data sets to the BSDS.

You can specify a maximum of 93 data sets per active log copy.

## Setting the size of active log data sets

You can modify DSNTIJIN installation job to change the size of your active log data set.

To choose the most effective size for your active log data set:

- When you calculate the size of the active log data set, identify the longest unit of work in your application programs. For example, if a batch application program commits only once every 20 minutes, the active log data set should be twice as large as the update information that is produced during this period by all of the application programs that are running.

For more information on determining and setting the size of your active log data sets, refer to *DB2 Installation Guide*.

- Allow time for possible operator interventions, I/O errors, and tape drive shortages if off-loading to tape. DB2 supports up to 20 tape volumes for a single archive log data set. If your archive log data sets are under the control of DFSMSHsm, also consider the Hierarchical Storage Manager recall time, if the data set has been migrated by Hierarchical Storage Manager.
- When archiving to disk, set the primary space quantity and block size for the archive log data set so that you can offload the active log data set without forcing the use of secondary extents in the archive log data set. This action avoids space abends when writing the archive log data set.
- Make the number of records for the active log be divisible by the blocking factor of the archive log (disk or tape). DB2 always writes complete blocks when it creates the archive log copy of the active log data set. If you make the archive log blocking factor evenly divisible into the number of active log records, DB2 does not have to pad the archive log data set with nulls to fill the block. This action can prevent REPRO errors if you should ever have to REPRO the archive log back into the active log data set, such as during disaster recovery.
- Make the number of records for the active log be divisible by the blocking factor of the archive log (disk or tape). DB2 always writes complete blocks when it creates the archive log copy of the active log data set. If you make the archive log blocking factor evenly divisible into the number of active log records, DB2

does not have to pad the archive log data set with nulls to fill the block. This action can prevent REPRO errors if you should ever have to REPRO the archive log back into the active log data set, such as during disaster recovery.

To determine the blocking factor of the archive log, divide the value specified on the BLOCK SIZE field of installation panel DSNTIPA by 4096 (that is, BLOCK SIZE / 4096). Then modify the DSNTIJIN installation job so that the number of records in the DEFINE CLUSTER field for the active log data set is a multiple of the blocking factor.

- If you offload to tape, consider adjusting the size of each of your active log data sets to contain the same amount of space as can be stored on a nearly full tape volume. Doing so minimizes tape handling and volume mounts and maximizes the use of the tape resource.

If you change the size of your active log data set to fit on one tape volume, remember that the bootstrap data set is copied to the tape volume along with the copy of the active log data set. Therefore, decrease the size of your active log data set to offset the space that is required on the archive tape for the bootstrap data set.

---

## Controlling the amount of log data

Certain processes such as the LOAD and REORG utility and certain SQL statements can cause a large amount of information to be logged, requiring a large amount of log space.

### Controlling log size for utilities

The REORG and LOAD LOG(YES) utilities cause all reorganized or loaded data to be logged.

For example, if a table space contains 200 million rows of data, this data, along with control information, is logged when this table space is the object of a REORG utility job. If you use REORG with the DELETE option to eliminate old data in a table and run CHECK DATA to delete rows that are no longer valid in dependent tables, you can use LOG(NO) to control log volume.

To reduce the log size:

- When populating a table with many records or reorganizing table spaces or indexes, specify LOG(NO) and take an inline copy or take a full image copy immediately after the LOAD or REORG.
- Specify LOGGED when adding less than 1% of the total table space. Doing so creates additional logging, but eliminates the need for a full image copy

### Controlling log size for SQL operations

The amount of logging performed for applications depends on how much data is changed.

Certain SQL statements are quite powerful, and a single statement can sometimes modify a large amount of data. Such statements include:

#### **INSERT with a fullselect**

A large amount of data can be inserted into table or view, depending on the result of the query.

**Mass deletes and mass updates (except for deleting all rows for a table in a segmented or universal table space)**

For non-segmented table spaces, each of these statements results in the logging of all database data that changes. For example, if a table contains 200 million rows of data, that data and control information are logged if all of the rows of a table are deleted with the SQL DELETE statement. No intermediate commit points are taken during this operation.

For segmented and universal table spaces, a mass delete results in the logging of the data of the deleted records when any of the following conditions are true:

- The table is the parent table of a referential constraint.
- The table is defined as DATA CAPTURE(CHANGES), which causes additional information to be logged for certain SQL operations.
- A delete trigger is defined on the table.

## TRUNCATE TABLE

Essentially a mass-delete that does not activate delete triggers

### Data definition statements

Logging for the entire database descriptor for which the change was made. For very large DBDs, this can be a significant amount of logging.

### Modification to rows that contain LOB data

To control the use of log space by powerful SQL statements:

- For mass delete operations, consider using segmented table spaces or universal table spaces. If segmented table spaces are not an option, and no triggers exist on the table or your application can safely ignore any triggers on the table, create one table per table space, and use TRUNCATE.
- For inserting a large amount of data, instead of using an SQL INSERT statement, use the LOAD utility with LOG(NO) and take an inline copy.
- For updates, consider your workload when defining a table's columns. The amount of data that is logged for update depends on whether the row contains all fixed-length columns or not. For fixed-length non-compressed rows, changes are logged only from the beginning of the first updated column to the end of the last updated column. Consequently, you should keep frequently updated columns close to each other to reduce log quantities.

For varying-length rows, data is logged from the first changed byte to the end of the last updated column. (A varying-length row contains one or more varying-length columns.) Keep varying-length columns at the end of the row to improve read performance, and keep all frequently updated columns near the end of the row to improve update performance. However, if only fixed-length columns are updated frequently, keep those columns close to each other at the beginning of the row.

To determine whether a workload is read-intensive or update-intensive, check the log data rate. Determine the average log size and divide that by 60 to get the average number of log bytes written per second.

- If you log less than 5 MB per second, the workload is read-intensive.
- If you log more than 5 MB per second, the workload is update-intensive.
- If you have many data definition statements (CREATE, ALTER, DROP) for a single database, issue them within a single unit of work to avoid logging the changed DBD for each data definition statement. However, be aware that the DBD is locked until the COMMIT is issued.
- Use the NOT LOGGED option for any **LOB or XML data that requires frequent updating** and for which the trade off of non-recoverability of LOB or XML data from the log is acceptable. (You can still use the RECOVER utility on LOB or

XML table spaces to recover control information that ensures physical consistency of the LOB or XML table space.) Because LOB and XML table spaces defined as NOT LOGGED are not recoverable from the DB2 log, you should make a recovery plan for that data. For example, if you run batch updates, be sure to take an image copy after the updates are complete.

- For data that is modified infrequently, except during certain periods such as year-end processing, when frequent or large changes to the data occur over a short time:
  1. Make an image copy of the data.
  2. Alter the table space to NOT LOGGED.
  3. Make the massive changes.
  4. Stop other activities that update the data.
  5. Make an image copy of the data.
  6. Alter the table space to LOGGED.
- For changes to tables, such as materialized query tables, that contain propagated data, use the NOT LOGGED option because the data exists elsewhere. If the data becomes damaged you can refresh the entire table from the original source.



---

## Chapter 9. Monitoring and tuning stored procedures and user-defined functions

Stored procedures that are created in DB2 and all user-defined functions must run in WLM-established address spaces.

Performance tuning for is the same for user-defined functions and stored procedures in WLM-established address spaces.

---

### Controlling address space storage

You can maximize the number of procedures or functions that can run concurrently in a WLM-established stored procedures address space.

Each task control block that runs in a WLM-established stored procedures address space uses approximately 200 KB below the 16-MB line. DB2 needs this storage for stored procedures and user-defined functions because you can create both main programs and subprograms, and DB2 must create an environment for each.

A stored procedure can invoke only one utility in one address space at any given time because of the resource requirements of utilities. On the WLM Application-Environment panel, set NUMTCB to 1. See Figure 7 on page 110. With NUMTCB=1 or NUMTCB being forced to 1, multiple WLM address spaces are created to run each concurrent utility request that comes from a stored procedure call.

To maximize the number of procedures or functions that can run concurrently in a WLM-established stored procedures address space:

- Set REGION size for the address spaces to REGION=0 to obtain the largest possible amount of storage below the 16-MB line.
- Limit storage required by application programs below the 16-MB line by:
  - Linking editing programs above the line with AMODE(31) and RMODE(ANY) attributes
  - Using the RES and DATA(31) compiler options for COBOL programs
- Limit storage required by Language Environment by using these run-time options:
  - HEAP(,,ANY) to allocate program heap storage above the 16-MB line
  - STACK(,,ANY) to allocate program stack storage above the 16-MB line
  - STORAGE(,,,4K) to reduce reserve storage area below the line to 4 KB
  - BELOWHEAP(4K,,) to reduce the heap storage below the line to 4 KB
  - LIBSTACK(4K,,) to reduce the library stack below the line to 4 KB
  - ALL31(ON) to indicate all programs contained in the stored procedure run with AMODE(31) and RMODE(ANY)

### Dynamically extending load libraries

Use partitioned data set extended (PDSEs) for load libraries containing stored procedures. Using PDSEs might eliminate your need to stop and start the stored procedures address space due to growth of the load libraries. If a load library grows from additions or replacements, the library might have to be extended.

If you use PDSEs for the load libraries, the new extent information is dynamically updated and you do not need to stop and start the address space. If PDSs are used, load failures might occur because the new extent information is not available.

## Assigning procedures and functions to WLM application environments

Workload manager routes work to address spaces based on the application environment name and service class associated with the stored procedure or function.

You must use WLM panels to associate an application environment name with the JCL procedure used to start an address space. See *z/OS MVS Planning: Workload Management* for details about workload management panels.

Other tasks must be completed before a stored procedure or user-defined function can run in a WLM-established stored procedures address space. Here is a summary of those tasks:

1. Make sure you have a numeric value specified in the TIMEOUT VALUE field of installation panel DSNTIPX. If you have problems with setting up the environment, this timeout value ensures that your request to execute a stored procedure does not wait for an unlimited amount of time.

2.

Unless a particular application environment or caller's service class is not used for a long time, WLM creates on demand at least one address space for each combination of application environment name and caller's service class that is encountered in the workload. For example, if five application environment names each have calling threads with six possible service classes, and all those combinations are in demand, it is possible to have 30 address spaces containing stored procedures or user-defined functions.

To prevent creating too many address spaces, create a relatively small number of WLM application environments and z/OS service classes.

3. Use the WLM application environment panels to associate the environment name with the JCL procedure. Figure 7 is an example of this panel.

Application-Environment
Notes
Options
Help

---

Create an Application Environment

Command ==>

Application Environment Name . : WLMENV2 Required  
Description . . . . . Large Stored Proc Env.  
Subsystem Type . . . . . DB2 Required  
Procedure Name . . . . . DSN1WLM  
Start Parameters . . . . . DB2SSN=DB2A,NUMTCB=2,APPLENV=WLMENV2

---

Limit on starting server address space per system
1
1. No limit
2. Single address space per system
3. Single address space per sysplex

Figure 7. WLM panel to create an application environment. You can also use the variable &IWMSSNM for the DB2SSN parameter (DB2SSN=&IWMSSNM). This variable represents the name of the subsystem for which you are starting this address space. This variable is useful for using the same JCL procedure for multiple DB2 subsystems.

4. Specify the WLM application environment name for the WLM\_ENVIRONMENT option on CREATE or ALTER PROCEDURE (or FUNCTION) to associate a stored procedure or user-defined function with an application environment.
5. Using the install utility in the WLM application, install the WLM service definition that contains information about this application environment into the couple data set.
6. Activate a WLM policy from the installed service definition.
7. Begin running stored procedures.



## Chapter 10. Accounting for nested activities

The accounting class 1 and class 2 CPU and elapsed times for triggers, stored procedures, and user-defined functions are accumulated in separate fields and exclude any time accumulated in other nested activity.

These CPU and elapsed times are accumulated for each category during the execution of each agent until agent deallocation. Package accounting can be used to break out accounting data for execution of individual stored procedures, user-defined functions, or triggers. The following figure shows an agent that executes multiple types of DB2 nested activities.

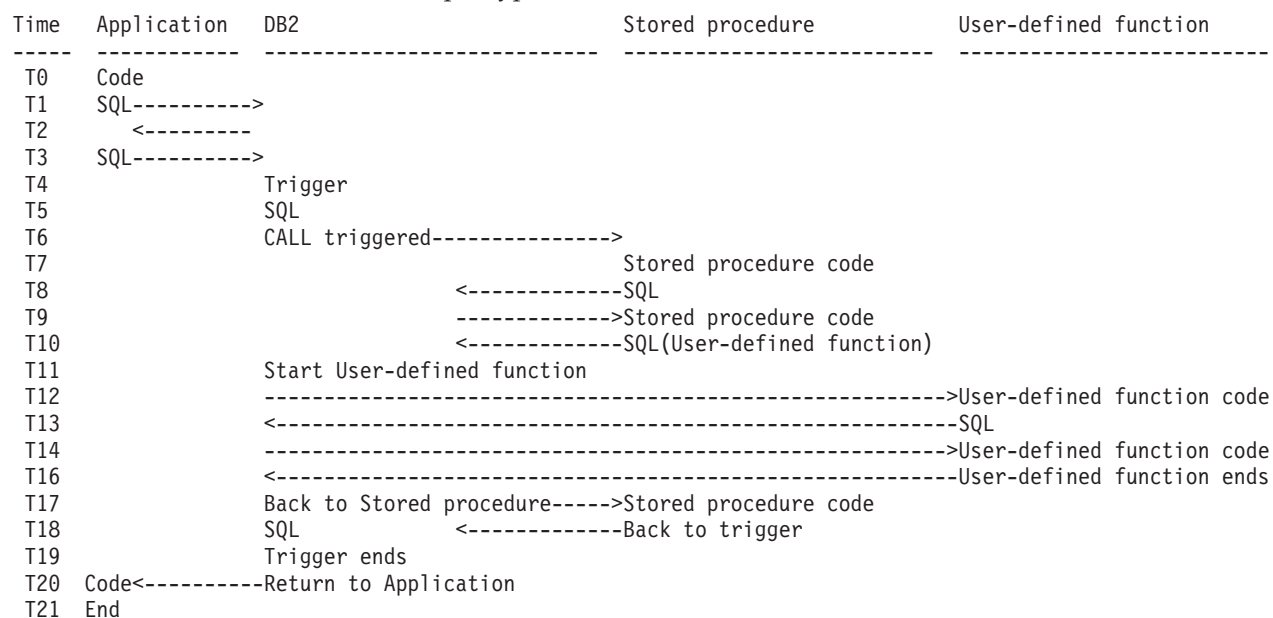


Figure 8. Time spent executing nested activities

Table 23 shows the formula used to determine time for nested activities.

Table 23. Sample for time used for execution of nested activities

Count for	Formula	Class
Application elapsed	T21-T0	1
Application task control block (TU)	T21-T0	1
Application in DB2 elapsed	T2-T1 + T4-T3 + T20-T19	2
Application in DB2 task control block (TU)	T2-T1 + T4-T3 + T20-T19	2
Trigger in DB2 elapsed	T6-T4 + T19-T18	2
Trigger in DB2 task control block (TU)	T6-T4 + T19-T18	2
Wait for STP time	T7-T6 + T18-T17	3
Stored procedure elapsed	T11-T6 + T18-T16	1

Table 23. Sample for time used for execution of nested activities (continued)

Count for	Formula	Class
Stored procedure task control block (TU)	$T11-T6 + T18-T16$	1
Stored procedure SQL elapsed	$T9-T8 + T11-T10 + T17-16$	2
Stored procedure SQL elapsed	$T9-T8 + T11-T10 + T17-T16$	2
Wait for user-defined function time	$T12-T11$	3
User-defined function elapsed	$T16-T11$	1
User-defined function task control block (TU)	$T16-T11$	1
User-defined function SQL elapsed	$T14-T13$	2
User-defined function SQL task control block (TU)	$T14-T13$	2
<b>Note:</b> TU = time used.		

The total class 2 time is the total of the "in DB2" times for the application, trigger, stored procedure, and user-defined function. The class 1 "wait" times for the stored procedures and user-defined functions need to be added to the total class 3 times.

---

## Chapter 11. Using materialized query tables to improve SQL performance

Materialized query tables can simplify query processing, greatly improve the performance of dynamic SQL queries, and are particularly effective in data warehousing applications, where you can use them to avoid costly aggregations and joins against large fact tables.

**PSPI** *Materialized query tables* are tables that contain information that is derived and summarized from other tables. Materialized query tables pre-calculate and store the results of queries with expensive join and aggregation operations. By providing this summary information, materialized query tables can simplify query processing and greatly improve the performance of dynamic SQL queries. Materialized query tables are particularly effective in data warehousing applications.

*Automatic query rewrite* is the process DB2 uses to access data in a materialized query table. If you enable automatic query rewrite, DB2 determines if it can resolve a dynamic query or part of the query by using a materialized query table.

To take advantage of eligible materialized query tables:

Rewrite the queries to use materialized query tables instead of the underlying base tables. Keep in mind that a materialized query table can yield query results that are not current if the base tables change after the materialized query table is updated. **PSPI**

---

### Configuring automatic query rewrite

You can enable DB2 to rewrite certain queries to use materialized query tables instead of base tables for faster processing.

**PSPI** To take advantage of automatic query rewrite with materialized query tables:

1. Define materialized query tables.
2. Populate materialized query tables.
3. Refresh materialized query tables periodically to maintain data currency with base tables. However, realize that refreshing materialized query tables can be an expensive process.
4. Enable automatic query rewrite, and exploit its functions by submitting read-only dynamic queries.
5. Evaluate the effectiveness of the materialized query tables. Drop under-utilized tables, and create new tables as necessary. **PSPI**

### Materialized query tables and automatic query rewrite

As the amount of data has increased, so has the demand for more interactive queries.

**PSPI**

Because databases have grown substantially over the years, queries must operate over huge amounts of data. For example, in a data warehouse environment, decision-support queries might need to operate over 1 to 10 terabytes of data, performing multiple joins and complex aggregation. As the amount of data has increased, so has the demand for more interactive queries.

Despite the increasing amount of data, these queries still require a response time in the order of minutes or seconds. In some cases, the only solution is to pre-compute the whole or parts of each query. You can store these pre-computed results in a materialized query table. You can then use the materialized query table to answer these complicated queries when the system receives them. Using a process called automatic query rewrite, DB2 recognizes when it can transparently rewrite a submitted query to use the stored results in a materialized query table. By querying the materialized query table instead of computing the results from the underlying base tables, DB2 can process some complicated queries much more efficiently. If the estimated cost of the rewritten query is less than the estimated cost of the original query, DB2 uses the rewritten query.

## Automatic query rewrite

When it uses *Automatic query rewrite*, DB2 compares user queries with the fullselect query that defined a materialized query table. It then determines whether the contents of a materialized query table overlap with the contents of a query. When an overlap exists, the query and the materialized query table are said to *match*. After discovering a match, DB2 rewrites the query to access the matched materialized query table instead of the one or more base tables that the original query specified. If a materialized query table overlaps with only part of a query, automatic query rewrite can use the partial match. Automatic query rewrite compensates for the non-overlapping parts of the query by accessing the tables that are specified in the original query.

Automatic query rewrite tries to search for materialized query tables that result in an access path with the lowest cost after the rewrite. DB2 compares the estimated costs of the rewritten query and of the original query and chooses the query with the lower estimated cost.

## Example

Suppose that you have a very large table named TRANS that contains one row for each transaction that a certain company processes. You want to tally the total amount of transactions by some time period. Although the table contains many columns, you are most interested in these four columns:

- YEAR, MONTH, DAY, which contain the date of a transaction
- AMOUNT, which contains the amount of the transaction

To total the amount of all transactions between 2001 and 2006, by year, you would use the following query:

```
SELECT YEAR, SUM(AMOUNT)
FROM TRANS
WHERE YEAR >= '2001' AND YEAR <= '2006'
GROUP BY YEAR
ORDER BY YEAR;
```

This query might be very expensive to run, particularly if the TRANS table is a very large table with millions of rows and many columns.


Now suppose that you define a materialized query table named STRANS by using the following CREATE TABLE statement:

```
CREATE TABLE STRANS AS
  (SELECT YEAR AS SYEAR,
        MONTH AS SMONTH,
        DAY AS SDAY,
        SUM(AMOUNT) AS SSUM
   FROM TRANS
   GROUP BY YEAR, MONTH, DAY)
DATA INITIALLY DEFERRED REFRESH DEFERRED;
```

After you populate STRANS with a REFRESH TABLE statement, the table contains one row for each day of each month and year in the TRANS table.



Using the automatic query rewrite process, DB2 can rewrite the original query into a new query. The new query uses the materialized query table STRANS instead of the original base table TRANS:

```
SELECT SYEAR, SUM(SSUM)
FROM STRANS
WHERE SYEAR >= '2001' AND SYEAR <= '2006'
GROUP BY SYEAR
ORDER BY SYEAR
```

If you maintain data currency in the materialized query table STRANS, the rewritten query provides the same results as the original query. The rewritten query offers better response time and requires less CPU time. 


### Queries that are eligible for rewrite

DB2 supports automatic query rewrite only for read-only, dynamic queries. DB2 cannot automatically rewrite statically bound queries.

 You can always refer to a materialized query table explicitly in a statically bound query or in a dynamically prepared query. However, you should consider updating materialized query table data more frequently if you frequently query the table directly. Also, allowing end users to refer directly to materialized query tables reduces an installation's flexibility of dropping and creating materialized query tables without affecting applications. 

### How DB2 considers automatic query rewrite

In general, DB2 considers automatic query rewrite at the query block level. A read-only, dynamic query can contain multiple query blocks.

 For example, it might contain a subselect of UNION or UNION ALL, temporarily materialized views, materialized table expressions, and subquery predicates. DB2 processes the query without automatic query rewrite if the query block is or contains any of the following items:

- A fullselect in the UPDATE SET statement.
- A fullselect in the INSERT statement.
- A fullselect in the materialized query table definition in the REFRESH TABLE statement.
- An outer join.
- A query block contains user-defined scalar or table functions with the EXTERNAL ACTION attribute or the NON-DETERMINISTIC attribute, or with the built-in function RAND.

If none of these items exist in the query block, DB2 considers automatic query rewrite. DB2 analyzes the query block in the user query and the fullselect in the materialized query table definition to determine if it can rewrite the query. The materialized query table must contain all of the data from the source tables (in terms of both columns and rows) that DB2 needs to satisfy the query. For DB2 to choose a rewritten query, the rewritten query must provide the same results as the user query. (DB2 assumes the materialized query table contains current data.) Furthermore, the rewritten query must offer better performance than the original user query.

DB2 performs a sophisticated analysis to determine whether it can obtain the results for the query from a materialized query table:

- DB2 compares the set of base tables that were used to populate the materialized query table to the set of base tables that are referenced by the user query. If these sets of tables share base tables in common, the query is a candidate for query rewrite.
- DB2 compares the predicates in the materialized query table fullselect to the predicates in the user query. The following factors influence the comparison:
  - The materialized query table fullselect might contain predicates that are not in the user query. If so, DB2 assumes that these predicates might have resulted in discarded rows when the materialized query table was refreshed. Thus, any rewritten query that makes use of the materialized query table might not give the correct results. The query is not a candidate for query rewrite.

**Exception:** DB2 behavior differs if a predicate joins a common base table to an extra table that is unique to the materialized query table fullselect. The predicate does not result in discarded data if you define a referential constraint between the two base tables to make the predicate *lossless*. However, the materialized query table fullselect must not have any local predicates that reference this extra table.

For an example of a lossless predicate, see Example 2 under “Automatic query rewrite—complex examples” on page 119.

- Referential constraints on the source tables are very important in determining whether automatic query rewrite uses a materialized query table.
- Predicates are much more likely to match if you code the predicate in the user query so that it is the same or very similar to the predicate in the materialized query table fullselect. Otherwise, the matching process might fail on some complex predicates.


For example, the matching process between the simple equal predicates such as  $COL1 = COL2$  and  $COL2 = COL1$  succeeds. Furthermore, the matching process between simple equal predicates such as  $COL1 * (COL2 + COL3) = COL5$  and  $COL5 = (COL3 + COL2) * COL1$  succeeds. However, the matching process between equal predicates such as  $(COL1 + 3) * 10 = COL2$  and  $COL1 * 10 + 30 = COL2$  fails.

- The items in an IN-list predicate do not need to be in exactly the same order for predicate matching to succeed.
- DB2 compares GROUP BY clauses in the user query to GROUP BY clauses in the materialized query table fullselect. If the user query requests data at the same or higher grouping level as the data in the materialized query table fullselect, the materialized query table remains a candidate for query rewrite. DB2 uses functional dependency information and column equivalence in this analysis.
- DB2 compares the columns that are requested by the user query with the columns in the materialized query table. If DB2 can derive the result columns

from one or more columns in the materialized query table, the materialized query table remains a candidate for query rewrite. DB2 uses functional dependency information and column equivalence in this analysis.


- DB2 examines predicates in the user query that are not identical to predicates in the materialized query table fullselect. Then, DB2 determines if it can derive references to columns in the base table from columns in the materialized query table instead. If DB2 can derive the result columns from the materialized query table, the materialized query table remains a candidate for query rewrite.

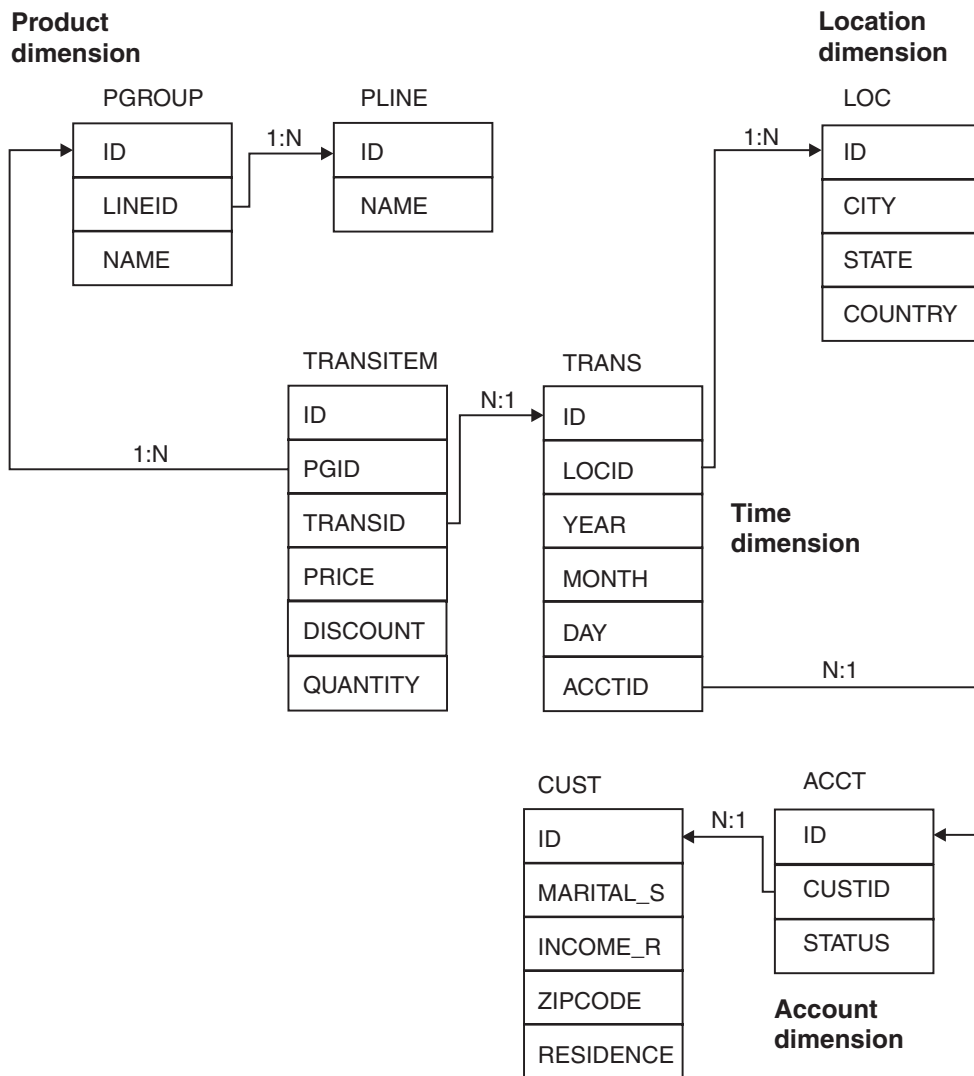
If all of the preceding analyses succeed, DB2 rewrites the user query. DB2 replaces all or some of the references to base tables with references to the materialized query table. If DB2 finds several materialized query tables that it can use to rewrite the query, it might use multiple tables simultaneously. If DB2 cannot use the tables simultaneously, it chooses which one to use according to a set of rules.

After writing the new query, DB2 determines the cost and the access path of that query. DB2 uses the rewritten query if the estimated cost of the rewritten query is less than the estimated cost of the original query. The rewritten query might give only approximate results if the data in the materialized query table is not up to date. 

### **Automatic query rewrite—complex examples**

These examples can help you understand how DB2 applies automatic query rewrite to avoid costly aggregations and joins against large fact tables.

 The following examples assume a scenario in which a data warehouse has a star schema. The star schema represents the data of a simplified credit card application, as shown in the following figure.




**Figure 9. Multi-fact star schema.** In this simplified credit card application, the fact tables **TRANSITEM** and **TRANS** form the hub of the star schema. The schema also contains four dimensions: product, location, account, and time.


The data warehouse records transactions that are made with credit cards. Each transaction consists of a set of items that are purchased together. At the center of the data warehouse are two large fact tables. **TRANS** records the set of credit card purchase transactions. **TRANSITEM** records the information about the items that are purchased. Together, these two fact tables are the hub of the star schema. The star schema is a multi-fact star schema because it contains these two fact tables. The fact tables are continuously updated for each new credit card transaction.

In addition to the two fact tables, the schema contains four dimensions that describe transactions: product, location, account, and time.

- The product dimension consists of two normalized tables, **PGROUP** and **PLINE**, that represent the product group and product line.
- The location dimension consists of a single, denormalized table, **LOC**, that contains city, state, and country.
- The account dimension consists of two normalized tables, **ACCT** and **CUST**, that represent the account and the customer.
- The time dimension consists of the **TRANS** table that contains day, month, and year.

Analysts of such a credit card application are often interested in the aggregation of the sales data. Their queries typically perform joins of one or more dimension tables with fact tables. The fact tables contain significantly more rows than the dimension tables, and complicated queries that involve large fact tables can be very costly. In many cases, you can use materialized query table to summarize and store information from the fact tables. Using materialized query tables can help you avoid costly aggregations and joins against large fact tables. 

## Example 1

 An analyst submits the following query to count the number of transactions that are made in the United States for each credit card. The analyst requests the results grouped by credit card account, state, and year:

```
UserQ1
-----
SELECT T.ACCTID, L.STATE, T.YEAR, COUNT(*) AS CNT
  FROM TRANS T, LOC L
 WHERE T.LOCID = L.ID AND
        L.COUNTRY = 'USA'
 GROUP BY T.ACCTID, L.STATE, T.YEAR;
```

Assume that the following CREATE TABLE statement created a materialized query table named TRANSCNT:

```
CREATE TABLE TRANSCNT AS
  (SELECT ACCTID, LOCID, YEAR, COUNT(*) AS CNT
   FROM TRANS
   GROUP BY ACCTID, LOCID, YEAR )
 DATA INITIALLY DEFERRED
 REFRESH DEFERRED;
```

If you enable automatic query rewrite, DB2 can rewrite UserQ1 as NewQ1. NewQ1 accesses the TRANSCNT materialized query table instead of the TRANS fact table.

```
NewQ1
-----
SELECT A.ACCTID, L.STATE, A.YEAR, SUM(A.CNT) AS CNT
  FROM TRANSCNT A, LOC L
 WHERE A.LOCID = L.ID      AND
        L.COUNTRY = 'USA'
 GROUP BY A.ACCTID, L.STATE, A.YEAR;
```

DB2 can use query rewrite in this case because of the following reasons:

- The TRANS table is common to both UserQ1 and TRANSCNT.
- DB2 can derive the columns of the query result from TRANSCNT.
- The GROUP BY in the query requests data that are grouped at a higher level than the level in the definition of TRANSCNT.

Because customers typically make several hundred transactions per year with most of them in the same city, TRANSCNT is about hundred times smaller than TRANS. Therefore, rewriting UserQ1 into a query that uses TRANSCNT instead of TRANS improves response time significantly.



## Example 2

### GUIP

Assume that an analyst wants to find the number of televisions, with a price over 100 and a discount greater than 0.1, that were purchased by each credit card account. The analyst submits the following query:

UserQ2

```
-----
SELECT T.ID, TI.QUANTITY * TI.PRICE * (1 - TI.DISCOUNT) AS AMT
FROM TRANSITEM TI, TRANS T, PGROUP PG
WHERE TI.TRANSID = T.ID AND
      TI.PGID = PG.ID AND
      TI.PRICE > 100 AND
      TI.DISCOUNT > 0.1 AND
      PG.NAME = 'TV';
```

If you define the following materialized query table TRANSIAB, DB2 can rewrite UserQ2 as NewQ2:

TRANSIAB

```
-----
CREATE TABLE TRANSIAB AS
(SELECT TI.TRANSID, TI.PRICE, TI.DISCOUNT, TI.PGID,
      L.COUNTRY, TI.PRICE * TI.QUANTITY as VALUE
FROM TRANSITEM TI, TRANS T, LOC L
WHERE TI.TRANSID = T.ID AND
      T.LOCID = L.ID AND
      TI.PRICE > 1 AND
      TI.DISCOUNT > 0.1)
DATA INITIALLY DEFERRED
REFRESH DEFERRED;
```

NewQ2

```
-----
SELECT A.TRANSID, A.VALUE * (1 - A.DISCOUNT) as AM
FROM TRANSIAB A, PGROUP PG
WHERE A.PGID = PG.ID AND
      A.PRICE > 100 AND
      PG.NAME = 'TV';
```

DB2 can rewrite UserQ2 as a new query that uses materialized query table TRANSIAB because of the following reasons:

- Although the predicate `T.LOCID = L.ID` appears only in the materialized query table, it does not result in rows that DB2 might discard. The referential constraint between the `TRANS.LOCID` and `LOC.ID` columns makes the join between `TRANS` and `LOC` in the materialized query table definition lossless. The join is *lossless* only if the foreign key in the constraint is `NOT NULL`.
- The predicates `TI.TRANSID = T.ID` and `TI.DISCOUNT > 0.1` appear in both the user query and the TRANSIAB fullselect.
- The fullselect predicate `TI.PRICE > 1` in TRANSIAB subsumes the user query predicate `TI.PRICE > 100` in UserQ2. Because the fullselect predicate is more inclusive than the user query predicate, DB2 can compute the user query predicate from TRANSIAB.
- The user query predicate `PG.NAME = 'TV'` refers to a table that is not in the TRANSIAB fullselect. However, DB2 can compute the predicate from the PGROUP table. A predicate like `PG.NAME = 'TV'` does not disqualify other predicates in a query from qualifying for automatic query rewrite. In this case PGROUP is a relatively small dimension table, so a predicate that refers to the table is not overly costly.

- DB2 can derive the query result from the materialized query table definition, even when the derivation is not readily apparent:
  - DB2 derives T.ID in the query from T.TRANSID in the TRANSIAB fullselect. Although these two columns originate from different tables, they are equivalent because of the predicate T.TRANSID = T.ID. DB2 recognizes such column equivalency through join predicates. Thus, DB2 derives T.ID from T.TRANSID, and the query qualifies for automatic query rewrite.
  - DB2 derives AMT in the query UserQ2 from DISCOUNT and VALUE in the TRANSIAB fullselect.

#### GUPI

### Example 3

#### GUPI

This example shows how DB2 matches GROUP BY items and aggregate functions between the user query and the materialized query table fullselect. Assume that an analyst submits the following query to find the average value of the transaction items for each year:

UserQ3

```
-----
SELECT YEAR, AVG(QUANTITY * PRICE) AS AVGVAL
FROM TRANSITEM TI, TRANS T
WHERE TI.TRANSID = T.ID
GROUP BY YEAR;
```

If you define the following materialized query table TRANSAVG, DB2 can rewrite UserQ3 as NewQ3:

TRANSAVG

```
-----
CREATE TABLE TRANSAVG AS
  (SELECT T.YEAR, T.MONTH, SUM(QUANTITY * PRICE) AS TOTVAL, COUNT(*) AS CNT
   FROM TRANSITEM TI, TRANS T
   WHERE TI.TRANSID = T.ID
   GROUP BY T.YEAR, T.MONTH )
DATA INITIALLY DEFERRED
REFRESH DEFERRED;
```

NewQ3

```
-----
SELECT YEAR, CASE WHEN SUM(CNT) = 0 THEN NULL
                  ELSE SUM(TOTVAL)/SUM(CNT)
                  END AS AVGVAL
FROM TRANSAVG
GROUP BY YEAR;
```

DB2 can rewrite UserQ3 as a new query that uses materialized query table TRANSAVG because of the following reasons:

- DB2 considers YEAR in the user query and YEAR in the materialized query table fullselect to match exactly.
- DB2 can derive the AVG function in the user query from the SUM function and the COUNT function in the materialized query table fullselect.
- The GROUP BY clause in the query NewQ3 requests data at a higher level than the level in the definition of TRANSAVG.

- DB2 can compute the yearly average in the user query by using the monthly sums and counts of transaction items in TRANSAVG. DB2 derives the yearly averages from the CNT and TOTVAL columns of the materialized query table by using a case expression.

#### GUPI

### Determining whether query rewrite occurred

You can use EXPLAIN to determine whether DB2 has rewritten a user query to use a materialized query table.

#### PSPI

When DB2 rewrites the query, the PLAN TABLE shows the name of the materialized query that DB2 uses. The value of the TABLE\_TYPE column is M to indicate that the table is a materialized query table.

### Example

Consider the following user query:

```
SELECT YEAR, AVG(QUANTITY * PRICE) AS AVGVAL
FROM TRANSITEM TI, TRANS T
WHERE TI.TRANSID = T.ID
GROUP BY YEAR;
```

If DB2 rewrites the query to use a materialized query table, a portion of the plan table output might look like the following table.

Table 24. Plan table output for an example with a materialized query table

PLANNO	METHOD	TNAME	JOIN_TYPE	TABLE_TYPE
1	0	TRANSAVG	-	M
2	3	2	-	?

The value M in TABLE\_TYPE indicates that DB2 used a materialized query table. TNAME shows that DB2 used the materialized query table named TRANSAVG. You can also obtain this information from a performance trace (IFCID

0022).

#### PSPI

### Enabling automatic query rewrite

Whether DB2 can consider automatic query rewrite depends on properly defined materialized query tables, and the values of two special registers.

#### Prerequisites:

- The isolation levels of the materialized query tables must be equal to or higher than the isolation level of the dynamic query being considered for automatic query rewrite
- You must populate system-maintained materialized query tables before DB2 considers them in automatic query rewrite.

#### GUPI

The values of two special registers, CURRENT REFRESH AGE and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION determine whether DB2 can consider using materialized query tables in automatic query rewrite.

To enable automatic query rewrite:

- Specify ANY for the CURRENT REFRESH AGE special register.  
The value in special register CURRENT REFRESH AGE represents a refresh age. The *refresh age* of a materialized query table is the time since the REFRESH TABLE statement last refreshed the table. (When you run the REFRESH TABLE statement, you update the timestamp in the REFRESH\_TIME column in catalog table SYSVIEWS.) The special register CURRENT REFRESH AGE specifies the maximum refresh age that a materialized query table can have. Specifying the maximum age ensures that automatic query rewrite does not use materialized query tables with old data. The CURRENT REFRESH AGE has only two values: 0 or ANY. A value of 0 means that DB2 considers no materialized query tables in automatic query rewrite. A value of ANY means that DB2 considers all materialized query tables in automatic query rewrite.

The CURRENT REFRESH AGE field on installation panel DSNTIP4 determines the initial value of the CURRENT REFRESH AGE special. The default value for the CURRENT REFRESH AGE field is 0.

- Specify the appropriate value for the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register.

The refresh age of a user-maintained materialized query table might not truly represent the freshness of the data in the table. In addition to the REFRESH TABLE statement, user-maintained query tables can be updated with the INSERT, UPDATE, MERGE, TRUNCATE, and DELETE statements and the LOAD utility. Therefore, you can use the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register to determine which type of materialized query tables, system-maintained or user-maintained, DB2 considers in automatic query rewrite. The special register has four possible values that indicate which materialized query tables DB2 considers for automatic query rewrite:

**SYSTEM**

DB2 considers only system-maintained materialized query tables.

**USER** DB2 considers only user-maintained materialized query tables.

**ALL** DB2 considers both types of materialized query tables.

**NONE**

DB2 considers no materialized query tables.

The CURRENT MAINT TYPES field on installation panel DSNTIP4 determines the initial value of the CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special resgister. the the default value for CURRENT MAINT TYPES is SYSTEM.

The following table summarizes how to use the CURRENT REFRESH AGE and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special registers together. The table shows which materialized query tables DB2 considers in automatic query rewrite.

Table 25. The relationship between CURRENT REFRESH AGE and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special registers

Value of CURRENT REFRESH AGE	Value of CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION			
	SYSTEM	USER	ALL	None
ANY	All system-maintained materialized query tables	All user-maintained materialized query tables	All materialized query tables (both system-maintained and user-maintained)	None
0	None	None	None	None

## Creating a materialized query table

You can create a materialized query table, which is defined by the result of a query, to improve the performance of certain SQL applications.

### GUIP

You should create materialized query tables in a table space that is defined as NOT LOGGED to avoid the performance overhead created by logging changes to the data.

To create a new table as a materialized query table:

1. Write a CREATE TABLE statement, and specify a fullselect. You can explicitly specify the column names of the materialized query table or allow DB2 to derive the column names from the fullselect. The column definitions of a materialized query table are the same as those for a declared global temporary table that is defined with the same fullselect.
2. Include the DATA INITIALLY DEFERRED and REFRESH DEFERRED clauses when you define a materialized query table.

#### DATA INITIALLY DEFERRED clause

DB2 does not populate the materialized query table when you create the table. You must explicitly populate the materialized query table.

- For system-maintained materialized query tables, populate the tables for the first time by using the REFRESH TABLE statement.
- For user-maintained materialized query tables, populate the table by using the LOAD utility, INSERT statement, or REFRESH TABLE statement.

#### REFRESH DEFERRED clause

DB2 does not immediately update the data in the materialized query table when its base tables are updated. You can use the REFRESH TABLE statement at any time to update materialized query tables and maintain data currency with underlying base tables.

3. Specify who maintains the materialized query table:

#### MAINTAINED BY SYSTEM clause

Specifies that the materialized query table is a system-maintained materialized query table. You cannot update a system-maintained materialized query table by using the LOAD utility or the INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements. You can update a system-maintained materialized query table only by using the REFRESH TABLE statement. BY SYSTEM is the default behavior if you do not specify a MAINTAINED BY clause.

Create system-maintained materialized query tables in segmented or universal table spaces because the REFRESH TABLE statement triggers a mass delete operation.

#### MAINTAINED BY USER clause

Specifies that the table is a user-maintained materialized query table. You can update a user-maintained materialized query table by using

the LOAD utility, the INSERT, UPDATE, MERGE, TRUNCATE, and DELETE statements, as well as the REFRESH TABLE statement.

4. Specify whether query optimization is enabled.

**ENABLE QUERY OPTIMIZATION clause**

Specifies that DB2 can consider the materialized query table in automatic query rewrite. When you enable query optimization, DB2 is more restrictive of what you can select in the fullselect for a materialized query table.

**DISABLE QUERY OPTIMIZATION clause**

Specifies that DB2 cannot consider the materialized query table in automatic query rewrite.

**Recommendation:** When creating a user-maintained materialized query table, initially disable query optimization. Otherwise, DB2 might automatically rewrite queries to use the empty materialized query table. After you populate the user-maintained materialized query table, you can alter the table to enable query optimization.

The isolation level of the materialized table is the isolation level at which the CREATE TABLE statement is executed.

After you create a materialized query table, it looks and behaves like other tables in the database system, with a few exceptions. DB2 allows materialized query tables in database operations wherever it allows other tables, with a few restrictions. As with any other table, you can create indexes on the materialized query table; however, the indexes that you create must not be unique. Instead, DB2 uses the materialized query table's definition to determine if it can treat the index as a unique index for query optimization.

The following CREATE TABLE statement defines a materialized query table named TRANSCNT. The TRANSCNT table summarizes the number of transactions in table TRANS by account, location, and year:

```
CREATE TABLE TRANSCNT (ACCTID, LOCID, YEAR, CNT) AS
  (SELECT ACCOUNTID, LOCATIONID, YEAR, COUNT(*)
   FROM TRANS
   GROUP BY ACCOUNTID, LOCATIONID, YEAR )
DATA INITIALLY DEFERRED
REFRESH DEFERRED
MAINTAINED BY SYSTEM
ENABLE QUERY OPTIMIZATION;
```

The fullselect, together with the DATA INITIALLY DEFERRED clause and the REFRESH DEFERRED clause, defines the table as a materialized query table.


**GUIP**

## Rules for materialized query table

If one or more source tables in the materialized query table definition contain a security label column, certain rules apply to creating a materialized query table.

**GUIP**

- If only one source table contains a security label column, the following conditions apply:
  - You must define the security label column in the materialized query table definition with the AS SECURITY LABEL clause.

- The materialized query table inherits the security label column from the source table.
- The MAINTAINED BY USER option is allowed.
- If only one source table contains a security label column and the materialized query table is defined with the DEFINITION ONLY clause, the materialized query table inherits the values in the security label column from the source table. However, the inherited column is not a security label column.
- If more than one source table contains a security label column, DB2 returns an error code and the materialized query table is not created. 

## Registering an existing table as a materialized query table

You might already have manually created base tables that act like materialized query tables and have queries that directly access the base tables. These base tables are often referred to as *summary tables*.

To ensure the accuracy of data that is used in automatic query rewrite, ensure that the summary table is current before registering it as a materialized query table.

Alternatively, you can follow these steps:

1. Register the summary table as a materialized query table with automatic query rewrite disabled.
2. Update the newly registered materialized query table to refresh the data.
3. Use the ALTER TABLE statement on the materialized query table to enable automatic query rewrite.

### GUIP

To take advantage of automatic query rewrite for an existing summary table:

Use the ALTER TABLE statement with DATA INITIALLY DEFERRED and MAINTAINED BY USER clauses, to register the table as materialized query table. The fullselect must specify the same number of columns as the table you register as a materialized query table. The columns must have the same definitions and have the same column names in the same ordinal positions.

The DATA INITIALLY DEFERRED clause indicates that the table data is to remain the same when the ALTER statement completes. The MAINTAINED BY USER clause indicates that the table is user-maintained.

The table becomes immediately eligible for use in automatic query rewrite. The isolation level of the materialized query table is the isolation level at which the ALTER TABLE statement is executed.

You can continue to update the data in the table by using the LOAD utility or the INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements. You can also use the REFRESH TABLE statement to update the data in the table.

Assume that you have an existing summary table named TRANSCOUNT. The TRANSCOUNT tabl has four columns to track the number of transactions by account, location, and year. Assume that TRANSCOUNT was created with this CREATE TABLE statement:

```
CREATE TABLE TRANCOUNT
  (ACCTID    INTEGER NOT NULL
   LOCID     INTEGER NOT NULL
   YEAR      INTEGER NOT NULL
   CNT       INTEGER NOT NULL);
```

The following SELECT statement then populated TRANCOUNT with data that was derived from aggregating values in the TRANS table:

```
SELECT ACCTID, LOCID, YEAR, COUNT(*)
FROM TRANS
GROUP BY ACCTID, LOCID, YEAR ;
```

You could use the following ALTER TABLE statement to register TRANCOUNT as a materialized query table. The statement specifies the ADD MATERIALIZED QUERY clause:

```
ALTER TABLE TRANCOUNT ADD MATERIALIZED QUERY
  (SELECT ACCTID, LOCID, YEAR, COUNT(*) as cnt
   FROM TRANS
   GROUP BY ACCTID, LOCID, YEAR )
  DATA INITIALLY DEFERRED
  REFRESH DEFERRED
  MAINTAINED BY USER;
```

#### GUIP

### Altering an existing materialized query table

You can use the ALTER TABLE statement to change the attributes of a materialized query table or change a materialized query table to a base table.

#### GUIP :

In addition to using the ALTER TABLE statement, you can change a materialized query table by dropping the table and recreating the materialized query table with a different definition. **GUIP**

- Change the attributes of a materialized query table.

You can enable or disable automatic query rewrite for a materialized query table with the ENABLE QUERY OPTIMIZATION or DISABLE QUERY OPTIMIZATION clause. You can change the type of materialized query table between system-maintained and user-maintained by using the MAINTAINED BY SYSTEM or MAINTAINED BY USER clause.

Altering a materialized query table to enable it for query optimization makes the table immediately eligible for use in automatic query rewrite. You must ensure that the data in the materialized query table is current. Otherwise, automatic query rewrite might return results that are not current.

- Change a materialized query table into a base table. For example Assume that you no longer want TRANCOUNT to be a materialized query table. The following ALTER TABLE statement, which specifies the DROP MATERIALIZED QUERY clause, changes the materialized query table into a base table.

```
ALTER TABLE TRANCOUNT DROP MATERIALIZED QUERY;
```

The column definitions and the data in the table do not change. However, DB2 can no longer use the table in automatic query rewrite. You can no longer update the table with the REFRESH TABLE statement. One reason you might want to change a materialized query table into a base table is to perform table operations that are restricted for a materialized query table.


You might want to rotate the partitions on your partitioned materialized query table. In order to rotate the partitions, you must change your materialized query table into a base table. While the table is a base table, you can rotate the partitions. After you rotate the partitions, you can change the table back to a materialized query table.

## Populating and maintaining materialized query tables

After you define a materialized query table, you need to maintain the accuracy of the data in the table.

This maintenance includes populating the table for the first time and periodically refreshing the data in the table. You need to refresh the data because any changes that are made to the base tables are not automatically reflected in the materialized query table.

The only way to change the data in a system-maintained materialized query table is through the REFRESH TABLE statement. The INSERT, UPDATE, MERGE, TRUNCATE and DELETE statements, and the LOAD utility cannot refer to a system-maintained materialized query table as a target table. Therefore, a system-maintained materialized query table is read-only.

Any view or cursor that is defined on a system-maintained materialized query table is read-only. However, for a user-maintained materialized query table, you can alter the data with the INSERT, UPDATE, and DELETE statements, and the LOAD utility. 

### Populating a new materialized query table

You can populate a newly created table for the first time by using the REFRESH TABLE statement or by using INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements, or the LOAD utility.



When you create a materialized query table with the CREATE TABLE statement, the table is not immediately populated.

- Issue a REFRESH TABLE statement. For example, the following REFRESH TABLE statement populates a materialized query table named SALESCNT:

```
REFRESH TABLE SALESCNT;
```

You should avoid using the REFRESH TABLE statement to update user-maintained materialized query tables because the REFRESH TABLE statement uses a fullselect and can result in a long-running query.

The REFRESH TABLE statement is an explainable statement. The explain output contains rows for INSERT with the fullselect in the materialized query table definition.

- Use the INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements, or the LOAD utility.

You cannot use the INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements, or the LOAD utility to change system-maintained materialized query tables.

### Refreshing a system-maintained materialized query table

You can use the REFRESH TABLE statement to refresh the data in any materialized query table at any time.

To refresh an existing materialized query table:

Issue a REFRESH TABLE statement. When you issue the REFRESH TABLE statement DB2 performs the following actions:

1. Deletes all the rows in the materialized query table
2. Executes the fullselect in the materialized query table definition to recalculate the data from the tables that are specified in the fullselect with the isolation level for the materialized query table
3. Inserts the calculated result into the materialized query table
4. Updates the DB2 catalog with a refresh timestamp and the cardinality of the materialized query table

Although the REFRESH TABLE statement involves both deleting and inserting data, DB2 completes these operations in a single commit scope. Therefore, if a failure occurs during execution of the REFRESH TABLE statement, DB2 rolls back all changes that the statement made.

### Refreshing user-maintained materialized query tables

You can update the data in user-maintained materialized query tables by using the INSERT, UPDATE, MERGE, TRUNCATE, and DELETE statements, and the LOAD utility.

#### PSPI

You should avoid using the REFRESH TABLE statement to update user-maintained materialized query tables. Because the REFRESH TABLE statement uses a fullselect to refresh a materialized query table, the statement can result in a long-running query. Use insert, update, delete, or load operations might be more efficient than using the REFRESH TABLE statement.

Depending on the size and frequency of changes in base tables, you might use different strategies to refresh your materialized query tables. For example, for infrequent, minor changes to dimension tables, you could immediately propagate the changes to the materialized query tables by using triggers. For larger or more frequent changes, you might consider refreshing your user-maintained materialized query tables incrementally to improve performance.

To avoid refresh a user-maintained materialized query table:

Use INSERT, UPDATE, MERGE, TRUNCATE, or DELETE statements, or the LOAD utility. For example, you might find it faster to generate the data for your materialized query table and execute the LOAD utility to populate the data.

For example, assume that you need to add a large amount of data to a fact table. Then, you need to refresh your materialized query table to reflect the new data in the fact table. To do this, perform these steps:

- Collect and stage the new data in a separate table.
- Evaluate the new data and apply it to the materialized table as necessary.
- Merge the new data into the fact table

For an example of such code, see member DSNTEJ3M in DSN910.SDSNSAMP, which is shipped with DB2. 

## Updating statistics on materialized query tables

For optimal performance of materialized query tables, you need to provide DB2 with accurate catalog statistics for access path selection.

**PSPI** When you run the REFRESH TABLE statement, the only statistic that DB2 updates for the materialized query table is the cardinality statistic.

To keep catalog statistics current for materialized query tables:

Run the RUNSTATS utility after executing a REFRESH TABLE statement or after changing the materialized query table significantly. Otherwise, DB2 uses default or out-of-date statistics. The estimated performance of queries that are generated by automatic rewrite might inaccurately compare less favorably to the original query.

**PSPI**

## Rules for using materialized query tables in a multilevel security environment

If source tables have multilevel security with row-level granularity enabled, some additional rules apply to working with the materialized query table and the source tables.

**GUPI** Tables with multilevel security enabled contain a security label column, which is defined with the AS SECURITY LABEL clause. The values in the security label column indicate which users can access the data in each row.

### Creating a materialized query tablet

If one or more source tables in the materialized query table definition contain a security label column, certain apply to creating a materialized query table.

#### Only one source table contains a security label column

The following conditions apply:

- You must define the security label column in the materialized query table definition with the AS SECURITY LABEL clause.
- The materialized query table inherits the security label column from the source table.
- The MAINTAINED BY USER option is allowed.

#### Only one source table contains a security label column, and a DEFINITION ONLY clause was used

The materialized query table inherits the values in the security label column from the source table. However, the inherited column is not a security label column.

#### More than one source table contains a security label column

DB2 returns an error code, and the materialized query table is not created.

### Altering a source table

An ALTER TABLE statement to add a security label column to a table fails if the table is a source table for a materialized query table.

## Refreshing a materialized query table


The REFRESH TABLE statement deletes the data in a materialized query table and then repopulates the materialized query table according to its table definition. During this refresh process, DB2 does not check for multilevel security with

row-level granularity. 

## Enabling a materialized query table for automatic query rewrite

After you populate a user-maintained materialized query table, you can alter the table to enable query optimization.

**Prerequisite:** If the materialized query table is user-maintained, it is populated with data.

 When creating a user-maintained materialized query table, initially disable query optimization. Otherwise, DB2 might automatically rewrite queries to use the empty materialized query table.

To enable a materialized query table for automatic query rewrite:

Issue an ALTER TABLE statement and specify:

- The ENABLE QUERY OPTIMIZATION clause
- An isolation level equivalent or higher than that of the dynamic queries that might use the materialized query table. The isolation level of the table must be equal to or higher than the isolation level of the dynamic query being considered for automatic query rewrite.



## Recommendations for materialized query table and base table design

By following certain best practices, you might improve the performance of your materialized query tables and the queries that use them. These recommendations, however, do not represent a complete design theory.

### Designing materialized query tables for automatic query rewrite

By following these recommendations, you might improve the performance of queries that use materialized query tables.



To get better performance from your materialized query tables:

- Include aggregate functions strategically in the fullselect of a materialized query table definition:
  - Include COUNT(\*) and SUM(expression).
  - Include SUM(expression\*expression) only if you plan to query VAR(expression), STDDEV(expression), VAR\_SAMP(expression), or STDDEV\_SAMP(expression).
  - Include COUNT(expression) in addition to COUNT(\*) if expression is nullable.
  - Include MIN(expression) and MAX(expression) if you plan to query them.

- Do not include *AVG(expression)*, *VAR(expression)*, or *STDDEV(expression)* directly if you include either of the following parameter combinations:
  - *SUM(expression)*, *SUM(expression\*expression)*, and *COUNT(\*)*
  - *SUM(expression)*, *SUM(expression\*expression)*, and *COUNT(expression)*

DB2 can derive *AVG(expression)*, *VAR(expression)*, and *STDDEV(expression)* from *SUM(expression)*, *SUM(expression\*expression)*, and the appropriate *COUNT* aggregate function.

- Include the foreign key of a dimension table in the *GROUP BY* clause of a materialized query table definition. For example, if you include *PGROUP.ID*, also include *PGROUP.LINEID*. Then DB2 can use the materialized query table to derive a summary at the *LINEID* level, without rejoining *PGROUP*.
- Include all the higher-level columns in the materialized query table if DB2 does not know the functional dependency in a denormalized dimension table. For example, if you include *CITY* in a *GROUP BY* clause, also include *STATE* and *COUNTRY* in the clause. Similarly, if you include *MONTH* in the *GROUP BY* clause, also include *YEAR* in the clause.
- Do not use the *HAVING* clause in your materialized query tables. A materialized query table with a *HAVING* clause in its definition has limited usability during query rewrite.
- Create indexes on materialized query tables as you would for base tables.

#### GUPI

### Designing base tables for automatic query rewrite

These recommendations describe base table design strategies that might increase the performance and eligibility of your materialized query tables.

#### GUPI

To make your base tables work well with materialized query tables:

- Define referential integrity as either *ENFORCED* or *NOT ENFORCED* whenever possible.
- Define an index as unique if it is truly unique.
- Define all base table columns as *NOT NULL* if possible, so that *COUNT(x)* is the same as *COUNT(\*)*. Then you do not need to include *COUNT(x)* for each nullable column *x* in a materialized query table definition. If necessary, use a special value to replace *NULL*.
- Emphasize normalized dimension design over denormalized dimension design in your base tables. When you use normalized dimension design, you do not need to include non-primary key columns in a materialized query table, thereby saving you storage space. DB2 compensates for the lack of non-primary key columns by deriving these columns through a re-join of the dimension table. If normalization causes performance problems, you can define materialized query tables to denormalize the snowflake dimensions.

## Materialized query tables—examples shipped with DB2

In addition to the examples shown in this information, DB2 provides a number of samples to help you design materialized query tables for automatic query rewrite.


#### PSPI

The samples are based on a data warehouse with a star schema database. The star schema contains one fact table, *SALESFACT*, and these four hierarchical dimensions:

- A *TIME* dimension that consists of one dimension table

- A PRODUCT dimension that is a snowflake that consists of four fully normalized tables
- A LOCATION dimension that is a snowflake that consists of five partially normalized tables
- A CUSTOMER dimension that is a snowflake that consists of four fully normalized tables

See member DSNTEJ3M in data set DSN810.SDSNSAMP for all of the code, including the following items:

- SQL statements to create and populate the star schema
- SQL statements to create and populate the materialized query tables
- Queries that DB2 rewrites to use the materialized query table 



---

## Chapter 12. Managing DB2 threads

Threads are an important DB2 resource. A thread is a DB2 structure that describes a connection made by an application and traces its progress.

When you install DB2, you choose a maximum number of active allied and database access threads that can be allocated concurrently. Choosing a good number for maximum threads is important to keep applications from queuing and to provide good response time.

When writing an application, you should know when threads are created and terminated and when they can be reused, because thread allocation can be a significant part of the cost in a short transaction.

This information provides a general introduction to how DB2 uses threads, including the following information:

- A discussion of how to choose the maximum number of concurrent threads.
- A description of the steps in creating and terminating an allied thread, in.
- An explanation of the differences between allied threads and database access threads (DBATs) and a description of how DBATs are created, including how they become active or pooled and how to set performance goals for individual DDF threads.
- Design options for reducing thread allocations and improving performance generally.

---

### Setting thread limits

You can limit the maximum number of DB2 threads that can be allocated concurrently.

Set these values to provide good response time without wasting resources, such as virtual and real storage. The value you specify depends upon your machine size, your work load, and other factors. When specifying values for these fields, consider the following:

- Fewer threads than needed under utilize the processor and cause queuing for threads.
- More threads than needed do not improve the response time. They require more real storage for the additional threads and might cause more paging and, hence, performance degradation.

To limit the number of allied and database access threads that can be allocated concurrently:

1. Use the MAX USERS and MAX REMOTE ACTIVE fields on installation panel DSNTIPE. The combined maximum allowed for MAX USERS and MAX REMOTE ACTIVE is 2000. If virtual storage or real storage is the limiting factor, set MAX USERS and MAX REMOTE ACTIVE according to the available storage.
2. For the TSO and call attachment facilities, you limit the number of threads indirectly by choosing values for the MAX TSO CONNECT and MAX BATCH CONNECT fields of installation panel DSNTIPE. These values limit the number

of connections to DB2. The number of threads and connections allowed affects the amount of work that DB2 can process.

---

## Allied thread allocation

This information describes at a high level the steps in allocating an allied thread, and some of the factors related to the performance of those steps. This information does not explain how a database access thread is allocated.

### Step 1: Thread creation

During thread creation with `ACQUIRE(ALLOCATE)`, the resources needed to execute the application are acquired. During thread creation with `ACQUIRE(USE)`, only the thread is created.

The following list shows the main steps in thread creation.

1. Check the maximum number of threads.  
DB2 checks whether the maximum number of active threads, specified as `MAX USERS` for local threads or `MAX REMOTE ACTIVE` for remote threads on the Storage Sizes panel (`DSNTIPE`) when DB2 was installed, has been exceeded. If it has been exceeded, the request waits. The wait for threads is not traced, but the number of requests queued is provided in the performance trace record with IFCID 0073.
2. Check the plan authorization.  
The authorization ID for an application plan is checked in the `SYSPLANAUTH` catalog table (IFCID 0015). If this check fails, the table `SYSUSERAUTH` is checked for the `SYSADM` special privilege.
3. For an application plan, load the control structures associated with the plan.  
The control block for an application plan is divided into sections. The header and directory contain control information; SQL sections contain SQL statements from the application. A copy of the plan's control structure is made for each thread executing the plan. Only the header and directory are loaded when the thread is created.
4. Load the descriptors necessary to process the plan.  
Some of the control structures describe the DB2 table spaces, tables, and indexes used by the application. If `ACQUIRE(ALLOCATE)` is used, all the descriptors referred to in the plan are loaded now. If the plan is bound with `ACQUIRE(USE)`, they are loaded when SQL statements are executed.

The most relevant factors from a system performance point of view are:

#### Thread reuse

Thread creation is a significant cost for small and medium transactions. When execution of a transaction is terminated, the thread can sometimes be reused by another transaction using the same plan.

#### ACQUIRE option of BIND

`ACQUIRE(ALLOCATE)` causes all the resources referred to in the application to be allocated when the thread is created. `ACQUIRE(USE)` allocates the resources only when an SQL statement is about to be executed. In general, `ACQUIRE(USE)` is recommended. However, if most of the SQL is used in every execution of the transaction, `ACQUIRE(ALLOCATE)` is recommended.

#### EDM pool size

The size of the EDM pool influences the number of I/Os needed to load

the control structures necessary to process the plan or package. To avoid a large number of allocation I/Os, the EDM pool must be large enough to contain the structures that are needed.

## Step 2: Resource allocation

Some of the structures necessary to process the statement are stored in 4 KB pages. If they are not already present, those are read into database buffer pool BP0 and copied from there into the EDM pool.

If the plan was bound with ACQUIRE(USE), it acquires resources when the statement is about to execute.

1. Load the control structures necessary to process the SQL section.  
If it is not already in the EDM pool, DB2 loads the control structure's section corresponding to this SQL statement.
2. Load structures necessary to process statement.  
Load any structures referred to by this SQL statement that are not already in the EDM pool.
3. Allocate and open data sets.  
When the control structure is loaded, DB2 locks the resources used.

The most important performance factors for resource allocation are the same as the factors for thread creation.

## Step 3: SQL statement execution

If the statement resides in a package, the directory and header of the package's control structure is loaded at the time of the first execution of a statement in the package.

The control structure for the package is allocated at statement execution time. This is contrasted with the control structures for plans bound with ACQUIRE(ALLOCATE), which are allocated at thread creation time. The header of the plan's control structures is allocated at thread creation time regardless of ACQUIRE(ALLOCATE) or ACQUIRE(USE).

When the package is allocated, DB2 checks authorization using the package authorization cache or the SYSPACKAUTH catalog table. DB2 checks to see that the plan owner has execute authority on the package. On the first execution, the information is not in the cache; therefore, the catalog is used. Thereafter, the cache is used..

For dynamic bind, authorization checking also occurs at statement execution time.

A summary record, produced at the end of the statement (IFCID 0058), contains information about each scan that is performed. Included in the record is the following information:

- The number of updated rows
- The number of processed rows
- The number of deleted rows
- The number of examined rows
- The number of pages that are requested through a getpage operation
- The number of rows that are evaluated during the first stage (stage 1) of processing

- The number of rows that are evaluated during the second stage (stage 2) of processing
- The number of getpage requests that are issued to enforce referential constraints
- The number of rows that are deleted or set null to enforce referential constraints
- The number of inserted rows

From a system performance perspective, the most important factor in the performance of SQL statement execution is the size of the database buffer pool. If the buffer pool is large enough, some index and data pages can remain there and can be accessed again without an additional I/O operation.

#### **Related tasks**

“Tuning DB2 buffer, EDM, RID, and sort pools” on page 56

## **Step 4: Commit and thread termination**

Commit processing can occur many times while a thread is active.

For example, an application program running under the control structure of the thread could issue an explicit COMMIT or SYNCPOINT several times during its execution. When the application program or the thread terminates, an implicit COMMIT or SYNCPOINT is issued.

When a COMMIT or SYNCPOINT is issued from an IMS application running with DB2, the two-phase commit process begins if DB2 resources have been changed since the last commit point. In a CICS or RRSAP application, the two-phase commit process begins only if DB2 resources have changed and a non-DB2 resource has changed within the same commit scope.

The significant events that show up in a performance trace of a commit and thread termination operation occur in the following sequence:

### **1. Commit phase 1**

In commit phase 1 (IFCID 0084), DB2 writes an end of phase 1 record to the log (IFCIDs 0032 and 0033). The trace shows two I/Os, one to each active log data set (IFCIDs 0038 and 0039).

### **2. Commit phase 2**

In commit phase 2 (IFCID 0070), DB2 writes a beginning of phase 2 record to the log. Again, the trace shows two I/Os. Page and row locks (except those protecting the current position of cursors declared with the WITH HOLD option), held to a commit point, are released. An unlock (IFCID 0021) with a requested token of zeros frees any lock for the specified duration. A summary lock record (IFCID 0020) is produced, which gives the maximum number of page locks held and the number of lock escalations. DB2 writes an end of phase 2 record to the log.

If RELEASE(COMMIT) is used, the following events also occur:

- Table space locks are released.
- All the storage used by the thread is freed, including storage for control blocks, CTs and PTs, and working areas.
- The use counts of the DBDs are decreased by one. If space is needed in the EDM DBD cache, a DBD can be freed when its use count reaches zero.
- Those table spaces and index spaces with no claimers are made candidates for deferred close.

### **3. Thread termination**

When the thread is terminated, the accounting record is written. It does not report transaction activity that takes place before the thread is created.

If RELEASE(DEALLOCATE) is used to release table space locks, the DBD use count is decreased, and the thread storage is released.

#### **Related concepts**

“Understanding the CLOSE YES and CLOSE NO options” on page 81

## **Variations on thread management**

Transaction flow can vary in different environments and when dynamic SQL statements are executed.

### **TSO and call attachment facility**

You can use the TSO attachment facility and call attachment facility (CAF) to request that SQL statements be executed in TSO foreground and batch. The processes differ from CICS or IMS transactions in that:

- No sign-on is required. The user is identified when the TSO address space is connected.
- Commit requires only a single-phase and only one I/O operation to each log. Single phase commit records are IFCID 0088 and 0089.
- Threads cannot be reused because the thread is allocated to the user address space.

### **Resource Recovery Services attachment facility (RRSAF)**

With RRSAF, you have sign-on capabilities, the ability to reuse threads, and the ability to coordinate commit processing across different resource managers.

### **SQL under DB2 QMF**

DB2 QMF uses CAF to create a thread when a request for work, such as a SELECT statement, is issued. A thread is maintained until the end of the session only if the requester and the server reside in different DB2 subsystems. If the requester and the server are both in the local DB2 subsystem, the thread is not maintained.

For more information about DB2 QMF™ connections, see *DB2 Query Management Facility: Installing and Managing DB2 QMF for TSO/CICS*.

## **Reusing threads**

In general, you want transactions to reuse threads when transaction volume is high and the cost of creating threads is significant, but thread reuse is also useful for a lower volume of priority transactions.

For a transaction of five to ten SQL statements (10 I/O operations), the cost of thread creation can be 10% of the processor cost. But the steps needed to reuse threads can incur costs of their own.

#### **Related concepts**

“Reusing threads for remote connections” on page 147

#### **Related tasks**

“Setting CICS options for threads” on page 150

“Setting IMS options for threads” on page 150

## Reusing threads through bind options

In DB2, you can prepare allied threads for reuse when you bind the plan.

To prepare allied threads for reuse:

Bind the plan with the ACQUIRE(USE) and RELEASE(DEALLOCATE) options; otherwise, the allocation cost is not eliminated but only slightly reduced. Be aware of the following effects:

- ACQUIRE(ALLOCATE) acquires all resources needed by the plan, including locks, when the thread is created; ACQUIRE(USE) acquires resources only when they are needed to execute a particular SQL statement. If most of the SQL statements in the plan are executed whenever the plan is executed, ACQUIRE(ALLOCATE) costs less. If only a few of the SQL statements are likely to be executed, ACQUIRE(USE) costs less and improves concurrency. But with thread reuse, if most of your SQL statements eventually get issued, ACQUIRE(USE) might not be as much of an improvement.
- RELEASE(DEALLOCATE) does not free cursor tables (SKCTs) at a commit point; hence, the cursor table could grow as large as the plan. If you are using created temporary tables, the logical work file space is not released until the thread is deallocated. Thus, many uses of the same created temporary table do not cause reallocation of the logical work files, but be careful about holding onto this resource for long periods of time if you do not plan to use it.

## Using reports to tell when threads were reused

The NORMAL TERM., ABNORMAL TERM., and IN DOUBT sections of the OMEGAMON accounting report can help you identify, by plan, when threads were reused.

As shown in the following figure:

- NEW USER ( **A** ) tells how many threads were not terminated at the end of the previous transaction or query, and hence reused.
- DEALLOCATION ( **B** ) tells how many threads were terminated at the end of the query or transaction.
- APPL. PROGR. END ( **C** ) groups all the other reasons for accounting. Since the agent did not abend, these are considered normal terminations.

This technique is accurate in IMS but not in CICS, where threads are reused frequently by the same user. For CICS, also consider looking at the number of commits and aborts per thread. For CICS:

- NEW USER ( **A** ) is thread reuse with a different authorization ID or transaction code.
- RESIGN-ON ( **D** ) is thread reuse with the same authorization ID.

NORMAL TERM.	TOTAL	ABNORMAL TERM.	TOTAL	IN DOUBT	TOTAL
-----	-----	-----	-----	-----	-----
NEW USER <b>A</b>	0	APPL.PROGR. ABEND	0	APPL.PGM. ABEND	0
DEALLOCATION <b>B</b>	0	END OF MEMORY	0	END OF MEMORY	0
APPL.PROGR. END <b>C</b>	0	RESOL.IN DOUBT	0	END OF TASK	0
RESIGNON <b>D</b>	0	CANCEL FORCE	0	CANCEL FORCE	0
DBAT INACTIVE	0				
IN2 INACTIVE	193				
RRS COMMIT	0				
END U. THRESH	0				
BLOCK STORAGE	0				
STALENESS	0				

Figure 10. OMEGAMON accounting report - information about thread termination

---

## Using distributed database access threads

*Database access threads* are created to access data at a DB2 server on behalf of a requester.

A database access thread is created when a new connection is accepted from a remote requester, or if DB2 is configured with INACTIVE MODE and a new request is received from a remote requester pooled database access thread is unavailable to service the new request. Allied threads perform work at a requesting DB2 subsystem.

Database access threads differ from allied threads in the following ways:

- Database access threads have two modes of processing: ACTIVE MODE and INACTIVE MODE. These modes are controlled by setting the DDF THREADS field on the installation panel DSNTIPR.

### ACTIVE MODE

When the value of DDF THREADS is ACTIVE, a database access thread is always active from initial creation to termination.

### INACTIVE MODE

When the value of DDF THREADS is INACTIVE, a database access thread can be active or pooled. When a database access thread in INACTIVE MODE is active, it is processing requests from client connections within units of work. When a database access thread is pooled, it is waiting for the next request from a client to start a new unit of work.

- Database access threads run in enclave SRB mode.
- Only when in INACTIVE MODE, a database access thread is terminated after it has processed 200 units of work or after the database access thread has been idle in the pool for the amount of time specified in the POOL THREAD TIMEOUT field on the installation panel DSNTIP5. This termination does not prevent another database access thread from being created to meet processing demand, as long as the value of the MAX REMOTE ACTIVE field on panel DSNTIPE has not been reached.

**Recommendation:** Use INACTIVE MODE threads instead of ACTIVE MODE threads if you can.

## Setting thread limits for database access threads

When you install DB2, you choose a maximum number of active threads that can be allocated concurrently.

The MAX USERS field on panel DSNTIPE represents the maximum number of allied threads, and the MAX REMOTE ACTIVE field on panel DSNTIPE represents the maximum number of database access threads. Together, the values you specify for these fields cannot exceed 1999.

In the MAX REMOTE CONNECTED field of panel DSNTIPE, you can specify up to 150 000 as the maximum number concurrent remote connections that can concurrently exist within DB2. This upper limit is only obtained if you specify the recommended value INACTIVE for the DDF THREADS field of installation panel

DSNTIPR. Figure 11 illustrates the relationship among the number of active threads in the system and the total number of connections.

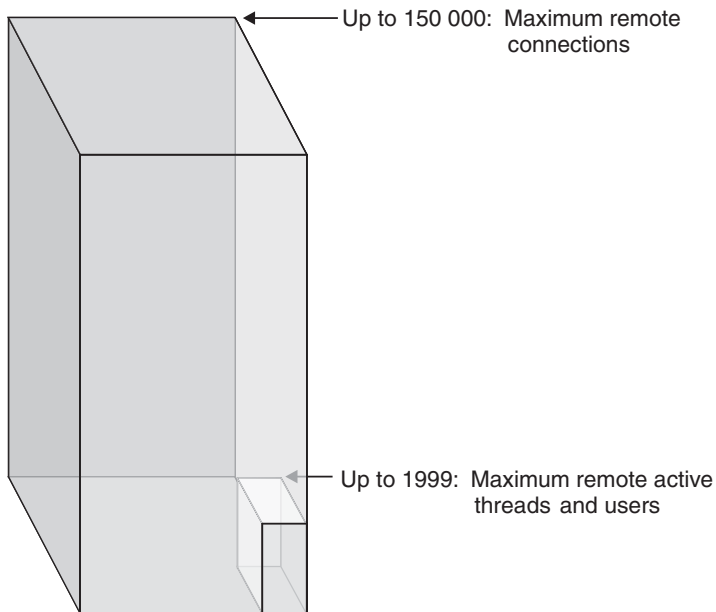


Figure 11. Relationship between active threads and maximum number of connections.

## Using threads in INACTIVE MODE for DRDA-only connections

When the value of DDF THREADS is INACTIVE, DRDA connections with a client cause a pooling behavior in database access threads.

A different database access behavior is used when private-protocol connections are involved. For information about private-protocol connections and database access threads, see “Using threads with private-protocol connections” on page 146.

A database access thread that is not currently processing a unit of work is called a pooled thread, and it is disconnected. DB2 always tries to pool database access threads, but in some cases cannot do so. The conditions listed in Table 26 determine if a thread can be pooled.

Table 26. Requirements for pooled threads

If there is...	Thread can be pooled?
A DRDA hop to another location	Yes
A package that is bound with RELEASE(COMMIT)	Yes
A package that is bound with RELEASE(DEALLOCATE)	Yes
A held cursor, a held LOB locator, or a package bound with KEEP_DYNAMIC(YES)	No
A declared temporary table that is active (the table was not explicitly dropped through the DROP TABLE statement or the ON COMMIT DROP TABLE clause on the DECLARE GLOBAL TEMPORARY TABLE statement)	No

When the conditions listed in Table 26 are true, the thread can be pooled when a COMMIT is issued. After a ROLLBACK, a thread can be pooled even if it had open cursors defined WITH HOLD or a held LOB locator because ROLLBACK

closes all cursors and LOB locators. ROLLBACK is also the only way to use the KEEP\_DYNAMIC(YES) bind option to clear information about a dynamic SQL statement.

## Understanding the advantages of database access threads in INACTIVE MODE

You can allow threads to be pooled to improve performance.

Allowing threads to be pooled has the following advantages:

- You can leave an application that is running on a workstation connected to DB2 from the time the application is first activated until the workstation is shut down and thus avoid the delay of repeated connections.
- DB2 can support a larger number of DDF connections (150,000 maximum; not limited by the maximum number of threads).
- Less storage is used for each DDF connection. (A connection uses significantly less storage than a database access thread.)
- You get an accounting trace record each time that a thread is pooled rather than once for the entire time that you are connected. When a pooled thread becomes active, the accounting fields for that thread are re-initialized. As a result, the accounting record contains information about active threads only, which makes it easier to study how distributed applications are performing. If a pooled mode thread remains active because of sections that are specified with the KEEP\_DYNAMIC(YES) bind option, an accounting record is still written.

**Exception:** If you employ account record accumulation, an accounting trace is not produced each time that a thread becomes pooled. Accounting records can be rolled up by concatenated combinations of the following values:

- End user ID
- End transaction name
- End user workstation name
- Each time that a thread is pooled, workload manager resets the information that it maintains on that thread. The next time that thread is activated, workload manager begins managing to the goals that you have set for the transactions that run in that service class. If you use multiple performance periods, it is possible to favor short-running units of work that use fewer resources while giving fewer resources over time to long running units of work. See “Establishing performance periods for DDF threads” on page 149 for more information.
- You can use response time goals, which is not recommended when using ACTIVE MODE threads.
- INACTIVE MODE threads can better take advantage of the ability to time out idle active threads, as described in “Timing out idle active threads.”
- Thread reuse lets DDF use a small pool of database access threads to support a large group of network clients.
- The response times reported by RMF include periods between requests and within the same unit of work. These times are shown as idle delays.

## Enabling threads to be pooled

You must specify INACTIVE on the DDF THREADS field of installation panel DSNTIPR to allow threads to be pooled.

## Timing out idle active threads

Active server threads that have remained idle for a specified period of time (in seconds) can be canceled by DB2.

When you install DB2, you choose a maximum IDLE THREAD TIMEOUT period, from 0 to 9999 seconds. The timeout period is an approximation. If a server thread has been waiting for a request from the requesting site for this period of time, it is canceled unless the thread is currently pooled or in doubt thread. A value of 0, the default, means that the server threads cannot be canceled because of an idle thread timeout.

**Recommendation:** Use the default option with the option ACTIVE for the DDF THREADS field on DSNTIPR. If you specify a timeout interval with ACTIVE, an application would have to start its next unit of work within the timeout period specification, or risk being canceled.

**TCP/IP keep\_alive interval for the DB2 subsystem:** For TCP/IP connections, it is a good idea to specify the IDLE THREAD TIMEOUT value in conjunction with a TCP/IP keep\_alive interval of 5 minutes or less to make sure that resources are not locked for a long time when a network outage occurs. It is recommended that you override the TCP/IP stack keep\_alive interval on a single DB2 subsystem by specifying a numeric value in the field TCP/IP KEEPALIVE on installation panel DSNTIPS.

---

## Using threads with private-protocol connections

Database access threads with private-protocol connections behave differently than threads with DRDA-only connections.

When a database access thread is associated with a private-protocol connection, whether inbound, outbound, or both, the database access thread remains with the connection or connections. During periods of inactivity, DB2 reduces the memory footprint of threads until the next unit of work begins.

The MAX INACTIVE DBATS setting determines whether DB2 reduces the memory footprint of a private-protocol connection that involves a database access thread. When a private-protocol connection ends a unit of work, DB2 first compares the number of current inactive database access threads to the value that is specified for your installation for MAX INACTIVE DBATS. Based on these values, DB2 either makes the thread inactive or allows it to remain active:

- If the current number of inactive database access threads is below the value in MAX INACTIVE DBATS, the thread becomes inactive. It cannot be used by another connection.
- If the current number of inactive database access threads meets or exceeds the value in MAX INACTIVE DBATS, the thread remains active. However, too many active threads (that is, more than MAX REMOTE ACTIVE) can cause the thread and its connection to be terminated.

To limit the number of inactive database access threads that can be created:

Specify a value in the MAX INACTIVE DBATS field of installation panel DSNTIPR. The default is 0, which means that any database access thread that involves a private-protocol-connection remains active.

While using a private-protocol connection, DB2 always tries to make threads inactive, but in some cases cannot do so. If MAX INACTIVE DBATS is greater than 0 and the value has not been exceeded, the conditions listed in the following table determine if a thread can be inactive.

Table 27. Requirements for inactive threads

If there is...	Thread can be inactive?
A hop to another location	Yes
A package that is bound with RELEASE(COMMIT)	Yes
A package that is bound with RELEASE(DEALLOCATE)	No
A held cursor, a held LOB locator, or a package bound with KEEP DYNAMIC(YES)	No
A declared temporary table that is active (the table was not explicitly dropped through the DROP TABLE statement)	No

## Reusing threads for remote connections

The cost to create a thread can be significant and reusing threads is a way to avoid that cost. DB2 for z/OS can reuse threads at the requester and at the server.

At the requester, a thread can be reused for an application that uses the CICS, IMS, or RRS attachment facility. As a server, DB2 can assign that connection to a pooled database access thread. Those threads can be shared and reused by thousands of client connections, which lets DB2 support very large client networks at minimal cost. (Inactive threads are only eligible to be reused by the same connection.)

If your server is not DB2 for z/OS, or some other server that can reuse threads, then reusing threads for your requesting CICS, IMS, or RRS applications is not a benefit for distributed access. Thread reuse occurs when sign-on occurs with a new authorization ID. If that request is bound for a server that does not support thread reuse, that change in the sign-on ID causes the connection between the requester and server to be released so that it can be rebuilt again for the new ID.

## Using z/OS workload management to set performance objectives

You can use z/OS workload management (WLM) support, to establish z/OS performance objectives for individual DDF server threads.

z/OS supports enclave system request blocks (SRBs). A z/OS enclave lets each thread have its own performance objective. For details on using workload management, see *z/OS MVS Planning: Workload Management*.

The z/OS performance objective of the DDF address space does not govern the performance objective of the user thread. As described in Chapter 6, “z/OS performance options for DB2,” on page 49, you should assign the DDF address space to a z/OS performance objective that is similar to the DB2 database services address space (*ssnmDBM1*). The z/OS performance objective of the DDF address space determines how quickly DB2 is able to perform operations associated with managing the distributed DB2 work load, such as adding new users or removing users that have terminated their connections. This performance objective should be a service class with a single velocity goal. This performance objective is assigned by modifying the WLM Classification Rules for started tasks (STC).

## Classifying DDF threads

You can classify DDF threads by, among other things, authorization ID and stored procedure name. The stored procedure name is only used as a classification if the first statement issued by the client after the CONNECT is an SQL CALL statement.

Use the WLM administrative application to define the service classes you want z/OS to manage. These service classes are associated with performance objectives. When a WLM-established stored procedure call originates locally, it inherits the performance objective of the caller, such as TSO or CICS.

**Important:** If you do not classify your DDF transactions into service classes, they are assigned to the default class, the *discretionary* class, which is at a very low priority.

### Classification attributes

Each of the WLM classification attributes has a two or three character abbreviation that you can use when entering the attribute on the WLM menus.

The following WLM classification attributes pertain to DB2 DDF threads:

- AI** Accounting information. The value of the DB2 accounting string associated with the DDF server thread, described by QMDAAINF in the DSNDQMDA mapping macro. WLM imposes a maximum length of 143 bytes for accounting information.
- CI** The DB2 correlation ID of the DDF server thread, described by QWHCCV in the DSNDQWHC mapping macro.
- CN** The DB2 collection name of the *first* SQL package accessed by the DRDA requester in the unit of work.
- LU** The VTAM LUNAME of the system that issued the SQL request.
- NET** The VTAM NETID of the system that issued the SQL request.
- PC** Process name. This attribute can be used to classify the application name or the transaction name. The value is defined by QWHCEUTX in the DSNDQWHC mapping macro.
- PK** The name of the **first** DB2 package accessed by the DRDA requester in the unit of work.
- PN** The DB2 plan name associated with the DDF server thread. For DB2 private protocol requesters and DB2 DRDA requesters that are at Version 3 or subsequent releases, this is the DB2 plan name of the requesting application. For other DRDA requesters, use 'DISTSERV' for PN.
- PR** Stored procedure name. This classification only applies if the first SQL statement from the client is a CALL statement.
- SI** Subsystem instance. The DB2 server's z/OS subsystem name.
- SPM** Subsystem parameter. This qualifier has a maximum length of 255 bytes. The first 16 bytes contain the client's user ID. The next 18 bytes contain the client's workstation name. The remaining 221 bytes are reserved.

**Important:** If the length of the client's user ID is less than 16 bytes, uses blanks after the user ID to pad the length. If the length of the client's workstation name is less than 18 bytes, uses blanks after the workstation name to pad the length.
- SSC** Subsystem collection name. When the DB2 subsystem is a member of a DB2 data sharing group, this attribute can be used to classify the data sharing group name. The value is defined by QWHADSGN in the DSNDQWHA mapping macro.
- UI** User ID. The DDF server thread's primary authorization ID, after inbound name translation.

Figure 12 shows how you can associate DDF threads with service classes.

Subsystem-Type Xref Notes Options Help																																																																	
Create Rules for the Subsystem Type			Row 1 to 5 of 5																																																														
Subsystem Type . . . . . <b>DDF</b> (Required)																																																																	
Description . . . Distributed DB2 Fold qualifier names? . . Y (Y or N)																																																																	
Enter one or more action codes: A=After B=Before C=Copy D=Delete M=Move I=Insert rule IS=Insert Sub-rule R=Repeat																																																																	
<table border="1"> <thead> <tr> <th colspan="4">-----Qualifier-----</th> <th colspan="2">-----Class-----</th> </tr> <tr> <th>Action</th> <th>Type</th> <th>Name</th> <th>Start</th> <th>Service</th> <th>Report</th> </tr> </thead> <tbody> <tr> <td colspan="6">DEFAULTS:</td> </tr> <tr> <td>_____</td> <td>1</td> <td>SI</td> <td>DB2P</td> <td>PRDBATCH</td> <td>_____</td> </tr> <tr> <td>_____</td> <td>2</td> <td>CN</td> <td>ONLINE</td> <td>PRDBATCH</td> <td>_____</td> </tr> <tr> <td>_____</td> <td>2</td> <td>PRC</td> <td>PAYPROC</td> <td>PRDONLIN</td> <td>_____</td> </tr> <tr> <td>_____</td> <td>2</td> <td>UI</td> <td>SYSADM</td> <td>PRDONLIN</td> <td>_____</td> </tr> <tr> <td>_____</td> <td>2</td> <td>PK</td> <td>QMFOSS2</td> <td>PRDQUERY</td> <td>_____</td> </tr> <tr> <td>_____</td> <td>1</td> <td>SI</td> <td>DB2T</td> <td>TESTUSER</td> <td>_____</td> </tr> <tr> <td>_____</td> <td>2</td> <td>PR</td> <td>PAYPROCT</td> <td>TESTPAYR</td> <td>_____</td> </tr> </tbody> </table>						-----Qualifier-----				-----Class-----		Action	Type	Name	Start	Service	Report	DEFAULTS:						_____	1	SI	DB2P	PRDBATCH	_____	_____	2	CN	ONLINE	PRDBATCH	_____	_____	2	PRC	PAYPROC	PRDONLIN	_____	_____	2	UI	SYSADM	PRDONLIN	_____	_____	2	PK	QMFOSS2	PRDQUERY	_____	_____	1	SI	DB2T	TESTUSER	_____	_____	2	PR	PAYPROCT	TESTPAYR	_____
-----Qualifier-----				-----Class-----																																																													
Action	Type	Name	Start	Service	Report																																																												
DEFAULTS:																																																																	
_____	1	SI	DB2P	PRDBATCH	_____																																																												
_____	2	CN	ONLINE	PRDBATCH	_____																																																												
_____	2	PRC	PAYPROC	PRDONLIN	_____																																																												
_____	2	UI	SYSADM	PRDONLIN	_____																																																												
_____	2	PK	QMFOSS2	PRDQUERY	_____																																																												
_____	1	SI	DB2T	TESTUSER	_____																																																												
_____	2	PR	PAYPROCT	TESTPAYR	_____																																																												
***** BOTTOM OF DATA *****																																																																	

Figure 12. Classifying DDF threads using z/OS workload management. You assign performance goals to service classes using the services classes menu of WLM.

The preceding figure shows the following classifications above:

- All DB2P applications accessing their first SQL package in the collection ONLINE are in service class PRDONLIN.
- All DB2P applications that call stored procedure PAYPROC first are in service class PRDONLIN.
- All work performed by DB2P user SYSADM is in service class PRDONLIN.
- Users other than SYSADM that run the DB2P PACKAGE QMFOSS2 are in the PRDQUERY class. (The QMFOSS2 package is not in collection ONLINE.
- All other work on the production system is in service class PRBBATCH.
- All users of the test DB2 system are assigned to the TESTUSER class except for work that first calls stored procedure PAYPROCT, which is in service class TESTPAYR.

## Establishing performance periods for DDF threads

You can establish performance periods for DDF threads, including threads that run in the WLM-established stored procedures address space.

By establishing multiple performance periods, you can cause the thread's performance objectives to change based upon the thread's processor consumption. Thus, a long-running unit of work can move down the priority order and let short-running transactions get in and out at a higher priority.

To design performance strategies for these threads:

- Take into account the events that cause a DDF thread to reset its z/OS performance period.
- Use velocity goals and use a single-period service class for threads that are always active. Because threads that are always active do not terminate the enclave and thus do not reset the performance period to period 1, a long-running thread always ends up in the last performance period. Any new business units of work that use that thread suffer the performance consequences. This makes performance periods unattractive for long-running threads.

## Establishing performance objectives for DDF threads

Threads are assigned a service class by the classification rules in the active WLM service policy. Each service class period has a performance objective (goal), and WLM raises or lowers that period's access to system resources as needed to meet the specified goal.

For example, the goal might be "application APPL8 should run in less than 3 seconds of elapsed time 90% of the time".

No matter what your service class goals are, a request to start an address space might time out, depending on the timeout value that is specified in the TIMEOUT VALUE field of installation DSNTIPX. If the timeout value is too small, you might need to increase it to account for a busy system.

To establish performance objectives for DDF threads and the related address spaces:

1. Create a WLM service definition that assigns service classes to the DDF threads under subsystem type DDF and to the DDF address space under subsystem type STC.
2. Install the service definition using the WLM menus and activate a policy (VARY WLM,POLICY=*policy*).

---

## Setting CICS options for threads

The CICS attachment facility provides a multithread connection to DB2 to allow you to operate DB2 with CICS

. Threads allow each CICS application transaction or DB2 command to access DB2 resources.

Use the CICS resource definition online (RDO) to tune the CICS attachment facility and define the characteristics of your threads.

When a transaction needs a thread, an existing thread can be reused or a new thread can be created. If no existing thread is available, and if the maximum number of threads has not been reached, a thread is created. For more information, see:

- *CICS Transaction Server for OS/390 Resource Definition Guide* for information about RDO
- *CICS Transaction Server for z/OS DB2 Guide* for information about DB2 performance considerations and setup of the CICS attachment facility

---

## Setting IMS options for threads

The IMS attachment facility provides a number of design options for threads.

For example, you can:

- Control the number of IMS regions connected to DB2. For IMS, this is also the maximum number of concurrent threads.
- A dependent region with a subsystem member (SSM) that is not empty is connected to DB2 at start up time. Regions with a null SSM cannot create a thread to DB2. A thread to DB2 is created at the first execution of an SQL statement in an IMS application schedule; it is terminated when the application terminates.

The maximum number of concurrent threads used by IMS can be controlled by the number of IMS regions that can connect to DB2 by transaction class assignments. You can control the number by doing the following:

- Minimize the number of regions needing a thread by the way in which you assign applications to regions.
- Provide an empty SSM member for regions that does not connect to DB2.
- Optimize the number of concurrent threads used by IMS.
- Provide efficient thread reuse for high volume transactions.

Thread creation and termination is a significant cost in IMS transactions. IMS transactions identified as wait for input (WFI) can reuse threads: they create a thread at the first execution of an SQL statement and reuse it until the region is terminated. In general, though, use WFI only for transactions that reach a region utilization of at least 75%.

Some degree of thread reuse can also be achieved with IMS class scheduling, queuing, and a PROCLIM count greater than one. IMS Fast Path (IFP) dependent regions always reuse the DB2 thread.

---

## Setting TSO options for threads

You can specify limits for the number of threads taken by the TSO and batch environments

To tune your TSO attachment facility:

- Specify values for the following parameters on the Storage Sizes installation panel (DSNTIPE):

### MAX TSO CONNECT

The maximum number of TSO foreground connections (including DB2I, DB2 QMF, and foreground applications)

### MAX BATCH CONNECT

The maximum number of TSO background connections (including batch jobs and utilities)

- Because DB2 must be stopped to set new values, consider setting a higher MAX BATCH CONNECT for batch periods. The statistics record (IFCID 0001) provides information on the create thread queue. The OMEGAMON statistics report shows (as shown in the example below) that information under the SUBSYSTEM SERVICES section.

### Example

For TSO or batch environments, having 1% of the requests queued is probably a good number to aim for by adjusting the MAX USERS value of installation panel DSNTIPE. Queuing at create thread time is not desirable in the CICS and IMS environments. If you are running IMS or CICS in the same DB2 subsystem as TSO and batch, use MAX BATCH CONNECT and MAX TSO CONNECT to limit the number of threads taken by the TSO and batch environments. The goal is to allow enough threads for CICS and IMS so that their threads do not queue. To determine the number of allied threads queued, see the QUEUED AT<sup>®</sup> CREATE THREAD field ( **A** ) of the OMEGAMON statistics report.

SUBSYSTEM SERVICES	QUANTITY
-----	-----
IDENTIFY	30757.00
CREATE THREAD	30889.00
SIGNON	0.00
TERMINATE	61661.00

ROLLBACK	644.00
COMMIT PHASE 1	0.00
COMMIT PHASE 2	0.00
READ ONLY COMMIT	0.00
UNITS OF RECOVERY INDOUBT	0.00
UNITS OF REC.INDBT RESOLVED	0.00
SYNCHS(SINGLE PHASE COMMIT)	30265.00
QUEUED AT CREATE THREAD <b>A</b>	0.00
SUBSYSTEM ALLIED MEMORY EOT	1.00
SUBSYSTEM ALLIED MEMORY EOM	0.00
SYSTEM EVENT CHECKPOINT	0.00

Figure 13. Thread queuing in the OMEGAMON statistics report

---

## Setting DB2 QMF options for threads

For more information on these aspects of DB2 QMF and how they affect performance, see *DB2 Query Management Facility: Installing and Managing DB2 QMF for TSO/CICS*.

To set DB2 QMF performance options:

Specify the following options:

- The DSQSIROW parameter of the ISPSTART command
- SPACE parameter of the user DB2 QMF profile (Q.PROFILES)
- DB2 QMF region size and the spill file attributes
- TRACE parameter of the user DB2 QMF profile (Q.PROFILES)

---

## Chapter 13. Designing DB2 statistics for performance

Accurate and up-to-date statistics are vital for maintaining good performance SQL processing performance from DB2.

---

### Maintaining statistics in the catalog

DB2 stores catalog statistics that the optimizer uses to determine the best access paths for optimal performance from your SQL statements.

The optimizer uses catalog statistics, database design, and the details of the SQL statement to choose access paths. The optimizer also considers the central processor model, number of central processors, buffer pool size, and RID pool size. The optimizer considers the number of processors only to determine appropriate degrees of parallelism.

Several important calculations for access path selection depend upon buffer pool statistics. The central processor model also affects the optimizer's access path selection. These two factors can change your queries' access paths from one system to another, even if all the catalog statistics are identical. You should keep this in mind when you migrate from a test system to a production system, or when you model a new application.

Mixed central processor models in a data sharing group can also affect access path selection.

### Statistics used for access path selection

The following table lists statistics in the catalog that DB2 uses for access path selection, the values that trigger the use of a default value, and the corresponding default values:

Information in the SYSTABLES and SYSTABLESPACE catalog tables indicates how much data is in your table and how many pages hold data. Information in the SYSINDEXES table lets you compare the available indexes on a table to determine which is the most efficient index for a query. The SYSCOLUMNS and SYSCOLDIST catalog tables provide information that helps DB2 estimate filter factors for predicates.

*Table 28. Catalog data that is used for access path selection or that is collected by RUNSTATS*

Column name	Set by RUNSTATS?	User can update?	Used for access paths? <sup>1</sup>	Description
In every table that RUNSTATS updates:				
STATTIME	Yes	Yes	No	If updated most recently by RUNSTATS, the date and time of that update, not updatable in SYSINDEXPART and SYSTABLEPART. Used for access path selection for SYSCOLDIST if duplicate column values exist for the same column (by user insertion).
SYSIBM.SYSCOLDIST				

Table 28. Catalog data that is used for access path selection or that is collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? <sup>1</sup>	Description
CARDF	Yes	Yes	Yes	For TYPE C, the number of distinct values gathered in the column group; for TYPE F, the number of distinct values for the column group -1; for TYPE='H', the number of distinct values in the column group of the interval indicated by the value of the QUANTILENO column.
COLGROUPCOLNO	Yes	Yes	Yes	The set of columns associated with the statistics. Contains an empty string if NUMCOLUMNS = 1.
COLVALUE	Yes	Yes	Yes	Frequently occurring value in the distribution.
FREQUENCYF	Yes	Yes	Yes	A number, which multiplied by 100, gives the percentage of rows that contain the value of COLVALUE; for TYPE='H', the percentage of rows with the value of COLVALUE that fall into the range between LOWVALUE and HIGHVALUE for the interval indicated by the value of the QUANTILENO column.
HIGHVALUE	Yes	Yes	Yes	For TYPE='H', the high bound for the interval indicated by the value of the QUANTILENO column.
LOWVALUE	Yes	Yes	Yes	For TYPE='H', the low bound for the interval indicated by the value of the QUANTILENO column.
NUMCOLUMNS	Yes	Yes	Yes	The number of columns that are associated with the statistics. The <b>default value</b> is 1.
TYPE	Yes	Yes	Yes	The type of statistics gathered:  C      Cardinality F      Frequent value P      Non-padded H      Histogram statistics
QUANTILENO	Yes	Yes	Yes	For histogram statistics, the ordinary sequence number of the quantile in the whole consecutive value range from low to high.
<b>SYSIBM.SYSCOLDISTSTATS:</b> contains statistics by partition				
CARDF	Yes	Yes	No	A number, which multiplied by 100, gives the percentage of rows that contain the value of COLVALUE; for TYPE='F' or TYPE='N' the number of rows or keys in the partition for which the FREQUENCYF value applies; for TYPE='H', the percentage of rows with the value of COLVALUE that fall into the range between LOWVALUE and HIGHVALUE for the interval indicated by the value of the QUANTILENO column.

Table 28. Catalog data that is used for access path selection or that is collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? <sup>1</sup>	Description
COLGROUPCOLNO	Yes	Yes	No	The set of columns associated with the statistics.
COLVALUE	Yes	Yes	No	Frequently occurring value in the distribution.
FREQUENCYF	Yes	Yes	No	A number, which multiplied by 100, gives the percentage of rows that contain the value of COLVALUE; for TYPE='H', the percentage of rows with the value of COLVALUE that fall into the range between LOWVALUE and HIGHVALUE for the interval indicated by the value of the QUANTILENO column.
HIGHVALUE	Yes	Yes	Yes	For TYPE='H', the high bound for the interval indicated by the value of the QUANTILENO column.
KEYCARDDATA	Yes	Yes	No	The internal representation of the estimate of the number of distinct values in the partition.
LOWVALUE	Yes	Yes	Yes	For TYPE='H', the low bound for the interval indicated by the value of the QUANTILENO column.
NUMCOLUMNS	Yes	Yes	No	The number of columns associated with the statistics. The <b>default value</b> is 1.
TYPE	Yes	Yes	No	The type of statistics gathered: <b>C</b> Cardinality <b>F</b> Frequent value <b>P</b> Non-padded <b>H</b> Histogram statistics
QUANTILENO	Yes	Yes	Yes	For histogram statistics, the ordinary sequence number of the quantile in the whole consecutive value range from low to high.
<b>SYSIBM.SYSCOLSTATS:</b> contains statistics by partition				
COLCARD	Yes	Yes	No	The number of distinct values in the partition. Do not update this column manually without first updating COLCARDATA to a value of length 0. For XML column indicators, NODEID columns, and XML tables, this value of this column is set to -2.
COLCARDATA	Yes	Yes	No	The internal representation of the estimate of the number of distinct values in the partition. A value appears here only if RUNSTATS TABLESPACE is run on the partition. Otherwise, this column contains a string of length 0, indicating that the actual value is in COLCARD.

Table 28. Catalog data that is used for access path selection or that is collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? <sup>1</sup>	Description
HIGHKEY	Yes	Yes	No	First 2000 bytes of the highest value of the column within the partition.If the partition is empty, the value is set to a string of length 0. For LOB columns, XML column indicators, NODEID columns and XML tables, the value of this column is set to blank.
HIGH2KEY	Yes	Yes	No	First 2000 bytes of the second highest value of the column within the partition. If the partition is empty, the value is set to a string of length 0. For LOB columns, XML column indicators, NODEID columns and XML tables, the value of this column is set to blank. This column is updated with decoded values if the column is a randomized key column.
LOWKEY	Yes	Yes	No	First 2000 bytes of the lowest value of the column within the partition. If the partition is empty, the value is set to a string of length 0.For LOB columns, XML column indicators, NODEID columns and XML tables, the value of this column is set to blank.
LOW2KEY	Yes	Yes	No	First 2000 bytes of the second lowest value of the column within the partition.If the partition is empty, the value is set to a string of length 0.For LOB columns, XML column indicators, NODEID columns and XML tables, the value of this column is set to blank. This column is updated with decoded values if the column is a randomized key column.
<b>SYSIBM.SYSCOLUMNS</b>				
COLCARDF	Yes	Yes	Yes	Estimated number of distinct values in the column, -1 to trigger use of the <b>default value</b> (25) and -2 for auxiliary indexes, XML column indicators, NODEID columns, and XML tables.
HIGH2KEY	Yes	Yes	Yes	First 2000 bytes of the second highest value in this column.If the table is empty, the value is set to a string of length 0. For auxiliary indexes, XML column indicators, NODEID columns and XML tables, the value of this column is set to blank.RUNSTATS does not update HIGH2KEY if the column is a randomized key column.

Table 28. Catalog data that is used for access path selection or that is collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? <sup>1</sup>	Description
LOW2KEY	Yes	Yes	Yes	First 2000 bytes of the second lowest value in this column.If the table is empty, the value is set to a string of length 0. For auxiliary indexes, XML column indicators, NODEID columns and XML tables, the value of this column is set to blank.RUNSTATS does not update LOW2KEY if the column is a randomized key column.
<b>SYSIBM.SYSINDEXES</b>				
AVGKEYLEN	Yes	Yes	No	Average key length.
CLUSTERED	Yes	Yes	No	Whether the table is actually clustered by the index. The value of this column is set to blank for auxiliary indexes, NODEID indexes, and XML indexes.
CLUSTERING	No	No	Yes	Whether the index was created using CLUSTER.
CLUSTERRATIOF	Yes	Yes	Yes	A number which, when multiplied by 100, gives the percentage of rows in clustering order. For example, 1 indicates that all rows are in clustering order and .87825 indicates that 87.825% of the rows are in clustering order. For a partitioned index, it is the weighted average of all index partitions in terms of the number of rows in the partition. The value of this column is set to -2 for auxiliary indexes, NODEID indexes, and XML indexes. If this columns contains the default, 0, DB2 uses the value in CLUSTERRATIO, a percentage, for access path selection.
FIRSTKEYCARDF	Yes	Yes	Yes	Number of distinct values of the first key column, or an estimate if updated while collecting statistics on a single partition, -1 to trigger use of the <b>default value</b> (25).
FULLKEYCARDF	Yes	Yes	Yes	Number of distinct values of the full key, -1 to trigger use of the <b>default value</b> (25).
NLEAF	Yes	Yes	Yes	Number of active leaf pages in the index, -1 to trigger use of the <b>default value</b> (SYSTABLES.CARD/300).
NLEVELS	Yes	Yes	Yes	Number of levels in the index tree, -1 to trigger use of the <b>default value</b> (2).
SPACEF	Yes	Yes	No	Disk storage in KB.
DATAPEATFACTORF	Yes	Yes	Yes	The number of times that data pages are repeatedly scanned after the index key is ordered. This number is -1 if statistics have not been collected. Valid values are -1 or any value that is equal to or greater than 1.
<b>SYSIBM.SYSINDEXPART:</b> contains statistics for space utilization				
AVGKEYLEN	Yes	Yes	No	Average key length.

Table 28. Catalog data that is used for access path selection or that is collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? <sup>1</sup>	Description
CARDF	Yes	No	No	Number of rows or LOBs referenced by the index or partition
DSNUM	Yes	Yes	No	Number of data sets.
EXTENTS	Yes	Yes	No	Number of data set extents (for multiple pieces, the value is for the extents in the last data set).
FAROFFPOSF	Yes	No	No	Number of referenced rows far from the optimal position because of an insert into a full page. The value of this column is set to -2 for NODEID indexes and XML indexes.
LEAFDIST	Yes	No	No	100 times the number of pages between successive leaf pages. The value of this column is set to -2 for NODEID indexes and XML indexes.
LEAFFAR	Yes	Yes	No	Number of leaf pages located physically far away from previous leaf pages for successive active leaf pages accessed in an index scan. See "LEAFNEAR and LEAFFAR columns" on page 734 for more information.
LEAFNEAR	Yes	Yes	No	Number of leaf pages located physically near previous leaf pages for successive active leaf pages. See "LEAFNEAR and LEAFFAR columns" on page 734 for more information.
LIMITKEY	No	No	Yes	The limit key of the partition in an internal format, 0 if the index is not partitioned.
NEAROFFPOSF	Yes	No	No	Number of referenced rows near but not at the optimal position because of an insert into a full page. The value of this column is set to -2 for NODEID indexes and XML indexes.
PQTY	Yes	No	No	The primary space allocation in 4K blocks for the data set.
PSEUDO_DEL_ENTRIES	Yes	Yes	No	Number of pseudo-deleted keys.
SECQTYI	Yes	No	No	Secondary space allocation in units of 4 KB, stored in integer format instead of small integer format supported by SQTY. If a storage group is not used, the value is 0.
SPACE	Yes	No	No	The number of KB of space currently allocated for all extents (contains the accumulated space used by all pieces if a page set contains multiple pieces)
SQTY	Yes	No	No	The secondary space allocation in 4 KB blocks for the data set.
SPACEF	Yes	Yes	No	Disk storage in KB.
<b>SYSIBM.SYSINDEXSTATS:</b> contains statistics by partition				

Table 28. Catalog data that is used for access path selection or that is collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? <sup>1</sup>	Description
CLUSTERRATIOF	Yes	Yes	No	A number which, when multiplied by 100, gives the percentage of rows in clustering order. For example, 1 indicates that all rows are in clustering order and .87825 indicates that 87.825% of the rows are in clustering order.
FIRSTKEYCARDF	Yes	Yes	No	Number of distinct values of the first key column, or an estimate if updated while collecting statistics on a single partition.
FULLKEYCARDDATA	Yes	Yes	No	The internal representation of the number of distinct values of the full key.
FULLKEYCARDF	Yes	Yes	No	Number of distinct values of the full key.
KEYCOUNTF	Yes	Yes	No	Number of rows in the partition, -1 to trigger use of the value in KEYCOUNT
NLEAF	Yes	Yes	No	Number of leaf pages in the index.
NLEVELS	Yes	Yes	No	Number of levels in the index tree.
DATAPEATFACTORF	Yes	Yes	Yes	The number of times that data pages are repeatedly scanned after the index key is ordered. This number is -1 if statistics have not been collected. Valid values are -1 or any value that is equal to or greater than 1.
<b>SYSIBM.SYSKEYTARGETS</b>				
CARDF	Yes	No	Yes	Number of distinct values for the key-target. The value of this column is set to -2 for NODEID indexes and XML indexes.
HIGH2KEY	Yes	Yes	Yes	Second highest key value
LOW2KEY	Yes	Yes	Yes	Second lowest key value
STATS_FORMAT	Yes	Yes	Yes	Type of statistics gathered:  <b>blank</b> No statistics have been collected, or VARCHAR column statistical values are padded  <b>N</b> Varchar statistical values are not padded
<b>SYSIBM.SYSKEYTARGETSTATS</b>				
HIGHKEY	Yes	Yes	Yes	Highest key value
HIGH2KEY	Yes	Yes	Yes	Second highest key value
LOWKEY	Yes	Yes	Yes	Lowest key value
LOW2KEY	Yes	Yes	Yes	Second lowest key value
STATS_FORMAT	Yes	Yes	Yes	Type of statistics gathered:  <b>blank</b> No statistics have been collected, or VARCHAR column statistical values are padded  <b>N</b> Varchar statistical values are not padded

Table 28. Catalog data that is used for access path selection or that is collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? <sup>1</sup>	Description
<b>SYSIBM.SYSKEYTGTDIST</b>				
CARDF	Yes	Yes	Yes	For TYPE C, Number of distinct values gathered in the key group; for TYPE F, number of distinct values for the key group -1; for TYPE='H', the number of distinct values in the column group of the interval indicated by the value in the QUANTILENO column.
KEYGROUPKEYNO	Yes	Yes	Yes	The set of KEYS associated with the statistics. Contains an empty string if NUMKEYS = 1.
KEYVALUE	Yes	Yes	Yes	Frequently occurring value in the distribution.
FREQUENCYF	Yes	Yes	Yes	A number, which multiplied by 100, gives the percentage of rows that contain the value of KEYVALUE; for TYPE='H', the percentage of rows with the value of KEYVALUE that fall into the range between LOWVALUE and HIGHVALUE for the interval indicated by the value of the QUANTILENO column.
HIGHVALUE	Yes	Yes	Yes	For TYPE='H', the high bound for the interval indicated by the value of the QUANTILENO column.
LOWVALUE	Yes	Yes	Yes	For TYPE='H', the low bound for the interval indicated by the value of the QUANTILENO column.
NUMKEYS	Yes	Yes	Yes	The number of keys associated with the statistics. The <b>default value</b> is 1.
TYPE	Yes	Yes	Yes	The type of statistics gathered:  <b>C</b> Cardinality <b>F</b> Frequent value <b>P</b> Non-padded <b>H</b> Histogram statistics
QUANTILENO	Yes	Yes	Yes	For histogram statistics, the ordinary sequence number of the quantile in the whole consecutive value range from low to high.
<b>SYSIBM.SYSKEYTGTDISTSTATS: contains statistics by partition</b>				
CARDF	Yes	Yes	No	A number, which multiplied by 100, gives the percentage of rows that contain the value of KEYVALUE; for TYPE='H', the percentage of rows with the value of COLVALUE that fall into the range between LOWVALUE and HIGHVALUE for the interval indicated by the value of the QUANTILENO column.

Table 28. Catalog data that is used for access path selection or that is collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? <sup>1</sup>	Description
KEYVALUE	Yes	Yes	No	The set of keys associated with the statistics
KEYGROUPKEYNO	Yes	Yes	No	Frequently occurring value in the distribution.
FREQUENCYF	Yes	Yes	No	A number, which multiplied by 100, gives the percentage of rows that contain the value of KEYVALUE; for TYPE='H', the percentage of rows with the value of KEYVALUE that fall into the range between LOWVALUE and HIGHVALUE for the interval indicated by the value of the QUANTILENO column.
HIGHVALUE	Yes	Yes	Yes	For TYPE='H', the high bound for the interval indicated by the value of the QUANTILENO column.
LOWVALUE	Yes	Yes	Yes	For TYPE='H', the low bound for the interval indicated by the value of the QUANTILENO column.
QUANTILENO	Yes	Yes	Yes	For histogram statistics, the ordinary sequence number of the quantile in the whole consecutive value range from low to high.
<b>SYSIBM.SYSLOBSTATS:</b> contains LOB table space statistics				
AVGSIZE	Yes	Yes	No	Average size of a LOB in bytes.
FREESPACE	Yes	Yes	No	The number of KB of available space in the LOB table space.
ORGRATIO	Yes	Yes	No	The percentage of organization in the LOB table space. A value of 100 indicates perfect organization of the LOB table space. A value of 1 indicates that the LOB table space is disorganized.  A value of 0.00 indicates that the LOB table space is totally disorganized. An empty LOB table space has an ORGRATIO value of 100.00.
<b>SYSIBM.SYSROUTINES:</b> Contains statistics for table functions. See "Updating catalog statistics" on page 308 for more information about using these statistics.				
CARDINALITY	No	Yes	Yes	The predicted cardinality of a table function, -1 to trigger use of the <b>default value</b> (10 000)
INITIAL_INSTS	No	Yes	Yes	Estimated number of instructions executed the first and last time the function is invoked, -1 to trigger use of the <b>default value</b> (40 000)
INITIAL_IOS	No	Yes	Yes	Estimated number of IOs performed the first and last time the function is invoked, -1 to trigger use of the <b>default value</b> (0)

Table 28. Catalog data that is used for access path selection or that is collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? <sup>1</sup>	Description
INSTS_PER_INVOC	No	Yes	Yes	Estimated number of instructions per invocation, -1 to trigger use of the <b>default value</b> (4 000)
IOS_PER_INVOC	No	Yes	Yes	Estimated number of IOs per invocation, -1 to trigger use of the <b>default value</b> (0)
<b>SYSIBM.SYSTABLEPART:</b> contains statistics for space utilization				
AVGROWLEN	Yes	No	No	Average row length
CARDF	Yes	No	No	Total number of rows in the table space or partition. For LOB table spaces, the number of LOBs in the table space.
DSNUM	Yes	Yes	No	Number of data sets.
EXTENTS	Yes	Yes	No	Number of data set extents (for multiple pieces, the value is for the extents in the last data set).
FARINDREF	Yes	No	No	Number of rows relocated far from their original page.
NEARINDREF	Yes	No	No	Number of rows relocated near their original page.
PAGESAVE	Yes	No	No	Percentage of pages, times 100, saved in the table space or partition as a result of using data compression.
PERCACTIVE	Yes	No	No	Percentage of space occupied by active rows, containing actual data from active tables, -2 for LOB table spaces.
PERCDROP	Yes	No	No	For non-segmented table spaces, the percentage of space occupied by rows of data from dropped tables; for segmented table spaces, 0.
PQTY	Yes	No	No	The primary space allocation in 4K blocks for the data set.
SECQTYI	Yes	No	No	Secondary space allocation in units of 4 KB, stored in integer format instead of small integer format supported by SQTY. If a storage group is not used, the value is 0.
SPACE	Yes	No	No	The number of KB of space currently allocated for all extents (contains the accumulated space used by all pieces if a page set contains multiple pieces).
SPACEF	Yes	Yes	No	Disk storage in KB.
SQTY	Yes	No	No	The secondary space allocation in 4K blocks for the data set
<b>SYSIBM.SYSTABLES:</b>				
AVGROWLEN	Yes	Yes	No	Average row length of the table specified in the table space.
CARDF	Yes	Yes	Yes	Total number of rows in the table or total number of LOBs in an auxiliary table, -1 to trigger use of the <b>default value</b> (10 000).

Table 28. Catalog data that is used for access path selection or that is collected by RUNSTATS (continued)

Column name	Set by RUNSTATS?	User can update?	Used for access paths? <sup>1</sup>	Description
EDPROC	No	No	Yes	Non-blank value if an edit exit routine is used.
NPAGES	Yes	Yes	Yes	Total number of pages on which rows of this table appear, -1 to trigger use of the <b>default value</b> (CEILING(1 + CARD/20))
NPAGESF	Yes	Yes	Yes	Number of pages used by the table.
PCTPAGES	Yes	Yes	No	For non-segmented table spaces, percentage of total pages of the table space that contain rows of the table; for segmented table spaces, the percentage of total pages in the set of segments assigned to the table that contain rows of the table.
PCTROWCOMP	Yes	Yes	Yes	Percentage of rows compressed within the total number of active rows in the table.
SPACEF	Yes	Yes	No	Disk storage in KB.
<b>SYSIBM.SYSTABLESPACE:</b>				
AVGROWLEN	Yes	Yes	No	Average row length.
NACTIVEF	Yes	Yes	Yes	Number of active pages in the table space, the number of pages touched if a cursor is used to scan the entire file, 0 to trigger use of the value in the NACTIVE column instead. If NACTIVE contains 0, DB2 uses the <b>default value</b> (CEILING(1 + CARD/20)).
SPACE	Yes	No	No	Disk storage in KB.
SPACEF	Yes	Yes	No	Disk storage in KB.
<b>SYSIBM.SYSTABSTATS:</b> contains statistics by partition				
CARDF	Yes	Yes	Yes	Total number of rows in the partition, -1 to trigger use of the value in the CARD column. If CARD is -1, DB2 uses a <b>default value</b> (10 000).
NACTIVE	Yes	Yes	No	Number of active pages in the partition.
NPAGES	Yes	Yes	Yes	Total number of pages on which rows of the partition appear, -1 to trigger use of the <b>default value</b> (CEILING(1 + CARD/20)).
PCTPAGES	Yes	Yes	No	Percentage of total active pages in the partition that contain rows of the table.
PCTROWCOMP	Yes	Yes	No	Percentage of rows compressed within the total number of active rows in the partition, -1 to trigger use of the <b>default value</b> (0).
<b>Note:</b> <sup>1</sup> Statistics on LOB-related values are not used for access path selection. SYSCOLDISTSTATS and SYSINDEXSTATS are not used for parallelism access paths. SYSCOLSTATS information (CARD, HIGHKEY, LOWKEY, HIGH2KEY, and LOW2KEY) is used to determine the degree of parallelism.				

## Filter factors and catalog statistics

DB2 needs an accurate estimate of the number of rows that qualify after applying each predicate in order to determine optimal access paths.

When multiple tables are accessed, filtering also affects the cost of join order and join method. The catalog tables SYSIBM.SYSCOLUMNS and SYSIBM.SYSCOLDIST are the main source of statistics for calculating predicate filter factors. The following columns are particularly important:

- SYSCOLUMNS.COLCARDF indicates whether statistics exist for a column or not. A positive value is an estimate of the number of distinct values in the column (cardinality). A value of '-1' results in the use of default statistics.  
The value of COLCARDF generated by RUNSTATS TABLESPACE is an estimate determined by a sampling method. If you know a more accurate number for COLCARDF, you can supply it by updating the catalog. If the column is the first column of an index, the value generated by RUNSTATS INDEX is exact.
- Columns in SYSCOLDIST contain statistics about the frequency (or distribution) of values for a single column. It can also contain statistics about the cardinality of a group of columns and the frequency of values for that group.

When frequency statistics do not exist, DB2 assumes that the data is uniformly distributed and all values in the column occur with the same frequency. This assumption can lead to an inaccurate estimate of the number of qualifying rows if the data is skewed, which can result in performance problems.

**Example:** Assume that a column (AGE\_CATEGORY) contains five distinct values (COLCARDF), each of which occur with the following frequencies:

AGE_CATEGORY	FREQUENCY
-----	-----
INFANT	5%
CHILD	15%
ADOLESCENT	25%
ADULT	40%
SENIOR	15%

Without this frequency information, DB2 would use a default filter factor of 1/5 (1/COLCARDF), or 20%, to estimate the number of rows that qualify for predicate AGE\_CATEGORY=ADULT. However, the actual frequency of that age category is 40%. Thus, the number of qualifying rows is underestimated by 50%.

When collecting statistics about indexes, you can specify the KEYCARD option of RUNSTATS to collect cardinality statistics on the specified indexes. You can also specify the FREQVAL option with KEYCARD to specify whether distribution statistics are collected for what number of concatenated index columns. By default, distribution statistics are collected on the first column of each index for the 10 most frequently occurring values. FIRSTKEYCARDF and FULLKEYCARDF are also collected by default.

When collecting statistics at the table level, you can specify the COLUMN option of RUNSTATS to collect cardinality statistics on just the specified columns. You can also use the COLGROUP option to specify a group of columns for which to collect cardinality statistics. If you use the FREQVAL option with COLGROUP, you can also collect distribution statistics for the column group.

To limit the resources required to collect statistics, you only need to collect column cardinality and frequency statistics that have changed. For example, a column on GENDER is likely to have a COLCARDF of 2, with M and F as the possible values. It is unlikely that the cardinality for this column ever changes. The distribution of the values in the column might not change often, depending on the volatility of the data.

**Recommendation:** If query performance is not satisfactory, consider the following actions:

- Collect cardinality statistics on all columns that are used as predicates in a WHERE clause.
- Collect frequencies for all columns with a low cardinality that are used as COL op constant predicates.
- Collect frequencies for a column when the column can contain default data, the default data is skewed, and the column is used as a COL op constant predicate.
- Collect KEYCARD on all candidate indexes.
- Collect column group statistics on all join columns.
- LOW2KEY and HIGH2KEY columns are limited to storing the first 2000 bytes of a key value. If the column is nullable, values are limited to 1999 bytes.
- The closer SYSINDEXES.CLUSTERRATIOF is to 100% (a value of 1), the more closely the ordering of the index entries matches the physical ordering of the table rows. Refer to Figure 113 on page 732 to see how an index with a high cluster ratio differs from an index with a low cluster ratio.

## Histogram statistics

Histogram statistics enable DB2 to improve access path selection by estimating predicate selectivity from value-distribution statistics that are collected over the entire range of values in a data set.

**Restriction:** RUNSTATS cannot collect histogram statistics on randomized key columns.

DB2 chooses the best access path for a query based on predicate selectivity estimation, which in turn relies heavily on data distribution statistics. Histogram statistics summarize data distribution on an interval scale by dividing the entire range of possible values within a data set into a number of intervals.

DB2 creates equal-depth histogram statistics, meaning that it divides the whole range of values into intervals that each contain about the same percentage of the total number rows. The following columns in a histogram statistics table define an interval:

### QUANTILENO

An ordinary sequence number that identifies the interval.

### HIGHVALUE

The value that serves as the upper bound for the interval.

### LOWVALUE

A value that serves as the lower bound for the interval.

Note the following characteristics of histogram statistics intervals:

- Each interval includes approximately the same number, or percentage, of the rows. A highly frequent single value might occupy an interval by itself.
- A single value is never broken into more than one interval, meaning that the maximum number of intervals is equal to the number of distinct values on the column. The maximum number of intervals cannot exceed 100, which is the maximum number that DB2 supports.
- Adjacent intervals sometime skip values that do not appear in the table, especially when doing so avoids a large range of skipped values within an interval. For example, if the value 30 above has 1% frequency, placing it in the

seventh interval would balance the percentage of rows in the 6th and 7th intervals. However, doing so would introduce a large skipped range to the seventh interval.

- HIGHVALUE and LOWVALUE can be inclusive or exclusive, but an interval generally represents a non-overlapped value range.
- NULL values, if any exist, occupy a single interval.
- Because DB2 cannot break any single value into two different intervals, the maximum number of intervals is limited to the number of distinct values in the column, and cannot exceed the DB2 maximum of 100 intervals.

## Statistics for partitioned table spaces

For a partitioned table space, DB2 keeps statistics separately by partition and also collectively for the entire table space.

The following table shows the catalog tables that contain statistics by partition and, for each one, the table that contains the corresponding aggregate statistics.

*Table 29. The catalog tables that contain statistics by partition and the table that contains the corresponding aggregate statistics*

Statistics by partition are in	Aggregate statistics are in
SYSTABSTATS	SYSTABLES
SYSINDEXSTATS	SYSINDEXES
SYSCOLSTATS	SYSCOLUMNS
SYSCOLDISTSTATS	SYSCOLDIST
SYSKEYTARGETSTATS	SYSKEYTARGETS
SYSKEYTGTDISTSTATS	SYSKEYTGTDIST

If you run RUNSTATS for separate partitions of a table space, DB2 uses the results to update the aggregate statistics for the entire table space. You should either run RUNSTATS once on the entire object before collecting statistics on separate partitions or use the appropriate option to ensure that the statistics are aggregated appropriately, especially if some partitions are not loaded with data. For recommendations about running RUNSTATS on separate partitions, see Chapter 30, “Gathering monitor statistics and update statistics,” on page 499.

## Setting default statistics for created temporary tables

You can establish default statistical values for the cardinality and the number of pages if you can estimate the normal cardinality and number of pages that used the values for a particular created temporary table.

When preparing an SQL statement that refers to a created temporary table, if the table has been instantiated, DB2 uses the cardinality and number of pages that are maintained for that table in storage. If the table has not been instantiated, DB2 looks at the CARDF and NPAGES columns of the SYSTABLES row for the created temporary table. These values are normally -1 because RUNSTATS cannot run against a created temporary table.

You can manually update the values in the CARDF and NPAGES columns of the SYSTABLES row for the created temporary table. These values become the default values that are used if more accurate values are not available or if more accurate values cannot be used. The more accurate values are available only for dynamic SQL statements that are prepared after the instantiation of the created temporary

table, but within the same unit of work. These more accurate values are not used if the result of the dynamic bind is destined for the dynamic statement cache.

## History statistics

Several catalog tables provide historical statistics for other catalog tables.

These catalog history tables include:

- SYSIBM.SYSCOLDIST\_HIST
- SYSIBM.SYSCOLUMNS\_HIST
- SYSIBM.SYSINDEXES\_HIST
- SYSIBM.SYSINDEXPART\_HIST
- SYSIBM.SYSINDEXSTATS\_HIST
- SYSIBM.SYSKEYTARGETS\_HIST
- SYSIBM.SYSKEYTGTDIST\_HIST
- SYSIBM.SYSLOBSTATS\_HIST
- SYSIBM.SYSTABLEPART\_HIST
- SYSIBM.SYSTABLES\_HIST
- SYSIBM.SYSTABSTATS\_HIST

For example, SYSIBM.SYSTABLESPACE\_HIST provides statistics for activity in SYSIBM.SYSTABLESPACE, SYSIBM.SYSTABLEPART\_HIST provides statistics for activity in SYSIBM.SYSTABLEPART, and so on.

When DB2 adds or changes rows in a catalog table, DB2 might also write information about the rows to the corresponding catalog history table. Although the catalog history tables are not identical to their counterpart tables, they do contain the same columns for access path information and space utilization information. The history statistics provide a way to study trends, to determine when utilities, such as REORG, should be run for maintenance, and to aid in space management.

Table 30 lists the catalog data that are collected for historical statistics. For information on how to gather these statistics, see Chapter 30, “Gathering monitor statistics and update statistics,” on page 499.

*Table 30. Catalog data collected for historical statistics*

Column name	Provides access path statistics <sup>1</sup>	Provides space statistics	Description
<b>SYSIBM.SYSCOLDIST_HIST</b>			
CARDF	Yes	No	For TYPE C, Number of distinct values gathered in the column group; for TYPE='H', the number of distinct values in the column group of the interval indicated by the value in the QUANTILENO column
COLGROUPCOLNO	Yes	No	Identifies the columns involved in multi-column statistics
COLVALUE	Yes	No	Frequently occurring value in the key distribution
FREQUENCYF	Yes	No	A number, which multiplied by 100, gives the percentage of rows that contain the value of COLVALUE; for TYPE='H', the percentage of rows with the value of COLVALUE that fall into the range between LOWVALUE and HIGHVALUE for the interval indicated by the value of the QUANTILENO column.

Table 30. Catalog data collected for historical statistics (continued)

Column name	Provides access path statistics <sup>1</sup>	Provides space statistics	Description
HIGHVALUE	Yes	No	For TYPE='H', the value of the high bound for the interval indicated by the value of the QUANTILENO column.
LOWVALUE	Yes	No	For TYPE='H', the value of the low bound for the interval indicated by the value of the QUANTILENO column.
NUMCOLUMNS	Yes	No	Number of columns involved in multi-column statistics
TYPE	Yes	No	The type of statistics gathered: <b>C</b> Cardinality <b>F</b> Frequent value <b>P</b> Non-padded <b>H</b> Histogram statistics
QUANTILENO	Yes	No	For histogram statistics, the ordinary sequence number of the quantile in the whole consecutive value range from low to high.
<b>SYSIBM.SYSCOLUMNS_HIST</b>			
COLCARDF	Yes	No	Estimated number of distinct values in the column
HIGH2KEY	Yes	No	Second highest value of the column, or blank
LOW2KEY	Yes	No	Second lowest value of the column, or blank
<b>SYSIBM.SYSINDEXES_HIST</b>			
CLUSTERING	Yes	No	Whether the index was created with CLUSTER
CLUSTERRATIOF	Yes	No	A number, when multiplied by 100, gives the percentage of rows in the clustering order
FIRSTKEYCARDF	Yes	No	Number of distinct values in the first key column
FULLKEYCARDF	Yes	No	Number of distinct values in the full key
NLEAF	Yes	No	Number of active leaf pages
NLEVELS	Yes	No	Number of levels in the index tree
DATAPEATFACTORF	Yes	No	The number of times that data pages are repeatedly scanned after the index key is ordered. This number is -1 if statistics have not been collected. Valid values are -1 or any value that is equal to or greater than 1.
<b>SYSIBM.SYSINDEXPART_HIST</b>			
CARDF	No	No	Number of rows or LOBs referenced
DSNUM	No	Yes	Number of data sets
EXTENTS	No	Yes	Number of data set extents (for multiple pieces, the value is for the extents in the last data set)
FAROFFPOSF	No	Yes	Number of rows referenced far from the optimal position
LEAFDIST	No	Yes	100 times the number of pages between successive leaf pages

Table 30. Catalog data collected for historical statistics (continued)

Column name	Provides access path statistics <sup>1</sup>	Provides space statistics	Description
LEAFFAR	No	Yes	Number of leaf pages located physically far away from previous leaf pages for successive active leaf pages accessed in an index scan
LEAFNEAR	No	Yes	Number of leaf pages located physically near previous leaf pages for successive active leaf pages
NEAROFFPOSF	No	Yes	Number of rows referenced near but not at the optimal position
PQTY	No	Yes	Primary space allocation in 4K blocks for the data set
PSEUDO_DEL_ENTRIES	No	Yes	Number of pseudo-deleted keys
SECQTYI	No	Yes	Secondary space allocation in 4K blocks for the data set.
SPACEF	No	Yes	Disk storage in KB
<b>SYSIBM.SYSINDEXSTATS_HIST</b>			
CLUSTERRATIO	Yes	No	A number, which when multiplied by 100, gives the percentage of rows in the clustering order
FIRSTKEYCARDF	Yes	No	Number of distinct values of the first key column
FULLKEYCARDF	Yes	No	Number of distinct values of the full key
KEYCOUNTF	Yes	No	Total number of rows in the partition
NLEAF	Yes	No	Number of leaf pages
NLEVELS	Yes	No	Number of levels in the index tree
DATAPEATFACTORF	Yes	No	The number of times that data pages are repeatedly scanned after the index key is ordered. This number is -1 if statistics have not been collected. Valid values are -1 or any value that is equal to or greater than 1.
<b>SYSIBM.SYSKEYTARGETS_HIST</b>			
KEYCARDF	Yes	No	For type C statistics, the number of distinct values for key-target
HIGH2KEY	Yes	No	The second highest key value
LOW2KEY	Yes	No	The second lowest key value
STATS_FORMAT	Yes	No	Type of statistics gathered:  <b>blank</b> No statistics have been collected, or VARCHAR column statistical values are padded  <b>N</b> Varchar statistical values are not padded
<b>SYSIBM.SYSKEYTGTDIST_HIST</b>			
CARDF	Yes	No	For TYPE C, Number of distinct values gathered in the key group; for TYPE='H', the number of distinct values in the key group of the interval indicated by the value in the QUANTILENO column

Table 30. Catalog data collected for historical statistics (continued)

Column name	Provides access path statistics <sup>1</sup>	Provides space statistics	Description
KEYGROUPKEYNO	Yes	No	Identifies the keys involved in multi-column statistics
KEYVALUE	Yes	No	Frequently occurring value in the key distribution
HIGHVALUE	Yes	No	For TYPE='H', the value of the high bound for the interval indicated by the value of the QUANTILENO column.
FREQUENCYF	Yes	No	A number, which multiplied by 100, gives the percentage of rows that contain the value of KEYVALUE; for TYPE='H', the percentage of rows with the value of KEYVALUE that fall into the range between LOWVALUE and HIGHVALUE for the interval indicated by the value of the QUANTILENO column.
LOWVALUE	Yes	No	For TYPE='H', the value of the low bound for the interval indicated by the value of the QUANTILENO column.
NUMKEYS	Yes	No	Number of keys involved in multi-key statistics
TYPE	Yes	No	The type of statistics gathered:  C      Cardinality F      Frequent value P      Non-padded H      Histogram statistics
QUANTILENO	Yes	No	For histogram statistics, the ordinary sequence number of the quantile in the whole consecutive value range from low to high.
<b>SYSIBM.SYSLOBSTATS_HIST</b>			
FREESPACE	No	Yes	The amount of free space in the LOB table space
ORGRATIO	No	Yes	The percentage of organization in the LOB table space. A value of 100 indicates perfect organization of the LOB table space. A value of 1 indicates that the LOB table space is disorganized.  A value of 0.00 indicates that the LOB table space is totally disorganized. An empty LOB table space has an ORGRATIO value of 100.00.
<b>SYSIBM.SYSTABLEPART_HIST</b>			
CARDF	No	Yes	Number of rows in the table space or partition
DSNUM	No	Yes	Number of data sets
EXTENTS	No	Yes	Number of data set extents (for multiple pieces, the value is for the extents in the last data set)
FARINDREF	No	Yes	Number of rows relocated far from their original position
NEARINDREF	No	Yes	Number of rows relocated near their original position
PAGESAVE	No	Yes	Percentage of pages saved by data compression

Table 30. Catalog data collected for historical statistics (continued)

Column name	Provides access path statistics <sup>1</sup>	Provides space statistics	Description
PERCACTIVE	No	Yes	Percentage of space occupied by active pages
PERCDROP	No	Yes	Percentage of space occupied by pages from dropped tables
PQTY	No	Yes	Primary space allocation in 4K blocks for the data set
SECQTYI	No	Yes	Secondary space allocation in 4K blocks for the data set.
SPACEF	No	Yes	The number of KB of space currently used
<b>SYSIBM.SYSTABLES_HIST</b>			
AVGROWLEN	No	Yes	Average row length of the table specified in the table space
CARDF	Yes	No	Number of rows in the table or number of LOBs in an auxiliary table
NPAGESF	Yes	No	Number of pages used by the table
PCTPAGES	No	Yes	Percentage of pages that contain rows
PCTROWCOMP	Yes	No	Percentage of active rows compressed
<b>SYSIBM.SYSTABSTATS_HIST</b>			
CARDF	Yes	No	Number of rows in the partition
NPAGES	Yes	No	Total number of pages with rows
<b>Note:</b> <sup>1</sup> The access path statistics in the history tables are collected for historical purposes and are not used for access path selection.			

## What other statistics provide index costs

Certain statistics in the SYSINDEXES table also give information about costs to process the index.

**FIRSTKEYCARDF:** The number of distinct values of the first index key column. When an indexable equal predicate is specified on the first index key column,  $1/\text{FIRSTKEYCARDF}$  is the filter factor for the predicate and the index. The higher the number is, the less the cost is.

**FULLKEYCARDF:** The number of distinct values for the entire index key. When indexable equal predicates are specified on all the index key columns,  $1/\text{FULLKEYCARDF}$  is the filter factor for the predicates and the index. The higher the number is, the less the cost is.

When the number of matching columns is greater than 1 and less than the number of index key columns, the filtering of the index is located between  $1/\text{FIRSTKEYCARDF}$  and  $1/\text{FULLKEYCARDF}$ .

**NLEAF:** The number of active leaf pages in the index. NLEAF is a portion of the cost to scan the index. The smaller the number is, the less the cost is. It is also less when the filtering of the index is high, which comes from FIRSTKEYCARDF, FULLKEYCARDF, and other indexable predicates.

**NLEVELS:** The number of levels in the index tree. NLEVELS is another portion of the cost to traverse the index. The same conditions as NLEAF apply. The smaller the number is, the less the cost is.

**DATA REPEAT FACTOR:** The number of times that data pages are repeatedly scanned after the index key is ordered.

---

## Modeling your production system

To see what access paths your production queries use, you can update the catalog statistics on your test system to be the same as your production system.

To do that, run RUNSTATS on your production tables to get current statistics for access path selection. Then retrieve them and use them to build SQL statements to update the catalog of the test system.

**Example:** You can use queries similar to the following queries to build those statements. To successfully model your production system, the table definitions must be the same on both the test and production systems. For example, they must have the same creator, name, indexes, number of partitions, and so on.

### PSPI

Use the following statements to update SYSTABLESPACE, SYSTABLES, SYSINDEXES, and SYSCOLUMNS:

```
SELECT DISTINCT 'UPDATE SYSIBM.SYSTABLESPACE SET NACTIVEF='
CONCAT STRIP(CHAR(NACTIVEF))
CONCAT ',NACTIVE='CONCAT STRIP(CHAR(NACTIVE))
CONCAT ' WHERE NAME='''CONCAT TS.NAME
CONCAT ''' AND DBNAME ='''CONCAT TS.DBNAME CONCAT''*'
FROM SYSIBM.SYSTABLESPACE TS, SYSIBM.SYSTABLES TBL
WHERE TS.NAME = TSNAME
      AND TBL.CREATOR IN (table_creator_list)
      AND TBL.NAME IN (table_list)
      AND (NACTIVEF >=0 OR NACTIVE >=0);

SELECT 'UPDATE SYSIBM.SYSTABLES SET CARDF='
CONCAT STRIP(CHAR(CARDF))
CONCAT ',NPAGES='CONCAT STRIP(CHAR(NPAGES))
CONCAT ',PCTROWCOMP='CONCAT STRIP(CHAR(PCTROWCOMP))
CONCAT ' WHERE NAME='''CONCAT NAME
CONCAT ''' AND CREATOR ='''CONCAT CREATOR CONCAT''*'
FROM SYSIBM.SYSTABLES WHERE
CREATOR IN (creator_list)
      AND NAME IN (table_list)
      AND CARDF >= 0;

SELECT 'UPDATE SYSIBM.SYSINDEXES SET FIRSTKEYCARDF='
CONCAT STRIP(CHAR(FIRSTKEYCARDF))
CONCAT ',FULLKEYCARDF='CONCAT STRIP(CHAR(FULLKEYCARDF))
CONCAT ',NLEAF='CONCAT STRIP(CHAR(NLEAF))
CONCAT ',NLEVELS='CONCAT STRIP(CHAR(NLEVELS))
CONCAT ',CLUSTERRATIO='CONCAT STRIP(CHAR(CLUSTERRATIO))
CONCAT ',CLUSTERRATIOF='CONCAT STRIP(CHAR(CLUSTERRATIOF))
CONCAT ',DATA REPEAT FACTORF='CONCAT STRIP(CHAR(DATA REPEAT FACTORF))
CONCAT ' WHERE NAME='''CONCAT NAME
CONCAT ''' AND CREATOR ='''CONCAT CREATOR CONCAT''*'
FROM SYSIBM.SYSINDEXES
WHERE TBcreator IN (creator_list)
      AND TBNAME IN (table_list)
      AND FULLKEYCARDF >= 0;

SELECT 'UPDATE SYSIBM.SYSCOLUMNS SET COLCARDF='
CONCAT STRIP(CHAR(COLCARDF))
```

```

CONCAT',HIGH2KEY= X''' CONCAT HEX(HIGH2KEY)
CONCAT''',LOW2KEY= X''' CONCAT HEX(LOW2KEY)
CONCAT''' WHERE TBNAME=''' CONCAT TBNAME CONCAT ''' AND COLNO='
CONCAT STRIP(CHAR(COLNO))
CONCAT ' AND TBCREATOR =''' CONCAT TBCREATOR CONCAT '*'
FROM SYSIBM.SYSCOLUMNS
WHERE TBCREATOR IN (creator_list)
      AND TBNAME IN (table_list)
      AND COLCARDF >= 0;

```

SYSTABSTATS and SYSCOLDIST require deletes and inserts.

Delete statistics from SYSTABSTATS on the test subsystem for the specified tables by using the following statement:

```

DELETE FROM (TEST_SUBSYSTEM).SYSTABSTATS
      WHERE OWNER IN (creator_list)
      AND   NAME IN (table_list)

```

Use INSERT statements to repopulate SYSTABSTATS with production statistics that are generated from the following statement:

```

SELECT 'INSERT INTO SYSIBM.SYSTABSTATS'
CONCAT '(CARD,NPAGES,PCTPAGES,NACTIVE,PCTROWCOMP'
CONCAT ',STATTIME,IBMREQD,DBNAME,TSNAME,PARTITION'
CONCAT ',OWNER,NAME,CARDF) VALUES('
CONCAT STRIP(CHAR(CARD))          CONCAT ' ',
CONCAT STRIP(CHAR(NPAGES))        CONCAT ' ',
CONCAT STRIP(CHAR(PCTPAGES))      CONCAT ' ',
CONCAT STRIP(CHAR(NACTIVE))       CONCAT ' ',
CONCAT STRIP(CHAR(PCTROWCOMP))    CONCAT ' ',
CONCAT '''' CONCAT CHAR(STATTIME) CONCAT ''',',
CONCAT '''' CONCAT IBMREQD        CONCAT ''',',
CONCAT '''' CONCAT STRIP(DBNAME)   CONCAT ''',',
CONCAT '''' CONCAT STRIP(TSNAME)   CONCAT ''',',
CONCAT STRIP(CHAR(PARTITION))     CONCAT ' ',
CONCAT '''' CONCAT STRIP(OWNER)    CONCAT ''',',
CONCAT '''' CONCAT STRIP(NAME)     CONCAT ''',',
CONCAT STRIP(CHAR(CARDF))         CONCAT ')*')
FROM SYSIBM.SYSTABSTATS
      WHERE OWNER IN (creator_list)
      AND   NAME IN (table_list);

```

#### PSPI

**GUIP** Delete statistics from SYSCOLDIST on the test subsystem for the specified tables by using the following statement:

```

DELETE FROM (TEST_SUBSYSTEM).SYSCOLDIST
      WHERE TBOWNER IN (creator_list)
      AND   TBNAME IN (table_list);

```

#### GUIP

**PSPI** Use INSERT statements to repopulate SYSCOLDIST with production statistics that are generated from the following statement:

```

SELECT 'INSERT INTO SYSIBM.SYSCOLDIST '
CONCAT '(FREQUENCY,STATTIME,IBMREQD,TBOWNER'
CONCAT ',TBNAME,NAME,COLVALUE,TYPE,CARDF,COLGROUPCOLNO'
CONCAT ',NUMCOLUMNS,FREQUENCYF) VALUES('
CONCAT STRIP(CHAR(FREQUENCY))      CONCAT ' ',
CONCAT '''' CONCAT CHAR(STATTIME)  CONCAT ''',',
CONCAT '''' CONCAT IBMREQD          CONCAT ''',',

```

```

CONCAT ' ' CONCAT STRIP(TBOWNER)          CONCAT ' ', '
CONCAT ' ' CONCAT STRIP(TBNAME)            CONCAT ' ', '
CONCAT ' ' CONCAT STRIP(NAME)              CONCAT ' ', '
CONCAT 'X' CONCAT STRIP(HEX(COLVALUE))     CONCAT ' ', '
CONCAT ' ' CONCAT TYPE                     CONCAT ' ', '
CONCAT STRIP(CHAR(CARDF))                  CONCAT ' ', '
CONCAT 'X' CONCAT STRIP(HEX(COLGROUPCOLNO)) CONCAT ' ', '
CONCAT CHAR(NUMCOLUMNS)                  CONCAT ' ', '
CONCAT STRIP(CHAR(FREQUENCYF))             CONCAT ')*'
FROM SYSIBM.SYSCOLDIST
      WHERE TBOWNER IN (creator_list)
      AND  TBNAME IN (table_list);

```

#### PSPI

#### Note about SPUI:

- If you use SPUI to execute the preceding SQL statements, you might need to increase the default maximum character column width to avoid truncation.
- Asterisks (\*) appear in the examples to avoid having the semicolon interpreted as the end of the SQL statement. Edit the result to change the asterisk to a semicolon.

**Access path differences from test to production:** When you bind applications on the test system with production statistics, access paths should be similar but still might be different to what you see when the same query is bound on your production system. The access paths from test to production could be different for the following possible reasons:

- The processor models are different.
- The number of processors are different. (Differences in the number of processors can affect the degree of parallelism that is obtained.)
- The buffer pool sizes are different.
- The RID pool sizes are different.
- Data in SYSIBM.SYSCOLDIST is mismatched. (This mismatch occurs only if some of the previously mentioned steps mentioned are not followed exactly).
- The service levels are different.
- The values of optimization subsystem parameters, such as STARJOIN, NPGTHRSH, and PARAMDEG (MAX DEGREE on installation panel DSNTIP4) are different.
- The use of techniques such as optimization hints and volatile tables are different.

If your production system is accessible from your test system, you can use DB2 PM EXPLAIN on your test system to request EXPLAIN information from your production system. This request can reduce the need to simulate a production system by updating the catalog.

You can also use the Visual Explain feature of IBM Optimization Service Center for DB2 for z/OS to display the current PLAN\_TABLE output or the graphed access paths for statements within any particular subsystem from your workstation environment. For example, if you have your test system on one subsystem and your production system on another subsystem, you can visually compare the PLAN\_TABLE outputs or access paths simultaneously with some window or view manipulation. You can then access the catalog statistics for certain referenced objects of an access path from either of the displayed instances of PLAN\_TABLE or access path graphs.

#### Related concepts

---

## Real-time statistics

The following topics provide detailed information about the real-time statistics tables.

### Setting up your system for real-time statistics

DB2 always generates in-memory statistics for each table space and index space in your system, including catalog objects. However, only certain global values are generated for the SYSRTS table space, and no incremental counters are maintained.

No statistics are generated for the directory. For partitioned spaces, DB2 generates information for each partition. However, you need to set the interval for writing statistics and establish base values for statistics.

### Setting the interval for writing real-time statistics

You can set the interval for writing real-time statistics when you install DB2, and you can subsequently update that interval online.

The installation field is REAL TIME STATS on panel DSNTIPO. In a data sharing environment, each member has its own interval for writing real-time statistics.

To update the interval:

Modify STATSINT system parameter. The default interval is 30 minutes.

### Establishing base values for real-time statistics

Many columns in the real-time statistics tables show the number of times an operation was performed between the last time a particular utility was run and when the real-time statistics are written.

For example, STATSINSERT in SYSTABLESPACESTATS indicates the number of records or LOBs that have been inserted after the last RUNSTATS utility was run on the table space or partition. Therefore, for each object for which you want real-time statistics, run the appropriate utility (REORG, RUNSTATS, LOAD REPLACE, REBUILD INDEX, or COPY) to establish a base value from which the delta value can be calculated.

## Contents of the real-time statistics tables

Real time statistics tables contain statistics for indexes and table spaces.

### Related information

“SYSIBM.SYSTABLESPACESTATS table” in (DB2 for z/OS SQL Reference)

“SYSIBM.SYSINDEXSPACESTATS table” (DB2 for z/OS SQL Reference)

## Operating with real-time statistics

To use the real-time statistics effectively, you need to understand when DB2 collects and externalizes them, and what factors in your system can affect the statistics.

### The DSNACCOX stored procedure

The DB2 real-time statistics stored procedure (DSNACCOX) is a sample stored procedure that makes recommendations to help you maintain your DB2 databases.

The DSNACCOX stored procedure requires DB2 Version 9.1 for z/OS new-function mode or later.

In particular, DSNACCOX performs these actions:

- Recommends when you should reorganize, image copy, or update statistics for table spaces or index spaces
- Indicates table spaces or index spaces that have exceeded their data set
- Indicates whether objects are in a restricted state

DSNACCOX uses data from the SYSIBM.SYSTABLESPACESTATS and SYSIBM.SYSSYSINDEXSPACESTATS real-time statistics tables to make its recommendations. DSNACCOX provides its recommendations in a result set.

DSNACCOX uses the set of criteria that are shown in “DSNACCOX formulas for recommending actions” on page 188 to evaluate table spaces and index spaces. By default, DSNACCOX evaluates **all** table spaces and index spaces in the subsystem that have entries in the real-time statistics tables. However, you can override this default through input parameters.

***Important information about DSNACCOX recommendations:***

- DSNACCOX makes recommendations based on general formulas that require input from the user about the maintenance policies for a subsystem. These recommendations might not be accurate for every installation.
- If the real-time statistics tables contain information for only a small percentage of your DB2 subsystem, the recommendations that DSNACCOX makes might not be accurate for the entire subsystem.
- Before you perform any action that DSNACCOX recommends, ensure that the object for which DSNACCOX makes the recommendation is available, and that the recommended action can be performed on that object. For example, before you can perform an image copy on an index, the index must have the COPY YES attribute.

## Environment for DSNACCOX

DSNACCOX must run in a WLM-established stored procedure address space.

DSNACCOX creates and uses declared temporary tables. Therefore, before you can invoke DSNACCOX, you need to create a TEMP database and segmented table spaces in the TEMP database.

You should bind the package for DSNACCOX with isolation UR to avoid lock contention. You can find the installation steps for DSNACCOX in job DSNTIJSG.

## Authorization required for DSNACCOX

To execute the CALL DSNACCOX statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for DSNACCOX
- Ownership of the package
- PACKADM authority for the package collection

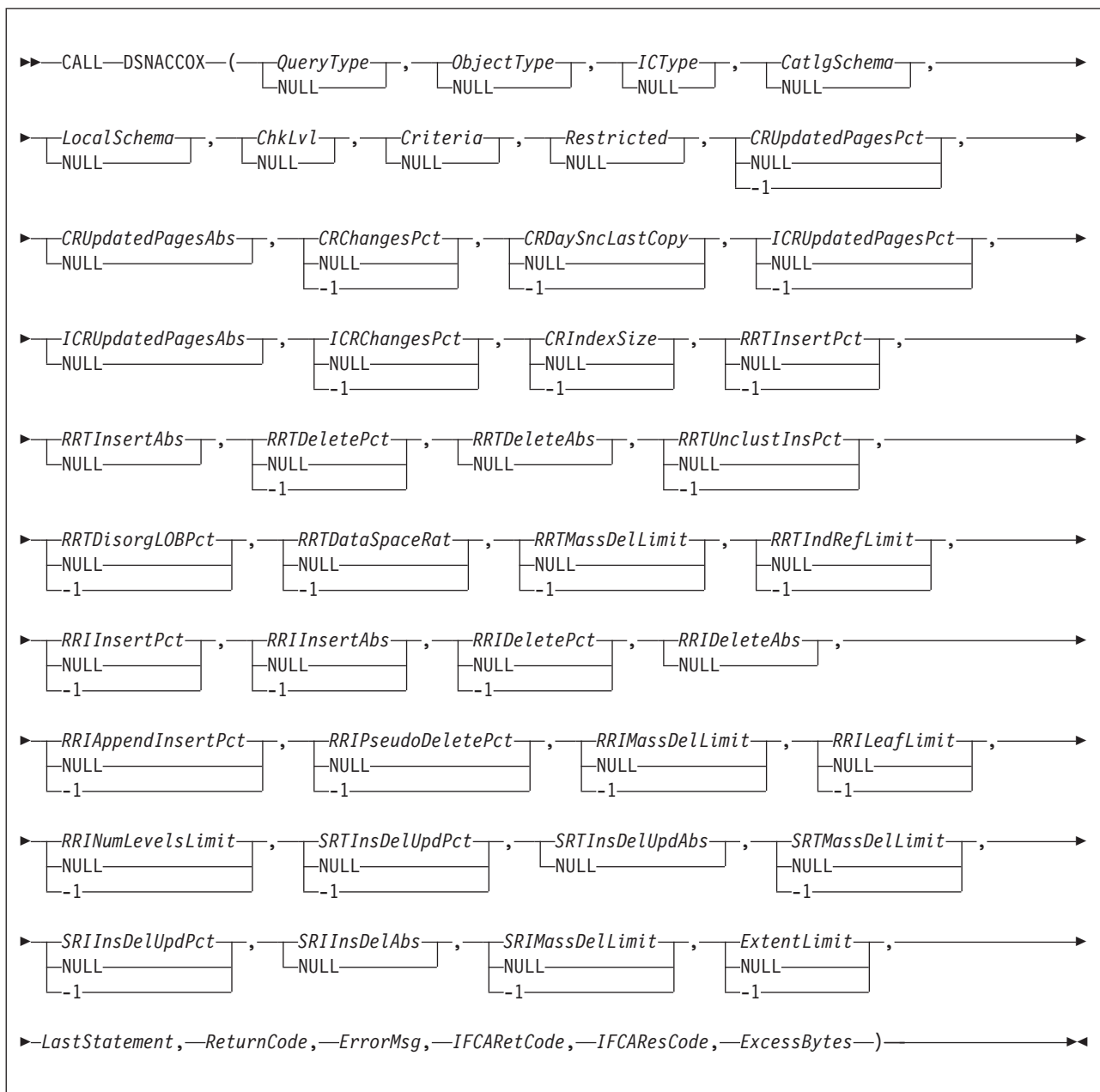
- SYSADM authority

The owner of the package or plan that contains the CALL statement must also have:

- SELECT authority on the real-time statistics tables
- The DISPLAY system privilege

### DSNACCOX syntax diagram

The following syntax diagram shows the CALL statement for invoking DSNACCOX. Because the linkage convention for DSNACCOX is GENERAL WITH NULLS, if you pass parameters in host variables, you need to include a null indicator with every host variable. Null indicators for input host variables must be initialized before you execute the CALL statement.



## DSNACCOX option descriptions

In the following option descriptions, the default value for an input parameter is the value that DSNACCOX uses if you specify a null value.

### *QueryType*

Specifies the types of actions that DSNACCOX recommends. This field contains one or more of the following values. Each value is enclosed in single quotation marks and separated from other values by a space.

**ALL** Makes recommendations for all of the following actions.

**COPY** Makes a recommendation on whether to perform an image copy.

#### **RUNSTATS**

Makes a recommendation on whether to perform RUNSTATS.

#### **REORG**

Makes a recommendation on whether to perform REORG. Choosing this value causes DSNACCOX to process the EXTENTS value also.

#### **EXTENTS**

Indicates when data sets have exceeded a user-specified extents limit.

#### **RESTRICT**

Indicates which objects are in a restricted state.

DSNACCOX recommends REORG on the table space when one of the following conditions is true, and REORG (or ALL) is also specified for the value of QUERYTYPE:

- The table space is in REORG-pending status.
- The table space is in advisory REORG-pending status as the result of an ALTER TABLE statement.

DSNACCOX recommends REORG on the index when on the following conditions is true and REORG (or ALL) is also specified for the value of QUERYTYPE::

- The index is in REORG-pending status.
- The index is in advisory REORG-pending as the result of an ALTER TABLE statement.

DSNACCOX recommends FULL COPY on the table space when on the following conditions is true and COPY (or ALL) is also specified for the value of QUERYTYPE::

- The table space is in COPY-pending status.
- The table space is in informational COPY-pending status.

DSNACCOX recommends FULL COPY on the index when on the following conditions is true and COPY (or ALL) is also specified for the value of QUERYTYPE: and SYSINDEX.COPY='Y':

- The index is in COPY-pending status.
- The index is in informational COPY-pending status.

*QueryType* is an input parameter of type VARCHAR(40). The default value is ALL.

### *ObjectType*

Specifies the types of objects for which DSNACCOX recommends actions:

**ALL** Table spaces and index spaces.

**TS** Table spaces only.

**IX** Index spaces only.

*ObjectType* is an input parameter of type VARCHAR(3). The default value is **ALL**.

#### *ICType*

Specifies the types of image copies for which DSNACCOX is to make recommendations:

**F** Full image copy.

**I** Incremental image copy. This value is valid for table spaces only.

**B** Full image copy or incremental image copy.

*ICType* is an input parameter of type VARCHAR(1). The default is **B**.

#### *CatlgSchema*

Specifies the qualifier for DB2 catalog table names. *CatlgSchema* is an input parameter of type VARCHAR(128). The default value is **SYSIBM**.

#### *LocalSchema*

Specifies the qualifier for the names of tables that DSNACCOX creates.

*LocalSchema* is an input parameter of type VARCHAR(128). The default value is **DSNACC**.

#### *ChkLvl*

Specifies the types of checking that DSNACCOX performs, and indicates whether to include objects that fail those checks in the DSNACCOX recommendations result set. This value is the sum of any combination of the following values:

**0** DSNACCOX performs none of the following actions.

**1** Exclude rows from the DSNACCOX recommendations result set for RUNSTATS on:

- Index spaces that are related to tables that are defined as VOLATILE.
- Table spaces for which all of the tables are defined as VOLATILE.

**2** Reserved for future use.

**4** Check whether rows that are in the DSNACCOX recommendations result set refer to objects that are in the exception table. For recommendations result set rows that have corresponding exception table rows, copy the contents of the QUERYTYPE column of the exception table to the INEXCEPTTABLE column of the recommendations result set.

**8** Check whether objects that have rows in the recommendations result set are restricted. Indicate the restricted status in the OBJECTSTATUS column of the result set.

**16** Reserved for future use.

**32** Exclude rows from the DSNACCOX recommendations result set for index spaces for which the related table spaces have been recommended for REORG.

**64** For index spaces that are listed in the DSNACCOX recommendations result set, check whether the related table spaces are listed in the exception table. For recommendations result set rows that have corresponding exception table rows, copy the contents of the

QUERYTYPE column of the exception table to the INEXCEPTTABLE column of the recommendations result set. Selecting CHKLVL64 also activates CHKLVLs 32 and 4.

*ChkLvl* is an input parameter of type INTEGER. The default is 5 (values 1+4).

#### *Criteria*

Narrows the set of objects for which DSNACCOX makes recommendations. This value is the search condition of an SQL WHERE clause. *Criteria* is an input parameter of type VARCHAR(4096). The default is that DSNACCOX makes recommendations for all table spaces and index spaces in the subsystem. The search condition can use any column in the result set and wildcards are allowed.

#### *Restricted*

A parameter that is reserved for future use. Specify the null value for this parameter. *Restricted* is an input parameter of type VARCHAR(80).

#### *CRUpdatedPagesPct*

Specifies, when combined with *CRUpdatedPagesAbs*, a criterion for recommending a full image copy on a table space or index space. If both of the following conditions are true for a table space, DSNACCOX recommends an image copy:

- The total number of distinct updated pages, divided by the total number of preformatted pages (expressed as a percentage) is greater than *CRUpdatedPagesPct*.
- The total number of distinct updates pages is greater than *CRUpdatedPagesABS*.

If all of the following conditions are true for an index space, DSNACCOX recommends an image copy:

- The total number of distinct updated pages, divided by the total number of preformatted pages (expressed as a percentage) is greater than *CRUpdatedPagesPct*.
- The total number of distinct updates pages is greater than *CRUpdatedPagesABS*.
- The number of active pages in the index space or partition is greater than *CRIndexSize*.

*CRUpdatedPagesPct* is an input parameter of type DOUBLE. The default is 20.0. A negative value turns off both this criteria and *CRUpdatedPagesABS*.

#### *CRUpdatedPagesABS*

Specifies, when combined with *CRUpdatedPagesPct*, a criterion for recommending a full image copy on a table space or index space. If both of the following conditions are true for a table space, DSNACCOX recommends an image copy:

- The total number of distinct updated pages, divided by the total number of preformatted pages (expressed as a percentage) is greater than *CRUpdatedPagesPct*.
- The total number of distinct updated pages is greater than *CRUpdatedPagesAbs*.

If all of the following conditions are true for an index space, DSNACCOX recommends an image copy:

- The total number of distinct updated pages, divided by the total number of preformatted pages (expressed as a percentage) is greater than *CRUpdatedPagesPct*.

- The total number of distinct updates pages is greater than *CRUpdatedPagesAbs*.
- The number of active pages in the index space or partition is greater than *CRIndexSize*.

*CRUpdatedPagesAbs* is an input parameter of type INTEGER. The default value is 0.

#### *CRChangesPct*

Specifies a criterion for recommending a full image copy on a table space or index space. If the following condition is true for a table space, DSNACCOX recommends an image copy:

The total number of insert, update, and delete operations since the last image copy, divided by the total number of rows or LOBs in a table space or partition (expressed as a percentage) is greater than *CRChangesPct*.

If both of the following conditions are true for an index table space, DSNACCOX recommends an image copy:

- The total number of insert and delete operations since the last image copy, divided by the total number of entries in the index space or partition (expressed as a percentage) is greater than *CRChangesPct*.
- The number of active pages in the index space or partition is greater than *CRIndexSize*.

*CRChangesPct* is an input parameter of type DOUBLE. The default is 10.0. A negative value turns off this criterion.

#### *CRDaySncLastCopy*

Specifies a criterion for recommending a full image copy on a table space or index space. If the number of days since the last image copy is greater than this value, DSNACCOX recommends an image copy.

*CRDaySncLastCopy* is an input parameter of type INTEGER. The default is 7. A negative value turns off this criterion.

#### *ICRUpdatedPagesPct*

Specifies a criterion for recommending an incremental image copy on a table space. If both of the following conditions are true, DSNACCOX recommends an incremental image copy:

- The number of distinct pages that were updated since the last image copy, divided by the total number of active pages in the table space or partition (expressed as a percentage) is greater than *ICRUpdatedPagesPct*.
- The number of distinct pages that were updated since last image copy is greater than *ICRUpdatedPagesAbs*.

*ICRUpdatedPagesPct* is an input parameter of type DOUBLE. The default value is 1.0. A negative value turns off this criterion and *ICRUpdatedPagesAbs*.

#### *ICRChangesPct*

Specifies a criterion for recommending an incremental image copy on a table space. If the following condition is true, DSNACCOX recommends an incremental image copy:

The ratio of the number of insert, update, or delete operations since the last image copy, to the total number of rows or LOBs in a table space or partition (expressed as a percentage) is greater than *ICRChangesPct*.

*ICRChangesPct* is an input parameter of type DOUBLE. The default is 1.0. A negative value turns off this criterion.

#### *CRIndexSize*

Specifies the minimum index size before checking the *ICRUpdatedPagesPct* or *ICRChangesPct* criteria for recommending a full image copy on an index space.

*CRIndexSize* is an input parameter of type INTEGER. The default is 50. A negative value turns off this criterion and *ICRChangesPct*.

#### *RRTInsertPct*

Specifies, when combined with *RRTInsertAbs*, a criterion for recommending that the REORG utility is to be run on a table space. If both of the following condition are true, DSNACCOX recommends running REORG:

- The sum of insert, update, and delete operations since the last REORG, divided by the total number of rows or in the table space or partition (expressed as a percentage) is greater than *RRTInsertPct*
- The sum of insert operations since the last REORG is greater than *RRTInsertAbs*.

*RRTInsertPct* is an input parameter of type DOUBLE. The default value is 25.0. A negative value turns off this criterion and *RRTInsertAbs*.

#### *RRTInsertAbs*

Specifies, when combined with *RRTInsertPct*, a criterion for recommending that the REORG utility is to be run on a table space. If both of the following condition are true, DSNACCOX recommends running REORG:

- The sum of insert operations since the last REORG, divided by the total number of rows or in the table space or partition (expressed as a percentage) is greater than *RRTInsertPct*
- The sum of insert operations since the last REORG is greater than *RRTInsertAbs*.

*RRTInsertAbs* is an input parameter of type INTEGER. The default value is 0.

#### *RRTDeletePct*

Specifies, when combined with *RRTDeleteAbs*, a criterion for recommending that the REORG utility is to be run on a table space. If both of the following condition are true, DSNACCOX recommends running REORG:

- The sum of delete operations since the last REORG, divided by the total number of rows or in the table space or partition (expressed as a percentage) is greater than *RRTDeletePct*
- The sum of insert operations since the last REORG is greater than *RRTDeleteAbs*.

*RRTInsertPct* is an input parameter of type DOUBLE. The default value is 25.0. A negative value turns off this criterion and *RRTDeleteAbs*.

#### *RRTDeleteAbs*

Specifies, when combined with *RRTDeletePct*, a criterion for recommending that the REORG utility is to be run on a table space. If both of the following condition are true, DSNACCOX recommends running REORG:

- The sum of delete operations since the last REORG, divided by the total number of rows or in the table space or partition (expressed as a percentage) is greater than *RRTDeletePct*
- The sum of delete operations since the last REORG is greater than *RRTDeleteAbs*.

*RRTDeleteAbs* is an input parameter of type INTEGER. The default value is 0.

#### *RRTUnclustInsPct*

Specifies a criterion for recommending that the REORG utility is to be run on a table space. If the following condition is true, DSNACCOX recommends running REORG:

The number of unclustered insert operations, divided by the total number of rows or LOBs in the table space or partition (expressed as a percentage) is greater than *RRTUnclustInsPct*.

*RRTUnclustInsPct* is an input parameter of type DOUBLE. The default is 10.0. A negative value will turn off this criterion.

#### *RRTDisorgLOBPct*

Specifies a criterion for recommending that the REORG utility is to be run on a table space. If the following condition is true, DSNACCOX recommends running REORG:

The number of imperfectly chunked LOBs, divided by the total number of rows or LOBs in the table space or partition (expressed as a percentage) is greater than *RRTDisorgLOBPct*.

*RRTDisorgLOBPct* is an input parameter of type INTEGER. The default is 50.0. A negative value will turn off this criterion.

#### *RRTDataSpaceRat*

Specifies a criterion for recommending that the REORG utility is to be run on table space for space reclamation. If the following condition is true, DSNACCOX recommends running REORG:

The SPACE allocated is greater than *RRTDataSpaceRat* multiplied by the actual space used. ( $\text{SPACE} > \text{RRTDataSpaceRat} \times (\text{DATASIZE}/1024)$ )

*RRTDataSpaceRat* is an input parameter of type DOUBLE. The default value is 2.0. A negative value turns off this criterion.

#### *RRTMassDelLimit*

Specifies a criterion for recommending that the REORG utility is to be run on a table space. If one of the following values is greater than *RRTMassDelLimit*, DSNACCOX recommends running REORG:

- The number of mass deletes from a segmented or LOB table space since the last REORG or LOAD REPLACE
- The number of dropped tables from a nonsegmented table space since the last REORG or LOAD REPLACE

*RRTMassDelLimit* is an input parameter of type INTEGER. The default is 0.

#### *RRTIndRefLimit*

Specifies a criterion for recommending that the REORG utility is to be run on a table space. If the following value is greater than *RRTIndRefLimit*, DSNACCOX recommends running REORG:

The total number of overflow records that were created since the last REORG or LOAD REPLACE, divided by the total number of rows or LOBs in the table space or partition (expressed as a percentage)

*RRTIndRefLimit* is an input parameter of type DOUBLE. The default is 5.0 in data sharing environment and 10.0 in a non-data sharing environment.

#### *RRIInsertPct*

Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the both of the following conditions are true, DSNACCOX recommends running REORG:

- The sum of the number of index entries that were inserted since the last REORG, divided by the total number of index entries in the index space or partition (expressed as a percentage) is greater than *RRInsertPct*.
- The sum of the number of index entries that were inserted since the last REORG is greater than *RRInsertAbs*.

*RRInsertPct* is an input parameter of type DOUBLE. The default is 30.0. A negative value turns off this criterion.

#### *RRInsertAbs*

Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the both of the following conditions are true, DSNACCOX recommends running REORG:

- The sum of the number of index entries that were inserted since the last REORG, divided by the total number of index entries in the index space or partition (expressed as a percentage) is greater than *RRInsertPct*.
- The sum of the number of index entries that were inserted since the last REORG is greater than *RRInsertAbs*.

*RRInsertAbs* is an input parameter of type INTEGER. The default is 0. A negative value turns off this criterion.

#### *RRDeletePct*

Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRDeletePct*, DSNACCOX recommends running REORG:

- The sum of the number of index entries that were inserted since the last REORG, divided by the total number of index entries in the index space or partition (expressed as a percentage) is greater than *RRDeletePct*.
- The sum of the number of index entries that were inserted since the last REORG is greater than *RRDeleteAbs*.

This is an input parameter of type DOUBLE. The default is 30.0. A negative value turns off this criterion.

#### *RRDeleteAbs*

Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRDeletePct*, DSNACCOX recommends running REORG:

- The sum of the number of index entries that were inserted since the last REORG, divided by the total number of index entries in the index space or partition (expressed as a percentage) is greater than *RRDeletePct*.
- The sum of the number of index entries that were inserted since the last REORG is greater than *RRDeleteAbs*.

This is an input parameter of type INTEGER. The default is 0. A negative value turns off this criterion.

#### *RRAppendInsertPct*

Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRAppendInsertPct*, DSNACCOX recommends running REORG:

The number of index entries that were inserted since the last REORG, REBUILD INDEX, or LOAD REPLACE with a key value greater than the maximum key value in the index space or partition, divided by the number of index entries in the index space or partition (expressed as a percentage)

*RRInsertDeletePct* is an input parameter of type DOUBLE. The default is 20.0. A negative value turns off this criterion.

#### *RRIPseudoDeletePct*

Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRIPseudoDeletePct*, DSNACCOX recommends running REORG:

The number of index entries that were pseudo-deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE, divided by the number of index entries in the index space or partition (expressed as a percentage)

*RRIPseudoDeletePct* is an input parameter of type DOUBLE. The default is 5.0 in data sharing and 10.0 in non data sharing environments. A negative value turns off this criterion.

#### *RRIMassDelLimit*

Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the number of mass deletes from an index space or partition since the last REORG, REBUILD, or LOAD REPLACE is greater than this value, DSNACCOX recommends running REORG.

*RRIMassDelLimit* is an input parameter of type INTEGER. The default is 0. A negative value turns off this criterion.

#### *RRLeafLimit*

Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRLeafLimit*, DSNACCOX recommends running REORG:

The number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE in which the higher part of the split page was far from the location of the original page, divided by the total number of active pages in the index space or partition (expressed as a percentage)

*RRLeafLimit* is an input parameter of type DOUBLE. The default is 10.0. A negative value turns off this criterion.

#### *RRNumLevelsLimit*

Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRNumLevelsLimit*, DSNACCOX recommends running REORG:

The number of levels in the index tree that were added or removed since the last REORG, REBUILD INDEX, or LOAD REPLACE

*RRNumLevelsLimit* is an input parameter of type INTEGER. The default is 0. A negative value turns off this criterion.

#### *SRTInsDelUpdPct*

Specifies, when combined with *SRTInsDelUpdAbs*, a criterion for recommending that the RUNSTATS utility is to be run on a table space. If both of the following conditions are true, DSNACCOX recommends running RUNSTATS:

- The number of insert, update, or delete operations since the last RUNSTATS on a table space or partition, divided by the total number of rows or LOBs in table space or partition (expressed as a percentage) is greater than *SRTInsDelUpdPct*.
- The sum of the number of inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRTInsDelUpdAbs*.

*SRTInsDelUpdPct* is an input parameter of type DOUBLE. The default is 20.0. A negative value turns off this criterion.

#### *SRTInsDelUpdAbs*

Specifies, when combined with *SRTInsDelUpdPct*, a criterion for recommending that the RUNSTATS utility is to be run on a table space. If both of the following conditions are true, DSNACCOX recommends running RUNSTATS:

- The number of insert, update, and delete operations since the last RUNSTATS on a table space or partition, divided by the total number of rows or LOBs in table space or partition (expressed as a percentage) is greater than *SRTInsDelUpdPct*.
- The sum of the number of inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRTInsDelUpdAbs*.

*SRTInsDelUpdAbs* is an input parameter of type INTEGER. The default is 0.

#### *SRTMassDelLimit*

Specifies a criterion for recommending that the RUNSTATS utility is to be run on a table space. If the following condition is true, DSNACCOX recommends running RUNSTATS:

- The number of mass deletes from a table space or partition since the last REORG or LOAD REPLACE is greater than *SRTMassDelLimit*.

*SRTMassDelLimit* is an input parameter of type DOUBLE. The default is 0.0. A negative value turns off this criterion.

#### *SRIInsDelPct*

Specifies, when combined with *SRIInsDelAbs*, a criterion for recommending that the RUNSTATS utility is to be run on an index space. If both of the following conditions are true, DSNACCOX recommends running RUNSTATS:

- The number of inserted and deleted index entries since the last RUNSTATS on an index space or partition, divided by the total number of index entries in the index space or partition (expressed as a percentage) is greater than *SRIInsDelUpdPct*.
- The sum of the number of inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRIInsDelUpdAbs*.

*SRIInsDelPct* is an input parameter of type DOUBLE. The default is 20.0. A negative value turns off this criterion.

#### *SRIInsDelAbs*

Specifies, when combined with *SRIInsDelPct*, specifies a criterion for recommending that the RUNSTATS utility is to be run on an index space. If the following condition is true, DSNACCOX recommends running RUNSTATS:

- The number of inserted and deleted index entries since the last RUNSTATS on an index space or partition, divided by the total number of index entries in the index space or partition (expressed as a percentage) is greater than *SRIInsDelUpdPct*.
- The sum of the number of inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRIInsDelUpdAbs*,

*SRIInsDelAbs* is an input parameter of type INTEGER. The default is 0.

#### *SRIMassDelLimit*

Specifies a criterion for recommending that the RUNSTATS utility is to be run on an index space. If the number of mass deletes from an index space or partition since the last REORG, REBUILD INDEX, or LOAD REPLACE is greater than this value, DSNACCOX recommends running RUNSTATS.

*SRIMassDelLimit* is an input parameter of type INTEGER. The default value is 0. A negative value turns off this criterion.

#### *ExtentLimit*

Specifies a criterion for recommending that the RUNSTATS or REORG utility is to be run on a table space or index space. Also specifies that DSNACCOX is to warn the user that the table space or index space has used too many extents. DSNACCOX recommends running RUNSTATS or REORG, and altering data set allocations if the following condition is true:

- The number of physical extents in the index space, table space, or partition is greater than *ExtentLimit*.

*ExtentLimit* is an input parameter of type INTEGER. The default value is 254. A negative value turns off this criterion.

#### *LastStatement*

When DSNACCOX returns a severe error (return code 12), this field contains the SQL statement that was executing when the error occurred. *LastStatement* is an output parameter of type VARCHAR(8012).

#### *ReturnCode*

The return code from DSNACCOX execution. Possible values are:

- |             |  |
|-------------|--|
| <b>0</b>    | DSNACCOX executed successfully.  |
| <b>4</b>    | DSNACCOX completed with a warning. The <i>ErrorMsg</i> parameter contains the input parameters that might be incompatible.   |
| <b>8</b>    | DSNACCOX terminated with errors. The <i>ErrorMsg</i> parameter contains a message that describes the error.  |
| <b>12</b>   | DSNACCOX terminated with severe errors. The <i>ErrorMsg</i> parameter contains a message that describes the error. The <i>LastStatement</i> parameter contains the SQL statement that was executing when the error occurred. |
| <b>14</b>   | DSNACCOX terminated because the real time statistics table were not yet migrated to the catalog.   |
| <b>15</b>   | DSNACCOX terminated because it encountered a problem with one of the declared temporary tables that it defines and uses.   |
| <b>16</b>   | DSNACCOX terminated because it could not define a declared temporary table.  |
| <b>NULL</b> | DSNACCOX terminated but could not set a return code.   |

*ReturnCode* is an output parameter of type INTEGER.

#### *ErrorMsg*

Contains information about DSNACCOX execution when DSNACCOX terminates with a non-zero value for *ReturnCode*.

#### *IFCARetCode*

Contains the return code from an IFI COMMAND call. DSNACCOX issues commands through the IFI interface to determine the status of objects. *IFCARetCode* is an output parameter of type INTEGER.

#### *IFCAResCode*

Contains the reason code from an IFI COMMAND call. *IFCAResCode* is an output parameter of type INTEGER.

### *ExcessBytes*

Contains the number of bytes of information that did not fit in the IFI return area after an IFI COMMAND call. *ExcessBytes* is an output parameter of type INTEGER.

## **DSNACCOX formulas for recommending actions**

The following formulas specify the criteria that DSNACCOX uses for its recommendations and warnings. The variables in italics are DSNACCOX input parameters. The capitalized variables are columns of the SYSIBM.SYSTABLESPACESTATS or SYSIBM.SYSINDEXSPACESTATS tables. The numbers to the right of selected items are reference numbers for the option descriptions in “DSNACCOX option descriptions” on page 178.

The figure below shows the formula that DSNACCOX uses to recommend a full image copy on a table space.

---

```
((QueryType='COPY' OR QueryType='ALL') AND
(ObjectType='TS' OR ObjectType='ALL') AND
(ICType='F' OR ICType='B')) AND
(COPYLASTTIME IS NULL OR
 REORGLASTTIME>COPYLASTTIME OR
 LOADRLASTTIME>COPYLASTTIME OR
 (CURRENT DATE-COPYLASTTIME)>CRDaySncLastCopy OR
 ((COPYUPDATEDPAGES×100)/NACTIVE>CRUpdatedPagesPct AND
 (COPYUPDATEDPAGES>CRUpdatedPagesAbs)) OR
 ((QueryType='RESTRICT' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL') AND
 The table space is in advisory or informational REORG pending status))
```

---

Figure 14. DSNACCOX formula for recommending a full image copy on a table space

The figure below shows the formula that DSNACCOX uses to recommend a full image copy on an index space.

---

```
((QueryType='COPY' OR QueryType='ALL') AND
(ObjectType='IX' OR ObjectType='ALL') AND
(ICType='F' OR ICType='B')) AND
(SYSINDEXES.COPY = 'Y')) AND
(COPYLASTTIME IS NULL OR
 REORGLASTTIME>COPYLASTTIME OR
 LOADRLASTTIME>COPYLASTTIME OR
 REBUILDLASTTIME>COPYLASTTIME OR
 (CURRENT DATE-COPYLASTTIME)>CRDaySncLastCopy OR
 (NACTIVE>CRIndexSize AND
 (((COPYUPDATEDPAGES×100)/NACTIVE>CRUpdatedPagesPct) AND
 (COPYUPDATEDPAGES>CRUpdatedPagesAbs)) OR
 (COPYCHANGES×100)/TOTALENTRIES>CRChangesPct)) OR
 ((QueryType='RESTRICT' OR QueryType='ALL') AND
 (ObjectType='IX' OR ObjectType='ALL') AND
 (SYSINDEXES.COPY = 'Y') AND
 The index space is in copy or informational copy pending status))
```

---

Figure 15. DSNACCOX formula for recommending a full image copy on an index space

The figure below shows the formula that DSNACCOX uses to recommend an incremental image copy on a table space.

---

```

((QueryType='COPY' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL') AND
 (ICType='I') AND
 COPYLASTTIME IS NOT NULL) AND
 (LOADRLASTTIME>COPYLASTTIME OR
 REORGLASTTIME>COPYLASTTIME OR
 ((COPYUPDATEDPAGES×100)/NACTIVE>ICRUpdatedPagesPct) AND
 (COPYUPDATEDPAGES>ICRUpdatedPagesAbs)) OR
 (COPYCHANGES×100)/TOTALROWS>ICRChangesPct)

```

---

Figure 16. DSNACCOX formula for recommending an incremental image copy on a table space

The figure below shows the formula that DSNACCOX uses to recommend a REORG on a table space. If the table space is a LOB table space, and CHCKLVL=1, the formula does not include EXTENTS>ExtentLimit.

---

```

(((QueryType='REORG' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL')) AND
 (REORGLASTTIME IS NULL AND LOADRLASTTIME IS NULL) OR
 (NACTIVE IS NULL OR NACTIVE > 5) AND
 (((REORGINSERTS×100)/TOTALROWS>RRTInsertPct) AND
 REORGINSERTS>RRTInsertAbs) OR
 (((REORGDELETE×100)/TOTALROWS>RRTDeletePct) AND
 REORGDELETE>RRTDeleteAbs) OR
 (REORGUNCLUSTINS×100)/TOTALROWS>RRTUnclustInsPct OR
 (REORGDISORGL0B×100)/TOTALROWS>RRTDisorgLOBPct OR
 (SPACE×1024)/DATASIZE>RRTDataSpaceRat OR
 ((REORGNEARINDREF+REORGFARINDREF)×100)/TOTALROWS>RRTIndRefLimit OR
 REORGMASDELETE>RRTMassDelLimit OR
 EXTENTS>ExtentLimit)) OR
 (QueryType='RESTRICT' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL') AND
 The table space is in advisory or informational reorg pending status))

```

---

Figure 17. DSNACCOX formula for recommending a REORG on a table space

The figure below shows the formula that DSNACCOX uses to recommend a REORG on an index space.

---

```
((QueryType='REORG' OR QueryType='ALL') AND
(ObjectType='IX' OR ObjectType='ALL') AND
(REORGLASTTIME IS NULL AND REBUILDLASTTIME IS NULL) OR
(NACTIVE IS NULL OR NACTIVE > 5) AND
(((REORGINSERTS×100)/TOTALENTRIES>RRIInsertPct) AND
 REORGINSERTS>RRIInsertAbs) OR
(((REORGDELETE×100)/TOTALENTRIES>RRIDeletePct) AND
 REORGDELETE>RRIDeleteAbs) OR
(REORGAPPENDINSERT×100)/TOTALENTRIES>RRIAppendInsertPct OR
(REORGSEUDODELETES×100)/TOTALENTRIES>RRIPseudoDeletePct OR
REORMASSDELETE>RRIMassDeleteLimit OR
(REORGLAFAFFAR×100)/NACTIVE>RRILeafLimit OR
REORGLAFAFFAR>RRINumLevelsLimit OR
EXTENTS>ExtentLimit)) OR
(QueryType='RESTRICT' OR QueryType='ALL') AND
(ObjectType='IX' OR ObjectType='ALL') AND
An index is in advisory-REBUILD-pending stats (ARBBDP)))
```

---

Figure 18. DSNACCOX formula for recommending a REORG on an index space

The figure below shows the formula that DSNACCOX uses to recommend RUNSTATS on a table space.

---

```
((QueryType='RUNSTATS' OR QueryType='ALL') AND
(ObjectType='TS' OR ObjectType='ALL') AND
Table Space is not cloned) AND
(STATSLASTTIME IS NULL OR
STATSLASTTIME>LOADRLASTTIME OR
STATSLASTTIME>REORGLASTTIME OR
(((STATSINSERTS+STATSDELETES+STATSUPDATES)×100)/TOTALROWS>SRTInsDelUpdPct AND
 (STATSINSERTS+STATSDELETES+STATSUPDATES)>SRTInsDelUpdAbs) OR
STATSMASDELETE>SRTMassDeleteLimit)))
```

---

Figure 19. DSNACCOX formula for recommending RUNSTATS on a table space

The figure below shows the formula that DSNACCOX uses to recommend RUNSTATS on an index space.

---

```
((QueryType='RUNSTATS' OR QueryType='ALL') AND
(ObjectType='IX' OR ObjectType='ALL')
Table Space for the index is not cloned ) AND
(STATSLASTTIME IS NULL OR
STATSLASTTIME>LOADRLASTTIME OR
STATSLASTTIME>REORGLASTTIME OR
(((STATSINSERTS+STATSDELETES)×100)/TOTALENTRIES>SRIInsDelUpdPct AND
 (STATSINSERTS+STATSDELETES)>SRIInsDelAbs) OR
STATSMASDELETE>SRIInsDelAbs)))
```

---

Figure 20. DSNACCOX formula for recommending RUNSTATS on an index space

### Using an exception table

An exception table is an optional, user-created DB2 table that you can use to place information in the INEXCEPTTABLE column of the recommendations result set. You can put any information in the INEXCEPTTABLE column, but the most

common use of this column is to filter the recommendations result set. Each row in the exception table represents an object for which you want to provide information for the recommendations result set.

To create the exception table, execute a CREATE TABLE statement similar to the following one. You can include other columns in the exception table, but you must include at least the columns that are shown.

```
CREATE TABLE DSNACC.EXCEPT_TBL
  (DBNAME CHAR(8) NOT NULL,
   NAME CHAR(8) NOT NULL,
   QUERYTYPE CHAR(40))
CCSID EBCDIC;
```

The meanings of the columns are:

#### **DBNAME**

The database name for an object in the exception table.

#### **NAME**

The table space name or index space name for an object in the exception table.

#### **QUERYTYPE**

The information that you want to place in the INEXCEPTTABLE column of the recommendations result set.

If you put a null value in this column, DSNACCOX puts the value YES in the INEXCEPTTABLE column of the recommendations result set row for the object that matches the DBNAME and NAME values.

**Recommendation:** If you plan to put many rows in the exception table, create a nonunique index on DBNAME, NAME, and QUERYTYPE.

After you create the exception table, insert a row for each object for which you want to include information in the INEXCEPTTABLE column.

**Example:** Suppose that you want the INEXCEPTTABLE column to contain the string 'IRRELEVANT' for table space STAFF in database DSNDB04. You also want the INEXCEPTTABLE column to contain 'CURRENT' for table space DSN8S91D in database DSN8D91A. Execute these INSERT statements:

```
INSERT INTO DSNACC.EXCEPT_TBL VALUES('DSNDB04 ', 'STAFF ', 'IRRELEVANT');
INSERT INTO DSNACC.EXCEPT_TBL VALUES('DSN8D91A', 'DSN8S91D', 'CURRENT');
```

To use the contents of INEXCEPTTABLE for filtering, include a condition that involves the INEXCEPTTABLE column in the search condition that you specify in your *Criteria* input parameter.

**Example:** Suppose that you want to include all rows for database DSNDB04 in the recommendations result set, except for those rows that contain the string 'IRRELEVANT' in the INEXCEPTTABLE column. You might include the following search condition in your *Criteria* input parameter:

```
DBNAME='DSNDB04' AND INEXCEPTTABLE<>'IRRELEVANT'
```

### **DSNACCOX output**

If DSNACCOX executes successfully, in addition to the output parameters described in “DSNACCOX option descriptions” on page 178, DSNACCOX returns two result sets.

The first result set contains the results from IFI COMMAND calls that DSNACCOX makes. The following table shows the format of the first result set.

*Table 31. Result set row for first DSNACCOX result set*

Column name	Data type	Contents
RS_SEQUENCE	INTEGER	Sequence number of the output line
RS_DATA	CHAR(80)	A line of command output

The second result set contains DSNACCOX's recommendations. This result set contains one or more rows for a table space or index space. A nonpartitioned table space or nonpartitioning index space can have at most one row in the result set. A partitioned table space or partitioning index space can have at most one row for each partition. A table space, index space, or partition has a row in the result set if both of the following conditions are true:

- If the *Criteria* input parameter contains a search condition, the search condition is true for the table space, index space, or partition.
- DSNACCOX recommends at least one action for the table space, index space, or partition.

The following table shows the columns of a result set row.

*Table 32. Result set row for second DSNACCOX result set*

Column name	Data type	Description
DBNAME	VARCHAR(24)	Name of the database that contains the object.
NAME	VARCHAR(128)	Table space or index space name.
PARTITION	INTEGER	Data set number or partition number.
INSTANCE	INTEGER	Indicates if the object is associated with a data set instance.
CLONE	CHAR(1)	'Y' or 'N', 'Y' indicates a cloned object.
OBJECTTYPE	CHAR(2)	DB2 object type: <ul style="list-style-type: none"> <li>• 'TS' for a table space</li> <li>• 'IX' for an index space</li> </ul>
OBJECTSTATUS	CHAR(36)	Status of the object: <ul style="list-style-type: none"> <li>• ORPHANED, if the object is an index space with no corresponding table space, or if the object does not exist</li> <li>• If the object is in a restricted state, one of the following values: <ul style="list-style-type: none"> <li>– TS=restricted-state, if OBJECTTYPE is TS</li> <li>– IX=restricted-state, if OBJECTTYPE is IX</li> </ul> <i>restricted-state</i> is one of the status codes that appear in DISPLAY DATABASE output. See <i>DB2 Command Reference</i> for details.</li> <li>• A, if the object is in an advisory state.</li> <li>• L, if the object is a logical partition, but not in an advisory state.</li> <li>• AL, if the object is a logical partition and in an advisory state.</li> </ul>

Table 32. Result set row for second DSNACCOX result set (continued)

Column name	Data type	Description
IMAGECOPY	CHAR(4)	COPY recommendation: <ul style="list-style-type: none"> <li>• If OBJECTTYPE is TS: FULL (full image copy), INC (incremental image copy), or NO</li> <li>• If OBJECTTYPE is IX: YES or NO</li> </ul>
RUNSTATS	CHAR(3)	RUNSTATS recommendation: YES or NO.
EXTENTS	CHAR(3)	Indicates whether the data sets for the object have exceeded <i>ExtentLimit</i> : YES or NO.
REORG	CHAR(3)	REORG recommendation: YES or NO.
INEXCEPTTABLE	CHAR(40)	A string that contains one of the following values: <ul style="list-style-type: none"> <li>• Text that you specify in the QUERYTYPE column of the exception table.</li> <li>• YES, if you put a row in the exception table for the object that this result set row represents, but you specify NULL in the QUERYTYPE column.</li> <li>• NO, if the exception table exists but does not have a row for the object that this result set row represents.</li> <li>• Null, if the exception table does not exist, or if the <i>ChkLvl</i> input parameter does not include the value 4.</li> </ul>
ASSOCIATEDTS	CHAR(8)	If OBJECTTYPE is IX this value is the name of the table space that is associated with the index space. Otherwise null.
COPYLASTTIME	TIMESTAMP	Timestamp of the last full image copy on the object. Null if COPY was never run, or if the last COPY execution is unknown.
LOADRLASTTIME	TIMESTAMP	Timestamp of the last LOAD REPLACE on the object. Null if LOAD REPLACE was never run, or if the last LOAD REPLACE execution is unknown.
REBUILDLASTTIME	TIMESTAMP	Timestamp of the last REBUILD INDEX on the object. Null if REBUILD INDEX was never run, or if the last REBUILD INDEX execution is unknown.
CRUPDPGSPCT	DOUBLE	IF OBJECTTYPE is TS or IX and IMAGECOPY is YES, the ratio of distinct updated pages to pre-formatted pages, expressed as a percentage. Otherwise null.  If the ratio of distinct updated pages to pre-formatted pages, does not exceed the <i>CRUpdatedPagesPct</i> (for table spaces) or <i>ICRUpdatedPagesPct</i> (for indexes), this value is null.
CRUPDPGSABS	INTEGER	IF OBJECTTYPE is TS or IX and IMAGECOPY is YES, the ratio of distinct updated pages to pre-formatted pages. Otherwise null.  If the ratio of distinct updated pages to pre-formatted pages, does not exceed the value specified for <i>CRUpdatedPagesPct</i> (for table spaces) or <i>ICRUpdatedPagesPct</i> (for indexes), this value is null.

Table 32. Result set row for second DSNACCOX result set (continued)

Column name	Data type	Description
CRCPYCHGPCT	INTEGER	<p>If OBJECTTYPE is TS and IMAGECOPY is YES, the ratio of the total number insert, update, and delete operations since the last image copy to the total number of rows or LOBs in the table space or partition, expressed as a percentage. If OBJECTTYPE is IX and IMAGECOPY is YES, the ratio of the total number of insert and delete operations since the last image copy to the total number of entries in the index space or partition, expressed as a percentage. Otherwise null.</p> <p>If the the ratio of the total number insert, update, and delete operations since the last image copy to the total number of rows or LOBs in the table space or partition does not exceed the value specified for <i>CRChangesPct</i> (for table spaces) or <i>ICRChangesPct</i> (for index spaces), this value is null.</p>
CRDAYSELSTCPY	INTEGER	<p>If OBJECTTYPE is TS or IX and IMAGECOPY is YES, the number of days since the last image copy. Otherwise null.</p> <p>If the number of days since the last image copy does not exceed the value specified for <i>CrDaySncLastCopy</i>, this value is null.</p>
CRINDEXSIZE	INTEGER	<p>If OBJECTTYPE is IX and IMAGECOPY is YES, the number of active pages in the index space or partition. Otherwise null.</p> <p>If the the number of active pages in the index space or partition does not exceed the value specified for <i>CRIndexSize</i>, this value is null.</p>
REORGLASTTIME	TIMESTAMP	Timestamp of the last REORG on the object. Null if REORG was never run, or if the last REORG execution was terminated.
RRTINSERTSPCT	DOUBLE	<p>If OBJECTTYPE is TS and REORG is YES, the ratio of the sum of insert, update, and delete operations since the last REORG to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.</p> <p>If the the ratio of the sum of insert, update, and delete operations since the last REORG to the total number of rows or LOBs in the table space or partition does not exceed the value specified for <i>RRTInsertPct</i></p>
RRTINSERTSABS	INTEGER	<p>If OBJECTTYPE is TS and REORG is YES, the ratio of the sum of insert operations since the last REORG to the total number of rows in the table space or partition. Otherwise null.</p> <p>If the the ratio of the sum of insert operations since the last REORG to the total number of rows in the table space or partition does not exceed the value specified for <i>RRTInsertAbs</i>, this value is null.</p>

Table 32. Result set row for second DSNACCOX result set (continued)

Column name	Data type	Description
RRTDELETESPCT	DOUBLE	<p>If the OBJECTTYPE is TS and REORG is YES, the ratio of the sum of delete operations since the last REORG to the total number of rows in the table space or partition, expressed as a percentage. Otherwise null.</p> <p>If the ratio of the sum of delete operations since the last REORG to the total number of rows in the table space or partition does not exceed the value specified for <i>RRTDeletePct</i>, this value is null.</p>
RRTDELETESABS	INTEGER	<p>If OBJECTTYPE is TS and REORG is YES, the total number of delete operations since the last REORG on a table space or partition. Otherwise null.</p> <p>If the the total number of delete operations since the last REORG does not exceed the value specified for <i>RRTDeleteAbs</i>, this value is null.</p>
RRTUNCINSPCT	DOUBLE	<p>If OBJECTTYPE is TS and REORG is YES, the ratio of the number of unclustered insert operations to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.</p> <p>If the the ratio of the number of unclustered insert operations to the total number of rows or LOBs does not exceed the value specifeid for <i>RRTUnclustInsPct</i>, this value is null.</p>
RRTDISORGLOBPCT	DOUBLE	<p>If OBJECTTYPE is TS and REORG is YES, the ratio of the number of imperfectly chunked LOBs to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.</p> <p>If the the ratio of the number of imperfectly chunked LOBs to the total number of rows or LOBs in the table space or partition does not exceed the value of <i>RRTDisorgLOBPct</i>, this value is null</p>
RRTMASSDELETE	INTEGER	<p>If OBJECTTYPE is TS, REORG is YES, and the table space is a segmented table space or LOB table space, the number of mass deletes since the last REORG or LOAD REPLACE. If OBJECTTYPE is TS, REORG is YES, and the table space is nonsegmented, the number of dropped tables since the last REORG or LOAD REPLACE. Otherwise null.</p> <p>If the number of the number of dropped tables since the last REORG or LOAD REPLACE does not exceed the value specified for <i>RRTMassDelLimit</i>, this value is null.</p>

Table 32. Result set row for second DSNACCOX result set (continued)

Column name	Data type	Description
RRTINDREF	INTEGER	<p>If OBJECTTYPE is TS, REORG is YES, the ratio of the total number of overflow records that were created since the last REORG or LOAD REPLACE to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.</p> <p>If the ratio of the total number of overflow records that were created since the last REORG or LOAD REPLACE to the total number of rows or LOBs does not exceed the value specified for <i>RRTIndRef</i>, this value is null.</p>
RRIINSERTPCT	DOUBLE	<p>If OBJECTTYPE is IX and REORG is YES, the ratio of the total number of insert operations since the last REORG to the total number of index entries in the index space or partition, expressed as a percentage. Otherwise null.</p> <p>If the ratio of the total number of insert operations since the last REORG to the total number of index entries does not exceed the value specified for <i>RRIInsertPct</i>, this value is null.</p>
RRIINSERTABS	INTEGER	<p>If OBJECTTYPE is IX and REORG is YES, the total number of insert operations since the last REORG. Otherwise null.</p> <p>If the total number of insert operations since the last REORG does not exceed the value specified for <i>RRIInsertAbs</i>, this value is null.</p>
RRIDELETEPCT	DOUBLE	<p>If OBJECTTYPE is IX and REORG is YES, the ratio of the total number of delete operations since the last REORG to the total number of index entries in the index space or partition, expressed as a percentage. Otherwise null.</p> <p>If the ratio of the total number of delete operations since the last REORG to the total number of index entries does not exceed the value specified for <i>RRIDeletePct</i>, this value is null.</p>
RRIDELETEABS	INTEGER	<p>If OBJECTTYPE is IX and REORG is YES, the total number of delete operations since the last REORG. Otherwise null.</p> <p>If the total number of delete operations since the last REORG does not exceed the value specified for <i>RRIDeleteAbs</i>, this value is null.</p>

Table 32. Result set row for second DSNACCOX result set (continued)

Column name	Data type	Description
RRIAPPINSPCT	DOUBLE	<p>If OBJECTTYPE is IX and REORG is YES, the ratio of the number of index entries that were inserted since the last REORG, REBUILD INDEX, or LOAD REPLACE that had a key value greater than the maximum key value in the index space or partition, to the number of index entries in the index space or partition, expressed as a percentage. Otherwise null.</p> <p>If the ratio of the number of index entries that were inserted, which had a key value greater than the maximum key value, to the number of index entries does not exceed the value specified for <i>RRIAppendInsertPct</i>, this value is null.</p>
RRIPSDDELPCT	INTEGER	<p>If OBJECTTYPE is IX and REORG is YES, the ratio of the number of index entries that were pseudo-deleted (the RID entry was marked as deleted) since the last REORG, REBUILD INDEX, or LOAD REPLACE to the number of index entries in the index space or partition, expressed as a percentage. Otherwise null.</p> <p>If the ratio of the number of index entries that were pseudo-deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE to the number of index entries does not exceed the value specified for <i>RRIPseudoDeletePct</i>, this value is null.</p>
RRIMASSDELETE	INTEGER	<p>If OBJECTTYPE is IX and REORG is YES, the number of mass deletes from the index space or partition since the last REORG, REBUILD, or LOAD REPLACE. Otherwise null.</p> <p>If the number of mass deletes from the index space or partition since the last REORG, REBUILD, or LOAD REPLACE does not exceed the value specified for <i>RRIMassDelLimit</i>, this value is null.</p>
RRILEAF	INTEGER	<p>If OBJECTTYPE is IX and REORG is YES, the ratio of the number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE in which the higher part of the split page was far from the location of the original page, to the total number of active pages in the index space or partition, expressed as a percentage. Otherwise null.</p> <p>If the ratio of the number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE to the total number of active pages does not exceed the value specified for <i>RRILeafLimit</i>, this value is null.</p>
RRINUMLEVELS	INTEGER	<p>If OBJECTTYPE is IX and REORG is YES, the number of levels in the index tree that were added or removed since the last REORG, REBUILD INDEX, or LOAD REPLACE. Otherwise null.</p> <p>If the number of levels in the index tree that were added or removed does not exceed the value specified for <i>RRINumLevelsLimit</i>, this value is null.</p>

Table 32. Result set row for second DSNACCOX result set (continued)

Column name	Data type	Description
STATSLASTTIME	TIMESTAMP	Timestamp of the last RUNSTATS on the object. Null if RUNSTATS was never run, or if the last RUNSTATS execution was unknown.
SRTINSDELUPDPCT	DOUBLE	<p>If OBJECTTYPE is TS and RUNSTATS is YES, the ratio of the total number of insert, update, and delete operations since the last RUNSTATS on a table space or partition, to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.</p> <p>If the ratio of the total number of insert, update, and delete operations since the last RUNSTATS to the total number of rows or LOBs does not exceed the value specified for <i>SRTInsDelUpdPct</i>, this value is null.</p>
SRTINSDELUPDABS	INTEGER	<p>If OBJECTTYPE is TS and RUNSTATS is YES, the total number of insert, update, and delete operations since the last RUNSTATS on a table space or partition. Otherwise null.</p> <p>If the total number of insert, update, and delete operations since the last RUNSTATS does not exceed the value specified for <i>SRTInsDelUpdAbs</i>, this value is null.</p>
SRTMASSDELETE	INTEGER	<p>If OBJECTTYPE is TS and RUNSTATS is YES, the number of mass deletes from the table space or partition since the last REORG or LOAD REPLACE. Otherwise null.</p> <p>If the number of mass deletes from the table space or partition since the last REORG or LOAD REPLACE does not exceed the value specified for <i>SRTMassDelLimit</i>, this value is null.</p>
SRIINSDELPCT	DOUBLE	<p>If OBJECTTYPE is IX and RUNSTATS is YES, the ratio of the total number of insert and delete operations since the last RUNSTATS on the index space or partition, to the total number of index entries in the index space or partition, expressed as a percentage. Otherwise null.</p> <p>If the ratio of the total number of insert and delete operations since the last RUNSTATS, to the total number of index entries does not exceed the value specified for <i>SRIInsDelPct</i>, this value is null.</p>
SRIINSDELABS	INTEGER	<p>If OBJECTTYPE is IX and RUNSTATS is YES, the number insert and delete operations since the last RUNSTATS on the index space or partition. Otherwise null.</p> <p>does not exceed the value specified for , this value is null.</p>

Table 32. Result set row for second DSNACCOX result set (continued)

Column name	Data type	Description
SRIMASSDELETE	INTEGER	<p>If OBJECTTYPE is IX and RUNSTATS is YES, the number of mass deletes from the index space or partition since the last REORG, REBUILD INDEX, or LOAD REPLACE. Otherwise, this value is null.</p> <p>If the number of mass deletes does not exceed the value specified for <i>SRIMassDelete</i>, this value is null.</p>
TOTALEXTENTS	SMALLINT	<p>If EXTENTS is YES, the number of physical extents in the table space, index space, or partition. Otherwise, this value is null.</p> <p>If the number of physical extents does not exceed the value specified for <i>ExtentLimit</i>, this value is null.</p>

### PSPI

## The DSNACCOR stored procedure

The DB2 real-time statistics stored procedure (DSNACCOR) is a sample stored procedure that makes recommendations to help you maintain your DB2 databases.

### PSPI

In particular, DSNACCOR performs these actions:

- Recommends when you should reorganize, image copy, or update statistics for table spaces or index spaces
- Indicates table spaces or index spaces that have exceeded their data set
- Indicates whether objects are in a restricted state

DSNACCOR uses data from the SYSIBM.SYSTABLESPACESTATS and SYSIBM.SYSSYSINDEXSPACESTATS real-time statistics tables to make its recommendations. DSNACCOR provides its recommendations in a result set.

DSNACCOR uses the set of criteria that are shown in “DSNACCOR formulas for recommending actions” on page 209 to evaluate table spaces and index spaces. By default, DSNACCOR evaluates **all** table spaces and index spaces in the subsystem that have entries in the real-time statistics tables. However, you can override this default through input parameters.

### Important information about DSNACCOR recommendations:

- DSNACCOR makes recommendations based on general formulas that require input from the user about the maintenance policies for a subsystem. These recommendations might not be accurate for every installation.
- If the real-time statistics tables contain information for only a small percentage of your DB2 subsystem, the recommendations that DSNACCOR makes might not be accurate for the entire subsystem.
- Before you perform any action that DSNACCOR recommends, ensure that the object for which DSNACCOR makes the recommendation is available, and that the recommended action can be performed on that object. For example, before you can perform an image copy on an index, the index must have the COPY YES attribute.

## Environment for DSNACCOR

DSNACCOR must run in a WLM-established stored procedure address space.

DSNACCOR creates and uses declared temporary tables. Therefore, before you can invoke DSNACCOR, you need to create a TEMP database and segmented table spaces in the TEMP database. .

You should bind the package for DSNACCOR with isolation UR to avoid lock contention. You can find the installation steps for DSNACCOR in job DSNTIJSJG.

## Authorization required for DSNACCOR

To execute the CALL DSNACCOR statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

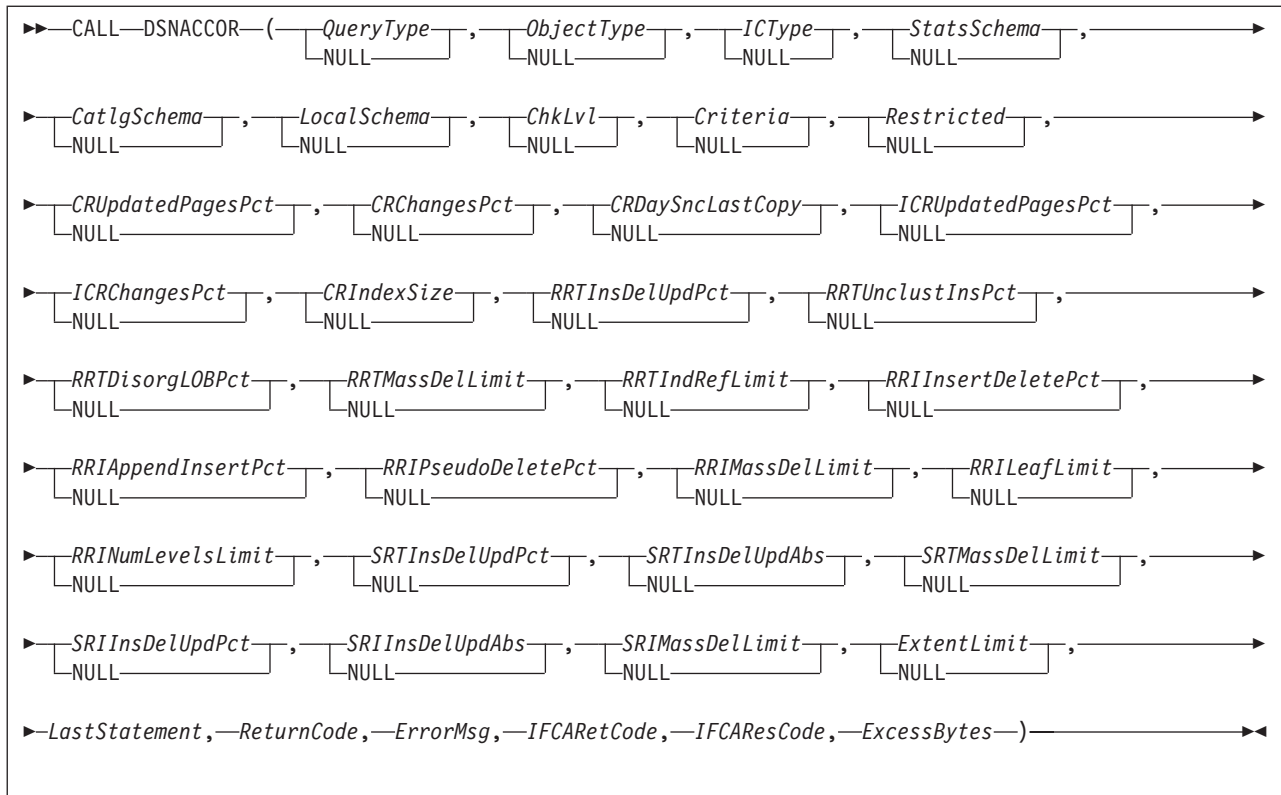
- The EXECUTE privilege on the package for DSNACCOR
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

The owner of the package or plan that contains the CALL statement must also have:

- SELECT authority on the real-time statistics tables
- The DISPLAY system privilege

## DSNACCOR syntax diagram

The following syntax diagram shows the CALL statement for invoking DSNACCOR. Because the linkage convention for DSNACCOR is GENERAL WITH NULLS, if you pass parameters in host variables, you need to include a null indicator with every host variable. Null indicators for input host variables must be initialized before you execute the CALL statement.



## DSNACCOR option descriptions

In the following option descriptions, the default value for an input parameter is the value that DSNACCOR uses if you specify a null value.

### *QueryType*

Specifies the types of actions that DSNACCOR recommends. This field contains one or more of the following values. Each value is enclosed in single quotation marks and separated from other values by a space.

**ALL** Makes recommendations for all of the following actions.

**COPY** Makes a recommendation on whether to perform an image copy.

### **RUNSTATS**

Makes a recommendation on whether to perform RUNSTATS.

### **REORG**

Makes a recommendation on whether to perform REORG. Choosing this value causes DSNACCOR to process the EXTENTS value also.

### **EXTENTS**

Indicates when data sets have exceeded a user-specified extents limit.

### **RESTRICT**

Indicates which objects are in a restricted state.

*QueryType* is an input parameter of type VARCHAR(40). The default is ALL.

### *ObjectType*

Specifies the types of objects for which DSNACCOR recommends actions:

**ALL** Table spaces and index spaces.

**TS** Table spaces only.

**IX** Index spaces only.

*ObjectType* is an input parameter of type VARCHAR(3). The default is **ALL**.

*ICType*

Specifies the types of image copies for which DSNACCOR is to make recommendations:

**F** Full image copy.

**I** Incremental image copy. This value is valid for table spaces only.

**B** Full image copy or incremental image copy.

*ICType* is an input parameter of type VARCHAR(1). The default is **B**.

*StatsSchema*

Specifies the qualifier for the real-time statistics table names. *StatsSchema* is an input parameter of type VARCHAR(128). The default is **SYSIBM**.

*CatlgSchema*

Specifies the qualifier for DB2 catalog table names. *CatlgSchema* is an input parameter of type VARCHAR(128). The default is **SYSIBM**.

*LocalSchema*

Specifies the qualifier for the names of tables that DSNACCOR creates. *LocalSchema* is an input parameter of type VARCHAR(128). The default is **DSNACC**.

*ChkLvl*

Specifies the types of checking that DSNACCOR performs, and indicates whether to include objects that fail those checks in the DSNACCOR recommendations result set. This value is the sum of any combination of the following values:

**0** DSNACCOR performs none of the following actions.

**1** For objects that are listed in the recommendations result set, check the SYSTABLESPACE or SYSINDEXES catalog tables to ensure that those objects have not been deleted. If value 16 is **not** also chosen, exclude rows for the deleted objects from the recommendations result set.

DSNACCOR excludes objects from the recommendations result set if those objects are not in the SYSTABLESPACE or SYSINDEXES catalog tables.

When this setting is specified, DSNACCOR does not use EXTENTS>*ExtentLimit* to determine whether a LOB table space should be reorganized.

**2** For index spaces that are listed in the recommendations result set, check the SYSTABLES, SYSTABLESPACE, and SYSINDEXES catalog tables to determine the name of the table space that is associated with each index space.

Choosing this value causes DSNACCOR to also check for rows in the recommendations result set for objects that have been deleted but have entries in the real-time statistics tables (value 1). This means that if value 16 is **not** also chosen, rows for deleted objects are excluded from the recommendations result set.

**4** Check whether rows that are in the DSNACCOR recommendations result set refer to objects that are in the exception table. For recommendations result set rows that have corresponding exception

table rows, copy the contents of the QUERYTYPE column of the exception table to the INEXCEPTTABLE column of the recommendations result set.

- 8 Check whether objects that have rows in the recommendations result set are restricted. Indicate the restricted status in the OBJECTSTATUS column of the result set.
- 16 For objects that are listed in the recommendations result set, check the SYSTABLESPACE or SYSINDEXES catalog tables to ensure that those objects have not been deleted (value 1). In result set rows for deleted objects, specify the word ORPHANED in the OBJECTSTATUS column.
- 32 Exclude rows from the DSNACCOR recommendations result set for index spaces for which the related table spaces have been recommended for REORG. Choosing this value causes DSNACCOR to perform the actions for values 1 and 2.
- 64 For index spaces that are listed in the DSNACCOR recommendations result set, check whether the related table spaces are listed in the exception table. For recommendations result set rows that have corresponding exception table rows, copy the contents of the QUERYTYPE column of the exception table to the INEXCEPTTABLE column of the recommendations result set.

*ChkLvl* is an input parameter of type INTEGER. The default is 7 (values 1+2+4).

#### *Criteria*

Narrows the set of objects for which DSNACCOR makes recommendations. This value is the search condition of an SQL WHERE clause. *Criteria* is an input parameter of type VARCHAR(4096). The default is that DSNACCOR makes recommendations for all table spaces and index spaces in the subsystem. The search condition can use any column in the result set and wildcards are allowed.

#### *Restricted*

A parameter that is reserved for future use. Specify the null value for this parameter. *Restricted* is an input parameter of type VARCHAR(80).

#### *CRUpdatedPagesPct*

Specifies a criterion for recommending a full image copy on a table space or index space. If the following condition is true for a table space, DSNACCOR recommends an image copy:

The total number of distinct updated pages, divided by the total number of preformatted pages (expressed as a percentage) is greater than *CRUpdatedPagesPct*.

See item 2 in Figure 21 on page 209. If both of the following conditions are true for an index space, DSNACCOR recommends an image copy:

- The total number of distinct updated pages, divided by the total number of preformatted pages (expressed as a percentage) is greater than *CRUpdatedPagesPct*.
- The number of active pages in the index space or partition is greater than *CRIndexSize*. See items 2 and 3 in Figure 22 on page 210.

*CRUpdatedPagesPct* is an input parameter of type INTEGER. The default is 20.

#### *CRChangesPct*

Specifies a criterion for recommending a full image copy on a table space or index space. If the following condition is true for a table space, DSNACCOR recommends an image copy:

The total number of insert, update, and delete operations since the last image copy, divided by the total number of rows or LOBs in a table space or partition (expressed as a percentage) is greater than *CRChangesPct*.

See item 3 in Figure 21 on page 209. If both of the following conditions are true for an index table space, DSNACCOR recommends an image copy:

- The total number of insert and delete operations since the last image copy, divided by the total number of entries in the index space or partition (expressed as a percentage) is greater than *CRChangesPct*.
- The number of active pages in the index space or partition is greater than *CRIndexSize*.

See items 2 and 4 in Figure 22 on page 210. *CRChangesPct* is an input parameter of type INTEGER. The default is 10.

#### *CRDaySncLastCopy*

Specifies a criterion for recommending a full image copy on a table space or index space. If the number of days since the last image copy is greater than this value, DSNACCOR recommends an image copy. (See item 1 in Figure 21 on page 209 and item 1 in Figure 22 on page 210.) *CRDaySncLastCopy* is an input parameter of type INTEGER. The default is 7.

#### *ICRUpdatedPagesPct*

Specifies a criterion for recommending an incremental image copy on a table space. If the following condition is true, DSNACCOR recommends an incremental image copy:

The number of distinct pages that were updated since the last image copy, divided by the total number of active pages in the table space or partition (expressed as a percentage) is greater than *ICRUpdatedPagesPct*.

(See item 1 in Figure 23 on page 210.) *ICRUpdatedPagesPct* is an input parameter of type INTEGER. The default is 1.

#### *ICRChangesPct*

Specifies a criterion for recommending an incremental image copy on a table space. If the following condition is true, DSNACCOR recommends an incremental image copy:

The ratio of the number of insert, update, or delete operations since the last image copy, to the total number of rows or LOBs in a table space or partition (expressed as a percentage) is greater than *ICRChangesPct*.

(See item 2 in Figure 23 on page 210.) *ICRChangesPct* is an input parameter of type INTEGER. The default is 1.

#### *CRIndexSize*

Specifies, when combined with *ICRUpdatedPagesPct* or *CRChangesPct*, a criterion for recommending a full image copy on an index space. (See items 2, 3, and 4 in Figure 22 on page 210.) *CRIndexSize* is an input parameter of type INTEGER. The default is 50.

#### *RRTInsDelUpdPct*

Specifies a criterion for recommending that the REORG utility is to be run on a table space. If the following condition is true, DSNACCOR recommends running REORG:

The sum of insert, update, and delete operations since the last REORG, divided by the total number of rows or LOBs in the table space or partition (expressed as a percentage) is greater than *RRTInsDelUpdPct*

(See item 1 in Figure 24 on page 210.) *RRTInsDelUpdPct* is an input parameter of type INTEGER. The default is 20.

#### *RRTUnclustInsPct*

Specifies a criterion for recommending that the REORG utility is to be run on a table space. If the following condition is true, DSNACCOR recommends running REORG:

The number of unclustered insert operations, divided by the total number of rows or LOBs in the table space or partition (expressed as a percentage) is greater than *RRTUnclustInsPct*.

(See item 2 in Figure 24 on page 210.) *RRTUnclustInsPct* is an input parameter of type INTEGER. The default is 10.

#### *RRTDisorgLOBPct*

Specifies a criterion for recommending that the REORG utility is to be run on a table space. If the following condition is true, DSNACCOR recommends running REORG:

The number of imperfectly chunked LOBs, divided by the total number of rows or LOBs in the table space or partition (expressed as a percentage) is greater than *RRTDisorgLOBPct*.

(See item 3 in Figure 24 on page 210.) *RRTDisorgLOBPct* is an input parameter of type INTEGER. The default is 10.

#### *RRTMassDelLimit*

Specifies a criterion for recommending that the REORG utility is to be run on a table space. If one of the following values is greater than *RRTMassDelLimit*, DSNACCOR recommends running REORG:

- The number of mass deletes from a segmented or LOB table space since the last REORG or LOAD REPLACE
- The number of dropped tables from a nonsegmented table space since the last REORG or LOAD REPLACE

(See item 5 in Figure 24 on page 210.) *RRTMassDelLimit* is an input parameter of type INTEGER. The default is 0.

#### *RRTIndRefLimit*

Specifies a criterion for recommending that the REORG utility is to be run on a table space. If the following value is greater than *RRTIndRefLimit*, DSNACCOR recommends running REORG:

The total number of overflow records that were created since the last REORG or LOAD REPLACE, divided by the total number of rows or LOBs in the table space or partition (expressed as a percentage)

(See item 4 in Figure 24 on page 210.) *RRTIndRefLimit* is an input parameter of type INTEGER. The default is 10.

#### *RRIInsertDeletePct*

Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRIInsertDeletePct*, DSNACCOR recommends running REORG:

The sum of the number of index entries that were inserted and deleted since the last REORG, divided by the total number of index entries in the index space or partition (expressed as a percentage)

(See item 1 in Figure 25 on page 211.) This is an input parameter of type INTEGER. The default is 20.

*RRIAppendInsertPct*

Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRIAppendInsertPct*, DSNACCOR recommends running REORG:

The number of index entries that were inserted since the last REORG, REBUILD INDEX, or LOAD REPLACE with a key value greater than the maximum key value in the index space or partition, divided by the number of index entries in the index space or partition (expressed as a percentage)

(See item 2 in Figure 25 on page 211.) *RRIInsertDeletePct* is an input parameter of type INTEGER. The default is 10.

*RRIPseudoDeletePct*

Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRIPseudoDeletePct*, DSNACCOR recommends running REORG:

The number of index entries that were pseudo-deleted since the last REORG, REBUILD INDEX, or LOAD REPLACE, divided by the number of index entries in the index space or partition (expressed as a percentage)

(See item 3 in Figure 25 on page 211.) *RRIPseudoDeletePct* is an input parameter of type INTEGER. The default is 10.

*RRIMassDelLimit*

Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the number of mass deletes from an index space or partition since the last REORG, REBUILD, or LOAD REPLACE is greater than this value, DSNACCOR recommends running REORG.

(See item 4 in Figure 25 on page 211.) *RRIMassDelLimit* is an input parameter of type INTEGER. The default is 0.

*RRILeafLimit*

Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRILeafLimit*, DSNACCOR recommends running REORG:

The number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE in which the higher part of the split page was far from the location of the original page, divided by the total number of active pages in the index space or partition (expressed as a percentage)

(See item 5 in Figure 25 on page 211.) *RRILeafLimit* is an input parameter of type INTEGER. The default is 10.

*RRINumLevelsLimit*

Specifies a criterion for recommending that the REORG utility is to be run on an index space. If the following value is greater than *RRINumLevelsLimit*, DSNACCOR recommends running REORG:

The number of levels in the index tree that were added or removed since the last REORG, REBUILD INDEX, or LOAD REPLACE

(See item 6 in Figure 25 on page 211.) *RRINumLevelsLimit* is an input parameter of type INTEGER. The default is 0.

*SRTInsDelUpdPct*

Specifies, when combined with *SRTInsDelUpdAbs*, a criterion for

recommending that the RUNSTATS utility is to be run on a table space. If both of the following conditions are true, DSNACCOR recommends running RUNSTATS:

- The number of insert, update, or delete operations since the last RUNSTATS on a table space or partition, divided by the total number of rows or LOBs in table space or partition (expressed as a percentage) is greater than *SRTInsDelUpdPct*.
- The sum of the number of inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRTInsDelUpdAbs*.

(See items 1 and 2 in Figure 26 on page 211.) *SRTInsDelUpdPct* is an input parameter of type INTEGER. The default is 20.

#### *SRTInsDelUpdAbs*

Specifies, when combined with *SRTInsDelUpdPct*, a criterion for recommending that the RUNSTATS utility is to be run on a table space. If both of the following conditions are true, DSNACCOR recommends running RUNSTATS:

- The number of insert, update, and delete operations since the last RUNSTATS on a table space or partition, divided by the total number of rows or LOBs in table space or partition (expressed as a percentage) is greater than *SRTInsDelUpdPct*.
- The sum of the number of inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRTInsDelUpdAbs*.

(See items 1 and 2 in Figure 26 on page 211.) *SRTInsDelUpdAbs* is an input parameter of type INTEGER. The default is 0.

#### *SRTMassDelLimit*

Specifies a criterion for recommending that the RUNSTATS utility is to be run on a table space. If the following condition is true, DSNACCOR recommends running RUNSTATS:

- The number of mass deletes from a table space or partition since the last REORG or LOAD REPLACE is greater than *SRTMassDelLimit*.

(See item 3 in Figure 26 on page 211.) *SRTMassDelLimit* is an input parameter of type INTEGER. The default is 0.

#### *SRIInsDelPct*

Specifies, when combined with *SRIInsDelAbs*, a criterion for recommending that the RUNSTATS utility is to be run on an index space. If both of the following conditions are true, DSNACCOR recommends running RUNSTATS:

- The number of inserted and deleted index entries since the last RUNSTATS on an index space or partition, divided by the total number of index entries in the index space or partition (expressed as a percentage) is greater than *SRIInsDelUpdPct*.
- The sum of the number of inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRIInsDelUpdAbs*.

(See items 1 and 2 in Figure 27 on page 211.) *SRIInsDelPct* is an input parameter of type INTEGER. The default is 20.

#### *SRIInsDelAbs*

Specifies, when combined with *SRIInsDelPct*, specifies a criterion for recommending that the RUNSTATS utility is to be run on an index space. If the following condition is true, DSNACCOR recommends running RUNSTATS:

- The number of inserted and deleted index entries since the last RUNSTATS on an index space or partition, divided by the total number of index entries in the index space or partition (expressed as a percentage) is greater than *SRIInsDelUpdPct*.
- The sum of the number of inserted and deleted index entries since the last RUNSTATS on an index space or partition is greater than *SRIInsDelUpdAbs*,

(See items 1 and 2 in Figure 27 on page 211.) *SRIInsDelAbs* is an input parameter of type INTEGER. The default is 0.

#### *SRIMassDelLimit*

Specifies a criterion for recommending that the RUNSTATS utility is to be run on an index space. If the number of mass deletes from an index space or partition since the last REORG, REBUILD INDEX, or LOAD REPLACE is greater than this value, DSNACCOR recommends running RUNSTATS.

(See item 3 in Figure 27 on page 211.) *SRIMassDelLimit* is an input parameter of type INTEGER. The **default** is 0.

#### *ExtentLimit*

Specifies a criterion for recommending that the RUNSTATS or REORG utility is to be run on a table space or index space. Also specifies that DSNACCOR is to warn the user that the table space or index space has used too many extents. DSNACCOR recommends running RUNSTATS or REORG, and altering data set allocations if the following condition is true:

- The number of physical extents in the index space, table space, or partition is greater than *ExtentLimit*.

(See Figure 28 on page 211.) *ExtentLimit* is an input parameter of type INTEGER. The default is 50.

#### *LastStatement*

When DSNACCOR returns a severe error (return code 12), this field contains the SQL statement that was executing when the error occurred. *LastStatement* is an output parameter of type VARCHAR(8012).

#### *ReturnCode*

The return code from DSNACCOR execution. Possible values are:

- |    |  |
|----|--|
| 0  | DSNACCOR executed successfully. The <i>ErrorMsg</i> parameter contains the approximate percentage of the total number of objects in the subsystem that have information in the real-time statistics tables.                  |
| 4  | DSNACCOR completed, but one or more input parameters might be incompatible. The <i>ErrorMsg</i> parameter contains the input parameters that might be incompatible.  |
| 8  | DSNACCOR terminated with errors. The <i>ErrorMsg</i> parameter contains a message that describes the error.  |
| 12 | DSNACCOR terminated with severe errors. The <i>ErrorMsg</i> parameter contains a message that describes the error. The <i>LastStatement</i> parameter contains the SQL statement that was executing when the error occurred. |
| 14 | DSNACCOR terminated because it could not access one or more of the real-time statistics tables. The <i>ErrorMsg</i> parameter contains the names of the tables that DSNACCOR could not access.                               |
| 15 | DSNACCOR terminated because it encountered a problem with one of the declared temporary tables that it defines and uses.   |

**16** DSNACCOR terminated because it could not define a declared temporary table. No table spaces were defined in the TEMP database.

**NULL** DSNACCOR terminated but could not set a return code.

*ReturnCode* is an output parameter of type INTEGER.

*ErrorMsg*

Contains information about DSNACCOR execution. If DSNACCOR runs successfully (*ReturnCode*=0), this field contains the approximate percentage of objects in the subsystem that are in the real-time statistics tables. Otherwise, this field contains error messages. *ErrorMsg* is an output parameter of type VARCHAR(1331).

*IFCARetCode*

Contains the return code from an IFI COMMAND call. DSNACCOR issues commands through the IFI interface to determine the status of objects. *IFCARetCode* is an output parameter of type INTEGER.

*IFCAResCode*

Contains the reason code from an IFI COMMAND call. *IFCAResCode* is an output parameter of type INTEGER.

*ExcessBytes*

Contains the number of bytes of information that did not fit in the IFI return area after an IFI COMMAND call. *ExcessBytes* is an output parameter of type INTEGER.

## DSNACCOR formulas for recommending actions

The following formulas specify the criteria that DSNACCOR uses for its recommendations and warnings. The variables in italics are DSNACCOR input parameters. The capitalized variables are columns of the SYSIBM.SYSTABLESPACESTATS or SYSIBM.SYSINDEXSPACESTATS tables. The numbers to the right of selected items are reference numbers for the option descriptions in “DSNACCOR option descriptions” on page 201.

The figure below shows the formula that DSNACCOR uses to recommend a full image copy on a table space.

---

```
((QueryType='COPY' OR QueryType='ALL') AND
(ObjectType='TS' OR ObjectType='ALL') AND
ICType='F') AND
(COPYLASTTIME IS NULL OR
REORGLASTTIME>COPYLASTTIME OR
LOADRLASTTIME>COPYLASTTIME OR
(CURRENT DATE-COPYLASTTIME)>CRDaySncLastCopy OR
(COPYUPDATEDPAGES*100)/NACTIVE>CRUpdatedPagesPct OR
(COPYCHANGES*100)/TOTALROWS>CRChangesPct)
```

1

2

3

---

Figure 21. DSNACCOR formula for recommending a full image copy on a table space

The figure below shows the formula that DSNACCOR uses to recommend a full image copy on an index space.

---

```

((QueryType='COPY' OR QueryType='ALL') AND
 (ObjectType='IX' OR ObjectType='ALL') AND
 (ICType='F' OR ICType='B')) AND
 (COPYLASTTIME IS NULL OR
 REORGLASTTIME>COPYLASTTIME OR
 LOADRLASTTIME>COPYLASTTIME OR
 REBUILDLASTTIME>COPYLASTTIME OR
 (CURRENT DATE-COPYLASTTIME)>CRDaySncLastCopy OR
 (NACTIVE>CRIndexSize AND
 ((COPYUPDATEDPAGES*100)/NACTIVE>CRUpdatedPagesPct OR
 (COPYCHANGES*100)/TOTALENTRIES>CRChangesPct)))

```

1

2

3

4

---

Figure 22. DSNACCOR formula for recommending a full image copy on an index space

The figure below shows the formula that DSNACCOR uses to recommend an incremental image copy on a table space.

---

```

((QueryType='COPY' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL') AND
 ICType='I' AND
 COPYLASTTIME IS NOT NULL) AND
 (LOADRLASTTIME>COPYLASTTIME OR
 REORGLASTTIME>COPYLASTTIME OR
 (COPYUPDATEDPAGES*100)/NACTIVE>ICRUpdatedPagesPct OR
 (COPYCHANGES*100)/TOTALROWS>ICRChangesPct))

```

1

2

---

Figure 23. DSNACCOR formula for recommending an incremental image copy on a table space

The figure below shows the formula that DSNACCOR uses to recommend a REORG on a table space. If the table space is a LOB table space, and CHCKLVL=1, the formula does not include EXTENTS>ExtentLimit.

---

```

((QueryType='REORG' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL')) AND
 (REORGLASTTIME IS NULL OR
 ((REORGINSERTS+REORGDELETES+REORGUPDATES)*100)/TOTALROWS>RRTInsDelUpdPct OR
 (REORGUNCLUSTINS*100)/TOTALROWS>RRTUnclustInsPct OR
 (REORGDISORGLQB*100)/TOTALROWS>RRTDisorgLOBPct OR
 ((REORGNearIndRef+REORGFARIndRef)*100)/TOTALROWS>RRTIndRefLimit OR
 REORGMASDELETE>RRTMassDelLimit OR
 EXTENTS>ExtentLimit)

```

1

2

3

4

5

6

---

Figure 24. DSNACCOR formula for recommending a REORG on a table space

The figure below shows the formula that DSNACCOR uses to recommend a REORG on an index space.

---

```

((QueryType='REORG' OR QueryType='ALL') AND
 (ObjectType='IX' OR ObjectType='ALL')) AND
 (REORGLASTTIME IS NULL OR
 ((REORGINSERTS+REORGDELETES)*100)/TOTALENTRIES>RRIInsertDeletePct OR
 (REORGAPPENDINSERT*100)/TOTALENTRIES>RRIAppendInsertPct OR
 (REORGPSEUDODELETES*100)/TOTALENTRIES>RRIPseudoDeletePct OR
 REORGMASDELETE>RRIMassDeleteLimit OR
 (REORGLEAFFAR*100)/NACTIVE>RRILeafLimit OR
 REORGNUMLEVELS>RRINumLevelsLimit OR
 EXTENTS>ExtentLimit)

```

1  
2  
3  
4  
5  
6  
7

---

Figure 25. DSNACCOR formula for recommending a REORG on an index space

The figure below shows the formula that DSNACCOR uses to recommend RUNSTATS on a table space.

---

```

((QueryType='RUNSTATS' OR QueryType='ALL') AND
 (ObjectType='TS' OR ObjectType='ALL')) AND
 (STATSLASTTIME IS NULL OR
 (((STATSINSERTS+STATSDELETES+STATSUPDATES)*100)/TOTALROWS>SRTInsDelUpdPct AND
 (STATSINSERTS+STATSDELETES+STATSUPDATES)>SRTInsDelUpdAbs) OR
 STATSMASDELETE>SRTMassDeleteLimit)

```

1  
2  
3

---

Figure 26. DSNACCOR formula for recommending RUNSTATS on a table space

The figure below shows the formula that DSNACCOR uses to recommend RUNSTATS on an index space.

---

```

((QueryType='RUNSTATS' OR QueryType='ALL') AND
 (ObjectType='IX' OR ObjectType='ALL')) AND
 (STATSLASTTIME IS NULL OR
 (((STATSINSERTS+STATSDELETES)*100)/TOTALENTRIES>SRIInsDelUpdPct AND
 (STATSINSERTS+STATSDELETES)>SRIInsDelPct) OR
 STATSMASDELETE>SRIInsDelAbs)

```

1  
2  
3

---

Figure 27. DSNACCOR formula for recommending RUNSTATS on an index space

The figure below shows the formula that DSNACCOR uses to that too many index space or table space extents have been used.

---

```

EXTENTS>ExtentLimit

```

---

Figure 28. DSNACCOR formula for warning that too many data set extents for a table space or index space are used

## Using an exception table

An exception table is an optional, user-created DB2 table that you can use to place information in the INEXCEPTTABLE column of the recommendations result set. You can put any information in the INEXCEPTTABLE column, but the most common use of this column is to filter the recommendations result set. Each row in the exception table represents an object for which you want to provide information for the recommendations result set.

To create the exception table, execute a CREATE TABLE statement similar to the following one. You can include other columns in the exception table, but you must include at least the columns that are shown.

```
CREATE TABLE DSNACC.EXCEPT_TBL  
  (DBNAME CHAR(8) NOT NULL,  
   NAME CHAR(8) NOT NULL,  
   QUERYTYPE CHAR(40))  
  CCSID EBCDIC;
```

The meanings of the columns are:

**DBNAME**

The database name for an object in the exception table.

**NAME**

The table space name or index space name for an object in the exception table.

**QUERYTYPE**

The information that you want to place in the INEXCEPTTABLE column of the recommendations result set.

If you put a null value in this column, DSNACCOR puts the value YES in the INEXCEPTTABLE column of the recommendations result set row for the object that matches the DBNAME and NAME values.

**Recommendation:** If you plan to put many rows in the exception table, create a nonunique index on DBNAME, NAME, and QUERYTYPE.

After you create the exception table, insert a row for each object for which you want to include information in the INEXCEPTTABLE column.

**Example:** Suppose that you want the INEXCEPTTABLE column to contain the string 'IRRELEVANT' for table space STAFF in database DSNDB04. You also want the INEXCEPTTABLE column to contain 'CURRENT' for table space DSN8S91D in database DSN8D91A. Execute these INSERT statements:

```
INSERT INTO DSNACC.EXCEPT_TBL VALUES('DSNDB04 ', 'STAFF ', 'IRRELEVANT');  
INSERT INTO DSNACC.EXCEPT_TBL VALUES('DSN8D91A', 'DSN8S91D', 'CURRENT');
```

To use the contents of INEXCEPTTABLE for filtering, include a condition that involves the INEXCEPTTABLE column in the search condition that you specify in your *Criteria* input parameter.

**Example:** Suppose that you want to include all rows for database DSNDB04 in the recommendations result set, except for those rows that contain the string 'IRRELEVANT' in the INEXCEPTTABLE column. You might include the following search condition in your *Criteria* input parameter:

```
DBNAME='DSNDB04' AND INEXCEPTTABLE<>'IRRELEVANT'
```

## Example of DSNACCOR invocation

The figure below is a COBOL example that shows variable declarations and an SQL CALL for obtaining recommendations for objects in databases DSN8D91A and DSN8D91L. This example also outlines the steps that you need to perform to retrieve the two result sets that DSNACCOR returns. These result sets are described in “DSNACCOR output” on page 216. See *DB2 Application Programming and SQL Guide* for more information about how to retrieve result sets from a stored procedure.

```

WORKING-STORAGE SECTION.
:
:
*****
* DSNACCOR PARAMETERS *
*****
01 QUERYTYPE.
   49 QUERYTYPE-LN          PICTURE S9(4) COMP VALUE 40.
   49 QUERYTYPE-DTA        PICTURE X(40)  VALUE 'ALL'.
01 OBJECTTYPE.
   49 OBJECTTYPE-LN         PICTURE S9(4) COMP VALUE 3.
   49 OBJECTTYPE-DTA        PICTURE X(3)   VALUE 'ALL'.
01 ICTYPE.
   49 ICTYPE-LN             PICTURE S9(4) COMP VALUE 1.
   49 ICTYPE-DTA            PICTURE X(1)   VALUE 'B'.
01 STATSSHEMA.
   49 STATSSHEMA-LN         PICTURE S9(4) COMP VALUE 128.
   49 STATSSHEMA-DTA        PICTURE X(128) VALUE 'SYSIBM'.
01 CATLGSCHEMA.
   49 CATLGSCHEMA-LN        PICTURE S9(4) COMP VALUE 128.
   49 CATLGSCHEMA-DTA        PICTURE X(128) VALUE 'SYSIBM'.
01 LOCALSCHEMA.
   49 LOCALSCHEMA-LN        PICTURE S9(4) COMP VALUE 128.
   49 LOCALSCHEMA-DTA        PICTURE X(128) VALUE 'DSNACC'.
01 CHKLVL          PICTURE S9(9) COMP VALUE +3.
01 CRITERIA.
   49 CRITERIA-LN          PICTURE S9(4) COMP VALUE 4096.
   49 CRITERIA-DTA         PICTURE X(4096) VALUE SPACES.
01 RESTRICTED.
   49 RESTRICTED-LN        PICTURE S9(4) COMP VALUE 80.
   49 RESTRICTED-DTA        PICTURE X(80)  VALUE SPACES.
01 CRUPDATEDPAGESPCT PICTURE S9(9) COMP VALUE +0.
01 CRCHANGESPCT      PICTURE S9(9) COMP VALUE +0.
01 CRDAYSNCCLASTCOPY  PICTURE S9(9) COMP VALUE +0.
01 ICRUPDATEDPAGESPCT PICTURE S9(9) COMP VALUE +0.
01 ICRCHANGESPCT      PICTURE S9(9) COMP VALUE +0.
01 CRINDEXSIZE        PICTURE S9(9) COMP VALUE +0.
01 RRTINSDELUPDPCT    PICTURE S9(9) COMP VALUE +0.
01 RRTUNCLUSTINSPCT   PICTURE S9(9) COMP VALUE +0.
01 RRTDISORGL0BPCT    PICTURE S9(9) COMP VALUE +0.
01 RRTMASSDELLIMIT    PICTURE S9(9) COMP VALUE +0.
01 RRTINDREFLIMIT     PICTURE S9(9) COMP VALUE +0.
01 RRIINSERTDELETEPCT PICTURE S9(9) COMP VALUE +0.
01 RRIAPPENDINSERTPCT PICTURE S9(9) COMP VALUE +0.
01 RRIPSEUDODELETEPCT PICTURE S9(9) COMP VALUE +0.
01 RRIMASSDELLIMIT    PICTURE S9(9) COMP VALUE +0.
01 RRILEAFLIMIT       PICTURE S9(9) COMP VALUE +0.
01 RRINUMLEVELSLIMIT  PICTURE S9(9) COMP VALUE +0.
01 SRTINSDELUPDPCT    PICTURE S9(9) COMP VALUE +0.
01 SRTINSDELUPDABS    PICTURE S9(9) COMP VALUE +0.
01 SRTMASSDELLIMIT    PICTURE S9(9) COMP VALUE +0.
01 SRIINSDELUPDPCT    PICTURE S9(9) COMP VALUE +0.
01 SRIINSDELUPDABS    PICTURE S9(9) COMP VALUE +0.
01 SRIMASSDELLIMIT    PICTURE S9(9) COMP VALUE +0.
01 EXTENTLIMIT        PICTURE S9(9) COMP VALUE +0.
01 LASTSTATEMENT.
   49 LASTSTATEMENT-LN     PICTURE S9(4) COMP VALUE 8012.
   49 LASTSTATEMENT-DTA    PICTURE X(8012) VALUE SPACES.
01 RETURNCODE         PICTURE S9(9) COMP VALUE +0.
01 ERRORMSG.
   49 ERRORMSG-LN          PICTURE S9(4) COMP VALUE 1331.
   49 ERRORMSG-DTA         PICTURE X(1331) VALUE SPACES.
01 IFCARETCODE         PICTURE S9(9) COMP VALUE +0.
01 IFCARESCODE         PICTURE S9(9) COMP VALUE +0.
01 EXCESSBYTES         PICTURE S9(9) COMP VALUE +0.

*****
* INDICATOR VARIABLES.          *
* INITIALIZE ALL NON-ESSENTIAL INPUT *

```

```

* VARIABLES TO -1, TO INDICATE THAT THE *
* INPUT VALUE IS NULL. *
*****
01 QUERYTYPE-IND          PICTURE S9(4) COMP-4 VALUE +0.
01 OBJECTTYPE-IND         PICTURE S9(4) COMP-4 VALUE +0.
01 ICTYPE-IND             PICTURE S9(4) COMP-4 VALUE +0.
01 STATSSHEMA-IND        PICTURE S9(4) COMP-4 VALUE -1.
01 CATLGSCHEMA-IND       PICTURE S9(4) COMP-4 VALUE -1.
01 LOCALSCHEMA-IND       PICTURE S9(4) COMP-4 VALUE -1.
01 CHKLVL-IND            PICTURE S9(4) COMP-4 VALUE -1.
01 CRITERIA-IND          PICTURE S9(4) COMP-4 VALUE -1.
01 RESTRICTED-IND        PICTURE S9(4) COMP-4 VALUE -1.
01 CRUPDATEDPAGESPCT-IND PICTURE S9(4) COMP-4 VALUE -1.
01 CRCHANGESPCT-IND     PICTURE S9(4) COMP-4 VALUE -1.
01 CRDAYSNCCLASTCOPY-IND PICTURE S9(4) COMP-4 VALUE -1.
01 ICRUPDATEDPAGESPCT-IND PICTURE S9(4) COMP-4 VALUE -1.
01 ICRCHANGESPCT-IND    PICTURE S9(4) COMP-4 VALUE -1.
01 CRINDEXSIZE-IND       PICTURE S9(4) COMP-4 VALUE -1.
01 RRTINSDELUPDPCT-IND   PICTURE S9(4) COMP-4 VALUE -1.
01 RRTUNCLUSTINSPT-IND   PICTURE S9(4) COMP-4 VALUE -1.
01 RRTDISORGL0BPCT-IND   PICTURE S9(4) COMP-4 VALUE -1.
01 RRTMASSDELLIMIT-IND   PICTURE S9(4) COMP-4 VALUE -1.
01 RRTINDREFLIMIT-IND    PICTURE S9(4) COMP-4 VALUE -1.
01 RRIINSERTDELETEPCT-IND PICTURE S9(4) COMP-4 VALUE -1.
01 RRIAPPENDINSERTPCT-IND PICTURE S9(4) COMP-4 VALUE -1.
01 RRIPEUDODELETEPCT-IND PICTURE S9(4) COMP-4 VALUE -1.
01 RRIASSDELLIMIT-IND    PICTURE S9(4) COMP-4 VALUE -1.
01 RRILEAFLIMIT-IND      PICTURE S9(4) COMP-4 VALUE -1.
01 RRINUMLEVELSLIMIT-IND PICTURE S9(4) COMP-4 VALUE -1.
01 SRTINSDELUPDPCT-IND   PICTURE S9(4) COMP-4 VALUE -1.
01 SRTINSDELUPDABS-IND   PICTURE S9(4) COMP-4 VALUE -1.
01 SRTMASSDELLIMIT-IND   PICTURE S9(4) COMP-4 VALUE -1.
01 SRIINSDELUPDPCT-IND   PICTURE S9(4) COMP-4 VALUE -1.
01 SRIINSDELUPDABS-IND   PICTURE S9(4) COMP-4 VALUE -1.
01 SRIMASSDELLIMIT-IND   PICTURE S9(4) COMP-4 VALUE -1.
01 EXTENTLIMIT-IND       PICTURE S9(4) COMP-4 VALUE -1.
01 LASTSTATEMENT-IND     PICTURE S9(4) COMP-4 VALUE +0.
01 RETURNCODE-IND        PICTURE S9(4) COMP-4 VALUE +0.
01 ERRORMSG-IND          PICTURE S9(4) COMP-4 VALUE +0.
01 IFCARETCODE-IND       PICTURE S9(4) COMP-4 VALUE +0.
01 IFCARESCODE-IND       PICTURE S9(4) COMP-4 VALUE +0.
01 EXCESSBYTES-IND       PICTURE S9(4) COMP-4 VALUE +0.

```

PROCEDURE DIVISION.

```

:
*****
* SET VALUES FOR DSNACCOR INPUT PARAMETERS: *
* - USE THE CHKLVL PARAMETER TO CAUSE DSNACCOR TO CHECK *
*   FOR ORPHANED OBJECTS AND INDEX SPACES WITHOUT *
*   TABLE SPACES, BUT INCLUDE THOSE OBJECTS IN THE *
*   RECOMMENDATIONS RESULT SET (CHKLVL=1+2+16=19) *
* - USE THE CRITERIA PARAMETER TO CAUSE DSNACCOR TO *
*   MAKE RECOMMENDATIONS ONLY FOR OBJECTS IN DATABASES *
*   DSN8D91A AND DSN8D91L. *
* - FOR THE FOLLOWING PARAMETERS, SET THESE VALUES, *
*   WHICH ARE LOWER THAN THE DEFAULTS: *
* CRUPDATEDPAGESPCT 4 *
* CRCHANGESPCT 2 *
* RRTINSDELUPDPCT 2 *
* RRTUNCLUSTINSPT 5 *
* RRTDISORGL0BPCT 5 *
* RRIAPPENDINSERTPCT 5 *
* SRTINSDELUPDPCT 5 *
* SRIINSDELUPDPCT 5 *
* EXTENTLIMIT 3 *
*****

```

MOVE 19 TO CHKLVL.

```

MOVE SPACES TO CRITERIA-DTA.
MOVE 'DBNAME = 'DSN8D91A'' OR DBNAME = 'DSN8D91L'''
    TO CRITERIA-DTA.
MOVE 46 TO CRITERIA-LN.
MOVE 4 TO CRUPDATEDPAGESPCT.
MOVE 2 TO CRCHANGESPCT.
MOVE 2 TO RRTINSDELUPDPCT.
MOVE 5 TO RRTUNCLUSTINSPECT.
MOVE 5 TO RRTDISORGLBPCT.
MOVE 5 TO RRIAPPENDINSERTPCT.
MOVE 5 TO SRTINSDELUPDPCT.
MOVE 5 TO SRIINSDELUPDPCT.
MOVE 3 TO EXTENTLIMIT.
*****
* INITIALIZE OUTPUT PARAMETERS *
*****
    MOVE SPACES TO LASTSTATEMENT-DTA.
    MOVE 1 TO LASTSTATEMENT-LN.
    MOVE 0 TO RETURNCODE-02.
    MOVE SPACES TO ERRORMSG-DTA.
    MOVE 1 TO ERRORMSG-LN.
    MOVE 0 TO IFCARETCODE.
    MOVE 0 TO IFCARESCODE.
    MOVE 0 TO EXCESSBYTES.
*****
* SET THE INDICATOR VARIABLES TO 0 FOR NON-NULL INPUT *
* PARAMETERS (PARAMETERS FOR WHICH YOU DO NOT WANT *
* DSNACCOR TO USE DEFAULT VALUES) AND FOR OUTPUT *
* PARAMETERS. *
*****
    MOVE 0 TO CHKLVL-IND.
    MOVE 0 TO CRITERIA-IND.
    MOVE 0 TO CRUPDATEDPAGESPCT-IND.
    MOVE 0 TO CRCHANGESPCT-IND.
    MOVE 0 TO RRTINSDELUPDPCT-IND.
    MOVE 0 TO RRTUNCLUSTINSPECT-IND.
    MOVE 0 TO RRTDISORGLBPCT-IND.
    MOVE 0 TO RRIAPPENDINSERTPCT-IND.
    MOVE 0 TO SRTINSDELUPDPCT-IND.
    MOVE 0 TO SRIINSDELUPDPCT-IND.
    MOVE 0 TO EXTENTLIMIT-IND.
    MOVE 0 TO LASTSTATEMENT-IND.
    MOVE 0 TO RETURNCODE-IND.
    MOVE 0 TO ERRORMSG-IND.
    MOVE 0 TO IFCARETCODE-IND.
    MOVE 0 TO IFCARESCODE-IND.
    MOVE 0 TO EXCESSBYTES-IND.
:
:
*****
* CALL DSNACCOR *
*****
EXEC SQL
    CALL SYSPROC.DSNACCOR
    (:QUERYTYPE           :QUERYTYPE-IND,
     :OBJECTTYPE           :OBJECTTYPE-IND,
     :ICTYPE               :ICTYPE-IND,
     :STATSSHEMA           :STATSSHEMA-IND,
     :CATLGSCHEMA          :CATLGSCHEMA-IND,
     :LOCALSCHEMA          :LOCALSCHEMA-IND,
     :CHKLVL               :CHKLVL-IND,
     :CRITERIA             :CRITERIA-IND,
     :RESTRICTED            :RESTRICTED-IND,
     :CRUPDATEDPAGESPCT    :CRUPDATEDPAGESPCT-IND,
     :CRCHANGESPCT         :CRCHANGESPCT-IND,
     :CRDAYSNCCLASTCOPY     :CRDAYSNCCLASTCOPY-IND,
     :ICRUPDATEDPAGESPCT    :ICRUPDATEDPAGESPCT-IND,
     :ICRCHANGESPCT        :ICRCHANGESPCT-IND,

```

```

:CRINDEXSIZE          :CRINDEXSIZE-IND,
:RTINSDELUPDPCT       :RTINSDELUPDPCT-IND,
:RTUNCLUSTINSPCT      :RTUNCLUSTINSPCT-IND,
:RTDISORGLOBPCT       :RTDISORGLOBPCT-IND,
:RTMASSDELLIMIT       :RTMASSDELLIMIT-IND,
:RTINDREFLIMIT        :RTINDREFLIMIT-IND,
:RIINSERTDELETEPCT    :RIINSERTDELETEPCT-IND,
:RIAPPENDINSERTPCT    :RIAPPENDINSERTPCT-IND,
:RIPEUDODELETEPCT     :RIPEUDODELETEPCT-IND,
:RIMASSDELLIMIT       :RIMASSDELLIMIT-IND,
:RILEAFLIMIT          :RILEAFLIMIT-IND,
:RRNUMLEVELSLIMIT     :RRNUMLEVELSLIMIT-IND,
:SRTINSDELUPDPCT      :SRTINSDELUPDPCT-IND,
:SRTINSDELUPDABS      :SRTINSDELUPDABS-IND,
:SRTMASSDELLIMIT      :SRTMASSDELLIMIT-IND,
:SRIINSDELUPDPCT      :SRIINSDELUPDPCT-IND,
:SRIINSDELUPDABS      :SRIINSDELUPDABS-IND,
:SRIMASSDELLIMIT      :SRIMASSDELLIMIT-IND,
:EXTENTLIMIT          :EXTENTLIMIT-IND,
:LASTSTATEMENT        :LASTSTATEMENT-IND,
:RETURNCODE           :RETURNCODE-IND,
:ERRORMSG             :ERRORMSG-IND,
:IFCARETCODE          :IFCARETCODE-IND,
:IFCARESCODE          :IFCARESCODE-IND,
:EXCESSBYTES          :EXCESSBYTES-IND)
END-EXEC.
*****
* ASSUME THAT THE SQL CALL RETURNED +466, WHICH MEANS THAT *
* RESULT SETS WERE RETURNED. RETRIEVE RESULT SETS.          *
*****
* LINK EACH RESULT SET TO A LOCATOR VARIABLE
  EXEC SQL ASSOCIATE LOCATORS (:LOC1, :LOC2)
  WITH PROCEDURE SYSPROC.DSNACCOR
  END-EXEC.
* LINK A CURSOR TO EACH RESULT SET
  EXEC SQL ALLOCATE C1 CURSOR FOR RESULT SET :LOC1
  END-EXEC.
  EXEC SQL ALLOCATE C2 CURSOR FOR RESULT SET :LOC2
  END-EXEC.
* PERFORM FETCHES USING C1 TO RETRIEVE ALL ROWS FROM FIRST RESULT SET
* PERFORM FETCHES USING C2 TO RETRIEVE ALL ROWS FROM SECOND RESULT SET

```

Figure 29. Example of DSNACCOR invocation

## DSNACCOR output

If DSNACCOR executes successfully, in addition to the output parameters described in “DSNACCOR option descriptions” on page 201, DSNACCOR returns two result sets.

The first result set contains the results from IFI COMMAND calls that DSNACCOR makes. The following table shows the format of the first result set.

Table 33. Result set row for first DSNACCOR result set

Column name	Data type	Contents
RS_SEQUENCE	INTEGER	Sequence number of the output line
RS_DATA	CHAR(80)	A line of command output

The second result set contains DSNACCOR’s recommendations. This result set contains one or more rows for a table space or index space. A nonpartitioned table

space or nonpartitioning index space can have at most one row in the result set. A partitioned table space or partitioning index space can have at most one row for each partition. A table space, index space, or partition has a row in the result set if both of the following conditions are true:

- If the *Criteria* input parameter contains a search condition, the search condition is true for the table space, index space, or partition.
- DSNACCOR recommends at least one action for the table space, index space, or partition.

The following table shows the columns of a result set row.

*Table 34. Result set row for second DSNACCOR result set*

Column name	Data type	Description
DBNAME	CHAR(8)	Name of the database that contains the object.
NAME	CHAR(8)	Table space or index space name.
PARTITION	INTEGER	Data set number or partition number.
OBJECTTYPE	CHAR(2)	DB2 object type: <ul style="list-style-type: none"> <li>• TS for a table space</li> <li>• IX for an index space</li> </ul>
OBJECTSTATUS	CHAR(36)	Status of the object: <ul style="list-style-type: none"> <li>• ORPHANED, if the object is an index space with no corresponding table space, or if the object does not exist</li> <li>• If the object is in a restricted state, one of the following values: <ul style="list-style-type: none"> <li>– TS=<i>restricted-state</i>, if OBJECTTYPE is TS</li> <li>– IX=<i>restricted-state</i>, if OBJECTTYPE is IX</li> </ul> <i>restricted-state</i> is one of the status codes that appear in DISPLAY DATABASE output. See <i>DB2 Command Reference</i> for details.</li> <li>• A, if the object is in an advisory state.</li> <li>• L, if the object is a logical partition, but not in an advisory state.</li> <li>• AL, if the object is a logical partition and in an advisory state.</li> </ul>
IMAGECOPY	CHAR(3)	COPY recommendation: <ul style="list-style-type: none"> <li>• If OBJECTTYPE is TS: FUL (full image copy), INC (incremental image copy), or NO</li> <li>• If OBJECTTYPE is IX: YES or NO</li> </ul>
RUNSTATS	CHAR(3)	RUNSTATS recommendation: YES or NO.
EXTENTS	CHAR(3)	Indicates whether the data sets for the object have exceeded <i>ExtentLimit</i> : YES or NO.
REORG	CHAR(3)	REORG recommendation: YES or NO.

Table 34. Result set row for second DSNACCOR result set (continued)

Column name	Data type	Description
INEXCEPTTABLE	CHAR(40)	A string that contains one of the following values: <ul style="list-style-type: none"> <li>Text that you specify in the QUERYTYPE column of the exception table.</li> <li>YES, if you put a row in the exception table for the object that this result set row represents, but you specify NULL in the QUERYTYPE column.</li> <li>NO, if the exception table exists but does not have a row for the object that this result set row represents.</li> <li>Null, if the exception table does not exist, or if the <i>ChkLvl</i> input parameter does not include the value 4.</li> </ul>
ASSOCIATEDTS	CHAR(8)	If OBJECTTYPE is IX and the <i>ChkLvl</i> input parameter includes the value 2, this value is the name of the table space that is associated with the index space. Otherwise null.
COPYLASTTIME	TIMESTAMP	Timestamp of the last full image copy on the object. Null if COPY was never run, or if the last COPY execution was terminated.
LOADRLASTTIME	TIMESTAMP	Timestamp of the last LOAD REPLACE on the object. Null if LOAD REPLACE was never run, or if the last LOAD REPLACE execution was terminated.
REBUILDLASTTIME	TIMESTAMP	Timestamp of the last REBUILD INDEX on the object. Null if REBUILD INDEX was never run, or if the last REBUILD INDEX execution was terminated.
CRUPDPGSPCT	INTEGER	If OBJECTTYPE is TS or IX and IMAGECOPY is YES, the ratio of distinct updated pages to preformatted pages, expressed as a percentage. Otherwise null.
CRCPYCHGPCT	INTEGER	If OBJECTTYPE is TS and IMAGECOPY is YES, the ratio of the total number insert, update, and delete operations since the last image copy to the total number of rows or LOBs in the table space or partition, expressed as a percentage. If OBJECTTYPE is IX and IMAGECOPY is YES, the ratio of the total number of insert and delete operations since the last image copy to the total number of entries in the index space or partition, expressed as a percentage. Otherwise null.
CRDAYSCELSTCPY	INTEGER	If OBJECTTYPE is TS or IX and IMAGECOPY is YES, the number of days since the last image copy. Otherwise null.
CRINDEXSIZE	INTEGER	If OBJECTTYPE is IX and IMAGECOPY is YES, the number of active pages in the index space or partition. Otherwise null.
REORGLASTTIME	TIMESTAMP	Timestamp of the last REORG on the object. Null if REORG was never run, or if the last REORG execution was terminated.
RRTINSDELUPDPCT	INTEGER	If OBJECTTYPE is TS and REORG is YES, the ratio of the sum of insert, update, and delete operations since the last REORG to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.
RRTUNCINSPCT	INTEGER	If OBJECTTYPE is TS and REORG is YES, the ratio of the number of unclustered insert operations to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.
RRTDISORGLBPCT	INTEGER	If OBJECTTYPE is TS and REORG is YES, the ratio of the number of imperfectly chunked LOBs to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.

Table 34. Result set row for second DSNACCOR result set (continued)

Column name	Data type	Description
RRTMASSDELETE	INTEGER	If OBJECTTYPE is TS, REORG is YES, and the table space is a segmented table space or LOB table space, the number of mass deletes since the last REORG or LOAD REPLACE. If OBJECTTYPE is TS, REORG is YES, and the table space is nonsegmented, the number of dropped tables since the last REORG or LOAD REPLACE. Otherwise null.
RRTINDREF	INTEGER	If OBJECTTYPE is TS, REORG is YES, the ratio of the total number of overflow records that were created since the last REORG or LOAD REPLACE to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.
RRIINSDELPCT	INTEGER	If OBJECTTYPE is IX and REORG is YES, the ratio of the total number of insert and delete operations since the last REORG to the total number of index entries in the index space or partition, expressed as a percentage. Otherwise null.
RRIAPPINSPECT	INTEGER	If OBJECTTYPE is IX and REORG is YES, the ratio of the number of index entries that were inserted since the last REORG, REBUILD INDEX, or LOAD REPLACE that had a key value greater than the maximum key value in the index space or partition, to the number of index entries in the index space or partition, expressed as a percentage. Otherwise null.
RRIPSDDELPCT	INTEGER	If OBJECTTYPE is IX and REORG is YES, the ratio of the number of index entries that were pseudo-deleted (the RID entry was marked as deleted) since the last REORG, REBUILD INDEX, or LOAD REPLACE to the number of index entries in the index space or partition, expressed as a percentage. Otherwise null.
RRIMASSDELETE	INTEGER	If OBJECTTYPE is IX and REORG is YES, the number of mass deletes from the index space or partition since the last REORG, REBUILD, or LOAD REPLACE. Otherwise null.
RRILEAF	INTEGER	If OBJECTTYPE is IX and REORG is YES, the ratio of the number of index page splits that occurred since the last REORG, REBUILD INDEX, or LOAD REPLACE in which the higher part of the split page was far from the location of the original page, to the total number of active pages in the index space or partition, expressed as a percentage. Otherwise null.
RRINUMLEVELS	INTEGER	If OBJECTTYPE is IX and REORG is YES, the number of levels in the index tree that were added or removed since the last REORG, REBUILD INDEX, or LOAD REPLACE. Otherwise null.
STATSLASTTIME	TIMESTAMP	Timestamp of the last RUNSTATS on the object. Null if RUNSTATS was never run, or if the last RUNSTATS execution was terminated.
SRTINSDELUPDPCT	INTEGER	If OBJECTTYPE is TS and RUNSTATS is YES, the ratio of the total number of insert, update, and delete operations since the last RUNSTATS on a table space or partition, to the total number of rows or LOBs in the table space or partition, expressed as a percentage. Otherwise null.
SRTINSDELUPDABS	INTEGER	If OBJECTTYPE is TS and RUNSTATS is YES, the total number of insert, update, and delete operations since the last RUNSTATS on a table space or partition. Otherwise null.

Table 34. Result set row for second DSNACCOR result set (continued)

Column name	Data type	Description
SRTMASSDELETE	INTEGER	If OBJECTTYPE is TS and RUNSTATS is YES, the number of mass deletes from the table space or partition since the last REORG or LOAD REPLACE. Otherwise null.
SRIINSDelpCT	INTEGER	If OBJECTTYPE is IX and RUNSTATS is YES, the ratio of the total number of insert and delete operations since the last RUNSTATS on the index space or partition, to the total number of index entries in the index space or partition, expressed as a percentage. Otherwise null.
SRIINSDelABS	INTEGER	If OBJECTTYPE is IX and RUNSTATS is YES, the number insert and delete operations since the last RUNSTATS on the index space or partition. Otherwise null.
SRIMASSDELETE	INTEGER	If OBJECTTYPE is IX and RUNSTATS is YES, the number of mass deletes from the index space or partition since the last REORG, REBUILD INDEX, or LOAD REPLACE. Otherwise, this value is null.
TOTALEXTENTS	SMALLINT	If EXTENTS is YES, the number of physical extents in the table space, index space, or partition. Otherwise, this value is null.



## When DB2 externalizes real-time statistics

DB2 externalizes real-time statistics in the following circumstances.

- When you issue STOP DATABASE(DSNCB06) SPACENAM(*space-name*).  
This command stops the in-memory statistics database and externalizes statistics for all objects in the subsystem.
- When you issue STOP DATABASE(*database-name*) SPACENAM(*space-name*)  
This command externalizes statistics only for *database-name* and *space-name*.
- At the end of the time interval that you specify during installation (the STATSINT system parameter).
- When you issue STOP DB2 MODE(QUIESCE).  
DB2 writes any statistics that are in memory when you issue this command to the statistics tables. However, if you issue STOP DB2 MODE(FORCE), DB2 does not write the statistics, and you lose them.
- During utility operations.  
Some utilities modify the statistics tables.  
DB2 does not maintain real-time statistics for any objects in the real-time statistics database. Therefore, if you run a utility with a utility list, and the list contains any real-time statistics objects, DB2 does not externalize real-time statistics during the execution of that utility for any of the objects in the utility list. DB2 does not maintain interval counter real-time statistics for SYSLGRNX and its indexes during utility operation. DB2 maintains statistics for these objects only during non-utility operation.

**Recommendation:** Do not include real-time statistics objects in utility lists.

DB2 does not externalize real-time statistics at a tracker site.

## How DB2 utilities affect the real-time statistics

In general, SQL INSERT, UPDATE, and DELETE statements cause DB2 to modify the real-time statistics.

However, certain DB2 utilities also affect the statistics. The following topics discuss the effect of each of those utilities on the statistics.

### How LOAD affects real-time statistics:

When you run LOAD REPLACE on a table space or table space partition, you change the statistics associated with that table space or partition.

The table below shows how running LOAD REPLACE on a table space or table space partition affects the SYSTABLESPACESTATS statistics.

*Table 35. Changed SYSTABLESPACESTATS values during LOAD REPLACE*

Column name	Settings for LOAD REPLACE after RELOAD phase
TOTALROWS	Number of loaded rows or LOBs <sup>1</sup>
NACTIVE	Actual value
SPACE	Actual value
EXTENTS	Actual value
LOADRLASTTIME	Current timestamp
REORGINSERTS	0
REORGDELETES	0
REORGUPDATES	0
REORGDISORGLOB	0
REORGUNCLUSTINS	0
REORMASSDELETE	0
REORGNEARINDREF	0
REORGFARINDEF	0
STATSLASTTIME	Current timestamp <sup>2</sup>
STATSINSERTS	0 <sup>2</sup>
STATSDELETES	0 <sup>2</sup>
STATSUPDATES	0 <sup>2</sup>
STATSMASSDELETE	0 <sup>2</sup>
COPYLASTTIME	Current timestamp <sup>3</sup>
COPYUPDATEDPAGES	0 <sup>3</sup>
COPYCHANGES	0 <sup>3</sup>
COPYUPDATELRN	Null <sup>3</sup>
COPYUPDATETIME	Null <sup>3</sup>

Table 35. Changed SYSTABLESPACESTATS values during LOAD REPLACE (continued)

Column name	Settings for LOAD REPLACE after RELOAD phase
<b>Notes:</b>	
1. Under certain conditions, such as a utility restart, the LOAD utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null. Some rows that are loaded into a table space and are included in this value might later be removed during the index validation phase or the referential integrity check. DB2 includes counts of those removed records in the statistics that record deleted records.	
2. DB2 sets this value only if the LOAD invocation includes the STATISTICS option.	
3. DB2 sets this value only if the LOAD invocation includes the COPYDDN option.	

The table below shows how running LOAD REPLACE affects the SYSINDEXSPACESTATS statistics for an index space or physical index partition.

Table 36. Changed SYSINDEXSPACESTATS values during LOAD REPLACE

Column name	Settings for LOAD REPLACE after BUILD phase
TOTALENTRIES	Number of index entries added <sup>1</sup>
NLEVELS	Actual value
NACTIVE	Actual value
SPACE	Actual value
EXTENTS	Actual value
LOADRLASTTIME	Current timestamp
REORGINSERTS	0
REORGDELETES	0
REORGAPPENDINSERT	0
REORGPSEUDODELETES	0
REORGMASDELETE	0
REORGLAFAFFAR	0
REORGLAFAFFAR	0
REORGNUMLEVELS	0
STATSLASTTIME	Current timestamp <sup>2</sup>
STATSINSERTS	0 <sup>2</sup>
STATSDELETES	0 <sup>2</sup>
STATSMASDELETE	0 <sup>2</sup>
COPYLASTTIME	Current timestamp <sup>3</sup>
COPYUPDATEDPAGES	0 <sup>3</sup>
COPYCHANGES	0 <sup>3</sup>
COPYUPDATELRN	Null <sup>3</sup>
COPYUPDATETIME	Null <sup>3</sup>

**Notes:**

1. Under certain conditions, such as a utility restart, the LOAD utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null.
2. DB2 sets this value only if the LOAD invocation includes the STATISTICS option.
3. DB2 sets this value only if the LOAD invocation includes the COPYDDN option.

For a logical index partition:

- A LOAD operation without the REPLACE option behaves similar to a SQL INSERT operation in that the number of records loaded are counted in the incremental counters such as REORGINSERTS, REORGAPPENDINSERT, STATSINSERTS, and COPYCHANGES. A LOAD operation without the REPLACE option affects the organization of the data and can be a trigger to run REORG, RUNSTATS or COPY.
- DB2 does not reset the nonpartitioned index when it does a LOAD REPLACE on a partition. Therefore, DB2 does not reset the statistics for the index. The REORG counters from the last REORG are still correct. DB2 updates LOADRLASTTIME when the entire nonpartitioned index is replaced.
- When DB2 does a LOAD RESUME YES on a partition, after the BUILD phase, DB2 increments TOTALENTRIES by the number of index entries that were inserted during the BUILD phase.

### How REORG affects real-time statistics:

When you run the REORG utility DB2 modifies some of the real-time statistics for the involved table space or index.

The table below shows how running REORG on a table space or table space partition affects the SYSTABLESPACESTATS statistics.

Table 37. Changed SYSTABLESPACESTATS values during REORG

Column name	Settings for REORG SHRLEVEL NONE after RELOAD phase	Settings for REORG SHRLEVEL REFERENCE or CHANGE after SWITCH phase
TOTALROWS	Number rows or LOBs loaded <sup>1</sup>	For SHRLEVEL REFERENCE: Number of loaded rows or LOBs during RELOAD phase  For SHRLEVEL CHANGE: Number of loaded rows or LOBs during RELOAD phase plus number of rows inserted during LOG APPLY phase minus number of rows deleted during LOG phase
NACTIVE	Actual value	Actual value
SPACE	Actual value	Actual value
EXTENTS	Actual value	Actual value
REORGLASTTIME	Current timestamp	Current timestamp
REORGINSERTS	0	Actual value <sup>2</sup>
REORGDELETES	0	Actual value <sup>2</sup>
REORGUPDATES	0	Actual value <sup>2</sup>
REORGDISORGLOB	0	Actual value <sup>2</sup>
REORGUNCLUSTINS	0	Actual value <sup>2</sup>
REORGMASDELETE	0	Actual value <sup>2</sup>
REORGNEARINDREF	0	Actual value <sup>2</sup>
REORGFARINDEF	0	Actual value <sup>2</sup>
STATSLASTTIME	Current timestamp <sup>3</sup>	Current timestamp <sup>3</sup>
STATSINSERTS	0 <sup>3</sup>	Actual value <sup>2</sup>

Table 37. Changed SYSTABLESPACESTATS values during REORG (continued)

Column name	Settings for REORG SHRLEVEL NONE after RELOAD phase	Settings for REORG SHRLEVEL REFERENCE or CHANGE after SWITCH phase
STATSDELETES	0 <sup>3</sup>	Actual value <sup>2</sup>
STATSUPDATES	0 <sup>3</sup>	Actual value <sup>2</sup>
STATSMASSDELETE	0 <sup>3</sup>	Actual value <sup>2</sup>
COPYLASTTIME	Current timestamp <sup>4</sup>	Current timestamp
COPYUPDATEDPAGES	0 <sup>4</sup>	Actual value <sup>2</sup>
COPYCHANGES	0 <sup>4</sup>	Actual value <sup>2</sup>
COPYUPDATELRSN	Null <sup>4</sup>	Actual value <sup>5</sup>
COPYUPDATETIME	Null <sup>4</sup>	Actual value <sup>5</sup>

**Notes:**

1. Under certain conditions, such as a utility restart, the REORG utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null. Some rows that are loaded into a table space and are included in this value might later be removed during the index validation phase or the referential integrity check. DB2 includes counts of those removed records in the statistics that record deleted records.
2. This is the actual number of inserts, updates, or deletes that are due to applying the log to the shadow copy.
3. DB2 sets this value only if the REORG invocation includes the STATISTICS option.
4. DB2 sets this value only if the REORG invocation includes the COPYDDN option.
5. This is the LRSN or timestamp for the first update that is due to applying the log to the shadow copy.

The table below shows how running REORG affects the SYSINDEXSPACESTATS statistics for an index space or physical index partition.

Table 38. Changed SYSINDEXSPACESTATS values during REORG

Column name	Settings for REORG SHRLEVEL NONE after RELOAD phase	Settings for REORG SHRLEVEL REFERENCE or CHANGE after SWITCH phase
TOTALENTRIES	Number of index entries added <sup>1</sup>	For SHRLEVEL REFERENCE: Number of added index entries during BUILD phase  For SHRLEVEL CHANGE: Number of added index entries during BUILD phase plus number of added index entries during LOG phase minus number of deleted index entries during LOG phase
NLEVELS	Actual value	Actual value
NACTIVE	Actual value	Actual value
SPACE	Actual value	Actual value
EXTENTS	Actual value	Actual value
REORGLASTTIME	Current timestamp	Current timestamp
REORGINSERTS	0	Actual value <sup>2</sup>
REORGDELETES	0	Actual value <sup>2</sup>
REORGAPPENDINSERT	0	Actual value <sup>2</sup>
REORGPSEUDODELETES	0	Actual value <sup>2</sup>
REORGMASSDELETE	0	Actual value <sup>2</sup>
REORGLEAFNEAR	0	Actual value <sup>2</sup>

Table 38. Changed SYSINDEXSPACESTATS values during REORG (continued)

Column name	Settings for REORG SHRLEVEL NONE after RELOAD phase	Settings for REORG SHRLEVEL REFERENCE or CHANGE after SWITCH phase
REORGLAFAFFAR	0	Actual value <sup>2</sup>
REORGNUMLEVELS	0	Actual value <sup>2</sup>
STATSLASTTIME	Current timestamp <sup>3</sup>	Current timestamp <sup>3</sup>
STATSINSERTS	0 <sup>3</sup>	Actual value <sup>2</sup>
STATSDELETES	0 <sup>3</sup>	Actual value <sup>2</sup>
STATSMASDELETE	0 <sup>3</sup>	Actual value <sup>2</sup>
COPYLASTTIME	Current timestamp <sup>4</sup>	Unchanged <sup>5</sup>
COPYUPDATEDPAGES	0 <sup>4</sup>	Unchanged <sup>5</sup>
COPYCHANGES	0 <sup>4</sup>	Unchanged <sup>5</sup>
COPYUPDATELRN	Null <sup>4</sup>	Unchanged <sup>5</sup>
COPYUPDATETIME	Null <sup>4</sup>	Unchanged <sup>5</sup>

**Notes:**

1. Under certain conditions, such as a utility restart, the REORG utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null.
2. This is the actual number of inserts, updates, or deletes that are due to applying the log to the shadow copy.
3. DB2 sets this value only if the REORG invocation includes the STATISTICS option.
4. DB2 sets this value only if the REORG invocation includes the COPYDDN option.
5. Inline COPY is not allowed for SHRLEVEL CHANGE or SHRLEVEL REFERENCE.

For a logical index partition, DB2 does not reset the nonpartitioned index when it does a REORG on a partition. Therefore, DB2 does not reset the statistics for the index. The REORG counters and REORGLASTTIME are relative to the last time the entire nonpartitioned index is reorganized. In addition, the REORG counters might be low because, due to the methodology, some index entries are changed during REORG of a partition.

**How REBUILD INDEX affects real-time statistics:**

Rebuilding an index has certain effects on the statistics for the index involved.

The table below shows how running REBUILD INDEX affects the SYSINDEXSPACESTATS statistics for an index space or physical index partition.

Table 39. Changed SYSINDEXSPACESTATS values during REBUILD INDEX

Column name	Settings after BUILD phase
TOTALENTRIES	Number of index entries added <sup>1</sup>
NLEVELS	Actual value
NACTIVE	Actual value
SPACE	Actual value
EXTENTS	Actual value
REBUILDLASTTIME	Current timestamp
REORGINSERTS	0
REORGDELETES	0

Table 39. Changed *SYSINDEXSPACESTATS* values during *REBUILD INDEX* (continued)

Column name	Settings after BUILD phase
REORGAPPENDINSERT	0
REORGPSEUDODELETES	0
REORGMASDELETE	0
REORGLAFAFNEAR	0
REORGLAFAFAR	0
REORGNUMLEVELS	0

**Note:**

1. Under certain conditions, such as a utility restart, the REBUILD utility might not have an accurate count of loaded records. In those cases, DB2 sets this value to null.

For a logical index partition, DB2 does not collect *TOTALENTRIES* statistics for the entire nonpartitioned index when it runs *REBUILD INDEX*. Therefore, DB2 does not reset the statistics for the index. The REORG counters from the last REORG are still correct. DB2 updates *REBUILDLASTTIME* when the entire nonpartitioned index is rebuilt.

**How RUNSTATS affects real-time statistics:**

When the RUNSTATS job starts, DB2 externalizes all in-memory statistics to the real-time statistics tables.

Only RUNSTATS UPDATE ALL affects the real-time statistics.

The table below shows how running RUNSTATS UPDATE ALL on a table space or table space partition affects the *SYSTABLESPACESTATS* statistics.

Table 40. Changed *SYSTABLESPACESTATS* values during *RUNSTATS UPDATE ALL*

Column name	During UTILINIT phase	After RUNSTATS phase
STATSLASTTIME	Current timestamp <sup>1</sup>	Timestamp of the start of RUNSTATS phase
STATSINSERTS	Actual value <sup>1</sup>	Actual value <sup>2</sup>
STATSDELETES	Actual value <sup>1</sup>	Actual value <sup>2</sup>
STATSUPDATES	Actual value <sup>1</sup>	Actual value <sup>2</sup>
STATSMASDELETE	Actual value <sup>1</sup>	Actual value <sup>2</sup>

**Notes:**

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

The table below shows how running RUNSTATS UPDATE ALL on an index affects the *SYSINDEXSPACESTATS* statistics.

Table 41. Changed *SYSINDEXSPACESTATS* values during *RUNSTATS UPDATE ALL*

Column name	During UTILINIT phase	After RUNSTATS phase
STATSLASTTIME	Current timestamp <sup>1</sup>	Timestamp of the start of RUNSTATS phase

*Table 41. Changed SYSINDEXSPACESTATS values during RUNSTATS UPDATE ALL (continued)*

Column name	During UTILINIT phase	After RUNSTATS phase
STATSINSERTS	Actual value <sup>1</sup>	Actual value <sup>2</sup>
STATSDELETES	Actual value <sup>1</sup>	Actual value <sup>2</sup>
STATSMASSDELETE	Actual value <sup>1</sup>	Actual value <sup>2</sup>

**Notes:**

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

**How COPY affects real-time statistics:**

When a COPY job starts, DB2 externalizes all in-memory statistics to the real-time statistics tables. Statistics are gathered for a full image copy or an incremental copy, but not for a data set copy.

The table below shows how running COPY on a table space or table space partition affects the SYSTABLESPACESTATS statistics.

*Table 42. Changed SYSTABLESPACESTATS values during COPY*

Column name	During UTILINIT phase	After COPY phase
COPYLASTTIME	Current timestamp <sup>1</sup>	Timestamp of the start of COPY phase
COPYUPDATEDPAGES	Actual value <sup>1</sup>	Actual value <sup>2</sup>
COPYCHANGES	Actual value <sup>1</sup>	Actual value <sup>2</sup>
COPYUPDATELRN	Actual value <sup>1</sup>	Actual value <sup>3</sup>
COPYUPDATETIME	Actual value <sup>1</sup>	Actual value <sup>3</sup>

**Notes:**

1. DB2 externalizes the current in-memory values.
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.
3. This value is null for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.

The table below shows how running COPY on an index affects the SYSINDEXSPACESTATS statistics.

*Table 43. Changed SYSINDEXSPACESTATS values during COPY*

Column name	During UTILINIT phase	After COPY phase
COPYLASTTIME	Current timestamp <sup>1</sup>	Timestamp of the start of COPY phase
COPYUPDATEDPAGES	Actual value <sup>1</sup>	Actual value <sup>2</sup>
COPYCHANGES	Actual value <sup>1</sup>	Actual value <sup>2</sup>
COPYUPDATELRN	Actual value <sup>1</sup>	Actual value <sup>3</sup>
COPYUPDATETIME	Actual value <sup>1</sup>	Actual value <sup>3</sup>

Table 43. Changed *SYSINDEXSPACESTATS* values during *COPY* (continued)

Column name	During UTILINIT phase	After COPY phase
<b>Notes:</b>		
1. DB2 externalizes the current in-memory values.		
2. This value is 0 for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.		
3. This value is null for SHRLEVEL REFERENCE, or the actual value for SHRLEVEL CHANGE.		

#### How RECOVER affects real-time statistics:

After recovery to the current state, the in-memory counter fields are still valid, so DB2 does not modify them. However, after a point-in-time recovery, the statistics might not be valid.

Consequently, DB2 sets all the REORG, STATS, and COPY counter statistics to null after a point-in-time recovery. After recovery to the current state, DB2 sets NACTIVE, SPACE, and EXTENTS to their new values. After a point-in-time recovery, DB2 sets NLEVELS, NACTIVE, SPACE, and EXTENTS to their new values.

#### How non-DB2 utilities affect real-time statistics

Non-DB2 utilities do not affect real-time statistics. Therefore, an object that is the target of a non-DB2 COPY, LOAD, REBUILD, REORG, or RUNSTATS job can cause incorrect statistics to be inserted in the real-time statistics tables.

Non-DB2 utilities do not affect real-time statistics. Therefore, an object that is the target of a non-DB2 COPY, LOAD, REBUILD, REORG, or RUNSTATS job can cause incorrect statistics to be inserted in the real-time statistics tables. Follow this process to ensure correct statistics when you run non-DB2 utilities:

1. Stop the table space or index on which you plan to run the utility. This action causes DB2 to write the in-memory statistics to the real-time statistics tables and initialize the in-memory counters. If DB2 cannot externalize the statistics, the STOP command does not fail.
2. Run the utility.
3. When the utility completes, update the statistics tables with new totals and timestamps, and put zero values in the incremental counter.

#### Real-time statistics on objects in work file databases and the TEMP database

Although you cannot run utilities on objects in the work files databases and TEMP database, DB2 records the NACTIVE, SPACE, and EXTENTS statistics on table spaces in those databases.

#### Real-time statistics for DEFINE NO objects

For objects that are created with DEFINE NO, no row is inserted into the real-time statistics table until the object is physically defined.

#### Real-time statistics on read-only or nonmodified objects

DB2 does not externalize the NACTIVE, SPACE, or EXTENTS statistics for read-only objects or objects that are not modified.

## How dropping objects affects real-time statistics

If you drop a table space or index, DB2 deletes its statistics from the real-time statistics tables.

However, if the real-time statistics database is not available when you drop a table space or index, the statistics remain in the real-time statistics tables, even though the corresponding object no longer exists. You need to use SQL DELETE statements to manually remove those rows from the real-time statistics tables.

If a row still exists in the real-time statistics tables for a dropped table space or index, and if you create a new object with the same DBID and PSID as the dropped object, DB2 reinitializes the row before it updates any values in that row.

## How SQL operations affect real-time statistics counters

SQL operations affect the counter columns in the real-time statistics tables. These are the columns that record the number of insert, delete, or update operations, as well as the total counters, TOTALROWS, and TOTALENTRIES.

### UPDATE

When you perform an UPDATE, DB2 increments the update counters.

### INSERT

When you perform an INSERT, DB2 increments the insert counters. DB2 keeps separate counters for clustered and unclustered INSERTs.

### DELETE

When you perform a DELETE, DB2 increments the delete counters.

### ROLLBACK

When you perform a ROLLBACK, DB2 increments the counters, depending on the type of SQL operation that is rolled back:

Rolled-back SQL statement	Incremented counters
UPDATE	Update counters
INSERT	Delete counters
DELETE	Insert counters

Notice that for INSERT and DELETE, the counter for the inverse operation is incremented. For example, if two INSERT statements are rolled back, the delete counter is incremented by 2.

## UPDATE of partitioning keys

If an update to a partitioning key causes rows to move to a new partition, the following real-time statistics are impacted:

Action	Incremented counters
When UPDATE is executed	Update count of old partition = +1 Insert count of new partition = +1

Action	Incremented counters
When UPDATE is committed	Delete count of old partition = +1
When UPDATE is rolled back	Update count of old partition = +1 (compensation log record) Delete count of new partition = +1 (remove inserted record)

If an update to a partitioning key does not cause rows to move to a new partition, the counts are accumulated as expected:

Action	Incremented counters
When UPDATE is executed	Update count of current partition = +1 NEAR/FAR indirect reference count = +1 (if overflow occurred)
When UPDATE is rolled back	Update count of current partition = +1 (compensation log record)

## Mass DELETE

Performing a mass delete operation on a table space does not cause DB2 to reset the counter columns in the real-time statistics tables. After a mass delete operation, the value in a counter column includes the count from a time prior to the mass delete operation, as well as the count after the mass delete operation.

## Real-time statistics in data sharing

In a data sharing environment, DB2 members update their statistics serially.

Each member reads the target row from the statistics table, obtains a lock, aggregates its in-memory statistics, and updates the statistics table with the new totals. Each member sets its own interval for writing real-time statistics.

DB2 does locking based on the lock size of the DSNRTSDB.DSNRTSTS table space. DB2 uses cursor stability isolation and CURRENTDATA(YES) when it reads the statistics tables.

At the beginning of a RUNSTATS job, all data sharing members externalize their statistics to the real-time statistics tables and reset their in-memory statistics. If all members cannot externalize their statistics, DB2 sets STATSLASTTIME to null. An error in gathering and externalizing statistics does not prevent RUNSTATS from running.

At the beginning of a COPY job, all data sharing members externalize their statistics to the real-time statistics tables and reset their in-memory statistics. If all members cannot externalize their statistics, DB2 sets COPYLASTTIME to null. An error in gathering and externalizing statistics does not prevent COPY from running.

Utilities that reset page sets to empty can invalidate the in-memory statistics of other DB2 members. The member that resets a page set notifies the other DB2 members that a page set has been reset to empty, and the in-memory statistics are invalidated. If the notify process fails, the utility that resets the page set does not

fail. DB2 sets the appropriate timestamp (REORGLASTTIME, STATSLASTTIME, or COPYLASTTIME) to null in the row for the empty page set to indicate that the statistics for that page set are unknown.

### **How the EXCHANGE command affects real-time statistics**

When the EXCHANGE command is used for clone tables real-time statistics are affected.

The values of the INSTANCE columns in the SYSTABLESPACESTATS and SYSINDEXSPACESTATS tables identify the VSAM data set that is associated with the real-time statistics row. For a cloned object, the real-time statistics table row might contain two rows, one for each instance of the object.

Utility operations and SQL operations can be run separately on each instance of the cloned object. Therefore, each instance of an object can be used to monitor activity, and allow recommendations on whether the base, the clone, or both objects require a REORG, RUNSTATS, or COPY operation.

### **Improving concurrency with real-time statistics**

Follow these recommendations to reduce the risk of timeouts and deadlocks when you work with the real-time statistics tables:

- When you run COPY, RUNSTATS, or REORG on the real-time statistics objects, use SHRLEVEL CHANGE.
- When you execute SQL statements to query the real-time statistics tables, use uncommitted read isolation.

### **Recovering the real-time statistics tables**

When you recover a DB2 subsystem after a disaster, DB2 starts with the ACCESS(MAINT) option. No statistics are externalized in this state.

Consequently, you need to perform the following actions on the real-time statistics database:

- Recover the real-time statistics objects after you recover the DB2 catalog and directory.
- Start the real-time statistics database explicitly, after DB2 restart.

### **Statistics accuracy**

In general, the real-time statistics values are very accurate.

However, several factors can affect the accuracy of the statistics:

- Certain utility restart scenarios
- Certain utility operations that leave indexes in a database restrictive state, such as RECOVER-pending (RECP)

Always consider the database restrictive state of objects before accepting a utility recommendation that is based on real-time statistics.

- A DB2 subsystem failure
- A notify failure in a data sharing environment

If you think that some statistics values might be inaccurate, you can correct the statistics by running REORG, RUNSTATS, or COPY on the objects for which DB2 generated the statistics.



---

## **Part 3. Programming DB2 applications for performance**

You can achieve better performance from DB2 by considering performance as you program and deploy your applications.



---

## Chapter 14. Tuning your queries

**PSPI** Before rewriting queries, you should consider running the REORG utility on your tables, and make sure that you have correct and up-to-date catalog statistics.

If you still have performance problems after you have tried the suggestions in this information, you can use other, more risky techniques. **PSPI**

---

### Coding SQL statements as simply as possible

Simpler SQL statements require less processing and generally perform better than more complex statements.

By following certain general guidelines when you write SQL statements you can keep them simple, limit the amount of processing that is required to execute the statement, and ensure better performance from the statements.

To get the best performance from your SQL statements:

- Avoid selecting unused columns.
- Do not include unnecessary ORDER BY or GROUP BY clauses.

### Coding queries with aggregate functions efficiently

If your query involves aggregate functions, you can take measure to increase the chances that they are evaluated when the data is retrieved, rather than afterward. Doing that can improve the performance of the query.

**PSPI** In general, an aggregate function performs best when evaluated during data access and next best when evaluated during DB2 sort. Least preferable is to have an aggregate function evaluated after the data has been retrieved. You can use EXPLAIN to determine when DB2 evaluates the aggregate functions.

To ensure that an aggregate function is evaluated when DB2 retrieves the data:

Queries that involve the functions MAX or MIN might be able to take advantage of one-fetch access. **PSPI**

Code the query so that every aggregate function that it contains meets the following criteria:

- No sort is needed for GROUP BY. Check this in the EXPLAIN output.
- No stage 2 (residual) predicates exist. Check this in your application.
- No distinct set functions exist, such as COUNT(DISTINCT C1).
- If the query is a join, all set functions must be on the last table joined. Check this by looking at the EXPLAIN output.
- All aggregate functions must be on single columns with no arithmetic expressions.
- The aggregate function is not one of the following aggregate functions:
  - STDDEV
  - STDDEV\_SAMP

- VAR
- VAR\_SAMP

## Using non-column expressions efficiently

DB2 can evaluate certain predicates at an earlier stage of processing called stage 1, so that the query that contains the predicate takes less time to run. When a predicate contains column and non-column expressions on the same side of the operator, DB2 must evaluate the predicate at a later stage.

**PSPI** To enable stage 1 processing of queries that contain non-column expressions:

Write each predicate so that all non-column expressions appear on the opposite side of the operator from any column expressions.

The following predicate combines a column, SALARY, with values that are not from columns on one side of the operator:

```
WHERE SALARY + (:hv1 * SALARY) > 50000
```

If you rewrite the predicate in the following way, DB2 can evaluate it more efficiently:

```
WHERE SALARY > 50000/(1 + :hv1)
```

In the second form, the column is by itself on one side of the operator, and all the other values are on the other side of the operator. The expression on the right is

called a *non-column expression*. **PSPI**

---

## Materialized query tables and query performance

One way to improve the performance of dynamic queries that operate on very large amounts of data is to generate the results of all or parts of the queries in advance, and store the results in materialized query tables.

**PSPI** Materialized query tables are user-created tables. Depending on how the tables are defined, they are user-maintained or system-maintained. If you have set subsystem parameters or an application sets special registers to tell DB2 to use materialized query tables, when DB2 executes a dynamic query, DB2 uses the contents of applicable materialized query tables if DB2 finds a performance advantage to doing so. **PSPI**

---

## Encrypted data and query performance

Encryption and decryption can degrade the performance of some queries.

**PSPI** However, you can lessen the performance impact of encryption and decryption by writing your queries carefully and designing your database with encrypted data in mind. **PSPI**

## XML data and query performance

XML data, by its nature, is more expensive to process, and might affect the performance of most SQL statements. Each row of an XML column contains an XML document, requires more processing, and requires more space in DB2.

### PSPI

When you use the XPath expression to search or extract the XML data, you can lessen the performance impact by avoiding the descendant or descendant-or-self axis in the expression. The XMLEXISTS predicate is always stage 2. However, you can use the XML index to reduce the number of rows, that is, the number of XML documents, that are to be searched at the second stage. For information about how to design your XML index so that the XPath predicates in the XMLEXISTS predicate can be used as the matching predicate in the matching index scan, see Matching index scan (MATCHCOLS>0.)

**Recommendations:** Creating and maintaining the XML index is more costly than for a non-XML index. You should, if possible, write your query to use non-XML indexes to filter as many rows as possible before the second stage.

### PSPI

## Best practices for XML performance in DB2

By observing certain best practices you can help to improve the performance of XML data that is stored in DB2 for z/OS.

### Choose the granularity of XML documents carefully

When you design your XML application, and your XML document structure in particular, you might have a choice to define which business data is kept together in a single XML document.

For example, in the department table and sample data shown in the following query and table uses one XML document per department.

```
create table dept(unitID char(8), deptdoc xml)
```

unitID	deptdoc
WWPR	<pre>&lt;dept deptID="PR27"&gt;   &lt;employee id="901"&gt;     &lt;name&gt;Jim Qu&lt;/name&gt;     &lt;phone&gt;408 555 1212&lt;/phone&gt;   &lt;/employee&gt;   &lt;employee id="902"&gt;     &lt;name&gt;Peter Pan&lt;/name&gt;     &lt;office&gt;216&lt;/office&gt;   &lt;/employee&gt; &lt;/dept&gt;</pre>
WWPR	<pre>&lt;dept deptID="V15"&gt;   &lt;employee id="673"&gt;     &lt;name&gt;Matt Foreman&lt;/name&gt;     &lt;phone&gt;416 891 7301&lt;/phone&gt;     &lt;office&gt;216&lt;/office&gt;   &lt;/employee&gt;   &lt;description&gt;This dept supports sales world wide&lt;/description&gt; &lt;/dept&gt;</pre>
S-USE	...
...	...

Figure 30. Sample data for the dept table

This intermediate granularity is a reasonable choice if a department is the predominant granularity at which your application accesses and processes the data. Alternatively, you might decide to combine multiple or many departments into a single XML document, such as those that belong to one unit. This coarse granularity, however, is sub-optimal if your application typically processes only one department at a time.

You might also choose one XML document per employee with an additional "dept" attribute for each employee to indicate which department he or she belongs to. This fine granularity would be a very good choice if employees use business objects of interest, which are often accessed and processed independently from the other employees in the same department. However, if the application typically processes all employees in one department together, one XML document per department might be the better choice.

## XML attributes versus elements

A common question related to XML document design, is when to use attributes instead of elements, and how that choice affects performance.

This question is much more relevant to data modeling than to performance. However, as a general rule, XML elements are more flexible than attributes because they can be repeated and nested.

For example, the department documents shown in the preceding example, use an element "phone" which allows multiple occurrences of "phone" for an employee who has multiple numbers. This design is also extensible in case we later need to break phone numbers into fragments, such as child elements for country code, area code, extension, and so on.

By contrast, if "phone" is an attribute of the employee element instead, it can exist only once per employee, and you could not add child elements. Such limitations might hinder future schema evolution.

Although you can probably model all of your data without using attributes, they can be a very intuitive choice for data items that are known not to repeat for a single element, nor have any sub-fields. Attributes can reduce the size of XML data slightly because they have only a single name-value pair, as opposed to elements, which have both a start tag and an end tag.

In DB2, you can use attributes in queries, predicates, and index definitions just as easily as elements. Because attributes are less extensible than elements, DB2 can apply certain storage and access optimizations. However, these advantages should be considered an extra performance bonus rather than an incentive to convert elements to attributes for the sake of performance, especially when data modeling considerations call for elements.

## Be aware of XML schema validation overhead

XML schema validation is an optional activity during XML parsing. Performance studies have shown that XML parsing in general is significantly more CPU-intensive if schema validation is enabled.

This overhead can vary drastically depending on the structure and size of your XML documents, and particularly on the size and complexity of the XML Schema used. For example, you might find 50% higher CPU consumption because of schema validation with moderately complex schemas. Unless your XML inserts are heavily I/O bound, the increased CPU consumption typically translates to reduced insert throughput.

An XML schema defines the structure, elements and attributes, data types, and value ranges, that are allowed in a set of XML documents. DB2 allows you to validate XML documents against XML schemas. If you choose to validate documents, you typically do so at insert time. Validation ensures that data inserted into the database is compliant with the schema definition, meaning that you prevent "junk data" from entering your tables.

Determine whether your application needs the stricter type checking for XML queries and XML schema compliance. For example, if you are using an application server which receives, validates, and processes XML documents before they are stored in the database, the documents probably do not need to be validated again in DB2. At that point you already know they are valid. Likewise, if the database receives XML documents from a trusted application, maybe even one that you control, and you know that the XML data is always valid, avoid schema validation for the benefit of higher insert performance. If, however, your DB2 database receives XML data from untrusted sources and you need to ensure schema compliance at the DB2 level, then you need to spend some extra CPU cycles on that.

## Specify full paths in XPath expressions when possible

When you know where in the structure of an XML document the desired element is located, it is best to provide that information in the form of a fully specified path to avoid unneeded overhead.

Consider the table that is created by the following SQL statement.

```
CREATE TABLE customer(info XML);
```

The following figure shows sample data in the info column.

```
<customerinfo Cid="1004">
  <name>Matt Foreman</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Toronto</city>
    <state/>Ontario
    <pcode>M3Z-5H9</pcode>
  </addr>
  <phone type="work">905-555-4789</phone>
  <phone type="home">416-555-3376</phone>
</customerinfo>
```

*Figure 31. Sample data in a customerinfo XML document*

If you want to retrieve customer phone numbers or the cities where they live, you can choose from several possible path expressions to get that data.

Both /customerinfo/phone and //phone would get you the phone numbers. Likewise, /customerinfo/addr/city and /customerinfo/\*/city both return the city. For best performance, the fully specified path is preferred over using either \* or //

because the fully specified path enables DB2 to navigate directly to the desired elements, skipping over non-relevant parts of the document.

In other words, if you know where in the document the desired element is located, it is best to provide that information in the form of a fully specified path to avoid unneeded overhead. If you ask for `//phone` instead of `/customerinfo/phone`, you ask for phone elements anywhere in the document. This requires DB2 to navigate down into the "addr" subtree of the document to look for phone elements at any level of the document.

Using `*` and `//` can also lead to undesired or unexpected query results. For example, if some of the "customerinfo" documents also contain "assistant" information, as shown in the following figure. The path `//phone` would return the customer phones and the assistant phone numbers, without distinguishing them. From the query result, you might mistakenly process the assistant's phone as a customer phone number.

```
<customerinfo Cid="1004">
  <name>Matt Foreman</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Toronto</city>
    <state>Ontario
    <pcode>M3Z-5H9</pcode>
  </addr>
  <phone type="work">905-555-4789</phone>
  <phone type="home">416-555-3376</phone>
  <assistant>
    <name>Peter Smith</name>
    <phone type="home">416-555-3426</phone>
  </assistant>
</customerinfo>
```

Figure 32. Sample data with phone and name elements at multiple levels

## Define lean indexes for XML data to avoid extra overhead

Assume that queries often search for "customerinfo" XML documents by customer name. An index on the customer name element, as shown in the following statements, can greatly improve the performance of such queries.

```
CREATE TABLE CUSTOMER (info XML);

CREATE INDEX custname1 ON customer(info)
GENERATE KEY USING XMLPATTERN '/customerinfo/name' as sql varchar(20);

CREATE INDEX custname2 ON customer(info)
GENERATE KEY USING XMLPATTERN '//name' as sql varchar(20);

SELECT * FROM customer
WHERE XMLEXISTS('$i/customerinfo[name = "Matt Foreman"]' passing info as $i);
```

Both of the indexes defined above are eligible to evaluate the `XMLEXISTS` predicate on the customer name. However, the `custname2` index might be substantially larger than the `custname1` index because it contains index entries not only for customer names but also for assistant names. This is because the XML pattern `//name` matches name elements anywhere in the document. However, if we never search by assistant name then we don't need them indexed.

For read operations, the custname1 index is smaller and therefore potentially better performing. For insert, update and delete operations, the custname1 index incurs index maintenance overhead only for customer names, while the custname2 index requires index maintenance for customer and assistant names. You certainly don't want to pay that extra price if you require maximum insert, update, and delete performance and you don't need indexed access based on assistant names.

## Use XMLEXISTS for predicates that filter at the document level

Consider the following table and sample data.

```
CREATE TABLE customer(info XML);

<customerinfo>
  <name>Matt Foreman</name>
  <phone>905-555-4789</phone>
</customerinfo>

<customerinfo>
  <name>Peter Jones</name>
  <phone>905-123-9065</phone>
</customerinfo>

<customerinfo>
  <name>Mary Clark</name>
  <phone>905-890-0763</phone>
</customerinfo>
```

Figure 33. Sample data in the customer table

Assume, for example, that you want to return the names of customers which have the phone number "905-555-4789". You might be tempted to write the following query.

```
SELECT XMLQUERY('$i/customerinfo[phone = "905-555-4789"]/name' passing info as "i")
FROM customer;
```

However, this query is not what you want for several reasons:

- It returns the following result set which has as many rows as there are rows in the table. This is because the SQL statement has no where clause and therefore cannot eliminate any rows. The result is shown in the following figure.

```
<name>Matt Foreman</name>

3 record(s) selected
```

Figure 34. Result for the preceding example query

- For each row in the table which doesn't match the predicate, a row containing an empty XML sequence is returned. This is because the XQuery expression in the XMLQUERY function is applied to one row, or document, at a time and never removes a row from the result set, only modifies its value. The value produced by that XQuery is either the customer's name element if the predicate is true, or the empty sequence otherwise. These empty rows are semantically correct, according to the SQL/XML standard, and must be returned if the query is written as shown.

- The performance of the query is poor. First, an index which might exist on /customerinfo/phone cannot be used because this query is not allowed to eliminate any rows. Secondly, returning many empty rows makes this query needlessly slow.

To resolve the performance issues and get the desired output, you should use the XMLQUERY function in the select clause only to extract the customer names, and move the search condition, which should eliminate rows, into an XMLEXISTS predicate in the WHERE clause. Doing so will allow index usage, row filtering, and avoid the overhead of empty results rows. You could write the query as shown in the following figure.

```
SELECT XMLQUERY('$i/customerinfo/name' passing info as "i")
FROM customer
WHERE XMLEXISTS('$i/customerinfo[phone = "905-555-4789"]' passing info as "i")
```

### Use square brackets too avoid Boolean predicates in XMLEXISTS

A common error is to write the previous query without the square brackets in the XMLEXISTS function, as shown in the following query.

```
SELECT XMLQUERY('$i/customerinfo/name' passing info as "i")
FROM customer
WHERE XMLEXISTS('$i/customerinfo/phone = "905-555-4789"' passing info as "i")
```

Writing the query this way produces the following results shown in the following figure.

```
<name>Matt Foreman</name>
<name>Peter Jones</name>
<name>Mary Clark</name>

3 record(s) selected
```

Figure 35. Sample results for the preceding example query

The expression in the XMLEXISTS predicate is written such that XMLEXISTS always evaluates to true. Hence, no rows are eliminated. For a given row, the XMLEXISTS predicate evaluates to false only if the XQuery expression inside returns the empty sequence. However, without the square brackets the XQuery expression is a Boolean expression which always returns a Boolean value and never the empty sequence. Note that XMLEXISTS truly checks for the existence of a value and evaluates to true if a value exists, even if that value happens to be the Boolean value "false". This behavior is correct according to the SQL/XML standard, although it is probably not what you intended to express.

The impact is again that an index on phone cannot be used because no rows will be eliminated, and you receive a lot more rows than you actually want. Also, beware not to make this same mistake when using two or more predicates, as shown in the following query.

```
SELECT XMLQUERY('$i/customerinfo/name' passing info as "i")
FROM customer
WHERE XMLEXISTS('$i/customerinfo[phone = "905-555-4789"] and
    $i/customerinfo[name = "Matt Foreman"]'
    passing info as "i")
```

The XQuery expression is still a Boolean expression because it has the form "exp1 and exp2." You would write the query as shown in the following query to filter rows and allow for index usage.

```
SELECT XMLQUERY('$i/customerinfo/name' passing info as "i")
from customer
WHERE XMLEXISTS('$i/customerinfo[phone = "905-555-4789" and name = "Matt Foreman"]'
    passing info as "i")
```

## Use RUNSTATS to collect statistics for XML data and indexes

The RUNSTATS utility has been extended to collect statistics on XML tables and indexes, and DB2 optimizer uses these statistics to generate efficient execution plan for XML queries. Consequently, continue to use RUNSTATS on XML tables and indexes as you would for relational data. You need to specify XML tablespace names explicitly or use LISTDEF to include ALL or XML objects to obtain the XML table statistics.

## Use SQL/XML publishing views to expose relational data as XML

You can include relational columns in a SQL/XML publishing view, and when querying the view, express any predicates on those columns rather than on the constructed XML.

SQL/XML publishing functions allow you to convert relational data into XML format. Hiding the SQL/XML publishing functions in a view definition can be beneficial. Applications or other queries can simply select the constructed XML documents from the view, instead of dealing with the publishing functions themselves. The following statements creates a view that contains hidden SQL/XML publishing functions.

```
CREATE TABLE unit( unitID char(8), name char(20), manager varchar(20));

CREATE VIEW UnitView(unitID, name, unitdoc) as
    SELECT unitID, name,
           XMLELEMENT(NAME "Unit",
                     XMLELEMENT(NAME "ID", u.unitID),
                     XMLELEMENT(NAME "UnitName", u.name),
                     XMLELEMENT(NAME "Mgr", u.manager)
                    )
    FROM unit u;
```

Note that the view definition includes relational columns. This does not create any physical redundancy because it is only a view, not a materialized view. Exposing the relational columns helps to query this view efficiently.

The following query uses a relational predicate to ensure that only the XML document for "WWPR" is constructed, resulting in a shorter runtime, especially on a large data set.

```
SELECT unitdoc
FROM UnitView
WHERE UnitID = "WWPR";
```

## Use XMLTABLE views to expose XML data in relational format

You might also want to use a view to expose XML data in relational format. Similar caution needs to be applied as before, but in the reverse way. In the following example the SQL/XML function XMLTABLE returns values from XML documents in tabular format.

```

CREATE TABLE customer(info XML);

CREATE VIEW myview(CustomerID, Name, Zip, Info) AS
SELECT T.*, info
FROM customer, XMLTABLE ('$c/customerinfo' passing info as "c"
    COLUMNS
    "CID"      INTEGER      PATH './@Cid',
    "Name"     VARCHAR(30)   PATH './name',
    "Zip"      CHAR(12)      PATH './addr/pcode' ) as T;

```

The view definition includes XML column info to help query the view efficiently. Assume that you want to retrieve a tabular list of customer IDs and names for a given ZIP code. Both of the following queries can do that, but the second one tends to perform better than the first.

In the first query, the filtering predicate is expressed on the CHAR column "Zip" generated by the XMLTABLE function. However, not all the relational predicates can be applied to the underlying XML column or indexes. Consequently, the query requires the view to generate rows for all customers and then picks out the one for zip code "95141".

```

SELECT CustomerID, Name
FROM myview
WHERE Zip = '95141';

```

The second query uses an XML predicate to ensure that only the rows for "95141" get generated, resulting in a shorter runtime, especially on a large data set.

```

SELECT CustomerID, Name
FROM myView
WHERE xmlexists('$i/customerinfo[addr/pcode = "95141"]' passing info as "i");

```

## Use SQL and XML statements with parameter markers for short queries and OLTP applications

The SQL/XML functions XMLQUERY, XMLTABLE and XMLEXISTS support external parameters.

Very short database queries often execute so fast that the time to compile and optimize them is a substantial portion of their total response time. Consequently, you might want to compile, or "prepare," them just once and only pass predicate literal values for each execution. This technique is recommended for applications with short and repetitive queries. The following query shows how you can use parameter markers to achieve the result of the preceding example.

```

SELECT info
FROM customer
WHERE xmlexists('$i/customerinfo[phone = $p]'
    passing info as "i", cast(? as varchar(12)) as "p")

```

## Avoid code page conversion during XML insert and retrieval

XML is different from other types of data in DB2 because it can be internally and externally encoded. Internally encoded means that the encoding of your XML data can be derived from the data itself. Externally encoded means that the encoding is derived from external information.

The data type of the application variables that you use to exchange XML data with DB2 determines how the encoding is derived. If your application uses character type variables for XML, then it is externally encoded. If you use binary application data types, then the XML data is considered internally encoded.

Internally encoded means that the encoding is determined by either a Unicode Byte-Order mark (BOM) or an encoding declaration in the XML document itself, such as: `<?xml version="1.0" encoding="UTF-8" ?>`

From a performance point of view, the goal is to avoid code page conversions as much as possible because they consume extra CPU cycles. Internally encoded XML data is preferred over externally encoded data because it can prevent unnecessary code page conversion.

This means that in your application you should prefer binary data types over character types. For example, in CLI when you use `SQLBindParameter()` to bind parameter markers to input data buffers, you should use `SQL_C_BINARY` data buffers rather than `SQL_C_CHAR`, `SQL_C_DBCHAR`, or `SQL_C_WCHAR`. In host applications, use XML AS BLOB as the host variable type.

When inserting XML data from Java applications, reading in the XML data as a binary stream (`setBinaryStream`) is better than as a string (`setString`). Similarly, if your Java application receives XML from DB2 and writes it to a file, code page conversion may occur if the XML is written as non-binary data.

When you retrieve XML data from DB2 into your application, it is serialized. Serialization is the inverse operation of XML parsing. It is the process that DB2 uses to convert internal XML format, which is a parsed, tree-like representation, into the textual XML format that your application can understand. In most cases it is best to let DB2 perform implicit serialization. This means your SQL/XML statements simply select XML-type values as shown in the following example, and that DB2 performs the serialization into your application variables as efficiently as possible.

```
CREATE TABLE customer(info XML);

SELECT info FROM customer WHERE...;

SELECT XMLQUERY('$i/customerinfo/name' passing info as "i")
FROM customer
WHERE...;
```

If your application deals with very large XML documents, it might benefit from using LOB locators for data retrieval. This requires explicit serialization to a LOB type, preferably BLOB, because explicit serialization into a character type such as CLOB can introduce encoding issues and unnecessary code page conversion. Explicit serialization uses the `XMLSERIALIZE` function as shown in the following query.

```
SELECT XMLSERIALIZE(info as BLOB(1M)) FROM customer WHERE...;
```

---

## Writing efficient predicates

*Predicates* are found in the WHERE, HAVING, or ON clauses of SQL statements; they describe attributes of data. Because SQL allows you to express the same query in different ways, knowing how predicates affect path selection helps you write queries that access data efficiently.

**PSPI** Predicates are usually based on the columns of a table and either qualify rows (through an index) or reject rows (returned by a scan) when the table is accessed. The resulting qualified or rejected rows are independent of the access path chosen for that table.

The following query has three predicates: an equal predicate on C1, a BETWEEN predicate on C2, and a LIKE predicate on C3.

```
SELECT * FROM T1
WHERE C1 = 10 AND
      C2 BETWEEN 10 AND 20 AND
      C3 NOT LIKE 'A%'
```



## Ensuring that predicates are coded correctly

Whether you code the predicates of your SQL statements correctly has a great effect on the performance of those queries.



To ensure the best performance from your predicates:

- Make sure all the predicates that you think should be indexable are coded so that they can be indexable. Refer to Table 45 on page 253 to see which predicates are indexable and which are not.
- Try to remove any predicates that are unintentionally redundant or not needed; they can slow down performance.
- For string comparisons other than equal comparisons, ensure that the declared length of a host variable is less than or equal to the length attribute of the table column that it is compared to. For languages in which character strings are null-terminated, the string length can be less than or equal to the column length plus 1. If the declared length of the host variable is greater than the column length, in a non-equal comparison, the predicate is stage 1 but cannot be a matching predicate for an index scan.

For example, assume that a host variable and an SQL column are defined as follows:

C language declaration	SQL definition
char string_hv[15]	STRING_COL CHAR(12)

A predicate such as WHERE STRING\_COL > :string\_hv is not a matching predicate for an index scan because the length of string\_hv is greater than the length of STRING\_COL. One way to avoid an inefficient predicate using character host variables is to declare the host variable with a length that is less than or equal to the column length:


```
char string_hv[12]
```

Because this is a C language example, the host variable length could be 1 byte greater than the column length:

```
char string_hv[13]
```

For numeric comparisons, a comparison between a DECIMAL column and a float or real host variable is stage 2 if the precision of the DECIMAL column is greater than 15. For example, assume that a host variable and an SQL column are defined as follows:

C language declaration	SQL definition
float float_hv	DECIMAL_COL DECIMAL(16,2)

A predicate such as WHERE DECIMAL\_COL = :float\_hv is not a matching predicate for an index scan because the length of DECIMAL\_COL is greater than 15. However, if DECIMAL\_COL is defined as DECIMAL(15,2), the predicate is stage 1 and indexable. 

## Properties of predicates

Predicates in the WHERE and ON clauses of a SQL statement affect how DB2 selects the access path for the statement.

 PSPI

Predicates in a HAVING clause are not used when DB2 selects access paths. Consequently, in this topic, the term 'predicate' means a predicate after WHERE or ON. A predicate influences the selection of an access path according to the following factors:

- The *type* of predicate, according to its operator or syntax.
- Whether the predicate is *indexable*.
- Whether the predicate is *stage 1* or *stage 2*.
- Whether the predicate contains a ROWID column.
- Whether the predicates in part of an ON clause.

The following terms are used to differentiate and classify certain kinds of predicates:


### Simple or compound

A *compound* predicate is the result of two predicates, whether simple or compound, connected together by AND or OR Boolean operators. All others are *simple*.

### Local or join

*Local predicates* reference only one table. They are local to the table and restrict the number of rows returned for that table. *Join predicates* involve more than one table or correlated reference. They determine the way rows are joined from two or more tables.

### Boolean term

Any predicate that is not contained by a compound OR predicate structure is a *Boolean term*. If a Boolean term is evaluated false for a particular row, the whole WHERE clause is evaluated false for that row. 

## Predicate types

The type of a predicate depends on its operator or syntax. The type determines what type of processing and filtering occurs when DB2 evaluates the predicate.

 PSPI

The following table shows the different predicate types.

Table 44. Definitions and examples of predicate types

Type	Definition	Example
Subquery	Any predicate that includes another SELECT statement.	C1 IN (SELECT C10 FROM TABLE1)

Table 44. Definitions and examples of predicate types (continued)

Type	Definition	Example
Equal	Any predicate that is not a subquery predicate and has an equal operator and no NOT operator. Also included are predicates of the form C1 IS NULL and C1 IS NOT DISTINCT FROM.	C1=100
Range	Any predicate that is not a subquery predicate and contains one of the following operators: > >= < <= LIKE BETWEEN	C1>100
IN-list	A predicate of the form column IN (list of values).	C1 IN (5,10,15)
NOT	Any predicate that is not a subquery predicate and contains a NOT operator. Also included are predicates of the form C1 IS DISTINCT FROM.	C1 <> 5 or C1 NOT BETWEEN 10 AND 20

The following two examples show how the predicate type can influence the way that DB2 chooses an access path. In each one, assume that a unique index, I1 (C1), exists on table T1 (C1, C2), and that all values of C1 are positive integers.

### Example equal predicate

The following query contains an equal predicate:

```
SELECT * FROM T1 WHERE C1 = 0;
```

DB2 chooses index access in this case because the index is highly selective on column C1.

### Example range predicate

The following query contains a range predicate:

```
SELECT C1, C2 FROM T1 WHERE C1 >= 0;
```

However, the predicate does not eliminate any rows of T1. Therefore, DB2 might determine during bind that a table space scan is more efficient than the index scan.



### Indexable and non-indexable predicates:

An *indexable* predicate can match index entries; predicates that cannot match index entries are said to be *non-indexable*.



To make your queries as efficient as possible, you can use indexable predicates in your queries and create suitable indexes on your tables. Indexable predicates allow the possible use of a matching index scan, which is often a very efficient access path.

Indexable predicates might or might not become matching predicates of an index; depending on the availability of indexes and the access path that DB2 chooses at bind time.

For example, if the employee table has an index on the column LASTNAME, the following predicate can be a matching predicate:

```
SELECT * FROM DSN8910.EMP WHERE LASTNAME = 'SMITH';
```

In contrast, the following predicate cannot be a matching predicate, because it is not indexable.

```
SELECT * FROM DSN8910.EMP WHERE SEX <> 'F';
```

#### PSPI

### Stage 1 and stage 2 predicates:

Rows retrieved for a query go through two stages of processing. Certain predicates can be applied during the first stage of processing, whereas other cannot be applied until the second stage of processing. You can improve the performance of your queries by using predicates that can be applied during the first stage whenever possible.

#### PSPI

Predicates that can be applied during the first stage of processing are called *Stage 1 predicates*. These predicates are also sometimes said to be *sargable*. Similarly, predicates that cannot be applied until the second stage of processing are called *stage 2 predicates*, and sometimes described as *nonsargable* or *residual* predicates.

Whether a predicate is stage 1 or stage 2 depends on the following factors:

- The syntax of the predicate.
- Data type and length of constants or columns in the predicate.

A simple predicate whose syntax classifies it as indexable and stage 1 might not be indexable or stage 1 because of data types that are associated with the predicate. For example, a predicate that is associated with either columns or constants of the DECFLOAT data type is never treated as stage 1. Similarly a predicate that contains constants or columns whose lengths are too long also might not be stage 1 or indexable.

For example, the following predicate is not indexable:

```
CHARCOL<'ABCDEFGH', where CHARCOL is defined as CHAR(6)
```

The predicate is not indexable because the length of the column is shorter than the length of the constant.

The following predicate is not stage 1:

```
DECCOL>34.5e0, where DECCOL is defined as DECIMAL(18,2)
```

The predicate is not stage 1 because the precision of the decimal column is greater than 15.

- Whether DB2 evaluates the predicate before or after a join operation. A predicate that is evaluated after a join operation is always a stage 2 predicate.
- Join sequence.

The same predicate might be stage 1 or stage 2, depending on the join sequence. *Join sequence* is the order in which DB2 joins tables when it evaluates a query. The join sequence is not necessarily the same as the order in which the tables appear in the predicate.

For example, the predicate might be stage 1 or stage 2:

$T1.C1 = T2.C1 + 1$

If T2 is the first table in the join sequence, the predicate is stage 1, but if T1 is the first table in the join sequence, the predicate is stage 2.

You can determine the join sequence by executing EXPLAIN on the query and examining the resulting plan table.

All indexable predicates are stage 1. The predicate `C1 LIKE %BC` is stage 1, but is not indexable.

#### PSPI

### Boolean term predicates:

You can improve the performance of queries by choosing Boolean term predicates over non-Boolean term predicates for join operations whenever possible.

#### PSPI

A *Boolean term predicate* is a simple or compound predicate that, when it is evaluated false for a particular row, makes the entire WHERE clause false for that particular row.

For example, in the following query P1, P2 and P3 are simple predicates:

```
SELECT * FROM T1 WHERE P1 AND (P2 OR P3);
```

- P1 is a simple Boolean term predicate.
- P2 and P3 are simple non-Boolean term predicates.
- P2 OR P3 is a compound Boolean term predicate.
- P1 AND (P2 OR P3) is a compound Boolean term predicate.

In single-index processing, only Boolean term predicates are chosen for matching predicates. Hence, only indexable Boolean term predicates are candidates for matching index scans. To match index columns by predicates that are not Boolean terms, DB2 considers multiple-index access.

In join operations, Boolean term predicates can reject rows at an earlier stage than can non-Boolean term predicates.

**Recommendation:** For join operations, choose Boolean term predicates over non-Boolean term predicates whenever possible.

#### PSPI

## Predicates in the ON clause

The ON clause supplies the join condition in an outer join. For a full outer join, the clause can use only equal predicates. For other outer joins, the clause can use any predicates except predicates that contain subqueries.

#### PSPI

For left and right outer joins, and for inner joins, join predicates in the ON clause are treated the same as other stage 1 and stage 2 predicates. A stage 2 predicate in the ON clause is treated as a stage 2 predicate of the inner table.

For full outer join, the ON clause is evaluated during the join operation like a stage 2 predicate.

In an outer join, predicates that are evaluated after the join are stage 2 predicates. Predicates in a table expression can be evaluated before the join and can therefore be stage 1 predicates.

For example, in the following statement, the predicate `EDLEVEL > 100` is evaluated before the full join and is a stage 1 predicate:

```
SELECT * FROM (SELECT * FROM DSN8910.EMP
               WHERE EDLEVEL > 100) AS X FULL JOIN DSN8910.DEPT
               ON X.WORKDEPT = DSN8910.DEPT.DEPTNO;
```

PSPI

---

## Using predicates efficiently

By following certain rules for how you write predicates, you can improve how DB2 processes SQL statements.

PSPI

To use predicates most efficiently in SQL statements:

- Use stage 1 predicates whenever possible. Stage 1 predicates are better than stage 2 predicates because they disqualify rows earlier and reduce the amount of processing that is needed at stage 2. In terms of resource usage, the earlier a predicate is evaluated, the better.
- Write queries to evaluate the most restrictive predicates first. When predicates with a high filter factor are processed first, unnecessary rows are screened as early as possible, which can reduce processing cost at a later stage. However, a predicate's restrictiveness is only effective among predicates of the same type and at the same evaluation stage.

PSPI

## When DB2 evaluates predicates

Two sets of rules determine the order of predicate evaluation.

PSPI

The first set of rules describes the order of predicate evaluation by stage:

1. Indexable predicates are applied first. All matching predicates on index key columns are applied first and evaluated when the index is accessed.  
Next, stage 1 predicates that have not been picked as matching predicates, but still refer to index columns, are applied to the index. This is called *index screening*.
2. Other stage 1 predicates are applied next.  
After data page access, stage 1 predicates are applied to the data.
3. Finally, the stage 2 predicates are applied on the returned data rows.

The second set of rules describes the order of predicate evaluation within each of the stages:

1. All equal predicates (including column *IN list*, where *list* has only one element, or column *BETWEEN value1 AND value1*) are evaluated.
2. All range predicates and predicates of the form *column IS NOT NULL* are evaluated.
3. All other predicate types are evaluated.

After both sets of rules are applied, predicates are evaluated in the order in which they appear in the query. Because you specify that order, you have some control over the order of evaluation.

**Exception:** Regardless of coding order, non-correlated subqueries are evaluated before correlated subqueries, unless DB2 correlates, de-correlates, or transforms the subquery into a join.



## Summary of predicate processing

The following table lists many of the simple predicates and tells whether those predicates are indexable or stage 1.

<b>PSPI</b>	The following terms are used:
<i>subq</i>	A correlated or noncorrelated subquery
<i>noncor subq</i>	A non-correlated subquery
<i>cor subq</i>	A correlated subquery
<i>op</i>	any of the operators >, >=, <, <=, ^>, ^<
<i>value</i>	A constant, host variable, or special register.
<i>pattern</i>	Any character string that does <i>not</i> start with the special characters for percent (%) or underscore (_).
<i>char</i>	Any character string that does <i>not</i> include the special characters for percent (%) or underscore (_).
<i>expression</i>	Any expression that contains arithmetic operators, scalar functions, aggregate functions, concatenation operators, columns, constants, host variables, special registers, or date or time expressions.
<i>noncol expr</i>	A non-column expression, which is any expression that does not contain a column. That expression can contain arithmetic operators, scalar functions, concatenation operators, constants, host variables, special registers, or date or time expressions.
	An example of a non-column expression is CURRENT DATE - 50 DAYS
<i>Tn col expr</i>	An expression that contains a column in table <i>Tn</i> . The expression might be only that column.
<i>predicate</i>	A predicate of any type.

In general, if you form a compound predicate by combining several simple predicates with OR operators, the result of the operation has the same characteristics as the simple predicate that is evaluated latest. For example, if two indexable predicates are combined with an OR operator, the result is indexable. If a stage 1 predicate and a stage 2 predicate are combined with an OR operator, the result is stage 2. Any predicate that is associated with the DECFLOAT data type is neither stage 1 nor indexable.

Table 45. Predicate types and processing

Predicate Type	Index- able?	Stage 1?	Notes
COL = <i>value</i>	Y	Y	16 on page 257
COL = <i>noncol expr</i>	Y	Y	9 on page 256, 11 on page 256, 12 on page 257, 15 on page 257
COL IS NULL	Y	Y	20 on page 258, 21 on page 258
COL <i>op value</i>	Y	Y	13 on page 257
COL <i>op noncol expr</i>	Y	Y	9 on page 256, 11 on page 256, 12 on page 257, 13 on page 257
COL BETWEEN <i>value1</i> AND <i>value2</i>	Y	Y	13 on page 257
COL BETWEEN <i>noncol expr1</i> AND <i>noncol expr2</i>	Y	Y	9 on page 256, 11 on page 256, 12 on page 257, 13 on page 257, 23 on page 258
<i>value</i> BETWEEN COL1 AND COL2	N	N	
COL BETWEEN COL1 AND COL2	N	N	10 on page 256
COL BETWEEN <i>expression1</i> AND <i>expression2</i>	Y	Y	6 on page 256, 7 on page 256, 11 on page 256, 12 on page 257, 13 on page 257, 14 on page 257, 15 on page 257, 27 on page 258
COL LIKE ' <i>pattern</i> '	Y	Y	5 on page 256
COL IN ( <i>list</i> )	Y	Y	17 on page 257, 18 on page 257
COL <> <i>value</i>	N	Y	8 on page 256, 11 on page 256
COL <> <i>noncol expr</i>	N	Y	8 on page 256, 11 on page 256
COL IS NOT NULL	Y	Y	21 on page 258
COL NOT BETWEEN <i>value1</i> AND <i>value2</i>	N	Y	
COL NOT BETWEEN <i>noncol exp1</i> AND <i>noncol expr2</i>	N	Y	
<i>value</i> NOT BETWEEN COL1 AND COL2	N	N	
COL NOT IN ( <i>list</i> )	N	Y	
COL NOT LIKE ' <i>char</i> '	N	Y	5 on page 256

Table 45. Predicate types and processing (continued)

Predicate Type	Index- able?	Stage 1?	Notes
COL LIKE '%char'	N	Y	1 on page 256, 5 on page 256
COL LIKE '_char'	N	Y	1 on page 256, 5 on page 256
COL LIKE <i>host variable</i>	Y	Y	2 on page 256, 5 on page 256
T1.COL = T2 <i>col expr</i>	Y	Y	6 on page 256, 9 on page 256, 11 on page 256, 12 on page 257, 14 on page 257, 15 on page 257, 25 on page 258, 27 on page 258
T1.COL <i>op</i> T2 <i>col expr</i>	Y	Y	6 on page 256, 9 on page 256, 11 on page 256, 12 on page 257, 13 on page 257, 14 on page 257, 15 on page 257
T1.COL <> T2 <i>col expr</i>	N	Y	8 on page 256, 11 on page 256, 27 on page 258
T1.COL1 = T1.COL2	N	N	3 on page 256, 25 on page 258
T1.COL1 <i>op</i> T1.COL2	N	N	3 on page 256
T1.COL1 <> T1.COL2	N	N	3 on page 256
COL=( <i>noncor subq</i> )	Y	Y	
COL = ANY ( <i>noncor subq</i> )	Y	Y	22 on page 258
COL = ALL ( <i>noncor subq</i> )	N	N	
COL <i>op</i> ( <i>noncor subq</i> )	Y	Y	28 on page 258
COL <i>op</i> ANY ( <i>noncor subq</i> )	Y	Y	22 on page 258
COL <i>op</i> ALL ( <i>noncor subq</i> )	Y	Y	
COL <> ( <i>noncor subq</i> )	N	Y	
COL <> ANY ( <i>noncor subq</i> )	N	N	22 on page 258
COL <> ALL ( <i>noncor subq</i> )	N	N	
COL IN ( <i>noncor subq</i> )	Y	Y	19 on page 258, 24 on page 258
(COL1,...COLn) IN ( <i>noncor subq</i> )	Y	Y	
COL NOT IN ( <i>noncor subq</i> )	N	N	
(COL1,...COLn) NOT IN ( <i>noncor subq</i> )	N	N	
COL = ( <i>cor subq</i> )	N	N	4 on page 256
COL = ANY ( <i>cor subq</i> )	Y	Y	19 on page 258, 22 on page 258
COL = ALL ( <i>cor subq</i> )	N	N	
COL <i>op</i> ( <i>cor subq</i> )	N	N	4 on page 256
COL <i>op</i> ANY ( <i>cor subq</i> )	N	N	22 on page 258

Table 45. Predicate types and processing (continued)

Predicate Type	Index- able?	Stage 1?	Notes
COL <i>op</i> ALL ( <i>cor subq</i> )	N	N	
COL <> ( <i>cor subq</i> )	N	N	4 on page 256
COL <> ANY ( <i>cor subq</i> )	N	N	22 on page 258
COL <> ALL ( <i>cor subq</i> )	N	N	
COL IN ( <i>cor subq</i> )	Y	Y	19 on page 258
(COL1,...COLn) IN ( <i>cor subq</i> )	N	N	
COL NOT IN ( <i>cor subq</i> )	N	N	
(COL1,...COLn) NOT IN ( <i>cor subq</i> )	N	N	
COL IS DISTINCT FROM <i>value</i>	N	Y	8 on page 256, 11 on page 256
COL IS NOT DISTINCT FROM <i>value</i>	Y	Y	16 on page 257
COL IS DISTINCT FROM <i>noncol expr</i>	N	Y	8 on page 256, 11 on page 256
COL IS NOT DISTINCT FROM <i>noncol expr</i>	Y	Y	9 on page 256, 11 on page 256, 12 on page 257, 15 on page 257
T1.COL1 IS DISTINCT FROM T2.COL2	N	N	3 on page 256
T1.COL1 IS NOT DISTINCT FROM T2.COL2	N	N	3 on page 256
T1.COL1 IS DISTINCT FROM T2 <i>col expr</i>	N	Y	8 on page 256, 11 on page 256
T1.COL1 IS NOT DISTINCT FROM T2 <i>col expr</i>	Y	Y	6 on page 256, 9 on page 256, 11 on page 256, 12 on page 257, 14 on page 257, 15 on page 257
COL IS DISTINCT FROM ( <i>noncor subq</i> )	N	Y	
COL IS NOT DISTINCT FROM ( <i>noncor subq</i> )	Y	Y	
COL IS DISTINCT FROM ANY ( <i>noncor subq</i> )	N	N	22 on page 258
COL IS NOT DISTINCT FROM ANY ( <i>noncor subq</i> )	N	N	22 on page 258
COL IS DISTINCT FROM ALL ( <i>noncor subq</i> )	N	N	
COL IS NOT DISTINCT FROM ALL ( <i>noncor subq</i> )	N	N	4 on page 256
COL IS NOT DISTINCT FROM ( <i>cor subq</i> )	N	N	4 on page 256
COL IS DISTINCT FROM ANY ( <i>cor subq</i> )	N	N	22 on page 258
COL IS DISTINCT FROM ANY ( <i>cor subq</i> )	N	N	22 on page 258
COL IS NOT DISTINCT FROM ANY ( <i>cor subq</i> )	N	N	22 on page 258
COL IS DISTINCT FROM ALL ( <i>cor subq</i> )	N	N	

Table 45. Predicate types and processing (continued)

Predicate Type	Index- able?	Stage 1?	Notes
COL IS NOT DISTINCT FROM ALL ( <i>cor subq</i> )	N	N	
EXISTS ( <i>subq</i> )	N	N	19 on page 258
NOT EXISTS ( <i>subq</i> )	N	N	
<i>expression</i> = <i>value</i>	N	N	27 on page 258
<i>expression</i> <> <i>value</i>	N	N	27 on page 258
<i>expression op value</i>	N	N	27 on page 258
<i>expression op (subq)</i>	N	N	
XML EXISTS	Y	N	26 on page 258
NOT XML EXISTS	N	N	

**Notes:**

1. Indexable only if an ESCAPE character is specified and used in the LIKE predicate. For example, COL LIKE '+%char' ESCAPE '+' is indexable.
2. Indexable only if the pattern in the host variable is an indexable constant (for example, host variable='char%').
3. If both COL1 and COL2 are from the same table, access through an index on either one is not considered for these predicates. However, the following query is an exception:

```
SELECT * FROM T1 A, T1 B WHERE A.C1 = B.C2;
```

By using correlation names, the query treats one table as if it were two separate tables. Therefore, indexes on columns C1 and C2 are considered for access.

4. If the subquery has already been evaluated for a given correlation value, then the subquery might not have to be reevaluated.
5. Not indexable or stage 1 if a field procedure exists on that column.
6. The column on the left side of the join sequence must be in a different table from any columns on the right side of the join sequence.
7. The tables that contain the columns in *expression1* or *expression2* must already have been accessed.
8. The processing for WHERE NOT COL = *value* is like that for WHERE COL <> *value*, and so on.
9. If *noncol expr*, *noncol expr1*, or *noncol expr2* is a noncolumn expression of one of these forms, then the predicate is not indexable:
  - *noncol expr* + 0
  - *noncol expr* - 0
  - *noncol expr* \* 1
  - *noncol expr* / 1
  - *noncol expr* CONCAT *empty string*
10. COL, COL1, and COL2 can be the same column or different columns. The columns are in the same table.
11. Any of the following sets of conditions make the predicate stage 2:

- The first value obtained before the predicate is evaluated is DECIMAL( $p,s$ ), where  $p>15$ , and the second value obtained before the predicate is evaluated is REAL or FLOAT.
  - The first value obtained before the predicate is evaluated is CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC, and the second value obtained before the predicate is evaluated is DATE, TIME, or TIMESTAMP.
12. The predicate is stage 1 but not indexable if the first value obtained before the predicate is evaluated is CHAR or VARCHAR, the second value obtained before the predicate is evaluated is GRAPHIC or VARGRAPHIC, and the first value obtained before the predicate is evaluated is not Unicode mixed.
  13. If both sides of the comparison are strings, any of the following sets of conditions makes the predicate stage 1 but not indexable:
    - The first value obtained before the predicate is evaluated is CHAR or VARCHAR, and the second value obtained before the predicate is evaluated is GRAPHIC or VARGRAPHIC.
    - Both of the following conditions are true:
      - Both sides of the comparison are CHAR or VARCHAR, or both sides of the comparison are BINARY or VARBINARY
      - The length the first value obtained before the predicate is evaluated is less than the length of the second value obtained before the predicate is evaluated.
    - Both of the following conditions are true:
      - Both sides of the comparison are GRAPHIC or VARGRAPHIC.
      - The length of the first value obtained before the predicate is evaluated is less than the length of the second value obtained before the predicate is evaluated.
    - Both of the following conditions are true:
      - The first value obtained before the predicate is evaluated is GRAPHIC or VARGRAPHIC, and the second value obtained before the predicate is evaluated is CHAR or VARCHAR.
      - The length of the first value obtained before the predicate is evaluated is less than the length of the second value obtained before the predicate is evaluated.
  14. If both sides of the comparison are strings, but the two sides have different CCSIDs, the predicate is stage 1 and indexable only if the first value obtained before the predicate is evaluated is Unicode and the comparison does not meet any of the conditions in note 13.
  15. Under either of these circumstances, the predicate is stage 2:
    - *noncol expr* is a case expression.
    - All of the following conditions are true:
      - *noncol expr* is the product or the quotient of two non-column expressions
      - *noncol expr* is an integer value
      - COL is a FLOAT or a DECIMAL column
  16. If COL has the ROWID data type, DB2 tries to use direct row access instead of index access or a table space scan.
  17. If COL has the ROWID data type, and an index is defined on COL, DB2 tries to use direct row access instead of index access.
  18. IN-list predicates are indexable and stage 1 if the following conditions are true:

- The IN list contains only simple items. For example, constants, host variables, parameter markers, and special registers.
- The IN list does not contain any aggregate functions or scalar functions.
- The IN list is not contained in a trigger's WHEN clause.
- For numeric predicates where the left side column is DECIMAL with precision greater than 15, none of the items in the IN list are FLOAT.
- For string predicates, the coded character set identifier is the same as the identifier for the left side column.
- For DATE, TIME, and TIMESTAMP predicates, the left side column must be DATE, TIME, or TIMESTAMP.

19. Certain predicates might become indexable and stage 1 depending on how they are transformed during processing.
20. The predicate types COL IS NULL and COL IS NOT NULL are stage 2 predicates when they query a column that is defined as NOT NULL.
21. If the predicate type is COL IS NULL and the column is defined as NOT NULL, the table is not accessed because C1 cannot be NULL.
22. The ANY and SOME keywords behave similarly. If a predicate with the ANY keyword is not indexable and not stage 1, a similar predicate with the SOME keyword is not indexable and not stage 1.
23. Under either of these circumstances, the predicate is stage 2:
  - *noncol expr* is a case expression.
  - *noncol expr* is the product or the quotient of two noncolumn expressions, that product or quotient is an integer value, and COL is a FLOAT or a DECIMAL column.
24. COL IN (*noncor subq*) is stage 1 for type N access only. Otherwise, it is stage 2.
25. If the inner table is an EBCDIC or ASCII column and the outer table is a Unicode column, the predicate is stage 1 and indexable.
26. The XMLEXISTS is always stage 2. But the same predicate can be indexable and become the matching predicate if an XML index can be used to evaluate the XPath expression in the predicate. The XMLEXISTS predicate can never be a screening predicate.
27. The predicate might be indexable by an index on expression if it contains an expression that is a column reference, invokes a built-in function, or contains a general expression.
28. This type of predicate is not stage 1 when a nullability mismatch is possible.

PSPI

## Examples of predicate properties

The included examples can help you to understand how and at which stage DB2 processes different predicates.

PSPI

Assume that predicate P1 and P2 are simple, stage 1, indexable predicates:  
 P1 AND P2 is a compound, stage 1, indexable predicate.  
 P1 OR P2 is a compound, stage 1 predicate, not indexable except by a union of RID lists from two indexes.

The following examples of predicates illustrate the general rules of predicate processing. In each case, assume that an index has been created on columns (C1,C2,C3,C4) of the table and that 0 is the lowest value in each column.

**WHERE C1=5 AND C2=7**

Both predicates are stage 1 and the compound predicate is indexable. A matching index scan could be used with C1 and C2 as matching columns.

**WHERE C1=5 AND C2>7**

Both predicates are stage 1 and the compound predicate is indexable. A matching index scan could be used with C1 and C2 as matching columns.

**WHERE C1>5 AND C2=7**

Both predicates are stage 1, but only the first matches the index. A matching index scan could be used with C1 as a matching column.

**WHERE C1=5 OR C2=7**

Both predicates are stage 1 but not Boolean terms. The compound is indexable. Multiple-index access for the compound predicate is not possible because no index has C2 as the leading column. For single-index access, C1 and C2 can be only index screening columns.

**WHERE C1=5 OR C2<>7**

The first predicate is indexable and stage 1, and the second predicate is stage 1 but not indexable. The compound predicate is stage 1 and not indexable.

**WHERE C1>5 OR C2=7**

Both predicates are stage 1 but not Boolean terms. The compound is indexable. Multiple-index access for the compound predicate is not possible because no index has C2 as the leading column. For single-index access, C1 and C2 can be only index screening columns.

**WHERE C1 IN (cor subq) AND C2=C1**

As written, both predicates are stage 2 and not indexable. The index is not considered for matching-index access, and both predicates are evaluated at stage 2. However, DB2 might transform the correlated subquery to a non-correlated subquery during processing, in which case both predicates become indexable and stage 1

**WHERE C1=5 AND C2=7 AND (C3 + 5) IN (7,8)**

The first two predicates only are stage 1 and indexable. The index is considered for matching-index access, and all rows satisfying those two predicates are passed to stage 2 to evaluate the third predicate.

**WHERE C1=5 OR C2=7 OR (C3 + 5) IN (7,8)**

The third predicate is stage 2. The compound predicate is stage 2 and all three predicates are evaluated at stage 2. The simple predicates are not Boolean terms and the compound predicate is not indexable.

**WHERE C1=5 OR (C2=7 AND C3=C4)**

The third predicate is stage 2. The two compound predicates (C2=7 AND C3=C4) and (C1=5 OR (C2=7 AND C3=C4)) are stage 2. All predicates are evaluated at stage 2.

**WHERE (C1>5 OR C2=7) AND C3 = C4**

The compound predicate (C1>5 OR C2=7) is indexable and stage 1. The simple predicate C3=C4 is not stage 1; so the index is not considered for matching-index access. Rows that satisfy the compound predicate (C1>5 OR C2=7) are passed to stage 2 for evaluation of the predicate C3=C4.

**WHERE C1= 17 and C2 <> 100**

In this example, assuming that a RANDOM ordering option has been specified on C2 in the CREATE INDEX statement, the query can use the index only in a limited way. The index is an effective filter on C1, but it

would not match on C2 because of the random values. The index is scanned for all values where C1=17 and only then ensure that values for C2 are not equal to 100.

```
WHERE (C1 = 1 OR C2 = 1) AND XMLEXISTS('/a/b[c = 1]' PASSING
XML_COL1) AND XMLEXISTS('/a/b[(e = 2 or f[g] = 3) and /h/i[j] = 4]' PASSING
XML_COL2)
```

The compound predicate (C1 = 1 OR C2 = 1) is indexable and stage 1. The first XMLEXISTS predicate is indexable and can become a matching predicate if the XML index /a/b/c has been created. The second XMLEXISTS predicate is indexable and can use multiple index access if the XML indexes, /a/b/e, /a/b/f/g, and /a/b/h/i/j, can be used to evaluate three XPath segments in the predicate. All rows satisfying the three indexable predicates (one compound and two XMLEXISTS) are passed to stage 2 to evaluate the same first and second XMLEXISTS predicates again.

PSPI

## Predicate filter factors

By understanding of how DB2 uses filter factors you can write more efficient predicates.

PSPI

The *filter factor* of a predicate is a number between 0 and 1 that estimates the proportion of rows in a table for which the predicate is true. Those rows are said to *qualify* by that predicate.

For example, suppose that DB2 can determine that column C1 of table T contains only five distinct values: A, D, Q, W and X. In the absence of other information, DB2 estimates that one-fifth of the rows have the value D in column C1. Then the predicate C1='D' has the filter factor 0.2 for table T.

### How DB2 uses filter factors:

DB2 uses filter factors to estimate the number of rows qualified by a set of predicates.

For simple predicates, the filter factor is a function of three variables:

- The constant value in the predicate; for instance, 'D' in the previous example.
- The operator in the predicate; for instance, '=' in the previous example and '<>' in the negation of the predicate.
- Statistics on the column in the predicate. In the previous example, those include the information that column T.C1 contains only five values.

**Recommendation:** Control the first two of those variables when you write a predicate. Your understanding of how DB2 uses filter factors should help you write more efficient predicates.

Values of the third variable, statistics on the column, are kept in the DB2 catalog. You can update many of those values, either by running the utility RUNSTATS or by executing UPDATE for a catalog table.

If you intend to update the catalog with statistics of your own choice, you should understand how DB2 uses filter factors and interpolation formulas.

PSPI

## Default filter factors for simple predicates

DB2 uses default filter factor values when no other statistics exist.

**PSPI** The following table lists default filter factors for different types of predicates.

Table 46. DB2 default filter factors by predicate type

Predicate Type	Filter Factor
Col = constant	1/25
Col <> constant	1 - (1/25)
Col IS NULL	1/25
Col IS NOT DISTINCT FROM	1/25
Col IS DISTINCT FROM	1 - (1/25)
Col IN (constant list)	(number of constants)/25
Col Op constant	1/3
Col LIKE constant	1/10
Col BETWEEN constant1 and constant2	1/10

**Note:** Op is one of these operators: <, <=, >, >=.

## Example

The default filter factor for the predicate C1 = 'D' is 1/25 (0.04). If D is actually not close to 0.04, the default probably does not lead to an optimal access path.

**PSPI**

## Filter factors for other predicate types:

Examples above represent only the most common types of predicates. If P1 is a predicate and F is its filter factor, then the filter factor of the predicate NOT P1 is (1 - F). But, filter factor calculation is dependent on many things, so a specific filter factor cannot be given for all predicate types. **PSPI**

## Filter factors for uniform distributions

In certain situations DB2 assumes that a data is distributed uniformly and calculates filter factors accordingly.

**PSPI** DB2 uses the filter factors in the following table if:

- The value in column COLCARD of catalog table SYSIBM.SYSCOLUMNS for the column "Col" is a positive value.
- No additional statistics exist for "Col" in SYSIBM.SYSCOLDIST.

Table 47. DB2 uniform filter factors by predicate type

Predicate type	Filter factor
Col = constant	1/COLCARD
Col <> constant	1 - (1/COLCARD)
Col IS NULL	1/COLCARD
Col IS NOT DISTINCT FROM	1/COLCARD


Table 47. DB2 uniform filter factors by predicate type (continued)

Predicate type	Filter factor
Col IS DISTINCT FROM	$1 - (1/\text{COLCARD})$
Col IN (constant list)	number of constants /COLCARD
Col <i>Op1</i> constant	interpolation formula
Col <i>Op2</i> constant	interpolation formula
Col LIKE constant	interpolation formula
Col BETWEEN constant1 and constant2	interpolation formula
<b>Notes:</b>	
<i>Op1</i> is < or <=, and the constant is not a host variable.	
<i>Op2</i> is > or >=, and the constant is not a host variable.	

## Example


If D is one of only five values in column C1, using RUNSTATS puts the value 5 in column COLCARD of SYSCOLUMNS. If no additional statistics are available, the filter factor for the predicate C1 = 'D' is 1/5 (0.2).

## Filter factors for other predicate types:

Examples above represent only the most common types of predicates. If P1 is a predicate and F is its filter factor, then the filter factor of the predicate NOT P1 is (1 - F). But, filter factor calculation is dependent on many things, so a specific filter factor cannot be given for all predicate types. 

## Interpolation formulas

For a predicate that uses a range of values, DB2 calculates the filter factor by an *interpolation formula*.

 The formula is based on an estimate of the ratio of the number of values in the range to the number of values in the entire column of the table.

## The formulas

The formulas that follow are rough estimates, which are subject to further modification by DB2. They apply to a predicate of the form *col op. constant*. The value of (Total Entries) in each formula is estimated from the values in columns HIGH2KEY and LOW2KEY in catalog table SYSIBM.SYSCOLUMNS for column *col*:  
Total Entries = (HIGH2KEY value - LOW2KEY value).

- For the operators < and <=, where the constant is not a host variable:  
 $(\text{constant value} - \text{LOW2KEY value}) / (\text{Total Entries})$
- For the operators > and >=, where the constant is not a host variable:  
 $(\text{HIGH2KEY value} - \text{constant value}) / (\text{Total Entries})$
- For LIKE or BETWEEN:  
 $(\text{High constant value} - \text{Low constant value}) / (\text{Total Entries})$

## Example

For column C2 in a predicate, suppose that the value of HIGH2KEY is 1400 and the value of LOW2KEY is 200. For C2, DB2 calculates *Total Entries* = 1400 - 200, or 1200.

For the predicate C1 BETWEEN 800 AND 1100, DB2 calculates the filter factor F as:

$$F = (1100 - 800)/1200 = 1/4 = 0.25$$

## Interpolation for LIKE

DB2 treats a LIKE predicate as a type of BETWEEN predicate. Two values that bound the range qualified by the predicate are generated from the constant string in the predicate. Only the leading characters found before the escape character ('%' or '\_') are used to generate the bounds. So if the escape character is the first character of the string, the filter factor is estimated as 1, and the predicate is estimated to reject no rows.

## Defaults for interpolation

DB2 might not interpolate in some cases; instead, it can use a default filter factor. Defaults for interpolation are:

- Relevant only for ranges, including LIKE and BETWEEN predicates
- Used only when interpolation is not adequate
- Based on the value of COLCARDF
- Used whether uniform or additional distribution statistics exist on the column if either of the following conditions is met:
  - The predicate does not contain constants
  - COLCARDF < 4.

The following table shows interpolation defaults for the operators <, <=, >, >= and for LIKE and BETWEEN.

Table 48. Default filter factors for interpolation

COLCARDF	Factor for Op	Factor for LIKE or BETWEEN
>=100000000	1/10,000	3/100000
>=10000000	1/3,000	1/10000
>=1000000	1/1,000	3/10000
>=100000	1/300	1/1000
>=10000	1/100	3/1000
>=1000	1/30	1/100
>=100	1/10	3/100
>=2	1/3	1/10
=1	1/1	1/1
<=0	1/3	1/10

**Note:** Op is one of these operators: <, <=, >, >=.



## Filter factors for all distributions

RUNSTATS can generate additional statistics for a column or set of columns. DB2 can use that information to calculate filter factors.

**PSPI** DB2 collects two kinds of distribution statistics:

### Frequency

The percentage of rows in the table that contain a value for a column or set of columns

### Cardinality

The number of distinct values in a set of columns

The table that follows lists the types of predicates on which these statistics are used.

Table 49. Predicates for which distribution statistics are used

Type of statistic	Single or concatenated columns	Predicates
Frequency	Single	COL= <i>constant</i> COL IS NULL COL IN ( <i>constant-list</i> ) COL <i>op</i> <i>constant</i> COL BETWEEN <i>constant</i> AND <i>constant</i> COL= <i>host-variable</i> COL1=COL2 T1.COL=T2.COL COL IS NOT DISTINCT FROM
Frequency	Concatenated	COL= <i>constant</i> COL IS NOT DISTINCT FROM
Cardinality	Single	COL= <i>constant</i> COL IS NULL COL IN ( <i>constant-list</i> ) COL <i>op</i> <i>constant</i> COL BETWEEN <i>constant</i> AND <i>constant</i> COL= <i>host-variable</i> COL1=COL2 T1.COL=T2.COL COL IS NOT DISTINCT FROM
Cardinality	Concatenated	COL= <i>constant</i> COL= <i>host-variable</i> COL1=COL2 COL IS NOT DISTINCT FROM

**Note:** *op* is one of these operators: <, <=, >, >=.

## How DB2 uses frequency statistics

Columns COLVALUE and FREQUENCYF in table SYSCOLDIST contain distribution statistics. Regardless of the number of values in those columns, running RUNSTATS deletes the existing values and inserts rows for frequent values.

You can run RUNSTATS without the FREQVAL option, with the FREQVAL option in the *correl-spec*, with the FREQVAL option in the *colgroup-spec*, or in both, with the following effects:

- If you run RUNSTATS without the FREQVAL option, RUNSTATS inserts rows for the 10 most frequent values for the first column of the specified index.
- If you run RUNSTATS with the FREQVAL option in the *correl-spec*, RUNSTATS inserts rows for concatenated columns of an index. The NUMCOLS option specifies the number of concatenated index columns. The COUNT option specifies the number of frequent values. You can collect most-frequent values, least-frequent values, or both.
- If you run RUNSTATS with the FREQVAL option in the *colgroup-spec*, RUNSTATS inserts rows for the columns in the column group that you specify. The COUNT option specifies the number of frequent values. You can collect most-frequent values, least-frequent values, or both.
- If you specify the FREQVAL option, RUNSTATS inserts rows for columns of the specified index and for columns in a column group.

DB2 uses the frequencies in column FREQUENCYF for predicates that use the values in column COLVALUE and assumes that the remaining data are uniformly distributed.

### Example: Filter factor for a single column

Suppose that the predicate is C1 IN ('3','5') and that SYSCOLDIST contains these values for column C1:


COLVALUE	FREQUENCYF
'3'	.0153
'5'	.0859
'8'	.0627

The filter factor is  $.0153 + .0859 = .1012$ .

### Example: Filter factor for correlated columns


Suppose that columns C1 and C2 are correlated. Suppose also that the predicate is C1='3' AND C2='5' and that SYSCOLDIST contains these values for columns C1 and C2:

COLVALUE	FREQUENCYF
'1' '1'	.1176
'2' '2'	.0588
'3' '3'	.0588
'3' '5'	.1176
'4' '4'	.0588
'5' '3'	.1764
'5' '5'	.3529
'6' '6'	.0588

The filter factor is .1176. 

### Histogram statistics

You can improve access path selection by specifying the HISTOGRAM option in RUNSTATS.

 RUNSTATS normally collects frequency statistics for a single-column or single multi-column data set. Because catalog space and bind time performance concerns make the collection of these types of statistics on every distinct value found in the target column or columns very impractical, such frequency statistics are commonly collected only on the most frequent or least frequent, and therefore most biased, values.

Such limited statistics often do not provide an accurate prediction of the value distribution because they require a rough interpolation across the entire range of values. For example, suppose that the YRS\_OF\_EXPERIENCE column on an EMPLOYEE table contains the following value frequencies:

*Table 50. Example frequency statistics for values on the YRS\_OF\_EXPERIENCE column in an EMPLOYEE table*

VALUE	FREQUENCY
2	10%
25	15%
26	15%
27	7%
12	0.02%
13	0.01%
40	0.0001%
41	0.00001%

### Example predicates that can benefit from histogram statistics

Some example predicates on values in this table include:

- Equality predicate with unmatched value:  

```
SELECT EMPID FROM EMPLOYEE T
WHERE T.YRS_OF_EXPERIENCE = 6;
```
- Range predicate:  

```
SELECT T.EMPID FROM EMPLOYEE T
WHERE T.YRS_OF_EXPERIENCE BETWEEN 5 AND 10;
```
- Non-local predicate:  

```
SELECT T1.EMPID FROM EMPLOYEE T1, OPENJOBS T2
WHERE T1.SPECIALTY = T2.AREA AND T1.YRS_OF_EXPERIENCE > T2.YRS_OF_EXPERIENCE;
```

For each of the above predicates, distribution statistics for any single value cannot help DB2 to estimate predicate selectivity, other than by uniform interpolation of filter factors over the uncollected part of the value range. The result of such interpolation might lead to inaccurate estimation and undesirable access path selection.

### How DB2 uses histogram statistics

DB2 creates a number of intervals such that each interval contains approximately the same percentage of rows from the data set. The number of intervals is specified by the value of NUMQUANTILES when you use the HISTOGRAM option of RUNSTATS. Each interval has an identifier value QUANTILENO, and values, the LOWVALUE and HIGHVALUE columns, that bound the interval. DB2 collects distribution statistics for each interval.

When you use RUNSTATS to collect statistics on a column that contains such wide-ranging frequency values, specify the histogram statistics option to collect more granular distribution statistics that account for the distribution of values across the entire range of values. The following table shows the result of collecting histogram statistics for the years of experience values in the employee table. In this example, the statistics have been collected with 7 intervals:

Table 51. Histogram statistics for the column YRS\_OF\_EXPERIENCE in an EMPLOYEE table.

QUANTILENO	LOWVALUE	HIGHVALUE	CARDF	FREQUENCYF
1	0	3	4	14%
2	4	15	8	14%
3	18	24	7	12%
4	25	25	1	15%
5	26	26	1	15%
6	27	30	4	16%
7	35	40	6	14%

#### PSPI

### How DB2 uses multiple filter factors to determine the cost of a query

When DB2 estimates the cost of a query, it determines the filter factor repeatedly and at various levels.

#### PSPI

For example, suppose that you execute the following query:

```
SELECT COLS FROM T1
  WHERE C1 = 'A'
  AND   C3 = 'B'
  AND   C4 = 'C';
```

Table T1 consists of columns C1, C2, C3, and C4. Index I1 is defined on table T1 and contains columns C1, C2, and C3.

Suppose that the simple predicates in the compound predicate have the following characteristics:

**C1='A'**

Matching predicate

**C3='B'** Screening predicate

**C4='C'**

Stage 1, nonindexable predicate

To determine the cost of accessing table T1 through index I1, DB2 performs these steps:


1. Estimates the matching index cost. DB2 determines the index matching filter factor by using single-column cardinality and single-column frequency statistics because only one column can be a matching column.
2. Estimates the total index filtering. This includes matching and screening filtering. If statistics exist on column group (C1,C3), DB2 uses those statistics. Otherwise DB2 uses the available single-column statistics for each of these columns.  
DB2 also uses FULLKEYCARDF as a bound. Therefore, it can be critical to have column group statistics on column group (C1, C3) to get an accurate estimate.
3. Estimates the table-level filtering. If statistics are available on column group (C1,C3,C4), DB2 uses them. Otherwise, DB2 uses statistics that exist on subsets of those columns.

**Important:** If you supply appropriate statistics at each level of filtering, DB2 is more likely to choose the most efficient access path.

You can use RUNSTATS to collect any of the needed statistics. 

### Filter factor estimation for the XMLEXISTS predicate


When the statistics are available for the XML NODEID index, from the NODEID index, the value FIRSTKEYCARD is the number of distinct DOCID values in the XML table.

 FIRSTKEYCARD/CARDF(base table) transforms the filtering of an XMLEXISTS predicate from the XML table to the base table. The filter factors of all XPath predicates in the XMLEXISTS predicate indicate the filtering on the XML table. After filter factors are multiplied with FIRSTKEYCARD/CARDF(base table), the filter factors on the XML tables are transformed to the filter factors on the base table.

From the NODEID index statistics, if the value FULLKEYCARD is available, the FULLKEYCARD is used as the default value of CARDF of the XML table.

When the statistics are available for the XML index, the value FIRSTKEYCARD can be used as the COLCARD of the comparison operand in the XPath predicates. For each comparison type, the following rules are used to calculate the filter factor for the XPath predicates:


- A default filter factor, without any statistic information, is the same as non-XPath predicates with the same comparison type.
- If the value index can be used to evaluate the XPath predicate, the default filter factor is redefined based on the FIRSTKEYCARD value. The filter factor is the same as non-XPath predicates with the same comparison type and the same COLCARD.
- Because the frequency statistics are not available, for range predicates, interpolation is performed based on HIGH2KEY/LOW2KEY and the key value of the comparison. When no statistics are available for the NODEID index and the value index, no statistics exist from indexes to facilitate the filter factor estimation for the XPath predicates in the XMLEXISTS predicate. The following default statistics are used for the value index in the index costing:
  - NLEAF reuses the current formula  $NLEAF = CARDF(xml\_table) / 300$
  - NLEVELS uses the current default value 1 for the NODEID index and the value index for the XMLEXISTS predicate. Because the index statistics are not available to help the default filter factor estimation, the predicate filter factor is set according to the predicate comparison type. FIRSTKEYCARD/

CARDF(base table) is set to value 1. 

## Avoiding problems with correlated columns

Two columns in a table are said to be *correlated* if the values in the columns do not vary independently.



DB2 might not determine the best access path when your queries include correlated columns. 

## Correlated columns

Two columns of data, A and B of a single table, are *correlated* if the values in column A do not vary independently of the values in column B.

**PSPI** For example, the following table is an excerpt from a large single table. Columns CITY and STATE are highly correlated, and columns DEPTNO and SEX are entirely independent.

Table 52. Data from the CREWINFO table

CITY	STATE	DEPTNO	SEX	EMPNO	ZIPCODE
Fresno	CA	A345	F	27375	93650
Fresno	CA	J123	M	12345	93710
Fresno	CA	J123	F	93875	93650
Fresno	CA	J123	F	52325	93792
New York	NY	J123	M	19823	09001
New York	NY	A345	M	15522	09530
Miami	FL	B499	M	83825	33116
Miami	FL	A345	F	35785	34099
Los Angeles	CA	X987	M	12131	90077
Los Angeles	CA	A345	M	38251	90091

In this simple example, for every value of column CITY that equals 'FRESNO', the STATE column contains the same value ('CA'). **PSPI**

### Impacts of correlated columns:

DB2 might not determine the best access path, table order, or join method when your query uses columns that are highly correlated.

**PSPI** Column correlation can make the estimated cost of operations cheaper than they actually are. Correlated columns affect both single-table queries and join queries.

### Column correlation on the best matching columns of an index

The following query selects rows with females in department A345 from Fresno, California. Two indexes are defined on the table, Index 1 (CITY,STATE,ZIPCODE) and Index 2 (DEPTNO,SEX).

#### Query 1

```
SELECT ... FROM CREWINFO WHERE
  CITY = 'FRESNO' AND STATE = 'CA'           (PREDICATE1)
  AND DEPTNO = 'A345' AND SEX = 'F';         (PREDICATE2)
```

Consider the two compound predicates (labeled PREDICATE1 and PREDICATE2), their actual filtering effects (the proportion of rows they select), and their DB2 filter factors. Unless the proper catalog statistics are gathered, the filter factors are calculated as if the columns of the predicate are entirely independent (not correlated).

When the columns in a predicate correlate but the correlation is not reflected in catalog statistics, the actual filtering effect to be significantly different from the DB2 filter factor. The following table shows how the actual filtering effect and the DB2 filter factor can differ, and how that difference can affect index choice and performance.

Table 53. Effects of column correlation on matching columns

	INDEX 1	INDEX 2
Matching predicates	Predicate1 CITY=FRESNO AND STATE=CA	Predicate2 DEPTNO=A345 AND SEX=F
Matching columns	2	2
DB2 estimate for matching columns (Filter Factor)	column=CITY, COLCARDF=4 Filter Factor=1/4 column=STATE, COLCARDF=3 Filter Factor=1/3	column=DEPTNO, COLCARDF=4 Filter Factor=1/4 column=SEX, COLCARDF=2 Filter Factor=1/2
Compound filter factor for matching columns	$1/4 \times 1/3 = 0.083$	$1/4 \times 1/2 = 0.125$
Qualified leaf pages based on DB2 estimations	$0.083 \times 10 = 0.83$ INDEX CHOSEN (.8 < 1.25)	$0.125 \times 10 = 1.25$
Actual filter factor based on data distribution	4/10	2/10
Actual number of qualified leaf pages based on compound predicate	$4/10 \times 10 = 4$	$2/10 \times 10 = 2$ BETTER INDEX CHOICE (2 < 4)

DB2 chooses an index that returns the fewest rows, partly determined by the smallest filter factor of the matching columns. Assume that filter factor is the only influence on the access path. The combined filtering of columns CITY and STATE seems very good, whereas the matching columns for the second index do not seem to filter as much. Based on those calculations, DB2 chooses Index 1 as an access path for Query 1.

The problem is that the filtering of columns CITY and STATE should not look good. Column STATE does almost no filtering. Since columns DEPTNO and SEX do a better job of filtering out rows, DB2 should favor Index 2 over Index 1.

#### Column correlation on index screening columns of an index

Correlation might also occur on nonmatching index columns, used for index screening. See “Nonmatching index scan (ACCESSTYPE='I' and MATCHCOLS=0)” on page 671 for more information. Index screening predicates help reduce the number of data rows that qualify while scanning the index. However, if the index screening predicates are correlated, they do not filter as many data rows as their filter factors suggest. To illustrate this, use Query 1 with the following indexes on Table 52 on page 269:

Index 3 (EMPNO,CITY,STATE)  
Index 4 (EMPNO,DEPTNO,SEX)

In the case of Index 3, because the columns CITY and STATE of Predicate 1 are correlated, the index access is not improved as much as estimated by the screening predicates and therefore Index 4 might be a better choice. (Note that index screening also occurs for indexes with matching columns greater than zero.)

### Multiple table joins

In Query 2, the data shown in the following table is added to the original query (see Query 1) to show the impact of column correlation on join queries.


Table 54. Data from the DEPTINFO table

CITY	STATE	MANAGER	DEPT	DEPTNAME
Fresno	CA	Smith	J123	ADMIN
Los Angeles	CA	Jones	A345	LEGAL

#### Query 2

```
SELECT ... FROM CREWINFO T1,DEPTINFO T2
WHERE T1.CITY = 'FRESNO' AND T1.STATE='CA'           (PREDICATE 1)
AND T1.DEPTNO = T2.DEPT AND T2.DEPTNAME = 'LEGAL';
```

The order that tables are accessed in a join statement affects performance. The estimated combined filtering of Predicate1 is lower than its actual filtering. So table CREWINFO might look better as the first table accessed than it should.

Also, due to the smaller estimated size for table CREWINFO, a nested loop join might be chosen for the join method. But, if many rows are selected from table CREWINFO because Predicate1 does not filter as many rows as estimated, then another join method or join sequence might be better. 

### Detecting correlated columns

The first indication that correlated columns might be a problem is poor response times when DB2 has chosen an inappropriate access path. If you suspect that you have problems with correlated columns, you can issue SQL statements to test whether columns are correlated.

To determine whether columns are correlated:

Issue SQL statements and compare the results as shown in the following example.

#### PSPI

If you suspect two columns in a table, such as the CITY and STATE columns in the CREWINFO table might be correlated, then you can issue the following SQL queries that reflect the relationships between the columns:

```
SELECT COUNT (DISTINCT CITY) AS CITYCOUNT,
COUNT (DISTINCT STATE) AS STATECOUNT FROM CREWINFO;
```

The result of the count of each distinct column is the value of COLCARD in the DB2 catalog table SYSCOLUMNS. Multiply the previous two values together to get a preliminary result:

CITYCOUNT x STATECOUNT = ANSWER1

Then issue the following SQL statement:

```
SELECT COUNT(*) FROM
  (SELECT DISTINCT CITY, STATE
   FROM CREWINFO) AS V1;      (ANSWER2)
```

Compare the result of the previous count (ANSWER2) with ANSWER1. If ANSWER2 is less than ANSWER1, then the suspected columns are correlated.

**PSPI**

## What to do about correlated columns

When correlated columns cause DB2 to choose an inappropriate access path, you can use several techniques to try to alter the access path.

**PSPI**

To address problems with correlated columns:

- For leading indexed columns, run the RUNSTATS utility with the KEYCARD option determine the column correlation. For all other column groups, run the RUNSTATS utility with the COLGROUP option
- Run the RUNSTATS utility to collect column correlation information for any column group with the COLGROUP option.
- Update the catalog statistics manually.
- Use SQL that forces access through a particular index.

The RUNSTATS utility collects the statistics that DB2 needs to make proper choices about queries. With RUNSTATS, you can collect statistics on the concatenated key columns of an index and the number of distinct values for those concatenated columns. This gives DB2 accurate information to calculate the filter factor for the query.

For example, RUNSTATS collects statistics that benefit queries like this:

```
SELECT * FROM T1
WHERE C1 = 'a' AND C2 = 'b' AND C3 = 'c' ;
```

Where:

- The first three index keys are used (MATCHCOLS = 3).
- An index exists on C1, C2, C3, C4, C5.
- Some or all of the columns in the index are correlated in some way.

**PSPI**

## DB2 predicate manipulation

In some specific cases, DB2 either modifies some predicates, or generates extra predicates. Although these modifications are transparent to you, they have a direct impact on the access path selection and your PLAN\_TABLE results.

**PSPI**

This manipulation occurs because DB2 always uses an index access path when it is cost effective. Generating extra predicates provides more indexable predicates potentially, which creates more chances for an efficient index access path.

Therefore, to understand your PLAN\_TABLE results, you must understand how DB2 manipulates predicates. The information in Table 45 on page 253 is also

helpful. **PSPI**

## Predicate modifications for IN-list predicates

If an IN-list predicate has only one item in its list, the predicate becomes an EQUAL predicate.

### PSPI

A set of simple, Boolean term, equal predicates on the same column that are connected by OR predicates can be converted into an IN-list predicate. For

example: C1=5 or C1=10 or C1=15 converts to C1 IN (5,10,15).

### PSPI

## When DB2 simplifies join operations

DB2 can simplify a join operation when the query contains a predicate or an ON clause that eliminates the null values that are generated by the join operation. When DB2 encounters a join operation that it can simplify, it attempts to do so.

### PSPI

However, because full outer joins are less efficient than left or right joins, and left and right joins are less efficient than inner joins, you should always try to use the simplest type of join operation in your queries.

## Example: ON clause that eliminates null values

Consider the following query:

```
SELECT * FROM T1 X FULL JOIN T2 Y
  ON X.C1=Y.C1
 WHERE X.C2 > 12;
```

The outer join operation yields these result table rows:

- The rows with matching values of C1 in tables T1 and T2 (the inner join result)
- The rows from T1 where C1 has no corresponding value in T2
- The rows from T2 where C1 has no corresponding value in T1

However, when you apply the predicate, you remove all rows in the result table that came from T2 where C1 has no corresponding value in T1. DB2 transforms the full join into a left join, which is more efficient:

```
SELECT * FROM T1 X LEFT JOIN T2 Y
  ON X.C1=Y.C1
 WHERE X.C2 > 12;
```

## Example: Predicate that eliminates null values

The predicate, X.C2>12, filters out all null values that result from the right join:

```
SELECT * FROM T1 X RIGHT JOIN T2 Y
  ON X.C1=Y.C1
 WHERE X.C2>12;
```

Therefore, DB2 can transform the right join into a more efficient inner join without changing the result:

```
SELECT * FROM T1 X INNER JOIN T2 Y
  ON X.C1=Y.C1
 WHERE X.C2>12;
```

### Example: Predicates that follow join operations

The predicate that follows a join operation must have the following characteristics before DB2 transforms an outer join into a simpler outer join or inner join:

- The predicate is a Boolean term predicate.
- The predicate is false if one table in the join operation supplies a null value for all of its columns.

These predicates are examples of predicates that can cause DB2 to simplify join operations:

```
T1.C1 > 10
T1.C1 IS NOT NULL
T1.C1 > 10 OR T1.C2 > 15
T1.C1 > T2.C1
T1.C1 IN (1,2,4)
T1.C1 LIKE 'ABC%'
T1.C1 BETWEEN 10 AND 100
12 BETWEEN T1.C1 AND 100
```

### Example: ON clause that eliminates unmatched values

This examples shows how DB2 can simplify a join operation because the query contains an ON clause that eliminates rows with unmatched values:

```
SELECT * FROM T1 X LEFT JOIN T2 Y
  FULL JOIN T3 Z ON Y.C1=Z.C1
  ON X.C1=Y.C1;
```

Because the last ON clause eliminates any rows from the result table for which column values that come from T1 or T2 are null, DB2 can replace the full join with a more efficient left join to achieve the same result:

```
SELECT * FROM T1 X LEFT JOIN T2 Y
  LEFT JOIN T3 Z ON Y.C1=Z.C1
  ON X.C1=Y.C1;
```

### Example: Full outer join processed as a left outer join


In one case, DB2 transforms a full outer join into a left join when you cannot write code to do it. This is the case where a view specifies a full outer join, but a subsequent query on that view requires only a left outer join.

**Example:** Consider this view:

```
CREATE VIEW V1 (C1,T1C2,T2C2) AS
  SELECT COALESCE(T1.C1, T2.C1), T1.C2, T2.C2
  FROM T1 X FULL JOIN T2 Y
  ON T1.C1=T2.C1;
```

This view contains rows for which values of C2 that come from T1 are null. However, if you execute the following query, you eliminate the rows with null values for C2 that come from T1:

```
SELECT * FROM V1
  WHERE T1C2 > 10;
```

Therefore, for this query, a left join between T1 and T2 would have been adequate. DB2 can execute this query as if the view V1 was generated with a left outer join so that the query runs more efficiently. 

## Predicates generated through transitive closure

When the set of predicates that belong to a query logically imply other predicates, DB2 can generate additional predicates to provide more information for access path selection.

### Rules for generating predicates

**PSPI** For single-table or inner join queries, DB2 generates predicates for transitive closure if any of the following conditions are true:

- The query has an equal type predicate: COL1=COL2. This could be:
  - A local predicate
  - A join predicate
- The query also has a Boolean term predicate on one of the columns in the first predicate, with one of the following formats:
  - COL1 *op value*  
*op* is =, <>, >, >=, <, or <=.  
*value* is a constant, host variable, or special register.
  - COL1 (NOT) BETWEEN *value1* AND *value2*
  - COL1=COL3

For outer join queries, DB2 generates predicates for transitive closure if the query has an ON clause of the form COL1=COL2 which comes before a join predicate that has one of the following formats:

- COL1 *op value*  
*op* is =, <>, >, >=, <, or <=
- COL1 (NOT) BETWEEN *value1* AND *value2*

DB2 generates a transitive closure predicate for an outer join query only if the generated predicate does not reference the table with unmatched rows. That is, the generated predicate cannot reference the left table for a left outer join or the right table for a right outer join.

For a multiple-CCSID query, DB2 does not generate a transitive closure predicate if the predicate that would be generated has any of the following characteristics:

- The generated predicate is a range predicate (*op* is >, >=, <, or <=).
- Evaluation of the query with the generated predicate results in different CCSID conversion from evaluation of the query without the predicate.

When a predicate meets the transitive closure conditions, DB2 generates a new predicate, whether or not it already exists in the WHERE clause.

The generated predicates have one of the following formats:

- COL *op value*  
*op* is =, <>, >, >=, <, or <=.  
*value* is a constant, host variable, or special register.
- COL (NOT) BETWEEN *value1* AND *value2*
- COL1=COL2 (for single-table or inner join queries only)

DB2 does not generate a predicate through transitive closure for any predicate that is associated with the DECFLOAT data type (column or constant).

Example of transitive closure for an inner join: Suppose that you have written this query, which meets the conditions for transitive closure:

```
SELECT * FROM T1, T2
WHERE T1.C1=T2.C1 AND
      T1.C1>10;
```

DB2 generates an additional predicate to produce this query, which is more efficient:

```
SELECT * FROM T1, T2
WHERE T1.C1=T2.C1 AND
      T1.C1>10 AND
      T2.C1>10;
```

### Example of transitive closure for an outer join

Suppose that you have written this outer join query:

```
SELECT * FROM
  (SELECT T1.C1 FROM T1 WHERE T1.C1>10) X
LEFT JOIN
  (SELECT T2.C1 FROM T2) Y
ON X.C1 = Y.C1;
```

The before join predicate, T1.C1>10, meets the conditions for transitive closure, so DB2 generates a query that has the same result as this more-efficient query:

```
SELECT * FROM
  (SELECT T1.C1 FROM T1 WHERE T1.C1>10) X
LEFT JOIN
  (SELECT T2.C1 FROM T2 WHERE T2.C1>10) Y
ON X.C1 = Y.C1;
```

### Predicate redundancy

A predicate is redundant if evaluation of other predicates in the query already determines the result that the predicate provides. You can specify redundant predicates or DB2 can generate them. However, DB2 does not determine that any of your query predicates are redundant. All predicates that you code are evaluated at execution time regardless of whether they are redundant. In contrast, if DB2 generates a redundant predicate to help select access paths, that predicate is ignored at execution.

### Adding extra predicates

DB2 performs predicate transitive closure only on equal and range predicates. However, you can help DB2 to choose a better access path by adding transitive closure predicates for other types of operators, such as IN or LIKE. For example, consider the following SELECT statement:

```
SELECT * FROM T1,T2
WHERE T1.C1=T2.C1
      AND T1.C1 LIKE 'A'
```

If T1.C1=T2.C1 is true, and T1.C1 LIKE 'A%' is true, then T2.C1 LIKE 'A%' must also be true. Therefore, you can give DB2 extra information for evaluating the query by adding T2.C1 LIKE 'A%':

```
SELECT * FROM T1,T2
WHERE T1.C1=T2.C1
      AND T1.C1 LIKE 'A%'
      AND T2.C1 LIKE 'A'
```

## Transformation of SQL predicates to XML predicates

DB2 sometimes transforms an SQL query to change the timing at which a predicate is applied to improve the performance of the query. DB2 might use such a transformation to push SQL predicates into the XPath expression embedded in the XMLTABLE function.

For example, a query finds all books that were published after 1991 and lists the year, title and publisher for each book.

```
SELECT X.*
FROM T1,
XMLTABLE('/bib/book'
  PASSING T1.bib_xml
  COLUMNS YEAR INT PATH '@year',
           TITLE VARCHAR(30) PATH 'title',
           PUBLISHER VARCHAR(30) PATH 'publisher') X
WHERE X.YEAR > 1991;
```

DB2 can rewrite the query to process the WHERE X.YEAR > 1991 predicate in the XMLTABLE function. In the rewritten query the original predicate becomes an XPath predicate that is associated with the *row-xpath-expression* of the XMLTABLE function:

```
SELECT X.*
FROM T1,
XMLTABLE('/bib/book[@year>1991]'
  PASSING T1.bib_xml
  COLUMNS YEAR INT PATH '@year',
           TITLE VARCHAR(30) PATH 'title',
           PUBLISHER VARCHAR(30) PATH 'publisher') X
```

## Implications of truncation and trailing blanks

Unlike SQL, in which trailing blanks have no significance, in XPath trailing blanks are significant. For example, the following query contains an additional predicate, X.publisher = 'Addison-Wesley':

```
SELECT *
FROM T1,
XMLTABLE('/bib/book'
  PASSING T1.bib_xml
  COLUMNS year INT PATH '@year',
           title VARCHAR(30) PATH 'title',
           publisher VARCHAR(30) PATH 'publisher') X
WHERE X.year > 1991
AND X.publisher = 'Addison-Wesley';
```

Because of the possible truncation when a publisher is cast to varchar(30), and the possibility of trailing blanks in the original XML data, DB2 must add an internal operator, db2:rtrim, to simulate the SQL semantics in order to push the predicate into XPath. As shown below, The predicate X.publisher = 'Addison-Wesley' is transformed into [db2:rtrim(publisher,30)="Addison-Wesley"] internally.

```
SELECT * FROM T1, XMLTABLE('/bib/book[@year>1991]
[db2:rtrim(publisher,30)="Addison-Wesley"]'
  PASSING T1.bib_xml
  COLUMNS year INT PATH '@year',
           title VARCHAR(30) PATH 'title',
           publisher VARCHAR(10) PATH 'publisher') X;
```

**Note:** The db2:rtrim operator is not supported externally, and the preceding SQL statement cannot be run, but it is used here to illustrate internal DB2 processing.

## Predicates that are eligible for transformation to XML predicates in XMLTABLE

A predicate that satisfies the following criteria is eligible for transformation to be processed by the XMLTABLE function:

- The predicate must have one of the following forms: (Where *op* stands for any of the following operators: =, <, >, <=, >=, or <>.)
    - Column *op* constant, parameter, or host variable, where the column is from the result table.
    - Column *op* column, where the column on the left hand side is from the result table and the column on the right hand side is from either the result table or one of the input tables.
    - Column *op* expression, where the column is from the result table and the expression is any SQL expression that only contains columns from the input table.
    - A BETWEEN predicate that can be transformed into one of the above forms.
    - COLUMN IS (NOT) NULL
    - A predicate that is composed of the above forms combined with AND and OR.
    - COLUMN (NOT) IN (expression 1, ..., expression *n*), where the column is from the result table and each of the expressions on either a column from the result table or an SQL expression that contains neither columns from the result table nor columns from a table that is not an input table.
  - The predicate is a boolean term predicate.
  - The predicate can be applied before any join operations.
  - The result column of the XMLTABLE function that is involved in the predicate is not of any of the following data types:
    - DATE
    - TIME
    - TIMESTAMP
    - DECFLOAT(16)
    - REAL
    - DOUBLE
- This restriction does not apply to IS (NOT) NULL predicate.
- The result column of the XMLTABLE function involved in the predicate does not have a default clause.
  - The XMLTABLE function does not have a FOR ORDINALITY column.

## Predicates with encrypted data

DB2 provides built-in functions for data encryption and decryption. These functions can secure sensitive data, but they can also degrade the performance of some statements if they are not used carefully.

**PSPI** If a predicate contains any operator other than = and <>, encrypted data must be decrypted before comparisons can be made. Decryption makes the predicates stage 2. **PSPI**

---

## Using host variables efficiently

When host variables or parameter markers are used in a query, the actual values are not known when you bind the package or plan that contains the query. DB2 therefore uses a default filter factor to determine the best access path for an SQL statement. If that access path proves to be inefficient, you can do several things to obtain a better access path.

**PSPI** Host variables require default filter factors. When you bind a static SQL statement that contains host variables, DB2 uses a default filter factor to determine the best access path for the SQL statement. DB2 often chooses an access path that performs well for a query with several host variables. However, in a new release or after maintenance has been applied, DB2 might choose a new access path that does not perform as well as the old access path. In many cases, the change in access paths is due to the default filter factors, which might lead DB2 to optimize the query in a different way.

To change the access path for a query that contains host variables, use one of the following options:

- Bind the package or plan that contains the query with the options REOPT(ALWAYS), REOPT(AUTO), or REOPT(ONCE).
- Rewrite the query. **PSPI**

## Changing the access path at run time

You can use bind options to control how DB2 determines the access path for SQL statements that contain variable values.

**PSPI** To control how DB2 determines the access path for SQL statements with variable values:

Specify one of the following bind options:

### REOPT(ALWAYS)

DB2 determines the access path for any SQL statement with variable values each time the statement is run.

### REOPT(AUTO)

DB2 automatically determines if a new access path is required to further optimize the performance of a statement for each execution.

REOPT(AUTO) only applies to dynamic statements that can be cached. If dynamic statement caching is turned off and DB2 executes a statement that is bound with REOPT(AUTO), no optimization occurs.

### REOPT(ONCE)

DB2 determines and caches the access path for any SQL statement with variable values only once at run time, using the first set of input variable values. If the statement is run multiple times, DB2 does not re-optimize each time unless the cached statement is invalidated or removed from the cache.

### REOPT(NONE)

DB2 determines the access path at bind time, and does not change the access path at run time. **PSPI**

## Specifying the REOPT(ALWAYS) bind option

You can specify a bind option when you want DB2 to determine access paths at both bind time and run time for statements that contain host variables and special registers

**PSPI** At run time, DB2 uses the values in these variables to determine the access paths. If the statement runs multiple times, DB2 determines the access path each time that the statement runs.

Consider using the REOPT(ALWAYS) bind option in the following circumstances:

- The SQL statement does not perform well with the access path that is chosen at bind time.
- The SQL statement takes a relatively long time to execute. For long-running SQL statements, the performance gain from the better access path that is chosen based on the input variable values for each run can be greater than the performance cost of reoptimizing the access path each time that the statement runs.

To use the REOPT(ALWAYS) bind option most efficiently: first determine which SQL statements in your applications perform poorly with the REOPT(NONE) bind option and the REOPT(ONCE) bind option. Separate the code containing those statements into units that you bind into packages with the REOPT(ALWAYS) option. Bind the rest of the code into packages using the REOPT(NONE) bind option or the REOPT(ONCE) bind option, as appropriate.

1. Determine which SQL statements in your applications perform poorly with the REOPT(NONE) bind option and the REOPT(ONCE) bind option.
2. Separate the code containing those statements into units that you bind into packages with the REOPT(ALWAYS) option.
3. Bind the rest of the code into packages using the REOPT(NONE) bind option or the REOPT(ONCE) bind option, as appropriate.
4. Then bind the plan with the REOPT(NONE) bind option.

Statements in the packages bound with REOPT(ALWAYS) are candidates for repeated reoptimization at run time.

### Example:

To determine which queries in plans and packages that are bound with the REOPT(ALWAYS) bind option will be reoptimized at run time, execute the following SELECT statements:

```
SELECT PLNAME,
       CASE WHEN STMTNOI <> 0
            THEN STMTNOI
            ELSE STMTNO
       END AS STMTNUM,
       SEQNO, TEXT
FROM   SYSIBM.SYSSTMT
WHERE  STATUS IN ('B','F','G','J')
ORDER BY PLNAME, STMTNUM, SEQNO;

SELECT COLLID, NAME, VERSION,
       CASE WHEN STMTNOI <> 0
            THEN STMTNOI
            ELSE STMTNO
       END AS STMTNUM,
       SEQNO, STMT
```

```
FROM SYSIBM.SYSPACKSTMT
WHERE STATUS IN ('B','F','G','J')
ORDER BY COLLID, NAME, VERSION, STMTNUM, SEQNO;
```

If you specify the bind option `VALIDATE(RUN)`, and a statement in the plan or package is not bound successfully, that statement is incrementally bound at run time. If you also specify the bind option `REOPT(ALWAYS)`, DB2 reoptimizes the access path during the incremental bind.

### Example:

To determine which plans and packages have statements that will be incrementally bound, execute the following `SELECT` statements:

```
SELECT DISTINCT NAME
  FROM SYSIBM.SYSSTMT
 WHERE STATUS = 'F' OR STATUS = 'H';
SELECT DISTINCT COLLID, NAME, VERSION
  FROM SYSIBM.SYSPACKSTMT
 WHERE STATUS = 'F' OR STATUS = 'H';
```

**PSPI**

## Specifying the `REOPT(AUTO)` bind option

You can specify the `REOPT(AUTO)` bind option when you want DB2 to automatically determine whether reoptimization is required at run time.

**PSPI**

For statements cached dynamic statements that reference parameter markers, DB2 can generate a new access path at any execution, depending on whether and how the parameter marker values change. The newly generated access path is cached in the statement cache, replacing the old access path for the query.

Consider using the `REOPT(AUTO)` bind option to achieve a better balance between the costs of reoptimization and the costs of processing a statement. You might use the `REOPT(AUTO)` bind options for many statements for which you could choose either the `REOPT(ALWAYS)` or `REOPT(NONE)` bind options, and especially in the following situations:

- The statement is a dynamic SQL statement and can be cached. If dynamic statement caching is not turned on when DB2 executes a statement specified with the `REOPT(AUTO)` bind option, no reoptimization occurs.
- The SQL statement sometimes takes a relatively long time to execute, depending on the values of referenced parameter markers, especially when parameter markers refer to non-uniform data that is joined to other tables. For such SQL statements, the performance gain from a new access path that is chosen based on the input variable values for each might or might not be greater than the performance cost of reoptimization when the statement runs.

**PSPI**

## Specifying the `REOPT(ONCE)` bind option

You can use the `REOPT(ONCE)` bind option to determine the access path for an SQL statement at run time.

**PSPI**

The `REOPT(ONCE)` bind option determines the access path for an SQL statement only once at run time and works only with dynamic SQL statements.

The REOPT(ONCE) bind option allows DB2 to store the access path for dynamic SQL statements in the dynamic statement cache.

Consider using the REOPT(ONCE) bind option in the following circumstances:

- The SQL statement is a dynamic SQL statement.
- The SQL statement does not perform well with the access path that is chosen at bind time.
- The SQL statement is relatively simple and takes a relatively short time to execute. For simple SQL statements, reoptimizing the access path each time that the statement runs can degrade performance more than using the access path from the first run for each subsequent run.
- The same SQL statement is repeated many times in a loop, or is run by many threads. Because of the dynamic statement cache, the access path that DB2 chooses for the first set of input variables performs well for subsequent executions of the same SQL statement, even if the input variable values are different each time.

To use the REOPT(ONCE) bind option most efficiently:

1. Determine which dynamic SQL statements in your applications perform poorly with the REOPT(NONE) bind option and the REOPT(ALWAYS) bind option.
2. Separate the code that contains those statements and bind it into packages with the REOPT(ONCE) option. A dynamic statement in a package that is bound with REOPT(ONCE) is a candidate for reoptimization the first time that the statement is run.
3. Bind the rest of the code into packages using the REOPT(NONE) bind option or the REOPT(ALWAYS) bind option, as appropriate.
4. Bind the plan with the REOPT(NONE) bind option.

#### **Example:**

To determine which queries in plans and packages that are bound with the REOPT(ONCE) bind option will be reoptimized at run time, execute the following SELECT statements:

```
SELECT PLNAME,
       CASE WHEN STMTNOI <> 0
         THEN STMTNOI
         ELSE STMTNO
       END AS STMTNUM,
       SEQNO, TEXT
FROM   SYSIBM.SYSSTMT
WHERE  STATUS IN ('J')
ORDER BY PLNAME, STMTNUM, SEQNO;

SELECT COLLID, NAME, VERSION,
       CASE WHEN STMTNOI <> 0
         THEN STMTNOI
         ELSE STMTNO
       END AS STMTNUM,
       SEQNO, STMT
FROM   SYSIBM.SYSPACKSTMT
WHERE  STATUS IN ('J')
ORDER BY COLLID, NAME, VERSION, STMTNUM, SEQNO;
```

#### **Example:**

If you specify the bind option `VALIDATE(RUN)`, and a statement in the plan or package is not bound successfully, that statement is incrementally bound at run time.

To determine which plans and packages have statements that will be incrementally bound, execute the following `SELECT` statements:

```
SELECT DISTINCT NAME
  FROM SYSIBM.SYSSTMT
 WHERE STATUS = 'F' OR STATUS = 'H';
SELECT DISTINCT COLLID, NAME, VERSION
  FROM SYSIBM.SYSPACKSTMT
 WHERE STATUS = 'F' OR STATUS = 'H';
```

PSPI

### Specifying the `REOPT(NONE)` bind option

You should use the `REOPT(NONE)` bind option when an SQL statement with variable values performs well with the access path that is chosen at bind time.

PSPI

Keep in mind that an SQL statement that performs well with the `REOPT(NONE)` bind option might perform even better with the bind options that change the access path at run time.

PSPI

---

## Writing efficient subqueries

A *subquery* is a `SELECT` statement within the `WHERE` or `HAVING` clause of an `INSERT`, `UPDATE`, `MERGE`, or `DELETE` SQL statement. By understanding how DB2 processes subqueries, you can estimate the best method to use when writing a given query when several methods can achieve the same result.

In many cases two or more different SQL statements can achieve identical results, particularly those that contain subqueries. The statements have different access paths, however, and probably perform differently.

Subqueries might also contain their own subqueries. Such *nested subqueries* can be either correlated or non-correlated. DB2 uses the same processing techniques with nested subqueries that it does for non-nested subqueries, and the same optimization techniques apply.

No absolute rules exist for deciding how or whether to code a subquery. DB2 might transform one type of subquery to another, depending on the optimizer estimation.

To ensure the best performance from SQL statements that contain subqueries:

Follow these general guidelines:

- If efficient indexes are available on the tables in the subquery, then a correlated subquery is likely to be the most efficient kind of subquery.
- If no efficient indexes are available on the tables in the subquery, then a non-correlated subquery would be likely to perform better.
- If multiple subqueries are in any parent query, make sure that the subqueries are ordered in the most efficient manner.

## Example

Assume that MAIN\_TABLE has 1000 rows:


```
SELECT * FROM MAIN_TABLE
WHERE TYPE IN (subquery 1) AND
      PARTS IN (subquery 2);
```

Assuming that subquery 1 and subquery 2 are the same type of subquery (either correlated or non-correlated) and the subqueries are stage 2, DB2 evaluates the subquery predicates in the order they appear in the WHERE clause. Subquery 1 rejects 10% of the total rows, and subquery 2 rejects 80% of the total rows:

- The predicate in subquery 1 (which is referred to as P1) is evaluated 1000 times, and the predicate in subquery 2 (which is referred to as P2) is evaluated 900 times, for a total of 1900 predicate checks. However, if the order of the subquery predicates is reversed, P2 is evaluated 1000 times, but P1 is evaluated only 200 times, for a total of 1200 predicate checks.
- Coding P2 before P1 appears to be more efficient if P1 and P2 take an equal amount of time to execute. However, if P1 is 100 times faster to evaluate than P2, then coding subquery 1 first might be advisable. If you notice a performance degradation, consider reordering the subqueries and monitoring the results.

If you are unsure, run EXPLAIN on the query with both a correlated and a non-correlated subquery. By examining the EXPLAIN output and understanding your data distribution and SQL statements, you should be able to determine which form is more efficient.

This general principle can apply to all types of predicates. However, because subquery predicates can potentially be thousands of times more processor- and I/O-intensive than all other predicates, the order of subquery predicates is particularly important.


Regardless of coding order, DB2 performs non-correlated subquery predicates before correlated subquery predicates, unless the subquery is transformed into a join. 

### Related concepts

“DB2 predicate manipulation” on page 272

## Correlated and non-correlated subqueries

Different subqueries require different approaches for efficient processing by DB2.

 All subqueries can be classified into either two categories: correlated and non-correlated

### Correlated subqueries

Correlated subqueries contain a reference to a table or column that is outside of the scope of the subquery.

In the following query, for example, the correlation name X is a value from a table that is not listed in the FROM clause of the subquery. The inclusion of X illustrates that the subquery references the outer query block:

```
SELECT * FROM DSN8910.EMP X
WHERE JOB = 'DESIGNER'
AND EXISTS (SELECT 1
```

```

FROM   DSN8910.PROJ
WHERE  DEPTNO = X.WORKDEPT
      AND MAJPROJ = 'MA2100');

```

## Non-correlated subqueries

Non-correlated subqueries do not refer to any tables or columns that are outside of the scope of the subquery.

The following example query refers only to tables are within the scope of the FROM clause.

```

SELECT * FROM DSN8910.EMP
WHERE  JOB = 'DESIGNER'
      AND WORKDEPT IN (SELECT DEPTNO
                       FROM   DSN8910.PROJ
                       WHERE  MAJPROJ = 'MA2100');

```

PSPI

## When DB2 transforms a subquery into a join

For a SELECT, UPDATE, or DELETE statement, DB2 can sometimes transform a subquery into a join between the result table of the subquery and the result table of the outer query.

PSPI

## SELECT statements

For a SELECT statement, DB2 transforms the subquery into a join if the following conditions are true:

- The transformation does not introduce redundancy.
- The subquery appears in a WHERE clause.
- The subquery does not contain GROUP BY, HAVING, ORDER BY, FETCH FIRST *n* ROWS ONLY, or aggregate functions.
- The subquery has only one table in the FROM clause.
- For a correlated subquery, the comparison operator of the predicate containing the subquery is IN, = ANY, or = SOME.
- For a noncorrelated subquery, the comparison operator of the predicate containing the subquery is IN, EXISTS, = ANY, or = SOME.
- For a noncorrelated subquery, the subquery select list has only one column, guaranteed by a unique index to have unique values.
- For a noncorrelated subquery, the left side of the predicate is a single column with the same data type and length as the subquery's column. (For a correlated subquery, the left side can be any expression.)
- Query parallelism is NOT enabled.

## Example

The following subquery can be transformed into a join because it meets the above conditions for transforming a SELECT statement:

```

SELECT * FROM EMP
WHERE DEPTNO IN
  (SELECT DEPTNO FROM DEPT
   WHERE LOCATION IN ('SAN JOSE', 'SAN FRANCISCO')
   AND DIVISION = 'MARKETING');

```

If a department in the marketing division has branches in both San Jose and San Francisco, the result of the SQL statement is not the same as if a join were performed. The join makes each employee in this department appear twice because it matches once for the department of location San Jose and again of location San Francisco, although it is the same department. Therefore, it is clear that to transform a subquery into a join, the uniqueness of the subquery select list must be guaranteed. For this example, a unique index on any of the following sets of columns would guarantee uniqueness:

- (DEPTNO)
- (DIVISION, DEPTNO)
- (DEPTNO, DIVISION).

The resulting query is:

```

SELECT EMP.* FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO AND
      DEPT.LOCATION IN ('SAN JOSE', 'SAN FRANCISCO') AND
      DEPT.DIVISION = 'MARKETING';

```

## UPDATE, DELETE, and other SELECT statements

For an UPDATE or DELETE statement, or a SELECT statement that does not meet the previous conditions for transformation, DB2 transforms a correlated subquery into a join if the following conditions are true:

- The transformation does not introduce redundancy.
- The subquery is correlated to its immediate outer query.
- The FROM clause of the subquery contains only one table, and the outer query for SELECT, UPDATE, or DELETE references only one table.
- If the outer predicate is a quantified predicate with an operator of =ANY or an IN predicate, and the following conditions are true:
  - The left side of the outer predicate is a single column.
  - The right side of the outer predicate is a subquery that references a single column.
  - The two columns have the same data type and length.
- The subquery does not contain the GROUP BY or DISTINCT clauses.
- The subquery does not contain aggregate functions.
- The SELECT clause of the subquery does not contain a user-defined function with an external action or a user-defined function that modifies data.
- The subquery predicate is a Boolean term predicate.
- The predicates in the subquery that provide correlation are stage 1 predicates.
- The subquery does not contain nested subqueries.
- The subquery does not contain a self-referencing UPDATE or DELETE.
- For a SELECT statement, the query does not contain the FOR UPDATE OF clause.
- For an UPDATE or DELETE statement, the statement is a searched UPDATE or DELETE.
- For a SELECT statement, parallelism is not enabled.

For a statement with multiple subqueries, DB2 transforms only the last subquery in the statement that qualifies for transformation.

### Example

The following subquery can be transformed into a join because it meets the above conditions for UPDATE statements

```
UPDATE T1 SET T1.C1 = 1
  WHERE T1.C1 = ANY
    (SELECT T2.C1 FROM T2
     WHERE T2.C2 = T1.C2);
```

PSPI

## When DB2 Correlates and de-correlates subqueries

Correlated and non-correlated subqueries have different processing advantages. Where possible, the DB2 optimizer transforms the query to the most efficient type, especially when a subquery does not meet the conditions for transformation into a join.

PSPI

The DB2 optimizer might transform a correlated query to a non-correlated, or *de-correlate* the subquery, to improve processing efficiency. Likewise, the Optimizer might *correlate* a non-correlated subquery. When a correlated and non-correlated subquery can achieve the same result, the most efficient way depends on the data.

The Optimizer chooses to correlate or de-correlate subqueries based on cost. Correlated subqueries allow more filtering to be done within the subquery. Non-correlated subqueries allow more filtering to be done on the table whose columns are being compared to the subquery result.

DB2 might correlate a non-correlated subquery, or de-correlate a correlated subquery, that cannot be transformed into a join to improve access path selection and processing efficiency.

### Example:

DB2 can transform the following non-correlated subquery into a correlated subquery:

```
SELECT * FROM T1
  WHERE T1.C1 IN (SELECT T2.C1 FROM T2, T3
                 WHERE T2.C1 = T3.C1)
```

Can be transformed to:

```
SELECT * FROM T1
  WHERE EXISTS (SELECT 1 FROM T2, T3
               WHERE T2.C1 = T3.C1 AND T2.C1 = T1.C1)
```

Some queries cannot be transformed from one form to another. Most set functions and grouping functions make it difficult to transform a subquery from one form to another. Expressions that can prevent such transformation include:

### Set functions and grouping functions

Most set functions and grouping functions make it difficult to transform a subquery from one form to another.

### Example:

In the following query, the non-correlated subquery cannot be correlated to T1 because it would change the result of the SUM function. Consequently, only the non-correlated form of the query can be considered.

```
SELECT * FROM T1
WHERE T1.C2 IN (SELECT SUM(T2.C2) FROM T2, T3
               WHERE T2.C1 = T3.C1
               GROUP BY T2.C1)
```

### Correlated ranges and <> comparisons

Some range comparisons involving correlated columns make it difficult to de-correlate the subquery. This is because when a correlated subquery is de-correlated we might have to remove duplicates in order to consider the virtual table in the outer position (see “Early Out” Processing). This duplicate removal requires a set of “equal-join” predicate columns as the key. Without equal-join predicates the early out process breaks down and doesn’t work. That means the virtual table can only be considered in correlated form (as the inner table of the join).

### Example:

DB2 cannot de-correlate the following query and use it to access T1 because removing duplicates on the T2.C2 subquery result does not guarantee that the range predicate correlation does not qualify multiple rows from T1.

```
SELECT * FROM T1
WHERE EXISTS (SELECT 1 FROM T2, T3
              WHERE T2.C1 = T3.C1 AND T2.C2 > T1.C2 AND T2.C2 < T1.C3)
```

PSPI

## Subquery tuning

DB2 automatically performs some subquery tuning by subquery to join transformation and through subquery correlation and de-correlation.

PSPI

However, you should be aware of the differences among the subqueries such as those in the following examples. You might need to code a query in one of the ways below for performance reasons that stem from restrictions to DB2 in transforming a given query, or restrictions to the DB2 optimizer accurately estimating the cost of the various transformation choices.

Each of the following three queries retrieves the same rows. All three retrieve data about all designers in departments that are responsible for projects that are part of major project MA2100. These three queries show that you can retrieve a desired result in several ways.

### Query A: A join of two tables

```
SELECT DISTINCT DSN8910.EMP.* FROM DSN8910.EMP, DSN8910.PROJ
WHERE JOB = 'DESIGNER'
AND WORKDEPT = 'DEPTNO'
AND MAJPROJ = 'MA2100';
```

## Query B: A correlated subquery


```
SELECT * FROM DSN8910.EMP X
WHERE JOB = 'DESIGNER'
AND EXISTS (SELECT 1 FROM DSN8910.PROJ
            WHERE DEPTNO = X.WORKDEPT
            AND MAJPROJ = 'MA2100');
```

## Query C: A noncorrelated subquery

```
SELECT * FROM DSN8910.EMP
WHERE JOB = 'DESIGNER'
AND WORKDEPT IN (SELECT DEPTNO FROM DSN8910.PROJ
                WHERE MAJPROJ = 'MA2100');
```

## Choosing between a subquery and a join

If you need columns from both tables EMP and PROJ in the output, you must use the join. Query A might be the one that performs best, and as a general practice you should code a subquery as a join whenever possible. However, in this example, PROJ might contain duplicate values of DEPTNO in the subquery, so that an equivalent join cannot be written. In that case, whether the correlated or non-correlated form is most efficient depends upon where the application of each predicate in the subquery provides the most benefit.

When looking at a problematic subquery, check if the query can be rewritten into another format, especially as a join, or if you can create an index to improve the performance of the subquery. Consider the sequence of evaluation for the different subquery predicates and for all other predicates in the query. If one subquery predicate is costly, look for another predicate that could be evaluated first to reject more rows before the evaluation of problem subquery predicate. 

---

## Using scrollable cursors efficiently

Scrollable cursors are a valuable tool for writing applications such as screen-based applications, in which the result table is small and you often move back and forth through the data.

 **PSPI**

To get the best performance from your scrollable cursors:

- Determine when scrollable cursors work best for you.  
Scrollable cursors require more DB2 processing than non-scrollable cursors. If your applications require large result tables or you only need to move sequentially forward through the data, use non-scrollable cursors.
- Declare scrollable cursors as SENSITIVE only if you need to see the latest data.  
If you do not need to see updates that are made by other cursors or application processes, using a cursor that you declare as INSENSITIVE requires less processing by DB2.  
If you need to see only some of the latest updates, and you do not need to see the results of insert operations, declare scrollable cursors as SENSITIVE STATIC.  
If you need to see all of the latest updates and inserts, declare scrollable cursors as SENSITIVE DYNAMIC.
- To ensure maximum concurrency when you use a scrollable cursor for positioned update and delete operations, specify ISOLATION(CS) and CURRENTDATA(NO) when you bind packages and plans that contain updatable scrollable cursors.


- Use the `FETCH FIRST n ROWS ONLY` clause with scrollable cursors when it is appropriate. In a distributed environment, when you need to retrieve a limited number of rows, `FETCH FIRST n ROWS ONLY` can improve your performance for distributed queries that use DRDA access by eliminating unneeded network traffic.

In a local environment, if you need to scroll through a limited subset of rows in a table, you can use `FETCH FIRST n ROWS ONLY` to make the result table smaller.

- In a distributed environment, if you do not need to use your scrollable cursors to modify data, do your cursor processing in a stored procedure. Using stored procedures can decrease the amount of network traffic that your application requires.
- In a work file database, create table spaces that are large enough for processing your scrollable cursors.

DB2 uses declared temporary tables for processing the following types of scrollable cursors:


- `SENSITIVE STATIC SCROLL`
- `INSENSITIVE SCROLL`
- `ASENSITIVE SCROLL`, if the cursor sensitivity is `INSENSITIVE`. A cursor that meets the criteria for a read-only cursor has an effective sensitivity of `INSENSITIVE`.

- Remember to commit changes often for the following reasons:
  - You frequently need to leave scrollable cursors open longer than non-scrollable cursors.
  - An increased chance of deadlocks with scrollable cursors occurs because scrollable cursors allow rows to be accessed and updated in any order. Frequent commits can decrease the chances of deadlocks.
  - To prevent cursors from closing after commit operations, declare your scrollable cursors `WITH HOLD`.
- While sensitive static sensitive scrollable cursors are open against a table, DB2 disallows reuse of space in that table space to prevent the scrollable cursor from fetching newly inserted rows that were not in the original result set. Although this is normal, it can result in a seemingly false out-of-space indication. The problem can be more noticeable in a data sharing environment with transactions that access LOBs. Consider the following preventive measures:
  - Check applications such that they commit frequently
  - Close sensitive scrollable cursors when no longer needed
  - Remove `WITH HOLD` parm for the sensitive scrollable cursor, if possible
  - Isolate LOB table spaces in a dedicated bufferpool in the data sharing environment 

---

## Efficient queries for tables with data-partitioned secondary indexes

The number of partitions that DB2 accesses to evaluate a query predicate can affect the performance of the query. A query that provides data retrieval through a data-partitioned secondary index (DPSI) might access some or all partitions of the DPSI.

 For a query that is based only on a DPSI key value or range, DB2 must examine all partitions. If the query also has predicates on the leading columns of

the partitioning key, DB2 does not need to examine all partitions. The removal from consideration of inapplicable partitions is known as *page range screening* or *limited partition scan*.

A limited partition scan can be determined at bind time or at run time. For example, a limited partition scan can be determined at bind time for a predicate in which a column is compared to a constant. A limited partition scan occurs at run time if the column is compared to a host variable, parameter marker, or special register.

### Example: limited partition scan

The following example demonstrates how you can use a partitioning index to enable a limited partition scan on a set of partitions that DB2 needs to examine to satisfy a query predicate.

Suppose that you create table Q1, with partitioning index DATE\_IX and DPSI STATE\_IX:

```
CREATE TABLESPACE TS1 Numparts 3;

CREATE TABLE Q1 (DATE DATE,
  CUSTNO CHAR(5),
  STATE CHAR(2),
  PURCH_AMT DECIMAL(9,2))
  IN TS1
  PARTITION BY (DATE)
  (PARTITION 1 ENDING AT ('2002-1-31'),
   PARTITION 2 ENDING AT ('2002-2-28'),
   PARTITION 3 ENDING AT ('2002-3-31'));

CREATE INDEX DATE_IX ON Q1 (DATE) PARTITIONED CLUSTER;

CREATE INDEX STATE_IX ON Q1 (STATE) PARTITIONED;
```

Now suppose that you want to execute the following query against table Q1:

```
SELECT CUSTNO, PURCH_AMT
FROM Q1
WHERE STATE = 'CA';
```

Because the predicate is based only on values of a DPSI key (STATE), DB2 must examine all partitions to find the matching rows.

Now suppose that you modify the query in the following way:

```
SELECT CUSTNO, PURCH_AMT
FROM Q1
WHERE DATE BETWEEN '2002-01-01' AND '2002-01-31' AND
STATE = 'CA';
```

Because the predicate is now based on values of a partitioning index key (DATE) and on values of a DPSI key (STATE), DB2 can eliminate the scanning of data partitions 2 and 3, which do not satisfy the query for the partitioning key. This can be determined at bind time because the columns of the predicate are compared to constants.

Now suppose that you use host variables instead of constants in the same query:

```
SELECT CUSTNO, PURCH_AMT
FROM Q1
WHERE DATE BETWEEN :hv1 AND :hv2 AND
STATE = :hv3;
```

DB2 can use the predicate on the partitioning column to eliminate the scanning of unneeded partitions at run time.

### Example: limited partition scan when correlation exists

Writing queries to take advantage of limited partition scan is especially useful when a correlation exists between columns that are in a partitioning index and columns that are in a DPSI.

For example, suppose that you create table Q2, with partitioning index DATE\_IX and DPSI ORDERNO\_IX:

```
CREATE TABLESPACE TS2 Numparts 3;

CREATE TABLE Q2 (DATE DATE,
  ORDERNO CHAR(8),
  STATE CHAR(2),
  PURCH_AMT DECIMAL(9,2))
  IN TS2
  PARTITION BY (DATE)
  (PARTITION 1 ENDING AT ('2004-12-31'),
   PARTITION 2 ENDING AT ('2005-12-31'),
   PARTITION 3 ENDING AT ('2006-12-31'));

CREATE INDEX DATE_IX ON Q2 (DATE) PARTITIONED CLUSTER;

CREATE INDEX ORDERNO_IX ON Q2 (ORDERNO) PARTITIONED;
```

Also suppose that the first 4 bytes of each ORDERNO column value represent the four-digit year in which the order is placed. This means that the DATE column and the ORDERNO column are correlated.

To take advantage of limited partition scan, when you write a query that has the ORDERNO column in the predicate, also include the DATE column in the predicate. The partitioning index on DATE lets DB2 eliminate the scanning of partitions that are not needed to satisfy the query. For example:

```
SELECT ORDERNO, PURCH_AMT
FROM Q2
WHERE ORDERNO BETWEEN '2005AAAA' AND '2005ZZZZ' AND
DATE BETWEEN '2005-01-01' AND '2005-12-31';
```

### Example: page range screening

Another way to increase query performance for tables with data-partitioned secondary indexes, is to use a join predicate for page range screening. In the following example assume that:

- Both T1 and T2 are partitioned on TRANS\_MONTH.
- The access path for the query is a nested join loop from T1 to T2, and access to T2 is through a DPSI index on T2.CUSTNUM


```
SELECT * FROM T1, T2
WHERE T1.TRANS_MONTH = T2.TRANS_MONTH
AND T1.CUSTNUM = T2.CUSTNUM
```

Because the tables are joined on the partitioning column, only one partition of the DPSI index on T2 actually needs to be accessed for each row from T1, and the elimination of partitions greatly improves the query performance.

You can also take advantage of *non-matching predicates* for page screening. These predicates on the partitioning key cannot be used as matching columns because they are not on the leading columns of the partitioning key. In the following example, assume that:

- T1 is partitioned on REGION, STATE, MONTH.
- The number of partitions is equal to the number of regions (4), times the number of states (50), times the number of months (12), for a total of 2400 partitions.
- Access to T1 is through a DSPI index on T1.STOR\_ID.


```
SELECT SUM(GROSS_SALES) FROM T1
WHERE T1.MONTH = ?
AND T1.STOR_ID = ?
```

Because only one month is of interest, and the page screening function uses the predicate on T1.MONTH, writing the query this way reduces the number of partitions accessed to one-twelfth of 2400, or 200. 


---

## Special techniques to influence access path selection

You can use certain special techniques to influence how DB2 chooses an access path for a SQL query, but you should do so carefully.

 The access path selection "tricks" that are described in these topics might cause significant performance degradation if they are not carefully implemented and monitored.

The selection method might change in a later release of DB2, causing your changes to degrade performance.

Save the old catalog statistics or SQL before you consider making any changes to control the choice of access path. Before and after you make any changes, take performance measurements. When you migrate to a new release, evaluate the performance again. Be prepared to back out any changes that have degraded performance. 

## Using package copies to alleviate performance regression

You can use package copies to automatically save pertinent catalog table and directory records when you rebind an existing package. If a performance regression occurs after you rebind a package, you can fall back to the older copy of the package.

You can use package copies with regular packages, non-native SQL procedures, external procedures and trigger packages.

Issue a REBIND PACKAGE or REBIND TRIGGER PACKAGE command, and specify the PLANMGMT option:

Option	Description
<b>PLANMGMT (OFF)</b>	DB2 purges all on-disk and in-memory pieces of the current package copy from the catalog tables and directory, compiles each static query in the package and create new records that represent the result of the compilation. This option does not purge any of the previous or original copies.
<b>PLANMGMT (BASIC)</b>	<p>DB2 creates or modifies only two copies of the package:</p> <p><b>Previous copy</b> DB2 discards any existing previous copy and replaces it with the old active copy.</p> <p><b>Active copy</b> DB2 saves the current active copy as the previous copy and the incoming copy becomes the active copy.</p>
<b>PLANMGMT (EXTENDED)</b>	<p>DB2 creates or modifies as many as three copies of the package:</p> <p><b>Original copy</b> If no original copy exists or an existing original copy is invalid, DB2 clones the current copy as the original. If a valid original exists, DB2 retains it as it is.</p> <p><b>Previous copy</b> DB2 discards any existing previous copy and replaces it with the old active copy.</p> <p><b>Active copy</b> DB2 saves the current active copy as the previous copy and the incoming copy becomes the active copy.</p>

When you rebind a package with the PLANMGMT (BASIC) or (EXTENDED) options, the following options must remain the same:

- OWNER
- QUALIFIER
- DBPROTOCOL
- ENABLE
- DISABLE
- PATH
- PATHDEFAULT
- IMMEDIATEWRITE

After the REBIND operation, records for the package remain available for restoration. In the event of a performance regression, you can fall back to the saved copies.

### Package copies

When a package is bound, DB2 stores relevant package information as records in several catalog tables and the directory.

These records include information about:

- Metadata
- Query text
- Dependencies
- Authorizations
- Access paths

By default, when you issue a REBIND PACKAGE command, DB2 deletes many of these records and replaces them with new records. However, you can specify for DB2 to retain such records in *package copies* when you rebind packages. When you specify this option, DB2 retains original, previous, and active copies of the package.

Package copies are similar to package versions in DB2, which can have the same location name, package name, and package name. However, two package copies can also have the same version or consistency ID.

You can use package copies with regular packages, non-native SQL procedures, external procedures and trigger packages.

### **Invalid package copies**

Packages can become invalid when any object that the package depends on is altered or dropped. For example a package can become invalid when an index or table is dropped, or when a table column is altered. The SYSIBM.SYSPACKDEP catalog table records package dependencies. Depending on the change, different copies of the packages can be affected differently. In the case of a dropped table, all of the copies of the package would be invalidated. Whereas, in the case of a dropped index, only certain copies that depend on the dropped index might be invalidated.

When DB2 finds that the active copy for a package is invalid, it uses auto-bind to replace the current copy. The auto-bind is not effected by invalid status on any previous or original copies, and auto-bind replaces the only active copy.

### **Switching to a different package copy**

If you use package copies when you rebind a package, you can revert to a saved copy of the package if you encounter performance regression after rebinding the package.

You can switch to a previous copy of the package only if you specified PLANMGMT (BASIC) or (EXTENDED) with the REBIND PACKAGE or REBIND TRIGGER PACKAGE statement. You can switch to an original copy of the package only if you specified PLANMGMT (EXTENDED).

To switch to a previous or original copy:

Issue a REBIND PACKAGE or REBIND TRIGGER PACKAGE command, and specify the SWITCH option:

Option	Description
<b>SWITCH (PREVIOUS)</b>	DB2 toggles the current and previous packages: <ul style="list-style-type: none"> <li>• The existing current package takes the place of the previous package</li> <li>• The existing previous package takes the place of the current package.</li> </ul>
<b>SWITCH (ORIGINAL)</b>	DB2 clones the original copy to take the place of the current copy: <ul style="list-style-type: none"> <li>• The existing current copy replaces the previous copy.</li> <li>• The existing previous copy is discarded.</li> </ul>

You can use wild cards (\*) in the syntax to restore the previous or original packages for multiple packages. If no previous or original package copies exist, DB2 issues the DSNT217I error message for each package the does not have copies and processes the remainder normally.

### Freeing package copies

Using package copies increases the amount of disk space used in the catalog and directory for those packages. You can free unneeded package copies to reclaim this space.

Disk space consumption for package copies relatively minor in the catalog, but disk space consumption in the directory increases substantially. For example, using the PLANMGMT (BASIC) option doubles the disk used in the directory. Similarly, using the PLANMGMT (EXTENDED) option triples the amount of disk space consumed in the directory.

To free disk space used by unneeded package copies:

Issue a FREE PACKAGE command, and specify a scope for the action by including a PLANMGMTSCOPE clause.

Option	Description
<b>PLANMGMTSCOPE (ALL)</b>	The entire package, including all topics is freed. This option is the default behavior.
<b>PLANMGMTSCOPE (INACTIVE)</b>	Only previous and original copies are freed. The operation will succeed even if a package has no inactive copies.

```
FREE PACAKGE (collection-name.package-name) PLANMGMTSCOPE (INACTIVE)
```

The PLANMGMTSCOPE option cannot be used for remote processing.

## Rewriting queries to influence access path selection

If DB2 chooses an unsatisfactory access path, you might consider using some “special tricks” to rewrite your SQL statement to get a different access path.

**Important:** The access path selection “tricks” that are described here cause significant performance degradation if they are not carefully implemented and monitored.

For example, the selection method might change in a later release of DB2, causing your changes to degrade performance, or the change to catalog statistics might help one SQL statement, but degrade many others.

Before and after you make any permanent changes, take performance measurements. When you migrate to a new release, evaluate the performance again. Be prepared to back out any changes that have degraded performance.

**PSPI**

The examples that follow identify potential performance problems and offer suggestions for tuning the queries. However, before you rewrite any query, you should consider whether you can use the REOPT(ALWAYS), REOPT(AUTO), or REOPT(ONCE) bind options to solve your access path problems at run time.

### *Example 1: An equal predicate*

An equal predicate has a default filter factor of 1/COLCARD. The actual filter factor might be quite different.

**Query:**

```
SELECT * FROM DSN8910.EMP
WHERE SEX = :HV1;
```

**Assumptions:** Because the column SEX has only two different values, 'M' and 'F', the value COLCARD for SEX is 2. If the numbers of male and female employees are not equal, the actual filter factor of 1/2 is larger or smaller than the default, depending on whether :HV1 is set to 'M' or 'F'.

**Recommendation:** One of these two actions can improve the access path:

- Bind the package or plan that contains the query with the REOPT(AUTO) bind option. This action enables DB2 to decide whether to reoptimize the query at run time, using the input values you provide.
- Write predicates to influence the DB2 selection of an access path, based on your knowledge of actual filter factors. For example, you can break the query into three different queries, two of which use constants. DB2 can then determine the exact filter factor for most cases when it binds the plan.

```
SELECT (HV1);
  WHEN ('M')
  DO;
    EXEC SQL SELECT * FROM DSN8910.EMP
      WHERE SEX = 'M';
  END;
  WHEN ('F')
  DO;
    EXEC SQL SELECT * FROM DSN8910.EMP
      WHERE SEX = 'F';
  END;
  OTHERWISE
  DO;
    EXEC SQL SELECT * FROM DSN8910.EMP
      WHERE SEX = :HV1;
  END;
END;
```

### *Example 2: Known ranges*

Table T1 has two indexes: T1X1 on column C1 and T1X2 on column C2.

**Query:**

```
SELECT * FROM T1
WHERE C1 BETWEEN :HV1 AND :HV2
AND C2 BETWEEN :HV3 AND :HV4;
```

**Assumptions:** You know that:

- The application always provides a narrow range on C1 and a wide range on C2.
- The desired access path is through index T1X1.

**Recommendation:** If DB2 does not choose T1X1, rewrite the query as follows, so that DB2 does not choose index T1X2 on C2:

```
SELECT * FROM T1
WHERE C1 BETWEEN :HV1 AND :HV2
AND (C2 BETWEEN :HV3 AND :HV4 OR 0=1);
```

**Example 3: Variable ranges**

Table T1 has two indexes: T1X1 on column C1 and T1X2 on column C2.

**Query:**

```
SELECT * FROM T1
WHERE C1 BETWEEN :HV1 AND :HV2
AND C2 BETWEEN :HV3 AND :HV4;
```

**Assumptions:** You know that the application provides both narrow and wide ranges on C1 and C2. Hence, default filter factors do not allow DB2 to choose the best access path in all cases. For example, a small range on C1 favors index T1X1 on C1, a small range on C2 favors index T1X2 on C2, and wide ranges on both C1 and C2 favor a table space scan.

**Recommendation:** If DB2 does not choose the best access path, try either of the following changes to your application:

- Use a dynamic SQL statement and embed the ranges of C1 and C2 in the statement. With access to the actual range values, DB2 can estimate the actual filter factors for the query. Preparing the statement each time it is executed requires an extra step, but it can be worthwhile if the query accesses a large amount of data.
- Include some simple logic to check the ranges of C1 and C2, and then execute one of these static SQL statements, based on the ranges of C1 and C2:

```
SELECT * FROM T1 WHERE C1 BETWEEN :HV1 AND :HV2
AND (C2 BETWEEN :HV3 AND :HV4 OR 0=1);
```

```
SELECT * FROM T1 WHERE C2 BETWEEN :HV3 AND :HV4
AND (C1 BETWEEN :HV1 AND :HV2 OR 0=1);
```

```
SELECT * FROM T1 WHERE (C1 BETWEEN :HV1 AND :HV2 OR 0=1)
AND (C2 BETWEEN :HV3 AND :HV4 OR 0=1);
```

**Example 4: ORDER BY**

Table T1 has two indexes: T1X1 on column C1 and T1X2 on column C2.

**Query:**

```
SELECT * FROM T1
WHERE C1 BETWEEN :HV1 AND :HV2
ORDER BY C2;
```

In this example, DB2 could choose one of the following actions:

- Scan index T1X1 and then sort the results by column C2
- Scan the table space in which T1 resides and then sort the results by column C2
- Scan index T1X2 and then apply the predicate to each row of data, thereby avoiding the sort

Which choice is best depends on the following factors:

- The number of rows that satisfy the range predicate
- The cluster ratio of the indexes

If the actual number of rows that satisfy the range predicate is significantly different from the estimate, DB2 might not choose the best access path.

**Assumptions:** You disagree with the DB2 choice.

**Recommendation:** In your application, use a dynamic SQL statement and embed the range of C1 in the statement. That allows DB2 to use the actual filter factor rather than the default, but requires extra processing for the PREPARE statement.

#### *Example 5: A join operation*

Tables A, B, and C each have indexes on columns C1, C2, C3, and C4.


**Assumptions:** The actual filter factors on table A are much larger than the default factors. Hence, DB2 underestimates the number of rows selected from table A and wrongly chooses that as the first table in the join.

**Recommendations:** You can:

- Reduce the estimated size of Table A by adding predicates
- Disfavor any index on the join column by making the join predicate on table A nonindexable

**Example:** The following query illustrates the second of those choices.

```
SELECT * FROM T1 A, T1 B, T1 C
WHERE (A.C1 = B.C1 OR 0=1)
      AND A.C2 = C.C2
      AND A.C2 BETWEEN :HV1 AND :HV2
      AND A.C3 BETWEEN :HV3 AND :HV4
      AND A.C4 < :HV5
      AND B.C2 BETWEEN :HV6 AND :HV7
      AND B.C3 < :HV8
      AND C.C2 < :HV9;
```

The result of making the join predicate between A and B a nonindexable predicate (which cannot be used in single index access) disfavors the use of the index on column C1. This, in turn, might lead DB2 to access table A or B first. Or, it might lead DB2 to change the access type of table A or B, thereby influencing the join sequence of the other tables. 

## Obtaining information about access paths

You can obtain information about DB2 access paths by using the following methods:

 PSPI

- Use Optimization Service Center for DB2 for z/OS.

With Optimization Service Center for DB2 for z/OS, an order-able feature and part of IBM DB2 for z/OS Accessories Suite, you can display and analyze information on access paths chosen by DB2. Optimization Service Center displays the access path information in both graphic and text formats. The tool provides you with an easy-to-use interface to the PLAN\_TABLE output and allows you to invoke EXPLAIN for dynamic SQL statements. You can also access the catalog statistics for certain referenced objects of an access path. In addition, the tool allows you to archive EXPLAIN output from previous SQL statements to analyze changes in your SQL environment.

- Run OMEGAMON DB2 Performance Monitor accounting reports.

Another way to track performance is with the DB2 Performance Monitor accounting reports. The accounting report, short layout, ordered by PLANNAME, lists the primary performance figures. Check the plans that contain SQL statements whose access paths you tried to influence. If the elapsed time, TCB time, or number of getpage requests increases sharply without a corresponding increase in the SQL activity, then there could be a problem. You can use OMEGAMON Online Monitor to track events after your changes have been implemented, providing immediate feedback on the effects of your changes.

- Specify the EXPLAIN bind option .

You can also use the EXPLAIN option when you bind or rebind a plan or package. Compare the new plan or package for the statement to the old one. If the new one has a table space scan or a nonmatching index space scan, but the old one did not, the problem is probably the statement. Investigate any changes in access path in the new plan or package; they could represent performance improvements or degradations. If neither the accounting report ordered by PLANNAME or PACKAGE nor the EXPLAIN statement suggest corrective action, use the OMEGAMON SQL activity reports for additional information.

 PSPI

## Fetching a limited number of rows: FETCH FIRST *n* ROWS ONLY

In some applications, you execute queries that can return a large number of rows, but you need only a small subset of those rows. Retrieving the entire result table from the query can be inefficient.

 PSPI

You can specify the FETCH FIRST *n* ROWS ONLY clause in a SELECT statement to limit the number of rows in the result table of a query to *n* rows. In addition, for a distributed query that uses DRDA access, FETCH FIRST *n* ROWS ONLY, DB2 prefetches only *n* rows.

**Example:** Suppose that you write an application that requires information on only the 20 employees with the highest salaries. To return only the rows of the employee table for those 20 employees, you can write a query like this:

```
SELECT LASTNAME, FIRSTNAME, EMPNO, SALARY
FROM EMP
ORDER BY SALARY DESC
FETCH FIRST 20 ROWS ONLY;
```

You can also use FETCH FIRST *n* ROWS ONLY within a subquery.

**Example:**

```
SELECT * FROM EMP
WHERE EMPNO IN (
  SELECT RESPEMP FROM PROJECT
ORDER BY PROJNO FETCH FIRST 3 ROWS ONLY)
```


**Interaction between OPTIMIZE FOR  $n$  ROWS and FETCH FIRST  $n$  ROWS ONLY:**

In general, if you specify FETCH FIRST  $n$  ROWS ONLY but not OPTIMIZE FOR  $n$  ROWS in a SELECT statement, DB2 optimizes the query as if you had specified OPTIMIZE FOR  $n$  ROWS.

When both the FETCH FIRST  $n$  ROWS ONLY clause and the OPTIMIZE FOR  $n$  ROWS clause are specified, the value for the OPTIMIZE FOR  $n$  ROWS clause is used for access path selection.


**Example:** Suppose that you submit the following SELECT statement:

```
SELECT * FROM EMP
FETCH FIRST 5 ROWS ONLY
OPTIMIZE FOR 20 ROWS;
```

The OPTIMIZE FOR value of 20 rows is used for access path selection. 

## Minimizing overhead for retrieving few rows: OPTIMIZE FOR $n$ ROWS

When an application executes a SELECT statement, DB2 assumes that the application retrieves all the qualifying rows.

 This assumption is most appropriate for batch environments. However, for interactive SQL applications, such as SPUFI, it is common for a query to define a very large potential result set but retrieve only the first few rows. The access path that DB2 chooses might not be optimal for those interactive applications.

This topic discusses the use of OPTIMIZE FOR  $n$  ROWS to affect the performance of interactive SQL applications. Unless otherwise noted, this information pertains to local applications.

**What OPTIMIZE FOR  $n$  ROWS does:** The OPTIMIZE FOR  $n$  ROWS clause lets an application declare its intent to do either of these things:

- Retrieve only a subset of the result set
- Give priority to the retrieval of the first few rows

DB2 uses the OPTIMIZE FOR  $n$  ROWS clause to choose access paths that minimize the response time for retrieving the first few rows. For distributed queries, the value of  $n$  determines the number of rows that DB2 sends to the client on each DRDA network transmission.

**Use OPTIMIZE FOR 1 ROW to avoid sorts:** You can influence the access path most by using OPTIMIZE FOR 1 ROW. OPTIMIZE FOR 1 ROW tells DB2 to select an access path that returns the first qualifying row quickly. This means that whenever possible, DB2 avoids any access path that involves a sort. If you specify a value for  $n$  that is anything but 1, DB2 chooses an access path based on cost, and you won't necessarily avoid sorts.

**How to specify OPTIMIZE FOR *n* ROWS for a CLI application:** For a Call Level Interface (CLI) application, you can specify that DB2 uses OPTIMIZE FOR *n* ROWS for all queries. To do that, specify the keyword OPTIMIZEFORNROWS in the initialization file.

**How many rows you can retrieve with OPTIMIZE FOR *n* ROWS:** The OPTIMIZE FOR *n* ROWS clause does not prevent you from retrieving all the qualifying rows. However, if you use OPTIMIZE FOR *n* ROWS, the total elapsed time to retrieve all the qualifying rows might be significantly greater than if DB2 had optimized for the entire result set.

**When OPTIMIZE FOR *n* ROWS is effective:** OPTIMIZE FOR *n* ROWS is effective only on queries that can be performed incrementally. If the query causes DB2 to gather the whole result set before returning the first row, DB2 ignores the OPTIMIZE FOR *n* ROWS clause, as in the following situations:

- The query uses SELECT DISTINCT or a set function distinct, such as COUNT(DISTINCT C1).
- Either GROUP BY or ORDER BY is used, and no index can give the necessary ordering.
- A aggregate function and no GROUP BY clause is used.
- The query uses UNION.

**Example:** Suppose that you query the employee table regularly to determine the employees with the highest salaries. You might use a query like this:

```
SELECT LASTNAME, FIRSTNAME, EMPNO, SALARY
FROM EMP
ORDER BY SALARY DESC;
```

An index is defined on column EMPNO, so employee records are ordered by EMPNO. If you have also defined a descending index on column SALARY, that index is likely to be very poorly clustered. To avoid many random, synchronous I/O operations, DB2 would most likely use a table space scan, then sort the rows on SALARY. This technique can cause a delay before the first qualifying rows can be returned to the application.

If you add the OPTIMIZE FOR *n* ROWS clause to the statement, DB2 probably uses the SALARY index directly because you have indicated that you expect to retrieve the salaries of only the 20 most highly paid employees.

**Example:** The following statement uses that strategy to avoid a costly sort operation:

```
SELECT LASTNAME, FIRSTNAME, EMPNO, SALARY
FROM EMP
ORDER BY SALARY DESC
OPTIMIZE FOR 20 ROWS;
```

**Effects of using OPTIMIZE FOR *n* ROWS:**

- The join method could change. Nested loop join is the most likely choice, because it has low overhead cost and appears to be more efficient if you want to retrieve only one row.
- An index that matches the ORDER BY clause is more likely to be picked. This is because no sort would be needed for the ORDER BY.
- List prefetch is less likely to be picked.

- Sequential prefetch is less likely to be requested by DB2 because it infers that you only want to see a small number of rows.
- In a join query, the table with the columns in the ORDER BY clause is likely to be picked as the outer table if an index created on that outer table gives the ordering needed for the ORDER BY clause.

**Recommendation:** For a local query, specify OPTIMIZE FOR *n* ROWS only in applications that frequently fetch only a small percentage of the total rows in a query result set. For example, an application might read only enough rows to fill the end user's terminal screen. In cases like this, the application might read the remaining part of the query result set only rarely. For an application like this, OPTIMIZE FOR *n* ROWS can result in better performance by causing DB2 to favor SQL access paths that deliver the first *n* rows as fast as possible.

When you specify OPTIMIZE FOR *n* ROWS for a remote query, a small value of *n* can help limit the number of rows that flow across the network on any given transmission.

You can improve the performance for receiving a large result set through a remote query by specifying a large value of *n* in OPTIMIZE FOR *n* ROWS. When you specify a large value, DB2 attempts to send the *n* rows in multiple transmissions. For better performance when retrieving a large result set, in addition to specifying OPTIMIZE FOR *n* ROWS with a large value of *n* in your query, do not execute other SQL statements until the entire result set for the query is processed. If retrieval of data for several queries overlaps, DB2 might need to buffer result set data in the DDF address space.

For local or remote queries, to influence the access path most, specify OPTIMIZE for 1 ROW. This value does not have a detrimental effect on distributed queries.

PSPI

## Favoring index access


One common database design involves tables that contain groups of rows that logically belong together. Within each group, the rows should be accessed in the same sequence every time.

PSPI

The sequence is determined by the primary key on the table. Lock contention can occur when DB2 chooses different access paths for different applications that operate on a table with this design.


To minimize contention among applications that access tables with this design, specify the VOLATILE keyword when you create or alter the tables. A table that is defined with the VOLATILE keyword is known as a *volatile table*. When DB2 executes queries that include volatile tables, DB2 uses index access whenever possible. One exception is for DELETE statements on a VOLATILE table in a segmented table space when no WHERE clause is specified. In this case, a table space scan is used. As well as minimizing contention, using index access preserves the access sequence that the primary key provides.

Defining a table as volatile has a similar effect on a query to setting the NPGTHRSRSH subsystem parameter to favor matching index access for all qualified tables. However, the effect of NPGTHRSRSH is subsystem-wide, and index access

might not be appropriate for many queries. Defining tables as volatile lets you limit the set of queries that favor index access to queries that involve the volatile tables. 

## Using the **CARDINALITY** clause to improve the performance of queries with user-defined table function references

The cardinality of a user-defined table function is the number of rows that are returned when the function is invoked. DB2 uses this number to estimate the cost of executing a query that invokes a user-defined table function.

 The cost of executing a query is one of the factors that DB2 uses when it calculates the access path. Therefore, if you give DB2 an accurate estimate of a user-defined table function's cardinality, DB2 can better calculate the best access path.

You can specify a cardinality value for a user-defined table function by using the **CARDINALITY** clause of the SQL **CREATE FUNCTION** or **ALTER FUNCTION** statement. However, this value applies to all invocations of the function, whereas a user-defined table function might return different numbers of rows, depending on the query in which it is referenced.

To give DB2 a better estimate of the cardinality of a user-defined table function for a particular query, you can use the **CARDINALITY** or **CARDINALITY MULTIPLIER** clause in that query. DB2 uses those clauses at bind time when it calculates the access cost of the user-defined table function. Using this clause is recommended only for programs that run on DB2 for z/OS because the clause is not supported on earlier versions of DB2.

### Example of using the **CARDINALITY** clause to specify the cardinality of a user-defined table function invocation

Suppose that when you created user-defined table function **TUDF1**, you set a cardinality value of 5, but in the following query, you expect **TUDF1** to return 30 rows:

```
SELECT *  
FROM TABLE(TUDF1(3)) AS X;
```

Add the **CARDINALITY 30** clause to tell DB2 that, for this query, **TUDF1** should return 30 rows:

```
SELECT *  
FROM TABLE(TUDF1(3) CARDINALITY 30) AS X;
```

### Example of using the **CARDINALITY MULTIPLIER** clause to specify the cardinality of a user-defined table function invocation

Suppose that when you created user-defined table function **TUDF2**, you set a cardinality value of 5, but in the following query, you expect **TUDF2** to return 30 times that many rows:

```
SELECT *  
FROM TABLE(TUDF2(10)) AS X;
```

Add the **CARDINALITY MULTIPLIER 30** clause to tell DB2 that, for this query, **TUDF1** should return  $5 \times 30$ , or 150, rows:

```
SELECT *
FROM TABLE(TUDF2(10) CARDINALITY MULTIPLIER 30) AS X;
```

**PSPI**

## Reducing the number of matching columns

You can discourage the use of a poorer performing index by reducing the index's matching predicate on its leading column.

**PSPI**

Consider the example in Figure 36 on page 306, where the index that DB2 picks is less than optimal.

```
CREATE TABLE PART_HISTORY (
    PART_TYPE CHAR(2),      IDENTIFIES THE PART TYPE
    PART_SUFFIX CHAR(10),   IDENTIFIES THE PART
    W_NOW      INTEGER,      TELLS WHERE THE PART IS
    W_FROM     INTEGER,      TELLS WHERE THE PART CAME FROM
    DEVIATIONS INTEGER,      TELLS IF ANYTHING SPECIAL WITH THIS PART
    COMMENTS   CHAR(254),
    DESCRIPTION CHAR(254),
    DATE1      DATE,
    DATE2      DATE,
    DATE3      DATE);

CREATE UNIQUE INDEX IX1 ON PART_HISTORY
(PART_TYPE,PART_SUFFIX,W_FROM,W_NOW);
CREATE UNIQUE INDEX IX2 ON PART_HISTORY
(W_FROM,W_NOW,DATE1);
```

Table statistics		Index statistics			IX1	IX2
CARDF	100,000	FIRSTKEYCARDF	1000			50
NPAGES	10,000	FULLKEYCARDF	100,000			100,000
		CLUSTERRATIO	99%			99%
		NLEAF	3000			2000
		NLEVELS	3			3
column		cardinality	HIGH2KEY	LOW2KEY		
Part_type		1000	'ZZ'	'AA'		
w_now		50	1000	1		
w_from		50	1000	1		

Q1:

```
SELECT * FROM PART_HISTORY -- SELECT ALL PARTS
WHERE PART_TYPE = 'BB'     P1 -- THAT ARE 'BB' TYPES
AND W_FROM = 3             P2 -- THAT WERE MADE IN CENTER 3
AND W_NOW = 3              P3 -- AND ARE STILL IN CENTER 3
```

ESTIMATED VALUES				WHAT REALLY HAPPENS			
Filter factor of these predicates.							
P1 = 1/1000= .001							
P2 = 1/50 = .02							
P3 = 1/50 = .02							
index	matchcols	filter factor	data rows	index	matchcols	filter factor	data rows
ix2	2	.02*.02	40	ix2	2	.02*.50	1000
ix1	1	.001	100	ix1	1	.001	100

Figure 36. Reducing the number of MATCHCOLS

DB2 picks IX2 to access the data, but IX1 would be roughly 10 times quicker. The problem is that 50% of all parts from center number 3 are still in Center 3; they have not moved. Assume that no statistics are available on the correlated columns in catalog table SYSCOLDIST. Therefore, DB2 assumes that the parts from center number 3 are evenly distributed among the 50 centers.

You can get the desired access path by changing the query. To discourage the use of IX2 for this particular query, you can change the third predicate to be nonindexable.

```
SELECT * FROM PART_HISTORY
  WHERE PART_TYPE = 'BB'
     AND W_FROM = 3
     AND (W_NOW = 3 + 0)      <-- PREDICATE IS MADE NONINDEXABLE
```

Now index I2 is not picked, because it has only one match column. The preferred index, I1, is picked. The third predicate is a nonindexable predicate, so an index is not used for the compound predicate.

You can make a predicate non-indexable in many ways. The recommended way is to add 0 to a predicate that evaluates to a numeric value or to concatenate an empty string to a predicate that evaluates to a character value.

Indexable	Nonindexable
T1.C3=T2.C4	(T1.C3=T2.C4 CONCAT '')
T1.C1=5	T1.C1=5+0

These techniques do not affect the result of the query and cause only a small amount of overhead.

The preferred technique for improving the access path when a table has correlated columns is to generate catalog statistics on the correlated columns. You can do that either by running RUNSTATS or by updating catalog table SYSCOLDIST manually.



## Indexes for efficient star schema processing

You can create indexes to enable DB2 to use special join methods for star schemas.



A *star schema* is a database design that, in its simplest form, consists of a large table called a *fact table*, and two or more smaller tables, called *dimension tables*. More complex star schemas can be created by breaking one or more of the dimension tables into multiple tables.

To access the data in a star schema design, you often write SELECT statements that include join operations between the fact table and the dimension tables, but no join operations between dimension tables. These types of queries are known as *star-join queries*.

For a star-join query, DB2 might use special join types, *star join* and *pair-wise join*, if the following conditions are true:

- The tables meet the conditions of a star join. (JOIN\_TYPE='S')

- The tables meet the conditions of a pair-wise join. (JOIN\_TYPE='P')
- The STARJOIN system parameter is set to ENABLE, and the number of tables in the query block is greater than or equal to the minimum number that is specified in the SJTABLES system parameter.

Whether DB2 uses star join, pair-wise join, or traditional join methods for processing a star schema query is based on which method results in the lowest cost access path. The existence of a star schema does not guarantee that either star join or pair-wise join access will be chosen.

#### PSPI

### Enabling efficient access for queries on star schemas

Pair-wise join processing simplifies index design by using single-column indexes to join a fact table and the associated dimension tables according to AND predicates.

#### PSPI

To design indexes to enable pair-wise join:

1. Create an index for each key column in the fact table that corresponds to a dimension table.
2. Partition by and cluster data according to commonly used dimension keys. Doing so can reduce the I/O that is required on the fact table for pair-wise join.

If you require further performance improvement for some star schema queries, consider the following index design recommendations to encourage DB2 to use star join access:

- Define a multi-column index on all key columns of the fact table. Key columns are fact table columns that have corresponding dimension tables.
- If you do not have information about the way that your data is used, first try a multi-column index on the fact table that is based on the correlation of the data. Put less highly correlated columns later in the index key than more highly correlated columns.
- As the correlation of columns in the fact table changes, reevaluate the index to determine if columns in the index should be reordered.
- Define indexes on dimension tables to improve access to those tables.
- When you have executed a number of queries and have more information about the way that the data is used, follow these recommendations:
  - Put more selective columns at the beginning of the multi-column index.
  - If a number of queries do not reference a dimension, put the column that corresponds to that dimension at the end of the index, or remove it completely.


#### PSPI

### Rearranging the order of tables in a FROM clause

The order of tables or views in the FROM CLAUSE can affect the access path that DB2 chooses for a SQL query.


#### PSPI

If your query performs poorly, it could be because the join sequence is inefficient. You can determine the join sequence within a query block from the PLANNO column in the PLAN\_TABLE. If you think that the join sequence is

inefficient, try rearranging the order of the tables and views in the FROM clause to match a join sequence that might perform better. 

## Updating catalog statistics

If you have the proper authority, you can influence access path selection by using an SQL UPDATE or INSERT statement to change statistical values in the DB2 catalog. However, doing so is not generally recommended except as a last resort.

 Although updating catalog statistics can help a certain query, other queries can be affected adversely. Also, the UPDATE statements must be repeated after RUNSTATS resets the catalog values. You should be very careful if you attempt to update statistics.

If you update catalog statistics for a table space or index manually, and you are using dynamic statement caching, you need to invalidate statements in the cache that involve those table spaces or indexes. To invalidate statements in the dynamic statement cache without updating catalog statistics or generating reports, you can run the RUNSTATS utility with the REPORT NO and UPDATE NONE options on the table space or the index that the query is dependent on.

The example shown in Figure 36 on page 306, involves this query:

```
SELECT * FROM PART_HISTORY  -- SELECT ALL PARTS
WHERE PART_TYPE = 'BB'      P1 -- THAT ARE 'BB' TYPES
    AND W_FROM = 3          P2 -- THAT WERE MADE IN CENTER 3
    AND W_NOW = 3           P3 -- AND ARE STILL IN CENTER 3
```

This query has a problem with data correlation. DB2 does not know that 50% of the parts that were made in Center 3 are still in Center 3. The problem was circumvented by making a predicate nonindexable. But suppose that hundreds of users are writing queries similar to that query. Having all users change their queries would be impossible. In this type of situation, the best solution is to change the catalog statistics.

For the query in Figure 36 on page 306, you can update the catalog statistics in one of two ways:

- Run the RUNSTATS utility, and request statistics on the correlated columns W\_FROM and W\_NOW. This is the preferred method.
- Update the catalog statistics manually.


**Updating the catalog to adjust for correlated columns:** One catalog table that you can update is SYSIBM.SYSCOLDIST, which gives information about a column or set of columns in a table. Assume that because columns W\_NOW and W\_FROM are correlated, only 100 distinct values exist for the combination of the two columns, rather than 2500 (50 for W\_FROM \* 50 for W\_NOW). Insert a row like this to indicate the new cardinality:

```
INSERT INTO SYSIBM.SYSCOLDIST
(FREQUENCY, FREQUENCYF, IBMREQD,
 TBOWNER, TBNAME, NAME, COLVALUE,
 TYPE, CARDF, COLGROUPCOLNO, NUMCOLUMNS)
VALUES(0, -1, 'N',
 'USRT001', 'PART_HISTORY', 'W_FROM', ' ',
 'C', 100, X'00040003', 2);
```

You can also use the RUNSTATS utility to put this information in SYSCOLDIST.

You tell DB2 about the frequency of a certain combination of column values by updating SYSIBM.SYSCOLDIST. For example, you can indicate that 1% of the rows in PART\_HISTORY contain the values 3 for W\_FROM and 3 for W\_NOW by inserting this row into SYSCOLDIST:

```
INSERT INTO SYSIBM.SYSCOLDIST
(FREQUENCY, FREQUENCYF, STATTIME, IBMREQD,
TBOWNER, TBNAME, NAME, COLVALUE,
TYPE, CARDF, COLGROUPCOLNO, NUMCOLUMNS)
VALUES(0, .0100, '2006-12-01-12.00.00.000000', 'N',
'USRT001', 'PART_HISTORY', 'W_FROM', X'00800000030080000003',
'F', -1, X'00040003', 2);
```

**Updating the catalog for joins with table functions:** Updating catalog statistics **might cause extreme performance problems** if the statistics are not updated correctly. Monitor performance, and be prepared to reset the statistics to their original values if performance problems arise. 

## Using subsystem parameters to improve performance


Many DB2 subsystem parameters affect the performance of DB2.

 To set subsystem parameters, modify and run installation job DSNTIJUZ.



### Using a subsystem parameter to favor matching index access

DB2 often scans a table space or nonmatching index when the data access statistics indicate that a table is small, even though matching index access is possible. This is a problem if the table is small or empty when statistics are collected, but the table is large when it is queried.

 In that case, the statistics are not accurate and can lead DB2 to pick an inefficient access path.

The best solution to the problem is to run RUNSTATS again after the table is populated. However, if you cannot do that, you can use subsystem parameter NPGTHRSH to cause DB2 to favor matching index access over a table space scan and over nonmatching index access.

The value of NPGTHRSH is an integer that indicates the tables for which DB2 favors matching index access. Values of NPGTHRSH and their meanings are:

- 0 DB2 selects the access path based on cost, and no tables qualify for special handling. This is the default.
- $n \geq 1$  If data access statistics have been collected for all tables, DB2 favors matching index access for tables for which the total number of pages on which rows of the table appear (NPAGES) is less than  $n$ .

Tables with default statistics for NPAGES (NPAGES = -1) are presumed to have 501 pages. For such tables, DB2 favors matching index access only when NPGTHRSH is set above 501.

**Recommendation:** Before you use NPGTHRSH, be aware that in some cases, matching index access can be more costly than a table space scan or nonmatching index access. Specify a small value for NPGTHRSH (10 or less), which limits the number of tables for which DB2 favors matching index access. If you need to use

matching index access only for specific tables, create or alter those tables with the VOLATILE parameter, rather than using the system-wide NPGTHRSH parameter.

**PSPI**

## Improving outer join processing

You can use a subsystem parameter to improve the how DB2 processes an outer join.

Set the OJPERFEH subsystem parameter to YES. DB2 takes the following actions, which can improve outer join processing in most cases:

- Does not merge table expressions or views if the parent query block of a table expression or view contains an outer join, and the merge would cause a column in a predicate to become an expression.
- Does not attempt to reduce work file usage for outer joins.
- Uses transitive closure for the ON predicates in outer joins.

These actions might not improve performance for some outer joins, and you should verify that performance improves. If the performance of queries that contain outer joins remains inadequate, set OJPERFEH to NO, restart DB2, and rerun those queries.

## Using a subsystem parameter to optimize queries with IN-list predicates

You can use the INLISTP parameter to control IN-list predicate optimization.

**PSPI**

If you set the INLISTP parameter to a number *n* that is between 1 and 5000, DB2 optimizes for an IN-list predicate with up to *n* values. If you set the INLISTP predicate to zero, the optimization is disabled. The default value for the INLISTP parameter is 50.

When you enable the INLISTP parameter, you enable two primary means of optimizing some queries that contain IN-list predicates:

- The IN-list predicate is pushed down from the parent query block into the materialized table expression.
- A correlated IN-list predicate in a subquery that is generated by transitive closure is moved up to the parent query block.

**PSPI**


## Controlling access path selection for clustered data

You can prevent undesirable access path changes to certain types of indexes by setting the STATCLUS subsystem parameter.

**PSPI**


If you execute the RUNSTATS utility on indexes that contain many duplicate key values, or key values that are highly clustered in reverse order, widespread changes to access paths can occur, especially when you migrate from DB2 Version 8. Performance measurements show that cluster ratio formula combined with DATAPEATFACTORF statistic is more accurate and provides better cost estimation. However, any statistics change can result in unexpected impacts.

If significant performance degradation occurs as a result of access paths that are associated with indexes with duplicate key values or reverse clustering:

Set the online updatable STATCLUS subsystem parameter to STANDARD and execute the RUNSTATS utility again. The default value of the STATCLUS parameter is ENHANCED, which means that access path selection uses the DATAREPEATFACTORF statistic. 

## Giving optimization hints to DB2

Experienced programmer analysts can tell DB2 how to process a query by giving *hints*DB2.

 The process of giving hints to DB2 is relatively simple but determining what those hints should be is not. Always test and analyze the results of any query that uses optimization hints.

Giving optimization hints to DB2 is useful in the following situations:

- You want to ensure consistency of response times across rebinds and across release migrations. When a plan or package is rebound, the access path is reformulated. If the database or application has changed, or if DB2 has new function that causes it to choose a different access path, it is handy to have the ability to use an old access path if the new one does not perform as well.  
For this reason, it is a good idea to save the access paths of your critical queries before migrating to a new release of DB2.

- You want to temporarily bypass the access path chosen by DB2. 

### Planning to use optimization hints

Before you can give hints to DB2, make sure your PLAN\_TABLE is of the correct format.

 To migrate your PLAN\_TABLE to the correct format:

1. Issue the following create table statement:

Figure 37. 59-column format of PLAN\_TABLE

```
CREATE TABLE userid.PLAN_TABLE
  (QUERYNO          INTEGER          NOT NULL,
   QBLOCKNO         SMALLINT        NOT NULL,
   APPLNAME          VARCHAR(24)     NOT NULL,
   PROGNAME          VARCHAR(128)    NOT NULL,
   PLANNO            SMALLINT        NOT NULL,
   METHOD             SMALLINT        NOT NULL,
   CREATOR           VARCHAR(128)    NOT NULL,
   TNAME             VARCHAR(128)    NOT NULL,
   TABNO             SMALLINT        NOT NULL,
   ACESSTYPE         CHAR(2)         NOT NULL,
   MATCHCOLS         SMALLINT        NOT NULL,
   ACCESSCREATOR     VARCHAR(128)    NOT NULL,
   ACCESSNAME        VARCHAR(128)    NOT NULL,
   INDEXONLY         CHAR(1)         NOT NULL,
   SORTN_UNIQ        CHAR(1)         NOT NULL,
   SORTN_JOIN        CHAR(1)         NOT NULL,
   SORTN_ORDERBY     CHAR(1)         NOT NULL,
   SORTN_GROUPBY     CHAR(1)         NOT NULL,
   SORTC_UNIQ        CHAR(1)         NOT NULL,
   SORTC_JOIN        CHAR(1)         NOT NULL,
   SORTC_ORDERBY     CHAR(1)         NOT NULL,
   SORTC_GROUPBY     CHAR(1)         NOT NULL,
   TSLOCKMODE        CHAR(3)         NOT NULL,
```

```

|          TIMESTAMP          CHAR(16)      NOT NULL,
|          REMARKS            VARCHAR(762)   NOT NULL,
|          PREFETCH           CHAR(1)        NOT NULL WITH DEFAULT,
|          COLUMN_FN_EVAL     CHAR(1)        NOT NULL WITH DEFAULT,
|          MIXOPSEQ           SMALLINT       NOT NULL WITH DEFAULT,
|          VERSION            VARCHAR(64)     NOT NULL WITH DEFAULT,
|          COLLID             VARCHAR(128)    NOT NULL WITH DEFAULT,
|          ACCESS_DEGREE      SMALLINT       ,
|          ACCESS_PGROUP_ID   SMALLINT       ,
|          JOIN_DEGREE        SMALLINT       ,
|          JOIN_PGROUP_ID     SMALLINT       ,
|          SORTC_PGROUP_ID    SMALLINT       ,
|          SORTN_PGROUP_ID    SMALLINT       ,
|          PARALLELISM_MODE    CHAR(1)        ,
|          MERGE_JOIN_COLS     SMALLINT       ,
|          CORRELATION_NAME    VARCHAR(128)   ,
|          PAGE_RANGE          CHAR(1)        NOT NULL WITH DEFAULT,
|          JOIN_TYPE           CHAR(1)        NOT NULL WITH DEFAULT,
|          GROUP_MEMBER        VARCHAR(24)     NOT NULL WITH DEFAULT,
|          IBM_SERVICE_DATA    VARCHAR(254)    FOR BIT DATA NOT NULL WITH DEFAULT,
|          WHEN_OPTIMIZE       CHAR(1)        NOT NULL WITH DEFAULT,
|          QBLOCK_TYPE         CHAR(6)        NOT NULL WITH DEFAULT,
|          BIND_TIME           TIMESTAMP      NOT NULL WITH DEFAULT,
|          OPTHINT             VARCHAR(128)    NOT NULL WITH DEFAULT,
|          HINT_USED           VARCHAR(128)    NOT NULL WITH DEFAULT,
|          PRIMARY_ACESSTYPE   CHAR(1)        NOT NULL WITH DEFAULT,
|          PARENT_QBLOCKNO     SMALLINT       NOT NULL WITH DEFAULT,
|          TABLE_TYPE         CHAR(1)        ,
|          TABLE_ENCODE       CHAR(1)        NOT NULL WITH DEFAULT,
|          TABLE_SCCSID       SMALLINT       NOT NULL WITH DEFAULT,
|          TABLE_MCCSID       SMALLINT       NOT NULL WITH DEFAULT,
|          TABLE_DCCSID       SMALLINT       NOT NULL WITH DEFAULT,
|          ROUTINE_ID          INTEGER        NOT NULL WITH DEFAULT,
|          CTEREF              SMALLINT       NOT NULL WITH DEFAULT,
|          STMTTOKEN           VARCHAR(240)... ,
|          PARENT_PLANNO       SMALLINT       NOT NULL WITH DEFAULT.)
|          IN database-name.table-space-name
|          CCSID UNICODE;


```

Figure 38. 59-column format of PLAN\_TABLE

- For best performance, create an ascending index on the following columns of PLAN\_TABLE:



- QUERYNO
- APPLNAME
- PROGNAME
- VERSION
- COLLID
- OPTHINT

The DB2 sample library, in member DSNTEESC, contains an appropriate

CREATE INDEX statement that you can modify. 

## Enabling optimization hints for the subsystem

You can specify whether DB2 tries to use any optimization hints.

 On the subsystem where the application is bound or where dynamic queries are prepared, specify YES in the OPTIMIZATION HINTS field of installation panel DSN TIP4. If you specify NO, DB2 ignores any hints. 

## Scenario: Preventing a change at rebind

By making an access path into a hint, you can tell DB2 to use that same access path when you rebind or migrate to a new release.

### PSPI

The following scenario assumes that DB2 uses an access path that you like for a particular query and that PLAN\_TABLE contains a row for that desired access path. By making that access path a hint, DB2 uses your hint when you rebind or migrate to a new release.

1. Determine the query number that currently exists for that query in the PLAN\_TABLE. To ensure that the query number is always correlated with that query in the application, modify the statement in the application to use the QUERYNO clause. (If you want to use some kind of numbering convention for queries that use access path hints, you can change the query number in PLAN\_TABLE. The important thing is to have the query in the application have a query number that is unique for that application and that matches the QUERYNO value in the PLAN\_TABLE.)

Here is an example of the QUERYNO clause:

```
SELECT * FROM T1
WHERE C1 = 10 AND
      C2 BETWEEN 10 AND 20 AND
      C3 NOT LIKE 'A%'
QUERYNO 100;
```

For more information about reasons to use the QUERYNO clause, see “Reasons to use the QUERYNO clause” on page 315.

2. Make the PLAN\_TABLE rows for that query (QUERYNO=100) into a hint by updating the OPTHINT column with the name you want to call the hint. In this case, the name is OLDPATH:

```
UPDATE PLAN_TABLE
SET OPTHINT = 'OLDPATH' WHERE
QUERYNO = 100 AND
APPLNAME = ' ' AND
PROGNAME = 'DSNTEP2' AND
VERSION = ' ' AND
COLLID = 'DSNTEP2';
```

3. Tell DB2 to use the hint, and indicate in the PLAN\_TABLE that DB2 used the hint. For dynamic SQL statements in the program, follow these steps:

- a. Execute the SET CURRENT OPTIMIZATION HINT statement in the program to tell DB2 to use OLDPATH. For example:  
SET CURRENT OPTIMIZATION HINT = 'OLDPATH';

If you do not explicitly set the CURRENT OPTIMIZATION HINT special register, the value that you specify for the bind option OPTHINT is used.

If you execute the SET CURRENT OPTIMIZATION HINT statement statically, rebind the plan or package to pick up the SET CURRENT OPTIMIZATION HINT statement.

- b. Execute the EXPLAIN statement on the SQL statements for which you have instructed DB2 to use OLDPATH. This step adds rows to the PLAN\_TABLE for those statements. The rows contain a value of OLDPATH in the HINT\_USED column.

If DB2 uses all of the hints that you provided, it returns SQLCODE +394 from the PREPARE of the EXPLAIN statement and from the PREPARE of SQL statements that use the hints. If any of your hints are invalid, or if any

duplicate hints were found, DB2 issues SQLCODE +395. If DB2 does not find an optimization hint, DB2 returns another SQLCODE. Usually, this SQLCODE is 0.

If the dynamic statement cache is enabled, DB2 does not use the hint to optimize a dynamic SQL statement. If the statement executes successfully in this case, DB2 usually returns SQLCODE 0.

4. For static SQL statements in the program, rebind the plan or package that contains the statements. Specify bind options EXPLAIN(YES) and OPTHINT('OLDPATH') to add rows for those statements in the PLAN\_TABLE that contain a value of OLDPATH in the HINT\_USED column.

If DB2 uses the hint you provided, it returns SQLCODE +394 from the rebind. If your hints are invalid, DB2 issues SQLCODE +395. If DB2 does not find an optimization hint, it returns another SQLCODE. Usually, this SQLCODE is 0.

5. Select from PLAN\_TABLE to see what was used:

```
SELECT *
FROM PLAN_TABLE
WHERE QUERYNO = 100
ORDER BY TIMESTAMP, QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
```

The PLAN\_TABLE in The following table shows the OLDPATH hint, indicated by a value in OPTHINT and it also shows that DB2 used that hint, indicated by OLDPATH in the HINT\_USED column.

Table 55. PLAN\_TABLE that shows that the OLDPATH optimization hint is used.

QUERYNO	METHOD	TNAME	OPTHINT	HINT_USED
100	0	EMP	OLDPATH	
100	4	EMPPROJACT	OLDPATH	
100	3		OLDPATH	
100	0	EMP		OLDPATH
100	4	EMPPROJECT		OLDPATH
100	3			OLDPATH



## Scenario: Modifying an existing access path

The following scenario assumes that DB2 uses a hybrid join where you know it can perform better with a sort merge join. The example assumes that the query is dynamic.



1. Put the old access path in the PLAN\_TABLE and associate it with a query number.

```
EXPLAIN ALL SET QUERYNO=200 FOR
SELECT X.ACTNO, X.PROJNO, X.EMPNO, Y.JOB, Y.EDLEVEL
FROM DSN8610.EMPPROJACT X, DSN8610.EMP Y
WHERE X.EMPNO = Y.EMPNO
AND X.EMPTIME > 0.5
AND (Y.JOB = 'DESIGNER' OR Y.EDLEVEL >= 12)
ORDER BY X.ACTNO, X.PROJNO;
```

2. Make the PLAN\_TABLE rows into a hint by updating the OPTHINT column with the name you want to call the hint. In this case, the name is NOHYB:

```

UPDATE PLAN_TABLE
SET OPTHINT = 'NOHYB' WHERE
QUERYNO = 200 AND
APPLNAME = ' ' AND
PROGNAME = 'DSNTEP2' AND
VERSION = '' AND
COLLID = 'DSNTEP2';

```

3. Change the access path so that merge scan join is used rather than hybrid join:

```

UPDATE PLAN_TABLE
SET METHOD = 2 WHERE
QUERYNO = 200 AND
APPLNAME = ' ' AND
PROGNAME = 'DSNTEP2' AND
VERSION = '' AND
COLLID = 'DSNTEP2' AND
OPTHINT = 'NOHYB' AND
METHOD = 4;

```

4. Tell DB2 to look for the NOHYB hint for this query:

```
SET CURRENT OPTIMIZATION HINT = 'NOHYB';
```

5. Explain the query again to check for the results:

```

EXPLAIN ALL SET QUERYNO=200 FOR
SELECT X.ACTNO, X.PROJNO, X.EMPNO, Y.JOB, Y.EDLEVEL
FROM DSN8610.EMPPROJECT X, DSN8610.EMP Y
WHERE X.EMPNO = Y.EMPNO
AND X.EMPTIME > 0.5
AND (Y.JOB = 'DESIGNER' OR Y.EDLEVEL >= 12)
ORDER BY X.ACTNO, X.PROJNO;

```

6. Select from the PLAN\_TABLE to verify the results:

```

SELECT *
FROM PLAN_TABLE
WHERE QUERYNO = 200
ORDER BY TIMESTAMP, QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;

```

The PLAN\_TABLE in The following table shows the NOHYB hint, indicated by a value in OPTHINT and it also shows that DB2 used that hint, indicated by NOHYB in the HINT\_USED column.

Table 56. PLAN\_TABLE that shows that the NOHYB optimization hint is used.

QUERYNO	METHOD	TNAME	OPTHINT	HINT_USED
200	0	EMP	NOHYB	
200	2	EMPPROJECT	NOHYB	
200	3		NOHYB	
200	0	EMP		NOHYB
200	2	EMPPROJECT		NOHYB
200	3			NOHYB

7. Analyze the performance of the statement to see if it is acceptable. 

## Reasons to use the QUERYNO clause


You do not need to assign a query number to use optimization hints. If you don't assign a query number, DB2 uses the statement number. However, query numbers can be useful in certain cases.

 Assigning a query number is especially useful in the following cases:

- For dynamic statements



The query number for dynamic applications is the statement number in the application *where the prepare occurs*. For some applications, such as DSNTEP2, the same statement in the application prepares each dynamic statement, resulting in the same query number for each dynamic statement. Assigning a query number to each statement that uses optimization hints eliminates ambiguity as to which rows in the PLAN\_TABLE are associated with each query.

- For static statements

If you change an application that has static statements, the statement number might change, causing rows in the PLAN\_TABLE to be out of sync with the modified application. Statements that use the QUERYNO clause are not dependent on the statement number. You can move those statements around without affecting the relationship between rows in the PLAN\_TABLE and the statements that use those rows in the application. 


### How DB2 locates the PLAN\_TABLE rows for a hint

DB2 uses the QUERYNO, APPLNAME, PROGNAME, VERSION, COLLID, and OPTHINT columns of the PLAN\_TABLE to determine the rows to use for a hint.

 For a PLAN\_TABLE row, the QUERYNO, APPLNAME, PROGNAME, VERSION, and COLLID values must match the corresponding values for an SQL statement before the SQL statement can use that row. In addition, the OPTHINT value for that row must match the value in the CURRENT OPTIMIZATION HINT special register if the SQL statement is executed dynamically. If the SQL statement is executed statically, the OPTHINT value for the row must match the value of bind option OPTHINT for the package or plan that contains the SQL statement. If no PLAN\_TABLE rows meet these conditions, DB2 determines the access path for the SQL statement without using hints. 

### How DB2 validates the hint

When you specify an optimization hint by using the OPTHINT bind option for static statements, or the CURRENT OPTIMIZATION HINT special register for dynamic statements, DB2 ensures that you choose a valid access path.

 To ensure a valid access path, DB2 validates the PLAN\_TABLE rows in which the value in the OPTHINT column matches the specified hint. If the access path that you specify has major problems, DB2 invalidates all hints for that query block. In that event, DB2 determines the access path as it normally does.

DB2 validates the information in **only** the PLAN\_TABLE columns in Table 57.

Table 57. PLAN\_TABLE columns that DB2 validates

Column	Correct values or other explanation
METHOD	Must be 0, 1, 2, 3, or 4. Any other value invalidates the hints.
CREATOR and TNAME	Must be specified and must name a table, materialized view, materialized nested table expression. Blank if method is 3. If a table is named that does not exist or is not involved in the query, then the hints are invalid.

Table 57. *PLAN\_TABLE* columns that DB2 validates (continued)

Column	Correct values or other explanation
TABNO	<p>Required only if CREATOR, TNAME, and CORRELATION_NAME do not uniquely identify the table. This situation might occur when the same table is used in multiple views (with the same CORRELATION_NAME).</p> <p>This field is ignored if it is not needed.</p>
ACCESSTYPE	<p>Must contain I, I1, N, M, R, RW, T, or V. Any other value invalidates the hints.</p> <p>Values of I, I1, and N all mean single index access. DB2 determines which of the three values to use based on the index specified in ACCESSNAME.</p> <p>M indicates multiple index access. DB2 uses only the first row in the authid.PLAN_TABLE for multiple index access (MIXOPSEQ=0). The choice of indexes, and the AND and OR operations involved, is determined by DB2. If multiple index access isn't possible, then the hints are invalidated.</p>
ACCESSCREATOR and ACCESSNAME	<p>Ignored if ACCESSTYPE is R or M. If ACCESSTYPE is I, I1, or N, then these fields must identify an index on the specified table.</p> <p>If the index doesn't exist, or if the index is defined on a different table, then the hints are invalid. Also, if the specified index can't be used, then the hints are invalid.</p>
SORTN_JOIN and SORTC_JOIN	<p>Must be Y, N or blank. Any other value invalidates the hints.</p> <p>This value determines if DB2 should sort the new (SORTN_JOIN) or composite (SORTC_JOIN) table. This value is ignored if the specified join method, join sequence, access type and access name dictate whether a sort of the new or composite tables is required.</p>
PREFETCH	<p>Must be S, L or blank. Any other value invalidates the hints.</p> <p>This value determines whether DB2 should use sequential prefetch (S), list prefetch (L), or no prefetch (blank). (A blank does not prevent sequential detection at run time.) This value is ignored if the specified access type and access name dictates the type of prefetch required.</p>
PAGE_RANGE	<p>Must be Y, N or blank. Any other value invalidates the hints.</p>
PARALLELISM_MODE	<p>This value is used only if it is possible to run the query in parallel; that is, the SET CURRENT DEGREE special register contains ANY, or the plan or package was bound with DEGREE(ANY).</p> <p>If parallelism is possible, this value must be I, C, X or null. All of the restrictions involving parallelism still apply when using access path hints. If the specified mode cannot be performed, the hints are either invalidated or the mode is modified by the optimizer, possibly resulting in the query being run sequentially. If the value is null then the optimizer determines the mode.</p>

Table 57. *PLAN\_TABLE* columns that DB2 validates (continued)

Column	Correct values or other explanation
ACCESS_DEGREE or JOIN_DEGREE	<p>If PARALLELISM_MODE is specified, use this field to specify the degree of parallelism. If you specify a degree of parallelism, this must be a number greater than zero, and DB2 might adjust the parallel degree from what you set here. If you want DB2 to determine the degree, do not enter a value in this field.</p> <p>If you specify a value for ACCESS_DEGREE or JOIN_DEGREE, you must also specify a corresponding ACCESS_PGROUP_ID and JOIN_PGROUP_ID.</p>
WHEN_OPTIMIZE	<p>Must be R, B, or blank. Any other value invalidates the hints.</p> <p>When a statement in a plan that is bound with REOPT(ALWAYS) qualifies for reoptimization at run time, and you have provided optimization hints for that statement, the value of WHEN_OPTIMIZE determines whether DB2 reoptimizes the statement at run time. If the value of WHEN_OPTIMIZE is blank or B, DB2 uses only the access path that is provided by the optimization hints at bind time. If the value of WHEN_OPTIMIZE is R, DB2 determines the access path at bind time using the optimization hints. At run time, DB2 searches the PLAN_TABLE for hints again, and if hints for the statement are still in the PLAN_TABLE and are still valid, DB2 optimizes the access path using those hints again.</p>
PRIMARY_ACCESTYPE	Must be D or blank. Any other value invalidates the hints.

#### PSPI

### Limitations on optimization hints

Optimization hints cannot force or undo query transformations, such as subquery transformation to join or materialization or merge of a view or table expression.

**PSPI** A query that is not transformed in one release of DB2 might be transformed in a later release of DB2. Therefore, if you include a hint in one release of DB2 for a query that is not transformed in that release, but in a later release of DB2, the query is transformed, DB2 does not use the hint in the later release. **PSPI**

---

## Chapter 15. Programming for concurrency

DB2 processes acquire, or avoid acquiring, locks based on certain general parameters.

**GUIP** To preserve data integrity, your application process acquires locks implicitly, under the control of DB2. It is not necessary for a process to explicitly request a lock to conceal uncommitted data. Therefore, do not always need to do anything about DB2 locks. However, you can make better use of your resources and improve concurrency by understanding the effects of the parameters that DB2 uses to control locks. **GUIP**

---

### Concurrency and locks defined

*Concurrency* is the ability of more than one application process to access the same data at essentially the same time.

**PSPI**

#### Example

An application for order entry is used by many transactions simultaneously. Each transaction makes inserts in tables of invoices and invoice items, reads a table of data about customers, and reads and updates data about items on hand. Two operations on the same data, by two simultaneous transactions, might be separated only by microseconds. To the users, the operations appear concurrent.

#### Conceptual background

Concurrency must be controlled to prevent lost updates and such possibly undesirable effects as unrepeatable reads and access to uncommitted data.

##### Lost updates

Without concurrency control, two processes, A and B, might both read the same row from the database, and both calculate new values for one of its columns, based on what they read. If A updates the row with its new value, and then B updates the same row, A's update is lost.

##### Access to uncommitted data

Also without concurrency control, process A might update a value in the database, and process B might read that value before it was committed. Then, if A's value is not later committed, but backed out, B's calculations are based on uncommitted (and presumably incorrect) data.

##### Unrepeatable reads

Some processes require the following sequence of events: A reads a row from the database and then goes on to process other SQL requests. Later, A reads the first row again and must find the same values it read the first time. Without control, process B could have changed the row between the two read operations.

To prevent those situations from occurring unless they are specifically allowed, DB2 might use *locks* to control concurrency.

## How DB2 uses locks

A lock associates a DB2 resource with an application process in a way that affects how other processes can access the same resource. The process associated with the resource is said to “hold” or “own” the lock. DB2 uses locks to ensure that no process accesses data that has been changed, but not yet committed, by another process. For XML and LOB locks, DB2 also uses locks to ensure that an application cannot access partial or incomplete data

## Effects of DB2 locks

Locks can cause situations that have a negative effect on DB2 performance.

The effects of locks that you want to avoid and minimize include the following situations.

### Suspension

An application process is *suspended* when it requests a lock that is already held by another application process and cannot be shared. The suspended process temporarily stops running.

**PSPI**

### Order of precedence for lock requests

Incoming lock requests are queued. Requests for lock promotion, and requests for a lock by an application process that already holds a lock on the same object, precede requests for locks by new applications. Within those groups, the request order is “first in, first out.”

### Example suspension

Using an application for inventory control, two users attempt to reduce the quantity on hand of the same item at the same time. The two lock requests are queued. The second request in the queue is suspended and waits until the first request releases its lock.

### Effects of suspension

The suspended process resumes running when:

- All processes that hold the conflicting lock release it.
- The requesting process times out or deadlocks and the process resumes to deal with an error condition.

**PSPI**

### Time out

An application process is said to *time out* when it is terminated because it has been suspended for longer than a preset interval.

**PSPI**

### Example time out

An application process attempts to update a large table space that is being reorganized by the utility REORG TABLESPACE with SHRLEVEL NONE. It is

likely that the utility job does not release control of the table space before the application process times out.


### Effects of time outs

DB2 terminates the process, issues two messages to the console, and returns SQLCODE -911 or -913 to the process (SQLSTATES '40001' or '57033'). Reason code 00C9008E is returned in the SQLERRD(3) field of the SQLCA. Alternatively, you can use the GET DIAGNOSTICS statement to check the reason code. If statistics trace class 3 is active, DB2 writes a trace record with IFCID 0196.

### Effects in IMS

If you are using IMS, and a timeout occurs, the following actions take place:

- In a DL/I batch application, the application process abnormally terminates with a completion code of 04E and a reason code of 00D44033 or 00D44050.
- In any IMS environment except DL/I batch:
  - DB2 performs a rollback operation on behalf of your application process to undo all DB2 updates that occurred during the current unit of work.
  - For a non-message driven BMP, IMS issues a rollback operation on behalf of your application. If this operation is successful, IMS returns control to your application, and the application receives SQLCODE -911. If the operation is unsuccessful, IMS issues user abend code 0777, and the application does not receive an SQLCODE.
  - For an MPP, IFP, or message driven BMP, IMS issues user abend code 0777, rolls back all uncommitted changes, and reschedules the transaction. The application does not receive an SQLCODE.

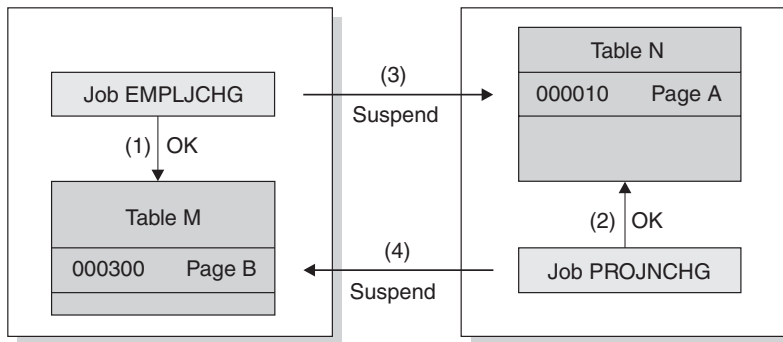
COMMIT and ROLLBACK operations do not time out. The command STOP DATABASE, however, might time out and send messages to the console, but it retries up to 15 times. 

### Deadlock

A *deadlock* occurs when two or more application processes each hold locks on resources that the others need and without which they cannot proceed.

### Example deadlock

 The following figure illustrates a deadlock between two transactions.



#### Notes:

1. Jobs EMPLJCHG and PROJNCHG are two transactions. Job EMPLJCHG accesses table M, and acquires an exclusive lock for page B, which contains record 000300.
2. Job PROJNCHG accesses table N, and acquires an exclusive lock for page A, which contains record 000010.
3. Job EMPLJCHG requests a lock for page A of table N while still holding the lock on page B of table M. The job is suspended, because job PROJNCHG is holding an exclusive lock on page A.
4. Job PROJNCHG requests a lock for page B of table M while still holding the lock on page A of table N. The job is suspended, because job EMPLJCHG is holding an exclusive lock on page B. The situation is a deadlock.

Figure 39. A deadlock example

### Effects of deadlocks

After a preset time interval (the value of DEADLOCK TIME), DB2 can roll back the current unit of work for one of the processes or request a process to terminate. That frees the locks and allows the remaining processes to continue. If statistics trace class 3 is active, DB2 writes a trace record with IFCID 0172. Reason code 00C90088 is returned in the SQLERRD(3) field of the SQLCA. Alternatively, you can use the GET DIAGNOSTICS statement to check the reason code. (The codes that describe the exact DB2 response depend on the operating environment.)

It is possible for two processes to be running on distributed DB2 subsystems, each trying to access a resource at the other location. In that case, neither subsystem can detect that the two processes are in deadlock; the situation resolves only when one process times out.

**Indications of deadlocks:** In some cases, a deadlock can occur if two application processes attempt to update data in the same page or table space.

### Deadlocks and TSO, Batch, and CAF

When a deadlock or timeout occurs in these environments, DB2 attempts to roll back the SQL for one of the application processes. If the ROLLBACK is successful, that application receives SQLCODE -911. If the ROLLBACK fails, and the application does not abend, the application receives SQLCODE -913.

### Deadlocks and IMS

If you are using IMS, and a deadlock occurs, the following actions take place:


- In a DL/I batch application, the application process abnormally terminates with a completion code of 04E and a reason code of 00D44033 or 00D44050.
- In any IMS environment except DL/I batch:

- DB2 performs a rollback operation on behalf of your application process to undo all DB2 updates that occurred during the current unit of work.
- For a non-message driven BMP, IMS issues a rollback operation on behalf of your application. If this operation is successful, IMS returns control to your application, and the application receives SQLCODE -911. If the operation is unsuccessful, IMS issues user abend code 0777, and the application does not receive an SQLCODE.
- For an MPP, IFP, or message driven BMP, IMS issues user abend code 0777, rolls back all uncommitted changes, and reschedules the transaction. The application does not receive an SQLCODE.

## Deadlocks and CICS

If you are using CICS and a deadlock occurs, the CICS attachment facility decides whether or not to roll back one of the application processes, based on the value of the ROLBE or ROLBI parameter. If your application process is chosen for rollback, it receives one of two SQLCODEs in the SQLCA:



- 911 A SYNCPOINT command with the ROLLBACK option was issued on behalf of your application process. All updates (CICS commands and DL/I calls, as well as SQL statements) that occurred during the current unit of work have been undone. (SQLSTATE '40001')
- 913 A SYNCPOINT command with the ROLLBACK option was not issued. DB2 rolls back only the incomplete SQL statement that encountered the deadlock or timed out. CICS does not roll back any resources. Your application process should either issue a SYNCPOINT command with the ROLLBACK option itself or terminate. (SQLSTATE '57033')

Consider using the DSNTIAC subroutine to check the SQLCODE and display the SQLCA. Your application must take appropriate actions before resuming. 

---

## Promoting basic concurrency

By following certain basic recommendations, you can promote concurrency in your DB2 system.

 Recommendations are grouped by their scope. 

## Using system and subsystem options to promote concurrency

Some performance problems can appear to be locking problems although they are really problems somewhere else in the system.

For example, a table space scan of a large table can result in timeouts.

To improve concurrency on your DB2 system:

- Resolve overall system, subsystem, and application performance problems to ensure that you not only eliminate locking symptoms but also correct other underlying performance problems.
- Consider reducing the number of threads or initiators, increasing the priority for the DB2 tasks, and providing more processing, I/O, and real memory. If a task is waiting or is swapped out and the unit of work has not been committed, then it still holds locks. When a system is heavily loaded, contention for processing, I/O, and storage can cause waiting.

## Designing your databases for concurrency

By following general recommendations and best practices for database design you can ensure improved concurrency on your DB2 system.

**PSPI** To design your database to promote concurrency:

- Keep like things together:
  - Give each application process that creates private tables a private database.
  - Put tables relevant to the same application into the same database.
  - Put tables together in a segmented table space if they are similar in size and can be recovered together.
- Use an adequate number of databases, schema or authorization-ID qualifiers, and table spaces to keep unlike things apart. Concurrency and performance is improved for SQL data definition statements, GRANT statements, REVOKE statements, and utilities. For example, a general guideline is a maximum of 50 tables per database.
- Plan for batch inserts. If your application does sequential batch insertions, excessive contention on the space map pages for the table space can occur.

This problem is especially apparent in data sharing, where contention on the space map means the added overhead of page P-lock negotiation. For these types of applications, consider using the MEMBER CLUSTER option of CREATE TABLESPACE. This option causes DB2 to disregard the clustering index (or implicit clustering index) when assigning space for the SQL INSERT statement.
- Use LOCKSIZE ANY until you have reason not to. LOCKSIZE ANY is the default for CREATE TABLESPACE.

It allows DB2 to choose the lock size, and DB2 usually chooses LOCKSIZE PAGE and LOCKMAX SYSTEM for non-LOB/non-XML table spaces. For LOB table spaces, DB2 chooses LOCKSIZE LOB and LOCKMAX SYSTEM. Similarly, for XML table spaces, DB2 chooses LOCKSIZE XML and LOCKMAX SYSTEM. You should use LOCKSIZE TABLESPACE or LOCKSIZE TABLE only for read-only table spaces or tables, or when concurrent access to the object is not needed. Before you choose LOCKSIZE ROW, you should estimate whether doing so increases the overhead for locking, especially from P-locks in a data sharing environment, and weigh that against the increase in concurrency.
- For small tables with high concurrency requirements, estimate the number of pages in the data and in the index. In this case, you can spread out your data to improve concurrency, or consider it a reason to use row locks. If the index entries are short or they have many duplicates, then the entire index can be one root page and a few leaf pages.
- Partition large tables to take advantage of parallelism for online queries, batch jobs, and utilities.

When batch jobs are run in parallel and each job goes after different partitions, lock contention is reduced. In addition, in data sharing environments, data sharing overhead is reduced when applications that are running on different members go after different partitions.

Certain utility operations, such as LOAD and REORG, cannot take advantage of parallelism in partition-by-growth table spaces.
- Partition secondary indexes. By using data-partitioned secondary indexes (DPSIs) you can promote partition independence and, thereby reduce lock contention and improve index availability, especially for utility processing, partition-level operations (such as dropping or rotating partitions), and recovery of indexes.

However, using data-partitioned secondary indexes does not always improve the performance of queries. For example, for a query with a predicate that references only the columns of a data-partitioned secondary index, DB2 must probe each partition of the index for values that satisfy the predicate if index access is chosen as the access path. Therefore, take into account data access patterns and maintenance practices when deciding to use a data-partitioned secondary index. Replace a nonpartitioned index with a partitioned index only if you will realize perceivable benefits such as improved data or index availability, easier data or index maintenance, or improved performance.

- Specify fewer rows of data per page. You can use the MAXROWS clause of CREATE or ALTER TABLESPACE, to specify the maximum number of rows that can be on a page. For example, if you use MAXROWS 1, each row occupies a whole page, and you confine a page lock to a single row. Consider this option if you have a reason to avoid using row locking, such as in a data sharing environment where row locking overhead can be greater.
- If multiple applications access the same table, consider defining the table as VOLATILE. DB2 uses index access whenever possible for volatile tables, even if index access does not appear to be the most efficient access method because of volatile statistics. Because each application generally accesses the rows in the table in the same order, lock contention can be reduced.
- Consider using randomized index key columns. In a data sharing environment, you can use randomized index key columns to reduce locking contention at the possible cost of more CPU usage, from increased locking and getpage operations, and more index page read and write I/Os.

This technique is effective for reducing contention on certain types of equality predicates. For example, if you create an index on a timestamp column, where the timestamp is always filled with the current time, every insertion on the index would be the greatest value and cause contention on the page at the end of the index. An index on a column of sequential values, such as invoice numbers, causes similar contention, especially in heavy transaction workload environments. In each case, using the RANDOM index order causes the values to be stored at random places in the index tree, and reduce the chance that consecutive insertions hit the same page and cause contention.

Although the randomized index can relieve contention problems for sets of similar or sequential values, it does not help with identical values. Identical values encode the same and each are inserted at the same place on the index

tree. 

## Programming your applications for concurrency

By following general recommendations and best practices for database design you can ensure improved concurrency on your DB2 system.

 **PSPI**

To design your applications for concurrency:

- When two different applications access the same data, try to make them do so in the same sequence. For, example if two applications access five rows of data, make both access rows 1,2,3,5 in that order. In that case, the first application to access the data delays the second, but the two applications cannot deadlock. For the same reason, try to make different applications access the same tables in the same order.
- To avoid unnecessary lock contention, issue a COMMIT statement as soon as possible after reaching a point of consistency, even in read-only applications. Statements issued through SPUFI can be committed immediately by the SPUFI autocommit feature.

Taking commit points frequently in a long running unit of recovery (UR) has the following benefits at the possible cost of more CPU usage and log write I/Os:

- Reduces lock contention, especially in a data sharing environment
- Improves the effectiveness of lock avoidance, especially in a data sharing environment
- Reduces the elapsed time for DB2 system restart following a system failure
- Reduces the elapsed time for a unit of recovery to rollback following an application failure or an explicit rollback request by the application
- Provides more opportunity for utilities, such as online REORG, to break in

Consider using the UR CHECK FREQUENCY field or the UR LOG WRITE CHECK field of installation panel DSNTIPN to help you identify those applications that are not committing frequently. UR CHECK FREQUENCY, which identifies when too many checkpoints have occurred without a UR issuing a commit, is helpful in monitoring overall system activity. UR LOG WRITE CHECK enables you to detect applications that might write too many log records between commit points, potentially creating a lengthy recovery situation for critical tables.

Even though an application might conform to the commit frequency standards of the installation under normal operational conditions, variation can occur based on system workload fluctuations. For example, a low-priority application might issue a commit frequently on a system that is lightly loaded. However, under a heavy system load, the use of the CPU by the application might be preempted, and, as a result, the application might violate the rule set by the UR CHECK FREQUENCY parameter. For this reason, add logic to your application to commit based on time elapsed since last commit, and not solely based on the amount of SQL processing performed. In addition, take frequent commit points in a long running unit of work that is read-only to reduce lock contention and to provide opportunities for utilities, such as online REORG, to access the data.

Committing frequently is equally important for objects that are not logged and objects that are logged. Make sure, for example, that you commit work frequently even if the work is done on a table space that is defined with the NOT LOGGED option. Even when a given transaction modifies only tables that reside in not logged table spaces, a unit of recovery is still established before updates are performed. Undo processing will continue to read the log in the backward direction looking for undo log records that must be applied until it detects the beginning of this unit of recovery as recorded on the log. Therefore, such transactions should perform frequent commits to limit the distance undo processing might have to go backward on the log to find the beginning of the unit of recovery.

- Include logic in a batch program so that it retries an operation after a deadlock or timeout. Such a method could help you recover from the situation without assistance from operations personnel.
  - Field SQLERRD(3) in the SQLCA returns a reason code that indicates whether a deadlock or timeout occurred.
  - Alternatively, you can use the GET DIAGNOSTICS statement to check the reason code.
- Close cursors. If you define a cursor using the WITH HOLD option, the locks it needs can be held past a commit point.
  - Use the CLOSE CURSOR statement as soon as possible in your application to cause those locks to be released and the resources that they hold to be freed at the first commit point that follows the CLOSE CURSOR statement.

- Use the `RELEASE LOCKS` parameter on the `DSNTIP4` installation panel to control whether page or row locks are held for `WITH HOLD` cursors. Closing cursors is particularly important in a distributed environment.

- Bind plans with the `ACQUIRE(USE)` option in most cases. `ACQUIRE(USE)`, which indicates that DB2 acquires table and table space locks when the objects are first used and not when the plan is allocated, is the best choice for concurrency. Packages are always bound with `ACQUIRE(USE)`, by default.

`ACQUIRE(ALLOCATE)`, which is an option for plans, but not for packages, can provide better protection against timeouts. Consider `ACQUIRE(ALLOCATE)` for applications that need gross locks instead of intent locks or that run with other applications that might request gross locks instead of intent locks. Acquiring the locks at plan allocation also prevents any one transaction in the application from incurring the cost of acquiring the table and table space locks. If you need `ACQUIRE(ALLOCATE)`, you might want to bind all DBRMs directly to the plan.

- Bind applications with the `ISOLATION(CS)` and `CURRENTDATA(NO)` options in most cases. `ISOLATION(CS)` lets DB2 release acquired row and page locks as soon as possible. `CURRENTDATA(NO)` lets DB2 avoid acquiring row and page locks as often as possible. When you use `ISOLATION(CS)` and `CURRENTDATA(NO)`, consider using the `SKIPUNCI` subsystem parameter value to `YES` so that readers do not wait for the outcome of uncommitted inserts.

If you do not use `ISOLATION(CS)` and `CURRENTDATA(NO)`, in order of decreasing preference for concurrency, use the bind options:

1. `ISOLATION(CS)` with `CURRENTDATA(YES)`, when data returned to the application must not be changed before your next `FETCH` operation.
  2. `ISOLATION(RS)`, when data returned to the application must not be changed before your application commits or rolls back. However, you do not care if other application processes insert additional rows.
  3. `ISOLATION(RR)`, when data evaluated as the result of a query must not be changed before your application commits or rolls back. New rows cannot be inserted into the answer set.
- Use `ISOLATION(UR)` option cautiously. The Resource Recovery Services attachment facility `UR` isolation acquires almost no locks on rows or pages. It is fast and causes little contention, but it reads uncommitted data. Do not use it unless you are sure that your applications and end users can accept the logical inconsistencies that can occur.

As an alternative, consider using the `SKIP LOCKED` data option if omitting data is preferable to reading uncommitted data in your application.

- Use sequence objects to generate unique, sequential numbers. Using an identity column is one way to generate unique sequential numbers.

However, as a column of a table, an identity column is associated with and tied to the table, and a table can have only one identity column. Your applications might need to use one sequence of unique numbers for many tables or several sequences for each table. As a user-defined object, sequences provide a way for applications to have DB2 generate unique numeric key values and to coordinate the keys across multiple rows and tables.

The use of sequences can avoid the lock contention problems that can result when applications implement their own sequences, such as in a one-row table that contains a sequence number that each transaction must increment. With DB2 sequences, many users can access and increment the sequence concurrently without waiting. DB2 does not wait for a transaction that has incremented a sequence to commit before allowing another transaction to increment the sequence again.

- Examine multi-row operations such as multi-row inserts, positioned updates, and positioned deletes, which have the potential of expanding the unit of work. This situation can affect the concurrency of other users that access the data. You can minimize contention by adjusting the size of the host-variable-array, committing between inserts, updates, and preventing lock escalation.
- Use global transactions. The Resource Recovery Services attachment facility (RRSAF) relies on a z/OS component called Resource Recovery Services (RRS). RRS provides system-wide services for coordinating two-phase commit operations across z/OS products. For RRSAF applications and IMS transactions that run under RRS, you can group together a number of DB2 agents into a single global transaction.

A global transaction allows multiple DB2 agents to participate in a single global transaction and thus share the same locks and access the same data. When two agents that are in a global transaction access the same DB2 object within a unit of work, those agents do not deadlock or timeout with each other. The following restrictions apply:

- Parallel Sysplex® is not supported for global transactions.
- Because each of the "branches" of a global transaction are sharing locks, uncommitted updates issued by one branch of the transaction are visible to other branches of the transaction.
- Claim/drain processing is not supported across the branches of a global transaction, which means that attempts to issue CREATE, DROP, ALTER, GRANT, or REVOKE might deadlock or timeout if they are requested from different branches of the same global transaction.
- LOCK TABLE might deadlock or timeout across the branches of a global transaction.

- Use optimistic concurrency control. Optimistic concurrency control represents a faster, more scalable locking alternative to database locking for concurrent data access. It minimizes the time for which a given resource is unavailable for use by other transactions.

When an application uses optimistic concurrency control, locks are obtained immediately before a read operation and released immediately. Update locks are obtained immediately before an update operation and held until the end of the transaction. Optimistic concurrency control uses the RID and a row change token to test whether data has been changed by another transaction since the last read operation.

Because DB2 can determine when a row was changed, it can ensure data integrity while limiting the time that locks are held. With optimistic concurrency control, DB2 releases the row or page locks immediately after a read operation. DB2 also releases the row lock after each FETCH, taking a new lock on a row only for a positioned update or delete to ensure data integrity.

To safely implement optimistic concurrency control, you must establish a row change timestamp column with a CREATE TABLE statement or an ALTER TABLE statement. The column must be defined as NOT NULL GENERATED ALWAYS FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP or NOT NULL GENERATED BY DEFAULT FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP. After you establish a row change timestamp column, DB2 maintains the contents of this column. When you want to use this change token as a condition when making an update, you can specify an

appropriate condition for this column in your WHERE clause. 


---

## Aspects of transaction locks

Understanding the sizes, durations, modes, and objects of transaction locks can help you understand why a process suspends or times out or why two processes deadlock, and how you might change the situation.

### Lock size

The *size* (sometimes *scope* or *level*) of a lock on data in a table describes the amount of data that is controlled by the lock. The same piece of data can be controlled by locks of different sizes.

 DB2 uses locks of the following sizes:

- Table space
- Table
- Partition
- Page
- Row

A table space lock (the largest size) controls the most data, all the data in an entire table space. A page or row lock controls only the data in a single page or row.

As the following figure suggests, row locks and page locks occupy an equal place in the hierarchy of lock sizes.

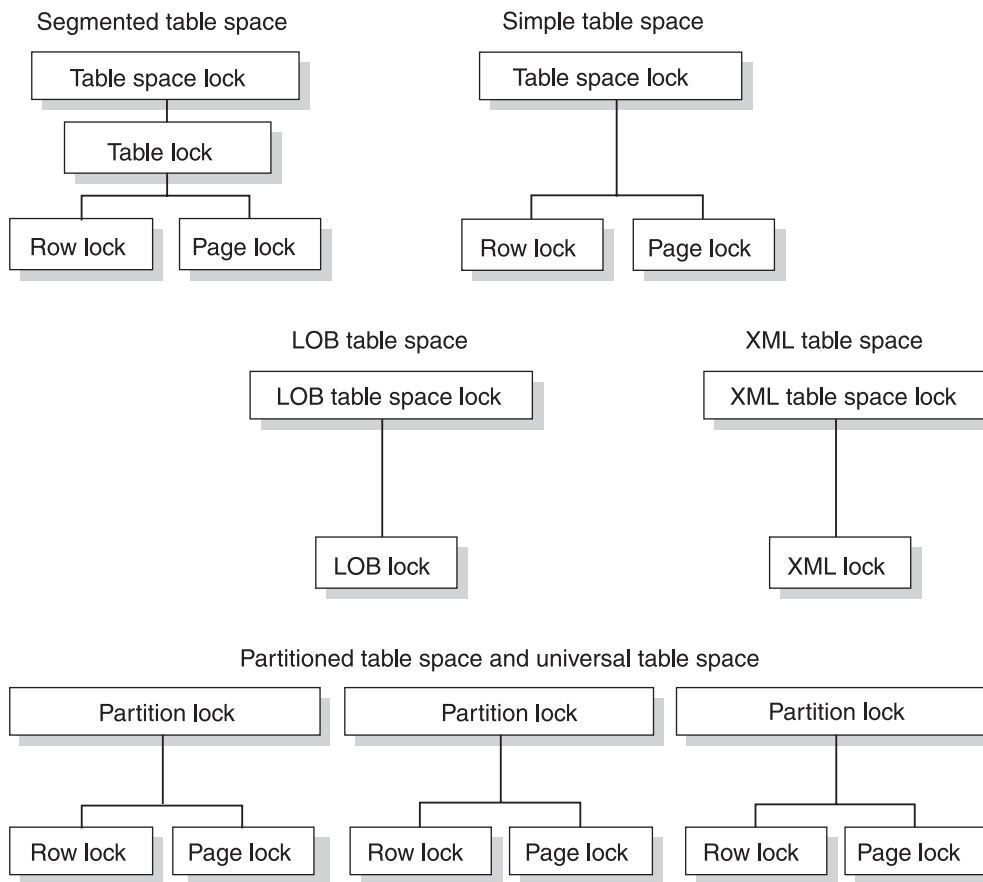


Figure 40. Sizes of objects locked



## Hierarchy of lock sizes

The same piece of data can be controlled by locks of different sizes.



A table space lock (the largest size) controls the most data, all the data in an entire table space. A page or row lock controls only the data in a single page or row.

As the following figure suggests, row locks and page locks occupy an equal place in the hierarchy of lock sizes.

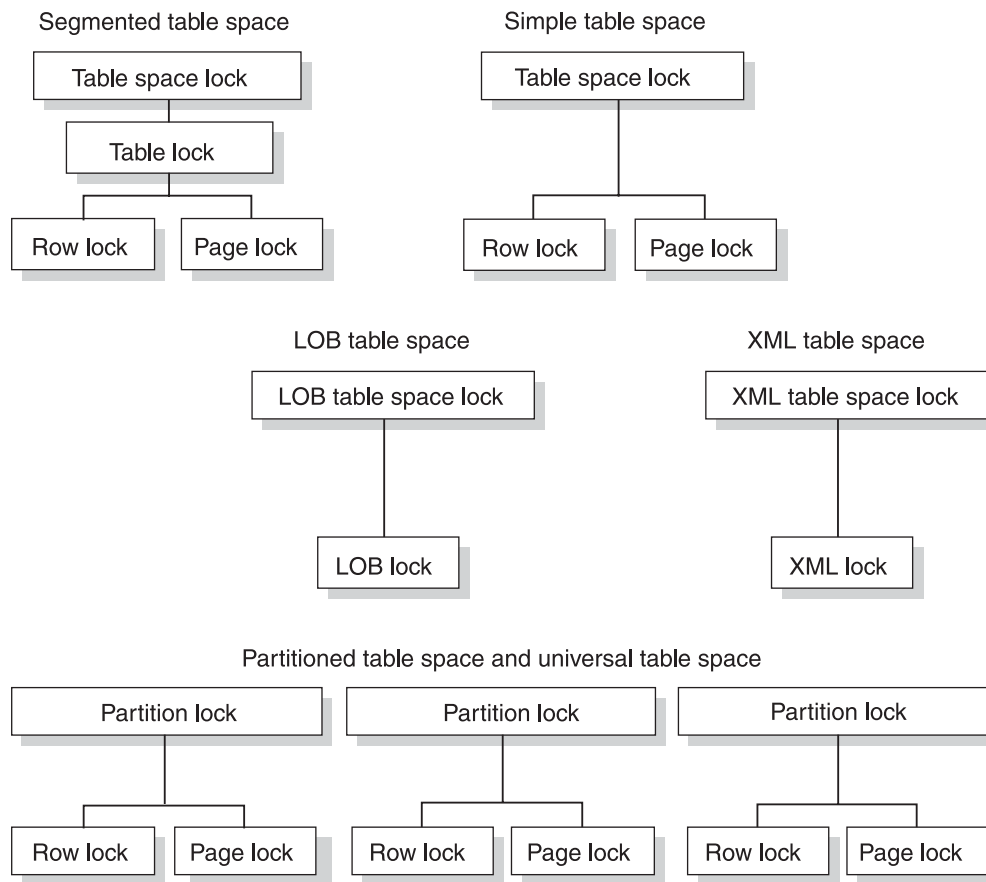


Figure 41. Sizes of objects locked

#### PSPI

### General effects of lock size

Locking larger or smaller amounts of data allows you to trade performance for concurrency.

#### PSPI

Using page or row locks instead of table or table space locks has the following effects:

- Concurrency usually improves, meaning better response times and higher throughput rates for many users.
- Processing time and use of storage increases. That is especially evident in batch processes that scan or update a large number of rows.

Using only table or table space locks has the following effects:

- Processing time and storage usage is reduced.
- Concurrency can be reduced, meaning longer response times for some users but better throughput for one user.

#### PSPI

### Lock sizes and table space type

DB2 different lock sizes depending on the type of table spaces where the locks are acquired.

## Partitioned and universal table space

In a partitioned table space or universal table space, locks are obtained at the partition level. Individual partitions are locked as they are accessed. Gross locks (S, U, or X) can be obtained on individual partitions instead of on the entire partitioned table space. (Partition locks are always acquired with the LOCKPART NO clause. For table spaces that are defined with the LOCKPART NO option, DB2 no longer locks the entire table space with one lock when any partition of the table space is accessed.)


**Restriction:** If any of the following conditions are true, DB2 must lock all partitions:

- The plan is bound with ACQUIRE(ALLOCATE).
- The table space is defined with LOCKSIZE TABLESPACE.
- The LOCK TABLE statement is used without the PART option.

## Segmented table space

In a segmented table space without partitions, rows from different tables are contained in different pages. Locking a page does not lock data from more than one table. Also, DB2 can acquire a table lock, which locks only the data from one specific table. Because a single row, of course, contains data from only one table, the effect of a row lock is the same as for a simple or partitioned table space: it locks one row of data from one table.

## Simple table space

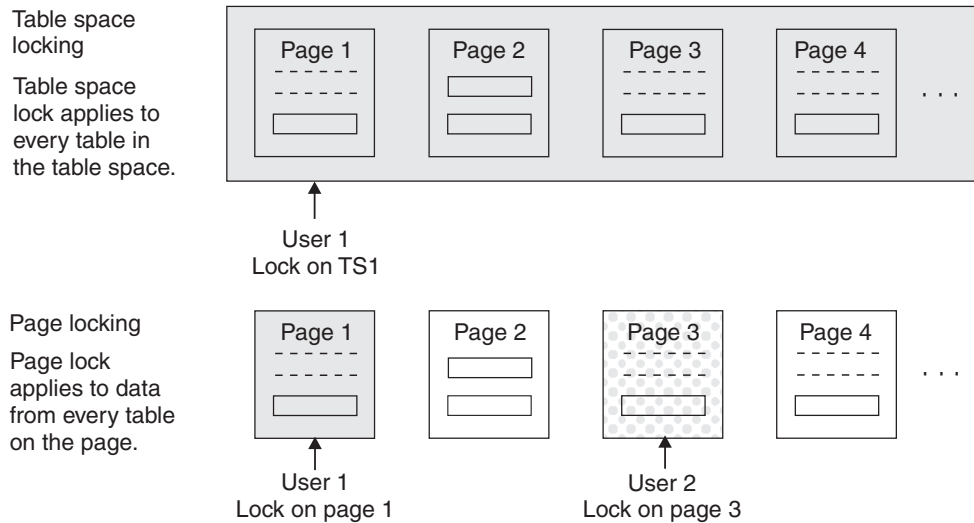
 DB2 no longer supports the creation of simple table spaces. However, an existing simple table space can contain more than one table. A lock on the table space locks all the data in every table. A single page of the table space can contain rows from every table. A lock on a page locks every row in the page, no matter what tables the data belongs to. Thus, a lock needed to access data from one table can make data from other tables temporarily unavailable. That effect can be partly undone by using row locks instead of page locks.

## Example: simple versus segmented table spaces

Suppose that tables T1 and T2 reside in table space TS1. In a simple table space, a single page can contain rows from both T1 and T2. If User 1 and User 2 acquire incompatible locks on different pages, such as exclusive locks for updating data, neither can access all the rows in T1 and T2 until one of the locks is released. (User 1 and User 2 can both hold a page lock on the same page when the mode of the locks are compatible, such as locks for reading data.)

As the figure also shows, in a segmented table space, a table lock applies only to segments assigned to a single table. Thus, User 1 can lock all pages assigned to the segments of T1 while User 2 locks all pages assigned to segments of T2. Similarly, User 1 can lock a page of T1 without locking any data in T2.

### Simple table space:



### Segmented table space:

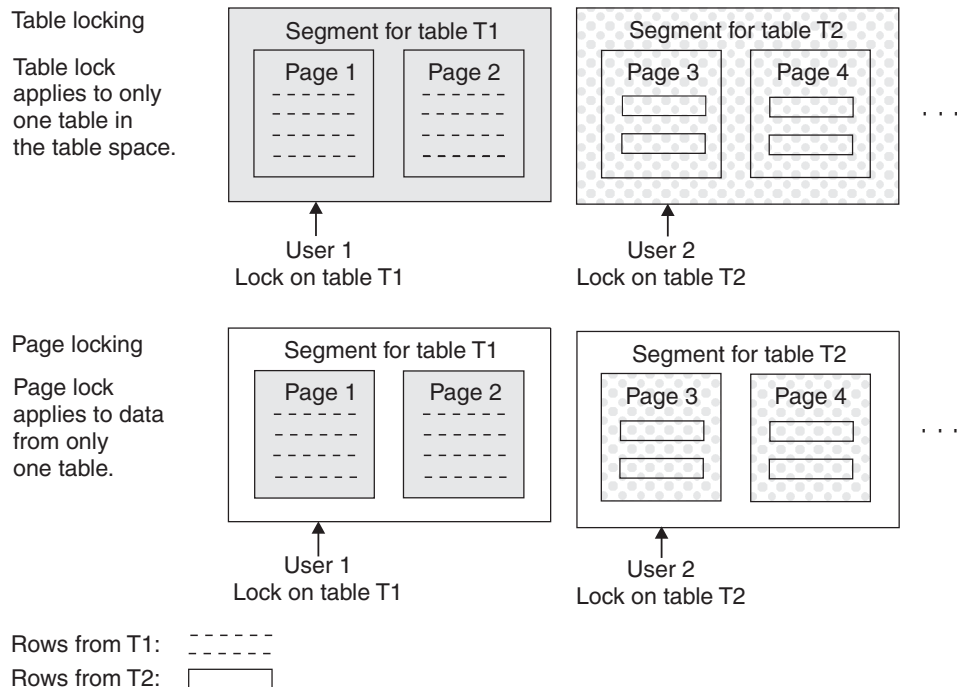


Figure 42. Page locking for simple and segmented table spaces

## LOB table space

In a LOB table space, pages are not locked. Because the concept of rows does not occur in a LOB table space, rows are not locked. Instead, LOBs are locked.

## XML table space

In an XML table space, XML locks are acquired in addition to locks on pages and rows.

## The duration of a lock

The *duration* of a lock is the length of time the lock is held. It varies according to when the lock is acquired and when it is released.

## Lock modes

The *mode* (sometimes *state*) of a lock tells what access to the locked object is permitted to the lock owner and to any concurrent processes.

PSPI

When a page or row is locked, the table, partition, or table space containing it is also locked. In that case, the table, partition, or table space lock has one of the *intent* modes: IS, IX, or SIX. The modes S, U, and X of table, partition, and table space locks are sometimes called *gross* modes. In the context of reading, SIX is a gross mode lock because you don't get page or row locks; in this sense, it is like an S lock.

**Example:** An SQL statement locates John Smith in a table of customer data and changes his address. The statement locks the entire table space in mode IX and the specific row that it changes in mode X.

PSPI

## Modes of page and row locks

Modes and their effects are listed in the order of increasing control over resources.

PSPI

Modes and their effects are listed in the order of increasing control over resources.

### S (SHARE)

The lock owner and any concurrent processes can read, but not change, the locked page or row. Concurrent processes can acquire S or U locks on the page or row or might read data without acquiring a page or row lock.

### U (UPDATE)

The lock owner can read, but not change, the locked page or row. Concurrent processes can acquire S locks or might read data without acquiring a page or row lock, but no concurrent process can acquire a U lock.

U locks reduce the chance of deadlocks when the lock owner is reading a page or row to determine whether to change it, because the owner can start with the U lock and then promote the lock to an X lock to change the page or row.

### X (EXCLUSIVE)

The lock owner can read or change the locked page or row. A concurrent process cannot acquire S, U, or X locks on the page or row; however, a concurrent process, such as those bound with the CURRENTDATA(NO) or ISO(UR) options or running with YES specified for the EVALUNC subsystem parameter, can read the data without acquiring a page or row lock.

PSPI

## Modes of table, partition, and table space locks

Modes and their effects are listed in the order of increasing control over resources.

PSPI

### IS (INTENT SHARE)

The lock owner can read data in the table, partition, or table space, but not change it. Concurrent processes can both read and change the data. The lock owner might acquire a page or row lock on any data it reads.

### IX (INTENT EXCLUSIVE)

The lock owner and concurrent processes can read and change data in the table, partition, or table space. The lock owner might acquire a page or row lock on any data it reads; it must acquire one on any data it changes.

### S (SHARE)

The lock owner and any concurrent processes can read, but not change, data in the table, partition, or table space. The lock owner does not need page or row locks on data it reads.

### U (UPDATE)


The lock owner can read, but not change, the locked data; however, the owner can promote the lock to an X lock and then can change the data. Processes concurrent with the U lock can acquire S locks and read the data, but no concurrent process can acquire a U lock. The lock owner does not need page or row locks.

U locks reduce the chance of deadlocks when the lock owner is reading data to determine whether to change it. U locks are acquired on a table space when the lock size is TABLESPACE and the statement is a SELECT with a FOR UPDATE clause. Similarly, U locks are acquired on a table when lock size is TABLE and the statement is a SELECT with a FOR UPDATE clause.

### SIX (SHARE with INTENT EXCLUSIVE)


The lock owner can read and change data in the table, partition, or table space. Concurrent processes can read data in the table, partition, or table space, but not change it. Only when the lock owner changes data does it acquire page or row locks.

### X (EXCLUSIVE)

The lock owner can read or change data in the table, partition, or table space. A concurrent process can access the data if the process runs with UR isolation or if data in a partitioned table space is running with CS isolation and CURRENTDATA((NO). The lock owner does not need page or row locks. 

## Compatibility of lock modes

DB2 uses lock mode to determine whether one lock is compatible with another.

 The major effect of the lock mode is to determine whether one lock is compatible with another.

Some lock modes do not shut out all other users. Assume that application process A holds a lock on a table space that process B also wants to access. DB2 requests, on behalf of B, a lock of some particular mode. If the mode of A's lock permits B's request, the two locks (or modes) are said to be *compatible*.

If the two locks are not compatible, B cannot proceed. It must wait until A releases its lock. (And, in fact, it must wait until all existing incompatible locks are released.)

## Compatible lock modes

Compatibility for page and row locks is easy to define. The table below shows whether page locks of any two modes, or row locks of any two modes, are compatible (Yes) or not (No). No question of compatibility of a page lock with a row lock can arise, because a table space cannot use both page and row locks.

Table 58. Compatibility of page lock and row lock modes

Lock Mode	S	U	X
S	Yes	Yes	No
U	Yes	No	No
X	No	No	No

Compatibility for table space locks is slightly more complex. The following table shows whether or not table space locks of any two modes are compatible.

Table 59. Compatibility of table and table space (or partition) lock modes

Lock Mode	IS	IX	S	U	SIX	X
IS	Yes	Yes	Yes	Yes	Yes	No
IX	Yes	Yes	No	No	No	No
S	Yes	No	Yes	Yes	No	No
U	Yes	No	Yes	No	No	No
SIX	Yes	No	No	No	No	No
X	No	No	No	No	No	No

PSPI

## The object of a lock

The *object* of a lock is the resource being locked.

PSPI

You might have to consider locks on any of the following objects:

### User data in target tables

A *target table* is a table that is accessed specifically in an SQL statement, and especially one that the statement updates, either by name or through a view. Locks on those tables are the most common concern, and the ones over which you have most control.

### User data in related tables


Operations subject to referential constraints can require locks on related tables. For example, if you delete from a parent table, DB2 might delete rows from the dependent table as well. In that case, DB2 locks data in the dependent table as well as in the parent table.

Similarly, operations on rows that contain LOB or XML values might require locks on the LOB or XML table space and possibly on LOB or XML values within that table space.

If your application uses triggers, any triggered SQL statements can cause additional locks to be acquired.


## DB2 internal objects

You might notice the following locks on internal objects:

- Portions of the **DB2 catalog**. For more information, see “Contention on the DB2 catalog.”
- The skeleton cursor table (SKCT) that represents an application plan.
- The skeleton package table (SKPT) that represents a package.
- The database descriptor (DBD) that represents a DB2 database. 


## Indexes and data-only locking

No index page locks are acquired during processing. Instead, DB2 uses a technique called *data-only locking* to serialize changes.

 Index page latches are acquired to serialize changes within a page and guarantee that the page is physically consistent. Acquiring page latches ensures that transactions accessing the same index page concurrently do not see the page in a partially changed state.

The underlying data page or row locks are acquired to serialize the reading and updating of index entries to ensure the data is logically consistent, meaning that the data is committed and not subject to rollback or abort. The data locks can be held for a long duration such as until commit. However, the page latches are only held for a short duration while the transaction is accessing the page. Because the index pages are not locked, hot spot insert scenarios (which involve several transactions trying to insert different entries into the same index page at the same time) do not cause contention problems in the index.


A query that uses index-only access might lock the data page or row, and that lock can contend with other processes that lock the data. However, using lock avoidance techniques can reduce the contention.

To provide better concurrency, when DB2 searches using XML values (the first key values) in the XML index key entries, it does not acquire the index page latch and does not lock either the base table data pages or rows, or the XML table space. When the matched-value index key entries are found, the corresponding DOCID values (the second key value) are retrieved. The retrieved DOCID values are used to retrieve the base table RIDs using the DOCID index. At this time, the regular data-only locking technique is applied on the DOCID index page and base table data page (or row). 

## Contention on the DB2 catalog

SQL data definition statements, GRANT statements, and REVOKE statements require locks on the DB2 catalog. If different application processes are issuing these types of statements, catalog contention can occur.

## Contention within the SYSDBASE table space

 SQL statements that update the catalog table space SYSDBASE contend with each other when those statements are on the same table space. Those statements are:

- CREATE TABLESPACE, TABLE, and INDEX
- ALTER TABLESPACE, TABLE, INDEX
- DROP TABLESPACE, TABLE, and INDEX


- CREATE VIEW, SYNONYM, and ALIAS
- DROP VIEW and SYNONYM, and ALIAS
- COMMENT ON and LABEL ON
- GRANT and REVOKE of table privileges
- RENAME TABLE
- RENAME INDEX
- ALTER VIEW

#### **Recommendations:**

- Reduce the concurrent use of statements that update SYSDBASE for the same table space.
- When you alter a table or table space, quiesce other work on that object.


#### **Contention independent of databases**


The following limitations on concurrency are independent of the referenced database:

- CREATE and DROP statements for a table space or index that uses a storage group contend significantly with other such statements.
- CREATE, ALTER, and DROP DATABASE, and GRANT and REVOKE database privileges all contend with each other and with any other function that requires a database privilege.
- CREATE, ALTER, and DROP STOGROUP contend with any SQL statements that refer to a storage group and with extensions to table spaces and indexes that use a storage group.
- GRANT and REVOKE for plan, package, system, or use privileges contend with other GRANT and REVOKE statements for the same type of privilege and with data definition statements that require the same type of privilege. 

#### **Contention independent of databases:**

Some processes result in contention that is independent of the database where the resources reside.

 The following limitations on concurrency are independent of the referenced database:

- CREATE and DROP statements for a table space or index that uses a storage group contend significantly with other such statements.
- CREATE, ALTER, and DROP DATABASE, and GRANT and REVOKE database privileges all contend with each other and with any other function that requires a database privilege.
- CREATE, ALTER, and DROP STOGROUP contend with any SQL statements that refer to a storage group and with extensions to table spaces and indexes that use a storage group.
- GRANT and REVOKE for plan, package, system, or use privileges contend with other GRANT and REVOKE statements for the same type of privilege and with data definition statements that require the same type of privilege. 

#### **Locks on the skeleton tables (SKCT and SKPT):**

The skeleton table of a plan (SKCT) or package (SKPT) is locked while the plan or package is running.

**PSPI** The following operations require incompatible locks on the SKCT or SKPT, whichever is applicable, and cannot run concurrently:

- Binding, rebinding, or freeing the plan or package
- Dropping a resource or revoking a privilege that the plan or package depends on
- In some cases, altering a resource that the plan or package depends on

**PSPI**

### Locks on the database descriptors (DBDs):

Whether a process locks a target DBD depends largely on whether the DBD is already in the EDM DBD cache.

**PSPI**

#### If the DBD is not in the EDM DBD cache

Most processes acquire locks on the database descriptor table space (DBD01). That has the effect of locking the DBD and can cause conflict with other processes.

#### If the DBD is in the EDM DBD cache

the lock on the DBD depends on the type of process, as shown in the following table:

Table 60. Contention for locks on a DBD in the EDM DBD cache

Process Type	Process	Lock acquired	Conflicts with process type
1	Static DML statements (SELECT, DELETE, INSERT, UPDATE) <sup>1</sup>	None	None
2	Dynamic DML statements <sup>2</sup>	S	3
3	Data definition statements (ALTER, CREATE, DROP)	X	2,3,4
4	Utilities	S	3

#### Notes:


1. Static DML statements can conflict with other processes because of locks on data.
2. If caching of dynamic SQL is turned on, no lock is taken on the DBD when a statement is prepared for insertion in the cache or for a statement in the cache.

**PSPI**


## How DB2 chooses lock types

Different types of SQL data manipulation statements acquire locks on target tables.

**PSPI**

The lock acquired because of an SQL statement is not always a constant throughout the time of execution. In certain situations, DB2 can change acquired locks during execution. Many other processes and operations acquire locks. 

## Locks acquired for SQL statements

 The following tables show the locks that certain SQL processes acquire and the modes of those locks. Whether locks are acquired at all and the mode of those locks depend on the following factors:

- The type of processing being performed
- The value of LOCKSIZE for the target table
- The isolation level of the plan, package, or statement
- The method of access to data
- Whether the application uses the SKIP LOCKED DATA option

### Example SQL statement

The following SQL statement and sample steps provide a way to understand the following tables.

```
EXEC SQL DELETE FROM DSN8910.EMP WHERE CURRENT OF C1;
```

Use the following sample steps to understand the table:

1. Find the portion of the table that describes DELETE operations using a cursor.
2. Find the row for the appropriate values of LOCKSIZE and ISOLATION. Table space DSN8910 is defined with LOCKSIZE ANY. The default value of ISOLATION is CS with CURRENTDATA (NO) by default.
3. Find the sub-row for the expected access method. The operation probably uses the index on employee number. Because the operation deletes a row, it must update the index. Hence, you can read the locks acquired in the sub-row for "Index, updated":
  - An IX lock on the table space
  - An IX lock on the table (but see the step that follows)
  - An X lock on the page containing the row that is deleted
4. Check the notes to the entries you use, at the end of the table. For this sample operation, see:
  - Note 2, on the column heading for "Table". If the table is not segmented, or if the table is segmented and partitioned, no separate lock is taken on the table.
  - Note 3, on the column heading for "Data Page or Row". Because LOCKSIZE for the table space is ANY, DB2 can choose whether to use page locks, table locks, or table space locks. Typically it chooses page locks.

### SELECT with read-only or ambiguous cursor, or with no cursor

The following table shows locks that are acquired during the processing of SELECT with read-only or ambiguous cursor, or with no cursor SQL statements. UR isolation is allowed and requires none of these locks.

Table 61. Locks acquired for SQL statements *SELECT* with read-only or ambiguous cursor

LOCKSIZE	ISOLATION	Access method <sup>1</sup>	Lock mode		
			Table space <sup>9</sup>	Table <sup>2</sup>	Data page or row <sup>3</sup>
TABLESPACE	CS RS RR	Any	S	n/a	n/a
TABLE <sup>2</sup>	CS RS RR	Any	IS	S	n/a
PAGE, ROW, or ANY	CS	Index, any use	IS <sup>4, 10</sup>	IS <sup>4</sup>	S <sup>5</sup>
		Table space scan	IS <sup>4, 11</sup>	IS <sup>4</sup>	S <sup>5</sup>
PAGE, ROW, or ANY	RS	Index, any use	IS <sup>4, 10</sup>	IS <sup>4</sup>	S <sup>5</sup> , U <sup>11</sup> , or X <sup>11</sup>
		Table space scan	IS <sup>4, 10</sup>	IS <sup>4</sup>	S <sup>5</sup> , U <sup>11</sup> , or X <sup>11</sup>
PAGE, ROW, or ANY	RR	Index/data probe	IS <sup>4</sup>	IS <sup>4</sup>	S <sup>5</sup> , U <sup>11</sup> , or X <sup>11</sup>
		Index scan <sup>6</sup>	IS <sup>4</sup> or S	S, IS <sup>4</sup> , or n/a	S <sup>5</sup> , U <sup>11</sup> , X <sup>11</sup> , or n/a
		Table space scan <sup>6</sup>	IS <sup>2</sup> or S	S or n/a	n/a

### INSERT, VALUES(...), or INSERT fullselect<sup>7</sup>

The following table shows locks that are acquired during the processing of INSERT, VALUES(...), or INSERT fullselect SQL statements.

Table 62. Locks acquired for SQL statements *INSERT ... VALUES(...)* or *INSERT ... fullselect*

LOCKSIZE	ISOLATION	Access method <sup>1</sup>	Lock mode		
			Table space <sup>9</sup>	Table <sup>2</sup>	Data page or row <sup>3</sup>
TABLESPACE	CS RS RR	Any	X	n/a	n/a
TABLE <sup>2</sup>	CS RS RR	Any	IX	X	n/a
PAGE, ROW, or ANY	CS RS RR	Any	IX	IX	X

### UPDATE or DELETE without cursor

The following table shows locks that are acquired during the processing of UPDATE or DELETE without cursor SQL statements. Data page and row locks apply only to selected data.

Table 63. Locks acquired for SQL statements *UPDATE*, or *DELETE* without cursor

LOCKSIZE	ISOLATION	Access method <sup>1</sup>	Lock mode		
			Table space <sup>9</sup>	Table <sup>2</sup>	Data page or row <sup>3</sup>
TABLESPACE	CS RS RR	Any	X	n/a	n/a
TABLE <sup>2</sup>	CS RS RR	Any	IX	X	n/a
PAGE, ROW, or ANY	CS	Index selection	IX	IX	<ul style="list-style-type: none"> <li>• For delete: X</li> <li>• For update: U→X</li> </ul>
		Index/data selection	IX	IX	U→X
		Table space scan	IX	IX	U-->→X

Table 63. Locks acquired for SQL statements UPDATE, or DELETE without cursor (continued)

LOCKSIZE	ISOLATION	Access method <sup>1</sup>	Lock mode		
			Table space <sup>9</sup>	Table <sup>2</sup>	Data page or row <sup>3</sup>
PAGE, ROW, or ANY	RS	Index selection	IX	IX	<ul style="list-style-type: none"> <li>• For update: S or U<sup>8</sup> → X</li> <li>• For delete: [S → X] or X</li> </ul>
		Index/data selection	IX	IX	S or U <sup>8</sup> → X
		Table space scan	IX	IX	S or U <sup>8</sup> → X
PAGE, ROW, or ANY	RR	Index selection	IX	IX	<ul style="list-style-type: none"> <li>• For update: [S or U<sup>8</sup> → X] or X</li> <li>• For delete: [S → X] or X</li> </ul>
		Index/data selection	IX	IX	S or U <sup>8</sup> → X
		Table space scan	IX <sup>2</sup> or X	X or n/a	n/a

## SELECT with FOR UPDATE OF

The following table shows locks that are acquired during the processing of SELECT with FOR UPDATE OF SQL statements. Data page and row locks apply only to selected data.

Table 64. Locks acquired for SQL statements SELECT with FOR UPDATE OF

LOCKSIZE	ISOLATION	Access method <sup>1</sup>	Lock mode		
			Table space <sup>9</sup>	Table <sup>2</sup>	Data page or row <sup>3</sup>
TABLESPACE	CS RS RR	Any	U	n/a	n/a
TABLE <sup>2</sup>	CS RS RR	Any	IS or IX	U	n/a
PAGE, ROW, or ANY	CS	Index, any use	IX	IX	U
		Table space scan	IX	IX	U
PAGE, ROW, or ANY	RS	Index, any use	IX	IX	S, U, or X <sup>8</sup>
		Table space scan	IX	IX	S, U, or X <sup>8</sup>
PAGE, ROW, or ANY	RR	Index/data probe	IX	IX	S, U, or X <sup>8</sup>
		Index scan <sup>6</sup>	IX or X	X, IX, or n/a	S, U, X <sup>8</sup> , or n/a
		Table space scan <sup>6</sup>	IX <sup>2</sup> or X	X or n/a	S, U, X <sup>8</sup> , or n/a

## UPDATE or DELETE with cursor

The following table shows locks that are acquired during the processing of xxx SQL statements.

Table 65. Locks acquired for SQL statements UPDATE or DELETE with cursor

LOCKSIZE	ISOLATION	Access method <sup>1</sup>	Lock mode		
			Table space <sup>9</sup>	Table <sup>2</sup>	Data page or row <sup>3</sup>
TABLESPACE	Any	Any	X	n/a	n/a
TABLE <sup>2</sup>	Any	Any	IX	X	n/a

Table 65. Locks acquired for SQL statements UPDATE or DELETE with cursor (continued)

LOCKSIZE	ISOLATION	Access method <sup>1</sup>	Lock mode		
			Table space <sup>9</sup>	Table <sup>2</sup>	Data page or row <sup>3</sup>
PAGE, ROW, or ANY	CS, RS, or RR	Index, updated	IX	IX	X
		Index not updated	IX	IX	X

## Mass delete or TRUNCATE

Lock modes for TRUNCATE depend solely on the type of tables space regardless of LOCKSIZE or isolation level:

### Simple table space

Locks the table space with an X lock

### Segmented table space (not partitioned)

Locks the table with an X lock and lock the table space with an IX lock

### Partitioned table space (including segmented)

Locks each partition with an X lock

## Notes for this topic

1. All access methods are either scan-based or probe-based. Scan-based means the index or table space is scanned for successive entries or rows. Probe-based means the index is searched for an entry as opposed to a range of entries, which a scan does. ROWIDs provide data probes to look for a single data row directly. The type of lock used depends on the backup access method. Access methods might be index-only, data-only, or index-to-data.

### Index-only

The index alone identifies qualifying rows and the return data.

### Data-only:

The data alone identifies qualifying rows and the return data, such as a table space scan or the use of ROWID for a probe.

### Index-to-data

The index is used or the index plus data are used to evaluate the predicate:


#### Index selection

The index is used to evaluate predicate and data is used to return values.

#### Index/data selection

The index and data are used to evaluate predicate and data is used to return values.


2. Used only for segmented table spaces that are not partitioned.
3. These locks are taken on pages if LOCKSIZE is PAGE or on rows if LOCKSIZE is ROW. When the maximum number of locks per table space (LOCKMAX) is reached, locks escalate to a table lock for tables in a segmented table space without partitions, or to a table space lock for tables in a non-segmented table space. Using LOCKMAX 0 in CREATE or ALTER TABLESPACE disables lock escalation.
4. If the table or table space is started for read-only access, DB2 attempts to acquire an S lock. If an incompatible lock already exists, DB2 acquires the IS lock.

5. SELECT statements that do not use a cursor, or that use read-only or ambiguous cursors and are bound with CURRENTDATA(NO), might not require any lock if DB2 can determine that the data to be read is committed. This is known as *lock avoidance*. If your application can tolerate incomplete or inconsistent results, you can also specify the SKIP LOCKED DATA option in your query to avoid lock wait times.
6. Even if LOCKMAX is 0, the bind process can promote the lock size to TABLE or TABLESPACE. If that occurs, SQLCODE +806 is issued.
7. The locks listed are acquired on the object into which the insert is made. A subselect acquires additional locks on the objects it reads, as if for SELECT with read-only cursor or ambiguous cursor, or with no cursor.
8. An installation option determines whether the lock is S, U, or X. . If you use the WITH clause to specify the isolation as RR or RS, you can use the USE AND KEEP UPDATE LOCKS option to obtain and hold a U lock instead of an S lock, or you can use the USE AND KEEP EXCLUSIVE LOCKS option to obtain and hold an X lock instead of an S lock.
9. Includes partition locks, and does not include LOB table space locks.
10. If the table space is partitioned, locks can be avoided on the partitions.
11. If you use the WITH clause to specify the isolation as RR or RS, you can use the USE AND KEEP UPDATE LOCKS option to obtain and hold a U lock instead of an S lock, or you can use the USE AND KEEP EXCLUSIVE LOCKS option to obtain and hold an X lock instead of an S lock. 


## Lock promotion

*Lock promotion* is the action of exchanging one lock on a resource for a more restrictive lock on the same resource, held by the same application process.

### Example


 An application reads data, which requires an IS lock on a table space. Based on further calculation, the application updates the same data, which requires an IX lock on the table space. The application is said to *promote* the table space lock from mode IS to mode IX.

### Effects

When promoting the lock, DB2 first waits until any incompatible locks held by other processes are released. When locks are promoted, they are promoted in the direction of increasing control over resources: from IS to IX, S, or X; from IX to SIX or X; from S to X; from U to X; and from SIX to X. 

## Lock escalation

*Lock escalation* is the act of releasing a large number of page, row, LOB, or XML locks, held by an application process on a single table or table space, to acquire a table or table space lock, or a set of partition locks, of mode S or X instead.

 When locks escalation occurs, DB2 issues message DSNI031I, which identifies the table space for which lock escalation occurred, and some information to help you identify what plan or package was running when the escalation occurred.

Lock counts are always kept on a table or table space level. For an application process that is accessing LOBs or XML, the LOB or XML lock count on the LOB or

XML table space is maintained separately from the base table space, and lock escalation occurs separately from the base table space.

When escalation occurs for a partitioned table space, only partitions that are currently locked are escalated. Unlocked partitions remain unlocked. After lock escalation occurs, any unlocked partitions that are subsequently accessed are locked with a gross lock.

For an application process that is using Sysplex query parallelism, the lock count is maintained on a member basis, not globally across the group for the process. Thus, escalation on a table space or table by one member does not cause escalation on other members.

### **Example lock escalation**

Assume that a segmented table space without partitions is defined with LOCKSIZE ANY and LOCKMAX 2000. DB2 can use page locks for a process that accesses a table in the table space and can escalate those locks. If the process attempts to lock more than 2000 pages in the table at one time, DB2 promotes its intent lock on the table to mode S or X and then releases its page locks.

If the process is using Sysplex query parallelism and a table space that it accesses has a LOCKMAX value of 2000, lock escalation occurs for a member only if more than 2000 locks are acquired for that member.

### **When lock escalation occurs**

Lock escalation balances concurrency with performance by using page or row locks while a process accesses relatively few pages or rows, and then changing to table space, table, or partition locks when the process accesses many. When it occurs, lock escalation varies by table space, depending on the values of LOCKSIZE and LOCKMAX, as described in Lock escalation is suspended during the execution of SQL statements for ALTER, CREATE, DROP, GRANT, and REVOKE.

### **Recommendations**

The DB2 statistics and performance traces can tell you how often lock escalation has occurred and whether it has caused timeouts or deadlocks. As a rough estimate, if one quarter of your lock escalations cause timeouts or deadlocks, then escalation is not effective for you. You might alter the table to increase LOCKMAX and thus decrease the number of escalations.

Alternatively, if lock escalation is a problem, use LOCKMAX 0 to disable lock escalation.

### **Example**

Assume that a table space is used by transactions that require high concurrency and that a batch job updates almost every page in the table space. For high concurrency, you should probably create the table space with LOCKSIZE PAGE and make the batch job commit every few seconds.

### **LOCKSIZE ANY**

LOCKSIZE ANY is a possible choice, if you take other steps to avoid lock escalation. If you use LOCKSIZE ANY, specify a LOCKMAX value large enough so

that locks held by transactions are not normally escalated. Also, LOCKS PER USER must be large enough so that transactions do not reach that limit.

If the batch job is:

#### Concurrent with transactions

It must use page or row locks and commit frequently: for example, every 100 updates. Review LOCKS PER USER to avoid exceeding the limit. The page or row locking uses significant processing time. Binding with ISOLATION(CS) might discourage lock escalation to an X table space lock for those applications that read a lot and update occasionally. However, this might not prevent lock escalation for those applications that are update intensive.

#### Non-concurrent with transactions

It need not use page or row locks. The application could explicitly lock the table in exclusive mode. 

### Modes of transaction locks for various processes

DB2 uses different lock modes for different types of processes.



 The rows in the following table show a sample of several types of DB2 processes. The columns show the most restrictive mode of locks used for different objects and the possible conflicts between application processes.

Table 66. Modes of DB2 transaction locks

Process	Catalog table spaces	Skeleton tables (SKCT and SKPT)	Database descriptor (DBD) (1)	Target table space (2)
Transaction with static SQL	IS (3)	S	n/a (4)	Any (5)
Query with dynamic SQL	IS (6)	S	S	Any (5)
BIND process	IX	X	S	n/a
SQL CREATE TABLE statement	IX	n/a	X	n/a
SQL ALTER TABLE statement	IX	X (7)	X	n/a
SQL ALTER TABLESPACE statement	IX	X (9)	X	n/a
SQL DROP TABLESPACE statement	IX	X (8)	X	n/a
SQL GRANT statement	IX	n/a	n/a	n/a
SQL REVOKE statement	IX	X (8)	n/a	n/a

#### Notes:

1. In a lock trace, these locks usually appear as locks on the DBD.
2. The target table space is one of the following table spaces:
  - Accessed and locked by an application process
  - Processed by a utility
  - Designated in the data definition statement
3. The lock is held briefly to check EXECUTE authority.
4. If the required DBD is not already in the EDM DBD cache, locks are acquired on table space DBD01, which effectively locks the DBD.
5. Except while checking EXECUTE authority, IS locks on catalog tables are held until a commit point.

6. The plan or package using the SKCT or SKPT is marked invalid if a referential constraint (such as a new primary key or foreign key) is added or changed, or the AUDIT attribute is added or changed for a table.
7. The plan or package using the SKCT or SKPT is marked invalid as a result of this operation.
8. These locks are not held when ALTER TABLESPACE is changing the following options: PRIQTY, SECQTY, PCTFREE, FREEPAGE, CLOSE, and ERASE. 

#### **Related reference**

“Locks acquired for SQL statements” on page 340


---

## **Options for tuning locks**

The following options affect how DB2 uses transaction locks.

### **IRLM startup procedure options**

You can control how DB2 uses locks by specifying certain options when you start the internal resource lock manager (IRLM).

 When you issue the z/OS START irlmproc command, the values of the options are passed to the startup procedure for the DB2 IRLM. (If an option is not explicitly specified on the command, the value of its corresponding installation parameter is used.)

The options that are relevant to DB2 locking are:

#### **SCOPE**

Whether IRLM is used for data sharing (GLOBAL) or not (LOCAL). Use LOCAL unless you are using data sharing. If you use data sharing, specify GLOBAL.

#### **DEADLOK**

The two values of this option specify:


1. The number of seconds between two successive scans for a local deadlock
2. The number of local scans that occur before a scan for global deadlock starts

**PC** Ignored by IRLM. However, PC is positional and must be maintained in the IRLM for compatibility.

#### **MAXCSA**

Ignored by IRLM. However, MAXCSA is positional and must be maintained in the IRLM for compatibility.

The maximum amount of storage available for IRLM locks is limited to 90% of the total space given to the IRLM private address space during the startup procedure. The other 10% is reserved for IRLM system services, z/OS system services, and “must complete” processes to prevent the IRLM address space from abending, which would bring down your DB2 system. When the storage limit is reached, lock requests are rejected with an out-of-storage reason code.


You can use the `F irlmproc,STATUS,STOR` command to monitor the amount of storage that is available for locks and the `MODIFY irlmproc,SET` command to dynamically change the maximum amount of IRLM private storage to use for locks. 

## Estimating the storage needed for locks

DB2 locks require storage space.

 That value is calculated when DB2 is installed.

To estimate the storage that is required for DB2 locks:

1. For a conservative starting figure, assume that DB2 locks require:
  - 540 bytes of storage for each lock.
  - All concurrent threads hold the maximum number of row or page locks (LOCKS PER USER on installation panel DSNTIPJ). The number of table and table space locks is negligible.
  - The maximum number of concurrent threads are active.
2. For a more accurate estimate, use the following calculation:  $\text{Storage} = 540 \times (\text{LOCKS PER USER}) \times (\text{MAX USERS})$  

## Setting installation options for wait times

These options determine how long it takes DB2 to identify that a process must be timed out or is deadlocked. They affect locking in your entire DB2 subsystem.

### Specifying the interval for detecting deadlocks


DB2 scans for deadlocked processes at regular intervals. The DEADLOCK TIME field on installation panel DSNTIPJ sets the length of the interval, in seconds.



The default value is 5 seconds.


Deadlock detection can cause latch suspensions. To determine the best value for deadlock detection

Consider your workload:

- For systems in which deadlocking is not a problem, have deadlock detection run less frequently for the best performance and concurrency (but do not choose a value greater than 5 seconds).
- If your system is prone to deadlocks, you want those detected as quickly as possible. In that case, choose 1. 

### Specifying the amount of inactive time before a timeout

The RESOURCE TIMEOUT field on installation panel DSNTIPI specifies a minimum number of seconds before a timeout can occur.

 A small value can cause a large number of timeouts. With a larger value, suspended processes more often resume normally, but they remain inactive for longer periods. The default value is 60 seconds.

Consider how long your system can afford to wait for a suspended process.

- If you can allow a suspended process to remain inactive for 60 seconds, use the defaults for both RESOURCE TIMEOUT and DEADLOCK TIME.
- If you specify a different a different inactive period, consider how DB2 calculates the wait time for timeouts. **GUPI**

## How DB2 calculates the wait time for timeouts

**PSPI** In a scanning schedule to determine whether a process waiting for a transaction lock has timed out, DB2 uses the following two factors:

- A timeout period
- An operation multiplier

### The timeout period

DB2 calculates a *timeout period* from the value of the RESOURCE TIMEOUT and DEADLOCK TIME options.

For example, assume that the value of the DEADLOCK TIME option is 5 and the value of the RESOURCE TIMEOUT option is 18. You can use the following calculations to see how DB2 calculates a *timeout period*.

1. Divide RESOURCE TIMEOUT by DEADLOCK TIME ( $18/5 = 3.6$ ). IRLM limits the result of this division to 255.
2. Round the result to the next largest integer (Round up 3.6 to 4).
3. Multiply the DEADLOCK TIME by that integer ( $4 * 5 = 20$ ).

The result, the timeout period (20 seconds), is always at least as large as the value of RESOURCE TIMEOUT (18 seconds), except when the RESOURCE TIMEOUT divided by DEADLOCK TIME exceeds 255.

### The timeout multiplier

Requests from different types of processes wait for different multiples of the timeout period according to the *timeout multiplier*. In a data sharing environment, you can add another multiplier to those processes to wait for retained locks.

In some cases, you can modify the multiplier value. The following table indicates the multiplier value by type of process, and whether you can change it.

Table 67. Timeout multiplier by type

Type	Multiplier <sup>1</sup>	Modifiable?
IMS MPP, IMS Fast Path Message Processing, CICS, DB2 QMF, CAF, TSO batch and online, RRSAP, global transactions	1	No
IMS BMPs	4	Yes
IMS DL/I batch	6	Yes
IMS Fast Path Non-message processing	6	No
BIND subcommand processing	3	No
STOP DATABASE command processing	10	No
Utilities	6	Yes
Retained locks for all types	0	Yes

Table 67. Timeout multiplier by type (continued)

Type	Multiplier <sup>1</sup>	Modifiable?
<b>Note:</b>		
1. If the transaction occurs on a table space that is not logged, the timeout multiplier is either three or the current timeout multiplier for the thread, whichever is greater.		

## Changing the multiplier for IMS BMP and DL/I batch

You can modify the multipliers for IMS BMP and DL/I batch by modifying the following subsystem parameters on installation panel DSNTIPI:

### IMS BMP TIMEOUT

The timeout multiplier for IMS BMP connections. A value from 1 to 254 is acceptable. The default is 4.

### DL/I BATCH TIMEOUT

The timeout multiplier for IMS DL/I batch connections. A value from 1 to 254 is acceptable. The default is 6.

## Additional multiplier for retained lock

For data sharing, you can specify an additional timeout multiplier to be applied to the connection's normal timeout multiplier. This multiplier is used when the connection is waiting for a retained lock, which is a lock held by a failed member of a data sharing group. A zero means don't wait for retained locks.

## The scanning schedule

The following figure illustrates the following example of scanning to detect a timeout:

- DEADLOCK TIME has the default value of 5 seconds.
- RESOURCE TIMEOUT was chosen to be 18 seconds. Therefore, the timeout period is 20 seconds.
- A bind operation starts 4 seconds before the next scan. The operation multiplier for a bind operation is 3.

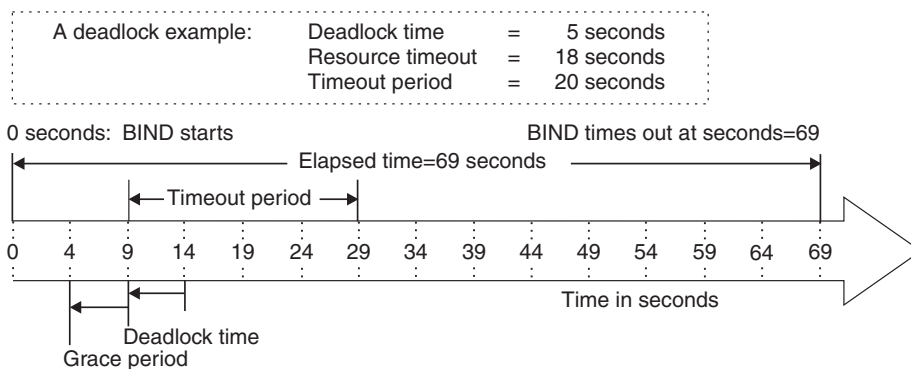



Figure 43. An example of scanning for timeout

The scans proceed through the following steps:

1. A scan starts 4 seconds after the bind operation requests a lock. As determined by the DEADLOCK TIME, scans occur every 5 seconds. The first scan in the example detects that the operation is inactive.
2. IRLM allows at least one full interval of DEADLOCK TIME as a “grace period” for an inactive process. After that, its lock request is judged to be waiting. At 9 seconds, the second scan detects that the bind operation is waiting.
3. The bind operation continues to wait for a multiple of the timeout period. In the example, the multiplier is 3 and the timeout period is 20 seconds. The bind operation continues to wait for 60 seconds longer.
4. The scan that starts 69 seconds after the bind operation detects that the process has timed out.


Consequently, an operation can remain inactive for longer than the value of RESOURCE TIMEOUT.

If you are in a data sharing environment, the deadlock and timeout detection process is longer than that for non-data-sharing systems.

You should carefully consider the length of inaction time when choosing your own values of DEADLOCK TIME and RESOURCE TIMEOUT. 

### Specifying how long an idle thread can use resources


The IDLE THREAD TIMEOUT field on installation panel DSNTIPR specifies a period for which an active distributed thread can use resources without doing any processing.

 After that period, a regular scan (at 3-minute intervals) detects that the thread has been idle for the specified period, and DB2 cancels the thread.

The cancellation applies only to active threads. If your installation permits distributed threads to be inactive and hold no resources, those threads are allowed to remain idle indefinitely.

The default value is 0. That value disables the scan to time out idle threads. The threads can then remain idle indefinitely.

If you have experienced distributed users leaving an application idle while it holds locks:

Specify an appropriate value other than 0 for this period. Because the scan occurs only at 3-minute intervals, your idle threads generally remain idle for somewhat longer than the value you specify. 

### Specifying how long utilities wait for resources

The UTILITY TIMEOUT field on installation panel DSNTIPI specifies an operation multiplier for utilities waiting for a drain lock, for a transaction lock, or for claims to be released.




The default value is 6.

**Recommendation:** With the default value, a utility generally waits longer for a resource than does an SQL application. To specify a different inactive period:

Consider how DB2 times out a process that is waiting for a drain. 

## Calculating wait times for drains

You can calculate how long DB2 waits for drains.

 A process that requests a drain might wait for two events:

### Acquiring the drain lock.

If another user holds the needed drain lock in an incompatible lock mode, then the drainer waits.

### Releasing all claims on the object.

Even after the drain lock is acquired, the drainer waits until all claims are released before beginning to process.

If the process drains more than one claim class, it must wait for those events to occur for each claim class that it drains.

To calculate the maximum amount of wait time:

1. Add the wait time for a drain lock and the wait time for claim release. Both wait times are based on the timeout period that is calculated by DB2. For the REORG, REBUILD, REBUILD INDEX, CHECK DATA or CHECK LOB utilities, with the SHRLEVEL CHANGE options you can use utility parameters to specify the wait time for a drain lock and to indicate if additional attempts should be made to acquire the drain lock..

#### Drainer:

Each wait time is:

**Utility** (timeout period) × (value of UTILITY TIMEOUT)

**Other process**  
timeout period

2. Add the wait time for claim release.
3. Multiply the result by the number of claim classes drained.

**Maximum wait time:** Because the maximum wait time for a drain lock is the same as the maximum wait time for releasing claims, you can calculate the total maximum wait time as follows:

#### For utilities

$$2 \times (\text{timeout period}) \times (\text{UTILITY TIMEOUT}) \times (\text{number of claim classes})$$

#### For other processes

$$2 \times (\text{timeout period}) \times (\text{operation multiplier}) \times (\text{number of claim classes})$$

For example, suppose that LOAD must drain 3 claim classes, that the timeout period is 20 seconds, and that the value of UTILITY TIMEOUT is 6. Use the following calculation to determine how long the LOAD might utility be suspended before being timed out:

$$\text{Maximum wait time} = 2 \times 20 \times 6 \times 3 = 720 \text{ seconds}$$

**Wait times less than maximum:** The maximum drain wait time is the longest possible time a drainer can wait for a drain, not the length of time it always waits.

For example, The following table lists the steps LOAD takes to drain the table space and the maximum amount of wait time for each step. A timeout can occur at any step. At step 1, the utility can wait 120 seconds for the repeatable read drain lock. If that lock is not available by then, the utility times out after 120 seconds. It does not wait 720 seconds.

*Table 68. Maximum drain wait times: LOAD utility*

Step	Maximum Wait Time (seconds)
1. Get repeatable read drain lock	120
2. Wait for all RR claims to be released	120
3. Get cursor stability read drain lock	120
4. Wait for all CS claims to be released	120
5. Get write drain lock	120
6. Wait for all write claims to be released	120
<b>Total</b>	<b>720</b>



## Bind options for locks

These options determine when an application process acquires and releases its locks and to what extent it isolates its actions from possible effects of other processes acting concurrently.

### Choosing ACQUIRE and RELEASE options

The ACQUIRE and RELEASE options of bind determine when DB2 locks an object (table, partition, or table space) your application uses and when it releases the lock. (The ACQUIRE and RELEASE options do not affect page, row, LOB, or XML locks.)

The options apply to static SQL statements, which are bound before your program executes. If your program executes dynamic SQL statements, the objects they lock are locked when first accessed and released at the next commit point though some locks acquired for dynamic SQL might be held past commit points.

The ACQUIRE and RELEASE options are:

#### ACQUIRE(ALLOCATE)

Acquires the lock when the object is allocated. This option is not allowed for BIND or REBIND PACKAGE.

#### ACQUIRE(USE)

Acquires the lock when the object is first accessed.

#### RELEASE(DEALLOCATE)

Releases the lock when the object is deallocated (the application ends). The value has no effect on dynamic SQL statements, which always use RELEASE(COMMIT), unless you are using dynamic statement caching. For information about the RELEASE option with dynamic statement caching. The value also has no effect on packages that are executed on a DB2 server through a DRDA connection with the client system.

## RELEASE(COMMIT)

Releases the lock at the next commit point, unless cursors. If the application accesses the object again, it must acquire the lock again.

The default options for ACQUIRE and RELEASE depend on the type of bind option as shown in the following table.

*Table 69. Default ACQUIRE and RELEASE values for different bind options*

Operation	Default values
BIND PLAN	ACQUIRE(USE) and RELEASE(COMMIT).
BIND PACKAGE	No option exists for ACQUIRE; ACQUIRE(USE) is always used. At the local server the default for RELEASE is the value used by the plan that includes the package in its package list. At a remote server the default is COMMIT.
REBIND PLAN or PACKAGE	The existing values for the plan or package that is being rebound.

**Partition locks:** Partition locks follow the same rules as table space locks, and all partitions are held for the same duration. Thus, if one package is using RELEASE(COMMIT) and another is using RELEASE(DEALLOCATE), all partitions use RELEASE(DEALLOCATE).

**Dynamic statement caching:** Generally, the RELEASE option has no effect on dynamic SQL statements with one exception. When you use the bind options RELEASE(DEALLOCATE) and KEEP DYNAMIC(YES), and your subsystem is installed with YES for field CACHE DYNAMIC SQL on installation panel DSNTIP4, DB2 retains prepared SELECT, INSERT, UPDATE, and DELETE statements in memory past commit points. For this reason, DB2 can honor the RELEASE(DEALLOCATE) option for these dynamic statements. The locks are held until deallocation, or until the commit after the prepared statement is freed from memory, in the following situations:

- The application issues a PREPARE statement with the same statement identifier.
- The statement is removed from memory because it has not been used.
- An object that the statement is dependent on is dropped or altered, or a privilege needed by the statement is revoked.
- RUNSTATS is run against an object that the statement is dependent on.

If a lock is to be held past commit and it is an S, SIX, or X lock on a table space or a table in a segmented table space, DB2 sometimes demotes that lock to an intent lock (IX or IS) at commit. DB2 demotes a gross lock if it was acquired for one of the following reasons:

- DB2 acquired the gross lock because of lock escalation.
- The application issued a LOCK TABLE.
- The application issued a mass delete (DELETE FROM *object* without a WHERE clause or TRUNCATE).

Choose a combination of values for ACQUIRE and RELEASE based on the characteristics of the particular application.

## Example

An application selects employee names and telephone numbers from a table, according to different criteria. Employees can update their own telephone numbers.

They can perform several searches in succession. The application is bound with the options ACQUIRE(USE) and RELEASE(DEALLOCATE), for these reasons:

- The alternative to ACQUIRE(USE), ACQUIRE(ALLOCATE), gets a lock of mode IX on the table space as soon as the application starts, because that is needed if an update occurs. But most uses of the application do not update the table and so need only the less restrictive IS lock. ACQUIRE(USE) gets the IS lock when the table is first accessed, and DB2 promotes the lock to mode IX if that is needed later.
- Most uses of this application do not update and do not commit. For those uses, little difference exists between RELEASE(COMMIT) and RELEASE(DEALLOCATE). However, administrators might update several phone numbers in one session with the application, and the application commits after each update. In that case, RELEASE(COMMIT) releases a lock that DB2 must acquire again immediately. RELEASE(DEALLOCATE) holds the lock until the application ends, avoiding the processing needed to release and acquire the lock several times.

### Combinations of ACQUIRE and RELEASE options:

Different combinations of bind options have advantages and disadvantages for certain situations.

### ACQUIRE(ALLOCATE) / RELEASE(DEALLOCATE)

#### GUPI

In some cases, this combination can avoid deadlocks by locking all needed resources as soon as the program starts to run. This combination is most useful for a long-running application that runs for hours and accesses various tables, because it prevents an untimely deadlock from wasting that processing.

- All tables or table spaces used in DBRMs bound directly to the plan are locked when the plan is allocated. (LOB and XML table spaces are not locked when the plan is allocated and are only locked when accessed.)
- All tables or table spaces are unlocked only when the plan terminates.
- The locks used are the most restrictive needed to execute all SQL statements in the plan regardless of whether the statements are actually executed.
- Restrictive states are not checked until the page set is accessed. Locking when the plan is allocated insures that the job is compatible with other SQL jobs. Waiting until the first access to check restrictive states provides greater availability; however, it is possible that an SQL transaction could:
  - Hold a lock on a table space or partition that is stopped
  - Acquire a lock on a table space or partition that is started for DB2 utility access only (ACCESS(UT))
  - Acquire an exclusive lock (IX, X) on a table space or partition that is started for read access only (ACCESS(RO)), thus prohibiting access by readers

**Disadvantages:** This combination reduces concurrency. It can lock resources in high demand for longer than needed. Also, the option ACQUIRE(ALLOCATE) turns off selective partition locking; if you are accessing a partitioned table space, all partitions are locked.

**Restriction:** This combination is not allowed for BIND PACKAGE. Use this combination if processing efficiency is more important than concurrency. It is a

good choice for batch jobs that would release table and table space locks only to reacquire them almost immediately. It might even improve concurrency, by allowing batch jobs to finish sooner. Generally, do not use this combination if your application contains many SQL statements that are often not executed.

#### ACQUIRE(USE) / RELEASE(DEALLOCATE)

This combination results in the most efficient use of processing time in most cases.

- A table, partition, or table space used by the plan or package is locked only if it is needed while running.
- All tables or table spaces are unlocked only when the plan terminates.
- The least restrictive lock needed to execute each SQL statement is used, with the exception that if a more restrictive lock remains from a previous statement, that lock is used without change.

**Disadvantages:** This combination can increase the frequency of deadlocks. Because all locks are acquired in a sequence that is predictable only in an actual run, more concurrent access delays might occur.

#### ACQUIRE(USE) / RELEASE(COMMIT)

This combination is the default combination and provides the greatest concurrency, but it requires more processing time if the application commits frequently.

- A table, partition, or table space is locked only when needed. That locking is important if the process contains many SQL statements that are rarely used or statements that are intended to access data only in certain circumstances.
- All tables and table spaces are unlocked when:

##### TSO, Batch, and CAF

An SQL COMMIT or ROLLBACK statement is issued, or your application process terminates

**IMS** A CHKP or SYNC call (for single-mode transactions), a GU call to the I/O PCB, or a ROLL or ROLB call is completed

**CICS** A SYNCPOINT command is issued.


##### Exception:

If the cursor is defined WITH HOLD, table or table space locks necessary to maintain cursor position are held past the commit point.

- Table, partition, or table space locks are released at the next commit point unless the cursor is defined WITH HOLD.
- The least restrictive lock needed to execute each SQL statement is used except when a more restrictive lock remains from a previous statement. In that case, that lock is used without change.

**Disadvantages:** This combination can increase the frequency of deadlocks. Because all locks are acquired in a sequence that is predictable only in an actual run, more concurrent access delays might occur.

#### ACQUIRE(ALLOCATE) / RELEASE(COMMIT)

This combination is not allowed; it results in an error message from BIND. 

## Choosing an ISOLATION option

Various *isolation levels* offer less or more concurrency at the cost of more or less protection from other application processes.

The ISOLATION option of an application specifies the degree to which operations are isolated from the possible effects of other operations acting concurrently. Based on this information, DB2 releases S and U locks on rows or pages as soon as possible.

Regardless of the isolation level that you specify, outstanding claims on DB2 objects can inhibit the execution of DB2 utilities or commands.

The default ISOLATION option differs for different types of bind operations, as shown in the following table.

Table 70. The default ISOLATION values for different types of bind operations

Operation	Default value
BIND PLAN	ISOLATION (CS) with CURRENTDATA (NO)
BIND PACKAGE	The value used by the plan that includes the package in its package list
REBIND PLAN or PACKAGE	The existing value for the plan or package being rebound

To ensure that your applications can access your data concurrently:

Choose an isolation level according to the needs and characteristics of the particular application. The recommended order of preference for isolation levels is:

1. Cursor stability (CS)
2. Uncommitted read (UR)
3. Read stability (RS)
4. Repeatable read (RR)

For ISOLATION(CS), the CURRENTDATA(NO) option is preferred over CURRENTDATA(YES).

### The ISOLATION (CS) option:

The ISOLATION (CS) or *cursor stability* option allows maximum concurrency with data integrity.



However, after the process leaves a row or page, another process can change the data. With CURRENTDATA(NO), the process does not have to leave a row or page to allow another process to change the data. If the first process returns to read the same row or page, the data is not necessarily the same. Consider these consequences of that possibility:

- For table spaces created with LOCKSIZE ROW, PAGE, or ANY, a change can occur even while executing a single SQL statement, if the statement reads the same row more than once. In the following statement, data read by the inner SELECT can be changed by another transaction before it is read by the outer SELECT.

```
SELECT * FROM T1
  WHERE C1 = (SELECT MAX(C1) FROM T1);
```

Therefore, the information returned by this query might be from a row that is no longer the one with the maximum value for C1.

- In another case, if your process reads a row and returns later to update it, that row might no longer exist or might not exist in the state that it did when your application process originally read it. That is, another application might have deleted or updated the row. **If your application is doing non-cursor operations on a row under the cursor, make sure that the application can tolerate “not found” conditions.**

Similarly, assume another application updates a row after you read it. If your process returns later to update it based on the value you originally read, you are, in effect, erasing the update made by the other process. **If you use ISOLATION(CS) with update, your process might need to lock out concurrent updates.** One method is to declare a cursor with the FOR UPDATE clause.

For packages and plans that contain updatable static scrollable cursors, ISOLATION(CS) lets DB2 use *optimistic concurrency control*. DB2 can use optimistic concurrency control to shorten the amount of time that locks are held in the following situations:

- Between consecutive fetch operations
- Between fetch operations and subsequent positioned update or delete operations

DB2 cannot use optimistic concurrency control for dynamic scrollable cursors. With dynamic scrollable cursors, the most recently fetched row or page from the base table remains locked to maintain position for a positioned update or delete.

The two following figures show processing of positioned update and delete operations with static scrollable cursors without optimistic concurrency control and with optimistic concurrency control.

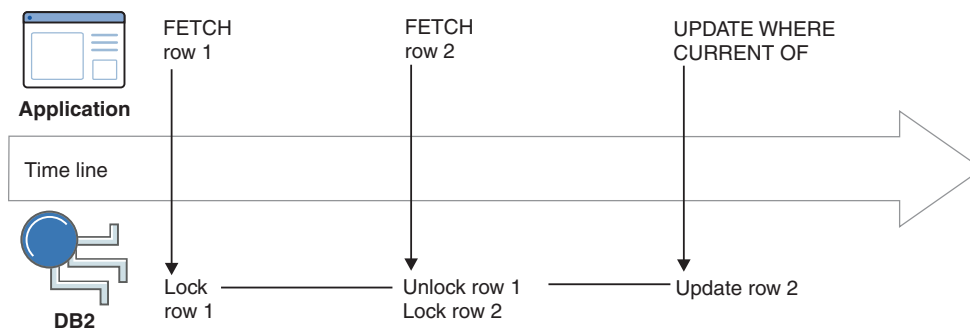


Figure 44. Positioned updates and deletes without optimistic concurrency control

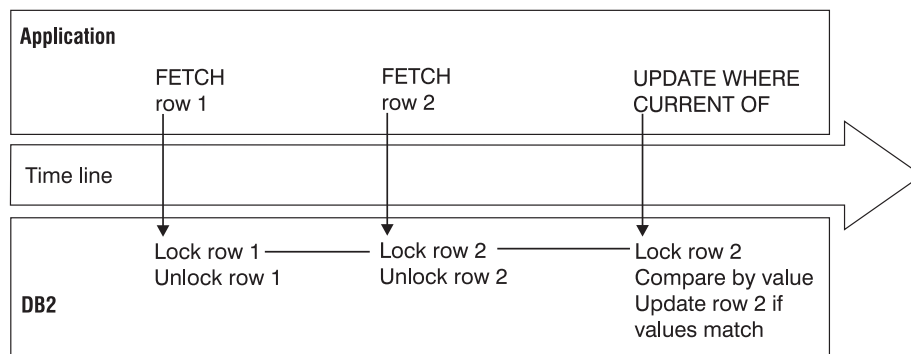



Figure 45. Positioned updates and deletes with optimistic concurrency control

Optimistic concurrency control consists of the following steps:

1. When the application requests a fetch operation to position the cursor on a row, DB2 locks that row, executes the FETCH, and releases the lock.
2. When the application requests a positioned update or delete operation on the row, DB2 performs the following steps:
  - a. Locks the row.
  - b. Reevaluates the predicate to ensure that the row still qualifies for the result table. 

#### The ISOLATION (UR) option:

The ISOLATION (UR) or *uncommitted read* option allows an application to read while acquiring few locks, at the risk of reading uncommitted data. UR isolation applies only to read-only operations: SELECT, SELECT INTO, or FETCH from a read-only result table.

#### Reading uncommitted data introduces an element of uncertainty.

For example, an application tracks the movement of work from station to station along an assembly line. As items move from one station to another, the application subtracts from the count of items at the first station and adds to the count of items at the second. Assume you want to query the count of items at all the stations, while the application is running concurrently.

If your query reads data that the application has changed but has not committed:

- If the application subtracts an amount from one record before adding it to another, *the query could miss the amount entirely.*
- If the application adds first and then subtracts, *the query could add the amount twice.*

If those situations can occur and are unacceptable, do not use UR isolation.

#### Restrictions for using ISOLATION (UR)

You cannot use the ISOLATION (UR) option for the following types of statements:

- INSERT, UPDATE, DELETE, and MERGE
- SELECT FROM INSERT, UPDATE, DELETE, or MERGE.
- Any cursor defined with a FOR UPDATE clause

If you bind with ISOLATION(UR) and the statement does not specify WITH RR or WITH RS, DB2 uses CS isolation for these types of statements.

When an application uses ISO(UR) and runs concurrently with applications that update variable-length records such that the update creates a double-overflow record, the ISO(UR) application might miss rows that are being updated.

### **When to use ISOLATION (UR)**

You can probably use UR isolation in cases such as the following examples:

#### **When errors cannot occur**

The follow examples describe situations in which errors can be avoided while using the ISOLATION (UR) option.

##### **Reference tables**

Like a tables of descriptions of parts by part number. Such tables are rarely updated, and reading an uncommitted update is probably no more damaging than reading the table 5 seconds earlier.

##### **Tables with limited access**

The employee table of Spiffy Computer, our hypothetical user. For security reasons, updates can be made to the table only by members of a single department. And that department is also the only one that can query the entire table. It is easy to restrict queries to times when no updates are being made and then run with UR isolation.

#### **When an error is acceptable**

Spiffy Computer wants to do some statistical analysis on employee data. A typical question is, "What is the average salary by sex within education level?" Because reading an occasional uncommitted record cannot affect the averages much, UR isolation can be used.

#### **When the data already contains inconsistent information**

Spiffy computer gets sales leads from various sources. The data is often inconsistent or wrong, and end users of the data are accustomed to dealing with that. Inconsistent access to a table of data on sales leads does not add to the problem.

### **When not to use ISOLATION (UR)**

Do not use uncommitted read, ISOLATION (UR), in the following cases:

- When computations must balance
- When the answer must be accurate
- When you are unsure whether using the ISOLATION (UR) might cause damage

### **The ISOLATION (RS) option:**

The ISOLATION (RS) or *read stability* option allows the application to read the same pages or rows more than once without allowing qualifying rows to be updated or deleted by another process.

It offers possibly greater concurrency than repeatable read, because although other applications cannot change rows that are returned to the original application, they can insert new rows or update rows that did not satisfy the original search condition of the application. Only those rows or pages that satisfy the stage 1

predicate (and all rows or pages evaluated during stage 2 processing) are locked until the application commits. The following figure illustrates this process. In the example, the rows held by locks L2 and L4 satisfy the predicate.

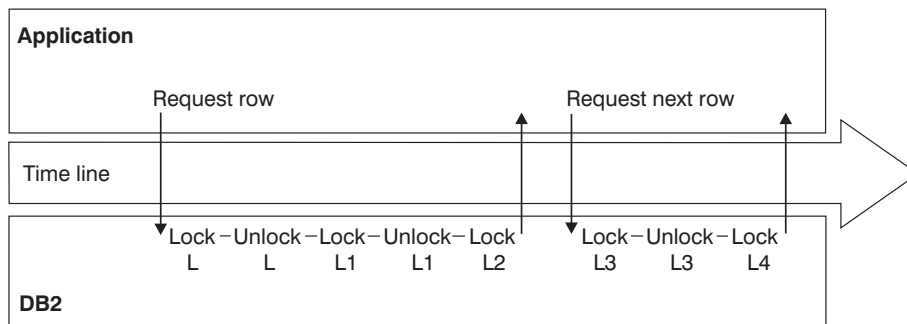


Figure 46. How an application using RS isolation acquires locks when no lock avoidance techniques are used. Locks L2 and L4 are held until the application commits. The other locks aren't held.

Applications using read stability can leave rows or pages locked for long periods, especially in a distributed environment.

If you do use read stability, plan for frequent commit points.

An installation option determines the mode of lock chosen for a cursor defined with the FOR UPDATE OF clause and bound with read stability.

#### The ISOLATION (RR) option:

The ISOLATION (RR) or *repeatable read* option allows the application to read the same pages or rows more than once without allowing any update, insert, or delete operations by another process. All accessed rows or pages are locked, even if they do not satisfy the predicate.

**GUPI** Applications that use repeatable read can leave rows or pages locked for longer periods, especially in a distributed environment, and they can claim more logical partitions than similar applications using cursor stability.

Applications that use repeatable read and access a nonpartitioned index cannot run concurrently with utility operations that drain all claim classes of the nonpartitioned index, even if they are accessing different logical partitions. For example, an application bound with ISOLATION(RR) cannot update partition 1 while the LOAD utility loads data into partition 2. Concurrency is restricted because the utility needs to drain all the repeatable-read applications from the nonpartitioned index to protect the repeatability of the reads by the application.

They are also subject to being drained more often by utility operations.

Because so many locks can be taken, lock escalation might take place. Frequent commits release the locks and can help avoid lock escalation.

With repeatable read, lock promotion occurs for table space scan to prevent the insertion of rows that might qualify for the predicate. (If access is via index, DB2 locks the key range. If access is via table space scans, DB2 locks the table, partition, or table space.)

An installation option determines the mode of lock chosen for a cursor defined with the FOR UPDATE OF clause and bound with repeatable read. **GUPI**

## Choosing a CURRENTDATA option

The CURRENTDATA option has different effects, depending on whether access is local or remote.

**GUPI**

### CURRENTDATA for local access:

For local access, the CURRENTDATA option tells whether the data upon which your cursor is positioned must remain identical to (or “current with”) the data in the local base table.

**GUPI**

For cursors positioned on data in a work file, the CURRENTDATA option has no effect. This effect only applies to read-only or ambiguous cursors in plans or packages bound with CS isolation.

### CURRENTDATA (YES)

CURRENTDATA (YES) Means that the data upon which the cursor is positioned cannot change while the cursor is positioned on it. If the cursor is positioned on data in a local base table or index, then the data returned with the cursor is current with the contents of that table or index. If the cursor is positioned on data in a work file, the data returned with the cursor is current only with the contents of the work file; it is not necessarily current with the contents of the underlying table or index.

The following figure shows locking with CURRENTDATA (YES).

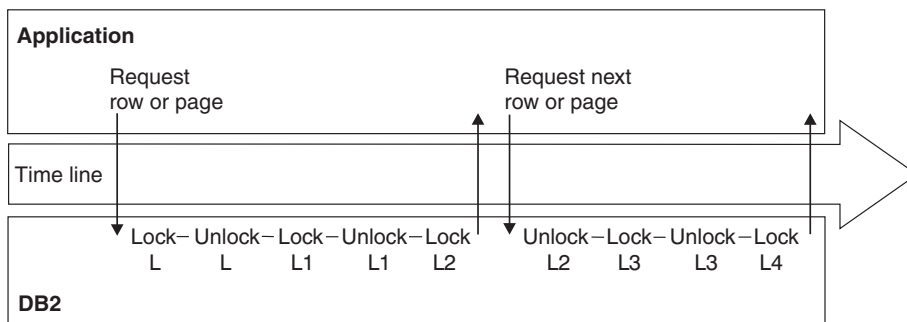


Figure 47. How an application using CS isolation with CURRENTDATA (YES) acquires locks. This figure shows access to the base table. The L2 and L4 locks are released after DB2 moves to the next row or page. When the application commits, the last lock is released.

As with work files, if a cursor uses query parallelism, data is not necessarily current with the contents of the table or index, regardless of whether a work file is used. Therefore, for work file access or for parallelism on read-only queries, the CURRENTDATA option has no effect.

If you are using parallelism but want to maintain currency with the data, you have the following options:

- Disable parallelism (Use SET DEGREE = '1' or bind with DEGREE(1)).
- Use isolation RR or RS (parallelism can still be used).

- Use the LOCK TABLE statement (parallelism can still be used).

### CURRENTDATA(NO)

Is similar to CURRENTDATA(YES) except for the case where a cursor is accessing a base table rather than a result table in a work file. In those cases, although CURRENTDATA(YES) can guarantee that the cursor and the base table are current, CURRENTDATA(NO) makes no such guarantee. **GUIP**

### CURRENTDATA for remote access:

For a request to a remote system, CURRENTDATA has an effect for ambiguous cursors using isolation levels RR, RS, or CS.

**GUIP** For access to a remote table or index, CURRENTDATA(YES) turns off block fetching for ambiguous cursors. The data returned with the cursor is current with the contents of the remote table or index for ambiguous cursors. Turning on block fetch offers best performance, but it means the cursor is not current with the base table at the remote site. **GUIP**

### Lock avoidance:

With CURRENTDATA(NO), you have much greater opportunity for avoiding locks.

**GUIP** DB2 can test to see if a row or page has committed data on it. If it has, DB2 does not have to obtain a lock on the data at all. Unlocked data is returned to the application, and the data can be changed while the cursor is positioned on the row. (For SELECT statements in which no cursor is used, such as those that return a single row, a lock is not held on the row unless you specify WITH RS or WITH RR on the statement.)

To take the best advantage of this method of avoiding locks, make sure all applications that are accessing data concurrently issue COMMIT statements frequently.

The following figure shows how DB2 can avoid taking locks and the table below summarizes the factors that influence lock avoidance.

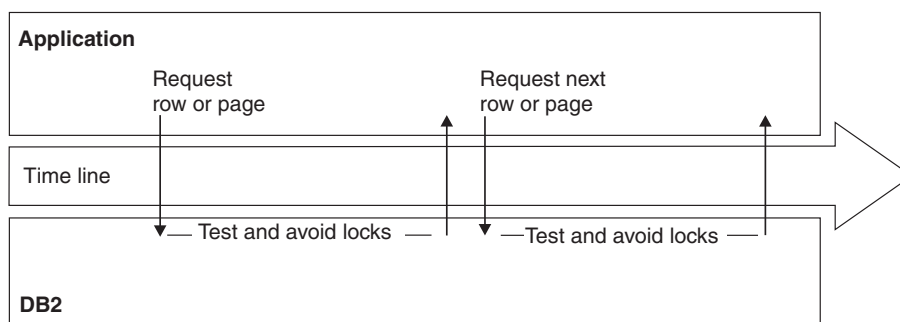


Figure 48. Best case of avoiding locks using CS isolation with CURRENTDATA(NO). This figure shows access to the base table. If DB2 must take a lock, then locks are released when DB2 moves to the next row or page, or when the application commits (the same as CURRENTDATA(YES)).

Table 71. Lock avoidance factors. “Returned data” means data that satisfies the predicate. “Rejected data” is that which does not satisfy the predicate.

Isolation	CURRENTDATA	Cursor type	Avoid locks on returned data?	Avoid locks on rejected data?
UR	N/A	Read-only	N/A	N/A
CS	YES	Any	No	Yes <sup>1</sup>
	NO	Read-only	Yes	
		Updatable	No	
		Ambiguous	Yes	
RS	N/A	Any	No	Yes <sup>1, 2</sup>
RR	N/A	Any	No	No

#### Notes:

1. Locks are avoided when the row is disqualified after stage 1 processing
2. When using ISO(RS) and multi-row fetch, DB2 releases locks that were acquired on Stage 1 qualified rows, but which subsequently failed to qualify for stage 2 predicates at the next fetch of the cursor. **GUPI**

#### Problems with ambiguous cursors:

A cursor is considered *ambiguous* if DB2 cannot tell whether it is used for update or read-only purposes.

**GUPI** If the cursor appears to be used only for read-only, but dynamic SQL could modify data through the cursor, then the cursor is ambiguous. If you use CURRENTDATA to indicate an ambiguous cursor is read-only when it is actually targeted by dynamic SQL for modification, you’ll get an error. Ambiguous cursors can sometimes prevent DB2 from using lock avoidance techniques. However, misuse of an ambiguous cursor can cause your program to receive a -510 SQLCODE, meaning:

- The plan or package is bound with CURRENTDATA(NO)
- An OPEN CURSOR statement is performed *before* a dynamic DELETE WHERE CURRENT OF statement against that cursor is prepared
- One of the following conditions is true for the open cursor:
  - Lock avoidance is successfully used on that statement.
  - Query parallelism is used.
  - The cursor is distributed, and block fetching is used.

In all cases, it is a good programming technique to eliminate the ambiguity by declaring the cursor with either the FOR FETCH ONLY or the FOR UPDATE

clause. **GUPI**

#### Conflicting plan and package bind options

A plan bound with one set of options can include packages in its package list that were bound with different sets of options.

**GUIP** In general, statements in a DBRM bound as a package use the options that the package was bound with, and statements in DBRMs bound to a plan use the options that the plan was bound with.

For example, the plan value for CURRENTDATA has no effect on the packages executing under that plan. If you do not specify a CURRENTDATA option explicitly when you bind a package, the default is CURRENTDATA(YES).

The rules are slightly different for the bind options RELEASE and ISOLATION. The values of those two options are set when the lock on the resource is acquired and usually stay in effect until the lock is released. But a conflict can occur if a statement that is bound with one pair of values requests a lock on a resource that is already locked by a statement that is bound with a different pair of values. DB2 resolves the conflict by resetting each option with the available value that causes the lock to be held for the greatest duration.

If the conflict is between RELEASE(COMMIT) and RELEASE(DEALLOCATE), then the value used is RELEASE(DEALLOCATE).

The table below shows how conflicts between isolation levels are resolved. The first column is the existing isolation level, and the remaining columns show what happens when another isolation level is requested by a new application process.

Table 72. Resolving isolation conflicts

	UR	CS	RS	RR
UR	n/a	CS	RS	RR
CS	CS	n/a	RS	RR
RS	RS	RS	n/a	RR
RR	RR	RR	RR	n/a

## GUIP

### Using SQL statements to override isolation levels

You can override the isolation level with which a plan or package is bound.

**GUIP** To override the isolation level for a specific SQL statement:

- Issue the SQL statements, and include a WITH *isolation level* clause. The WITH *isolation level* clause:
  - Can be used on these statements:
    - SELECT
    - SELECT INTO
    - Searched DELETE
    - INSERT from fullselect
    - Searched UPDATE
  - Cannot be used on subqueries.
  - Can specify the isolation levels that specifically apply to its statement. (For example, because WITH UR applies only to read-only operations, you cannot use it on an INSERT statement.)
  - Overrides the isolation level for the plan or package only for the statement in which it appears.

The following statement finds the maximum, minimum, and average bonus in the sample employee table.

```
SELECT MAX(BONUS), MIN(BONUS), AVG(BONUS)
  INTO :MAX, :MIN, :AVG
  FROM DSN8910.EMP
  WITH UR;
```

The statement is executed with uncommitted read isolation, regardless of the value of ISOLATION with which the plan or package containing the statement is bound.

- If you use the WITH RR or WITH RS clause, you can issue SELECT and SELECT INTO statements, and specify the following options:
  - USE AND KEEP EXCLUSIVE LOCKS
  - USE AND KEEP UPDATE LOCKS
  - USE AND KEEP SHARE LOCKS

To use these options, specify them as shown in the following example:

```
SELECT ...
  WITH RS USE AND KEEP UPDATE LOCKS;
```

By using one of these options, you tell DB2 to acquire and hold a specific mode of lock on all the qualified pages or rows. The following table shows which mode of lock is held on rows or pages when you specify the SELECT using the WITH RS or WITH RR isolation clause.

*Table 73. Which mode of lock is held on rows or pages when you specify the SELECT using the WITH RS or WITH RR isolation clause*

Option Value	Lock Mode
USE AND KEEP EXCLUSIVE LOCKS	X
USE AND KEEP UPDATE LOCKS	U
USE AND KEEP SHARE LOCKS	S

With read stability (RS) isolation, a row or page that is rejected during stage 2 processing might still have a lock held on it, even though it is not returned to the application.

With repeatable read (RR) isolation, DB2 acquires locks on all pages or rows that fall within the range of the selection expression.

All locks are held until the application commits. Although this option can reduce concurrency, it can prevent some types of deadlocks and can better serialize access to data. **GUPI**

## Locking a table explicitly

You can override the DB2 rules for choosing initial lock attributes.

**GUPI** You can use LOCK TABLE on any table, including the auxiliary tables of LOB and XML table spaces. However, LOCK TABLE has no effect on locks that are acquired at a remote server.

To lock a table explicitly:

Issue a LOCK TABLE statement. Two examples are:

```
LOCK TABLE table-name IN SHARE MODE;
LOCK TABLE table-name PART n IN EXCLUSIVE MODE;
```

Executing the statement requests a lock immediately, unless a suitable lock exists already. The bind option `RELEASE` determines when locks acquired by `LOCK`

`TABLE` or `LOCK TABLE` with the `PART` option are released. **GUIP**

#### When to use `LOCK TABLE`:

The `LOCK TABLE` statement is often appropriate for particularly high-priority applications.

**GUIP** It can improve performance if `LOCKMAX` disables lock escalation or sets a high threshold for it.

For example, suppose that you intend to execute an SQL statement to change job code 21A to code 23 in a table of employee data. The table is defined with:

- The name `PERSADM1.EMPLOYEE_DATA`
- `LOCKSIZE ROW`
- `LOCKMAX 0`, which disables lock escalation

Because the change affects about 15% of the employees, the statement can require many row locks of mode X.

- To avoid the overhead for locks, first execute the following statement:  
`LOCK TABLE PERSADM1.EMPLOYEE_DATA IN EXCLUSIVE MODE;`
- If `EMPLOYEE_DATA` is a partitioned table space, you can choose to lock individual partitions as you update them. For example:  
`LOCK TABLE PERSADM1.EMPLOYEE_DATA PART 1 IN EXCLUSIVE MODE;`

When the statement is executed, DB2 locks partition 1 with an X lock. The lock has no effect on locks that already exist on other partitions in the table space.

**GUIP**

#### The effect of `LOCK TABLE`:

Different locks are acquired when you issue a `LOCK TABLE` statement, depending on the mode of the lock and the type of table space.

**GUIP** The table below shows the modes of locks acquired in segmented and nonsegmented table spaces for the `SHARE` and `EXCLUSIVE` modes of `LOCK TABLE`. Auxiliary tables of LOB table spaces are considered nonsegmented table spaces and have the same locking behavior.

*Table 74. Modes of locks acquired by `LOCK TABLE`. `LOCK TABLE` on partitions behave the same as nonsegmented table spaces.*

LOCK TABLE IN	Nonsegmented or Universal Table Space	Segmented Table Space	
		Table	Table Space
EXCLUSIVE MODE	X	X	IX
SHARE MODE	S or SIX	S or SIX	IS

**Note:** The SIX lock is acquired if the process already holds an IX lock. `SHARE MODE` has no effect if the process already has a lock of mode SIX, U, or X.

## Recommendations for using LOCK TABLE:

You can use LOCK TABLE to prevent other application processes from changing any row in a table or partition that your process accesses.

**GUIP** For example, suppose that you access several tables. You can tolerate concurrent updates on all the tables except one; for that one, you need RR or RS isolation. You can handle the situation in several ways:

(If other tables exist in the same table space, see the caution that follows.) The LOCK TABLE statement locks out changes by any other process, giving the exceptional table a degree of isolation even more thorough than repeatable read. All tables in other table spaces are shared for concurrent update.

You might want to lock a table or partition that is normally shared for any of the following reasons:

### Taking a “snapshot”

If you want to access an entire table throughout a unit of work as it was at a particular moment, you must lock out concurrent changes. If other processes can access the table, use LOCK TABLE IN SHARE MODE. (RR isolation is not enough; it locks out changes only from rows or pages you have already accessed.)

### Avoiding overhead

If you want to update a large part of a table, it can be more efficient to prevent concurrent access than to lock each page as it is updated and unlock it when it is committed. Use LOCK TABLE IN EXCLUSIVE MODE.

### Preventing timeouts

Your application has a high priority and must not risk timeouts from contention with other application processes. Depending on whether your application updates or not, use either LOCK IN EXCLUSIVE MODE or LOCK TABLE IN SHARE MODE.

- Bind the application plan with RR or RS isolation. But that affects all the tables you access and might reduce concurrency.
- Design the application to use packages and access the exceptional table in only a few packages, and bind those packages with RR or RS isolation, and the plan with CS isolation. Only the tables accessed within those packages are accessed with RR or RS isolation.
- Add the clause WITH RR or WITH RS to statements that must be executed with RR or RS isolation. Statements that do not use WITH are executed as specified by the bind option ISOLATION.
- Bind the application plan with CS isolation and execute LOCK TABLE for the exceptional table. (If other tables exist in the same table space, see the caution that follows.) The LOCK TABLE statement locks out changes by any other process, giving the exceptional table a degree of isolation even more thorough than repeatable read. All tables in other table spaces are shared for concurrent update.

**Caution when using LOCK TABLE with simple table spaces:** The statement locks all tables in a simple table space, even though you name only one table. No other process can update the table space for the duration of the lock. If the

lock is in exclusive mode, no other process can read the table space, unless that process is running with UR isolation.

#### **GUIP**

### **How access paths effect locks:**

The access path that DB2 uses can affect the mode, size, and even the object of a lock.

#### **GUIP**

For example, an UPDATE statement using a table space scan might need an X lock on the entire table space. If rows to be updated are located through an index, the same statement might need only an IX lock on the table space and X locks on individual pages or rows.

If you use the EXPLAIN statement to investigate the access path chosen for an SQL statement, then check the lock mode in column TSLOCKMODE of the resulting PLAN\_TABLE. If the table resides in a nonsegmented table space, or is defined with LOCKSIZE TABLESPACE, the mode shown is that of the table space lock. Otherwise, the mode is that of the table lock.

Important points that you should consider when you work with DB2 locks include:

- You usually do not have to lock data explicitly in your program.
- DB2 ensures that your program does not retrieve uncommitted data unless you specifically allow that.
- Any page or row where your program updates, inserts, or deletes stays locked at least until the end of a unit of work, regardless of the isolation level. No other process can access the object in any way until then, unless you specifically allow that access to that process.
- Commit often for concurrency. Determine points in your program where changed data is consistent. At those points, you should issue:

#### **TSO, Batch, and CAF**

An SQL COMMIT statement

**IMS** A CHKP or SYNC call, or (for single-mode transactions) a GU call to the I/O PCB

**CICS** A SYNCPOINT command.

- Bind with ACQUIRE(USE) to improve concurrency.
- Set ISOLATION (usually RR, RS, or CS) when you bind the plan or package.
  - With RR (repeatable read), all accessed pages or rows are locked until the next commit point.
  - With RS (read stability), all qualifying pages or rows are locked until the next commit point.
  - With CS (cursor stability), only the pages or rows currently accessed can be locked, and those locks might be avoided. (You can access one page or row for each open cursor.)
- You can also set isolation for specific SQL statements, using WITH.
- A deadlock can occur if two processes each hold a resource that the other needs. One process is chosen as “victim”, its unit of work is rolled back, and an SQL error code is issued.
- You can lock an entire nonsegmented table space, or an entire table in a segmented table space, by the LOCK TABLE statement:

- To let other users retrieve, but not update, delete, or insert, issue the following statement:  
LOCK TABLE *table-name* IN SHARE MODE
- To prevent other users from accessing rows in any way, except by using UR isolation, issue the following statement:  
LOCK TABLE *table-name* IN EXCLUSIVE MODE

#### GUPI

### Skipping locked data

You can use the SKIP LOCKED DATA option to skip rows that are locked to increase the concurrency of applications and transactions that can tolerate incomplete or inaccurate results.

#### GUPI

The SKIP LOCKED DATA option allows a transaction to skip rows that are incompatibly locked by other transactions. Because the SKIP LOCKED DATA option skips these rows, the performance of some applications can be improved by eliminating lock wait time. However, you should use the SKIP LOCKED DATA option only for applications that can reasonably tolerate the absence of the skipped rows in the returned data. If your transaction uses the SKIP LOCKED DATA option, it does not read or modify data that is held by locks.

**Important:** When DB2 skips data because of the SKIP LOCKED DATA option, it does not issue a warning. Even if only a subset of the data that satisfies a query is returned or modified, the transaction completes as if no data was skipped. To use the SKIP LOCKED DATA option, you must be willing to accept inconsistent results.

To skip locked rows for applications and transactions that require fast results and data that is only somewhat accurate:

Specify the SKIP LOCKED DATA clause in one of the following SQL statements:

- SELECT
- SELECT INTO
- PREPARE
- Searched-UPDATE
- Searched-DELETE

You can also use the SKIP LOCKED DATA option with the UNLOAD utility.

### Isolation levels and SKIP LOCKED DATA:

You can use the SKIP LOCKED DATA option in conjunction with only certain isolation levels.

#### GUPI

You can use the SKIP LOCKED DATA option with cursor stability (CS) isolation and read stability (RS) isolation. However, you cannot use SKIP LOCKED DATA with uncommitted read (UR) or repeatable read (RR) isolation levels. For

UR and RR, DB2 ignores the SKIP LOCKED DATA clause. **GUPI**

### Lock sizes for SKIP LOCKED DATA:

The SKIP LOCKED DATA option works only with row locks and page locks.

**GUPI** If you specify SKIP LOCKED DATA for a transaction with row level locking, incompatibly locked rows are skipped. If you specify SKIP LOCKED DATA for a transaction with page level locking, all rows on pages with incompatible locks are skipped.

In general, the SKIP LOCKED DATA clause does not apply to table, partition, LOB, XML, or table space locks. When LOCKSIZE TABLE or LOCKSIZE TABLESPACE is specified for a table space or when a lock is escalated to a gross table, partition, or table space lock, DB2 ignores the SKIP LOCKED DATA clause. **GUPI**

### Lock mode compatibility and SKIP LOCKED DATA:

Lock mode compatibility for transactions that use the SKIP LOCKED DATA option is the same as lock mode compatibility for other page- and row-level locks.

**GUPI** However, when incompatible locks are held, a transaction that uses the SKIP LOCKED DATA option does not wait for the locks to be released and skips the locked data instead.

### Example

Suppose that Application A holds an s lock on a row that Process B also wants to access. The query in Process B specifies SKIP LOCKED DATA. The outcome of Process B depends on the mode of lock that it acquires. If Process B requires a compatible s or u lock, Process B can access the row on which Application A holds an s lock. If Process B requires an incompatible x lock, Process B cannot access the locked row. Because the SKIP LOCKED DATA option is specified, that row is skipped and the results of Process B are returned without that row. **GUPI**

### Examples of results with SKIP LOCKED DATA:

With some queries that use the SKIP LOCKED DATA clause, you can receive unexpected or inconsistent results.

#### **GUPI**

Suppose that a table EXTABLE exists in a table space with row-level locking, or in a table space with page-level locking and the rows of the table are distributed across several pages. EXTABLE has two columns, C1 and C2, that contain the following data:

C1	C2
1	AAAA
2	BBBB
3	CCCC
4	DDDD

Next, suppose that a transaction issues the following update statement:

```
UPDATE EXTABLE
  SET C1 = 99
  WHERE C1 < 3;
```

Suppose that a second transaction issues the following SELECT statement before the previous transaction commits:

```
SELECT COUNT (*) FROM EXTABLE
WHERE C2>=AAAA
SKIP LOCKED DATA;
```

If you do not have an index defined on C2, DB2 returns a count of 2 because DB2 skips the two rows that are locked by the uncommitted UPDATE statement.


However, you cannot always expect DB2 to skip this data. 

## Using other options to control locking

You can use various options to control such things as how many locks are used and which mode is used for certain locks such as information about the LOCKPART clause of CREATE and ALTER TABLESPACE.


### Specifying the maximum number of locks that a single process can hold

The LOCKS PER USER field of installation panel DSNTIPJ specifies the maximum number of page, row, LOB, or XML locks that can be held by a single process at any one time. It includes locks for both the DB2 catalog and directory and for user data

 When a request for a page, row, LOB, or XML lock exceeds the specified limit, it receives SQLCODE -904: "resource unavailable" (SQLSTATE '57011'). The requested lock cannot be acquired until some of the existing locks are released.


The default value is 10 000.

The default should be adequate for 90 percent of the work load when using page locks. If you use row locks on very large tables, you might want a higher value. If you use LOBs or XML data, you might need a higher value.

- Review application processes that require higher values to see if they can use table space locks rather than page, row, LOB, or XML locks. The accounting trace shows the maximum number of page, row, LOB, or XML locks a process held while a application runs.
- Remember that the value specified is for a single application. Each concurrent application can potentially hold up to the maximum number of locks specified. Do not specify zero or a very large number unless it is required to run your applications. 

### Specify the size of locks for a table space

The LOCKSIZE clause of CREATE and ALTER TABLESPACE statements specifies the size for locks held on a table or table space by any application process that accesses it.

 In addition to using the ALTER TABLESPACE statement to change the lock size for user data, you can also change the lock size of any DB2 catalog table space that is neither a LOB table space nor a table space that contains links. The relevant options are:

#### LOCKSIZE TABLESPACE

A process acquires no table, page, row, LOB, or XML locks within the table space. That improves performance by reducing the number of locks maintained, but greatly inhibits concurrency.

### **LOCKSIZE TABLE**

A process acquires table locks on tables in a segmented table space without partitions. If the table space contains more than one table, this option can provide acceptable concurrency with little extra cost in processor resources.

### **LOCKSIZE PAGE**

A process acquires page locks, plus table, partition, or table space locks of modes that permit page locks (IS, IX, or SIX). The effect is not absolute: a process can still acquire a table, partition, or table space lock of mode S or X, without page locks, if that is needed. In that case, the bind process issues a message warning that the lock size has been promoted as described under “Lock promotion” on page 344.

### **LOCKSIZE ROW**

A process acquires row locks, plus table, partition, or table space locks of modes that permit row locks (IS, IX, or SIX). The effect is not absolute: a process can still acquire a table, partition, or table space lock of mode S or X, without row locks, if that is needed. In that case, the bind process issues a message warning that the lock size has been promoted as described under “Lock promotion” on page 344.

### **LOCKSIZE ANY**

DB2 chooses the size of the lock, usually LOCKSIZE PAGE.

### **LOCKSIZE LOB**

If a LOB must be accessed, a process acquires LOB locks and the necessary LOB table space locks (IS or IX). This option is valid only for LOB table spaces. See “LOB locks” on page 378 for more information about LOB locking.

### **LOCKSIZE XML**

If XML must be accessed, a process acquires XML locks and the necessary XML table space locks (IS or IX). This option is valid only for XML table spaces. See “XML locks” on page 381 for more information about XML locking.

DB2 attempts to acquire an S lock on table spaces that are started with read-only access. If the LOCKSIZE is PAGE, ROW, or ANY and DB2 cannot get the S lock, it requests an IS lock. If a partition is started with read-only access, DB2 attempts to get an S lock on the partition that is started RO. For a complete description of how the LOCKSIZE clause affects lock attributes, see “How DB2 chooses lock types” on page 339.

The default option is LOCKSIZE ANY, and the LOCKRULE column of the SYSIBM.SYSTABLESPACE catalog table records the current value for each table space.

If you do not use the default, base your choice upon the results of monitoring applications that use the table space. When considering changing the lock size for a DB2 catalog table space:

Be aware that, in addition to user queries, DB2 internal processes such as bind and authorization checking and utility processing can access the DB2 catalog.

### **Page locks versus row locks:**

The question of whether to use row or page locks depends on your data and your applications. If you are experiencing contention on data pages of a table space now defined with LOCKSIZE PAGE, consider LOCKSIZE ROW. But consider also the trade-offs.

The resource required to acquire, maintain, and release a row lock is about the same as that required for a page lock. If your data has 10 rows per page, a table space scan or an index scan can require nearly 10 times as much resource for row locks as for page locks. But locking only a row at a time, rather than a page, might reduce the chance of contention with some other process by 90%, especially if access is random. (Row locking is not recommended for sequential processing.)

Lock avoidance is very important when row locking is used. Therefore, use ISOLATION(CS) CURRENTDATA(NO) or ISOLATION(UR) whenever possible. In many cases, DB2 can avoid acquiring a lock when reading data that is known to be committed. Thus, if only 2 of 10 rows on a page contain uncommitted data, DB2 must lock the entire page when using page locks, but might ask for locks on only the 2 rows when using row locks. Then, the resource required for row locks would be only twice as much, not 10 times as much, as that required for page locks.

On the other hand, if two applications update the same rows of a page, and not in the same sequence, then row locking might even increase contention. With page locks, the second application to access the page must wait for the first to finish and might time out. With row locks, the two applications can access the same page simultaneously, and might deadlock while trying to access the same set of rows.

In short, no single answer fits all cases.



## Specifying the maximum number of locks that a process can hold on a table space

You can specify the LOCKMAX clause of the CREATE and ALTER TABLESPACE statements for tables of user data and also for tables in the DB2 catalog, by using ALTER TABLESPACE.



### LOCKMAX *n*

Specifies the maximum number of page or row locks that a single application process can hold on the table space before those locks are escalated. For LOB table spaces, this value specifies the number of LOB locks that the application process can hold before escalating. For XML table spaces, this value specifies the number of page, row, and XML locks that the application process can hold before escalating. For an application that uses Sysplex query parallelism, a lock count is maintained on each member.

### LOCKMAX SYSTEM

Specifies that *n* is effectively equal to the system default set by the field LOCKS PER TABLE(SPACE) of installation panel DSNTIPJ.

### LOCKMAX 0

Disables lock escalation entirely.

The default value depends on the value of LOCKSIZE, as shown in the following table.

Table 75. How the default for LOCKMAX is determined

LOCKSIZE	Default for LOCKMAX
ANY	SYSTEM
other	0

**Note:** For XML table spaces, the default value of LOCKMAX is inherited from the base table space.

**Catalog record:** Column LOCKMAX of table SYSIBM.SYSTABLESPACE.

If you do not use the default:

- Base your choice upon the results of monitoring applications that use the table space.
- Aim to set the value of LOCKMAX high enough that, when lock escalation occurs, one application already holds so many locks that it significantly interferes with others. For example, if an application holds half a million locks on a table with a million rows, it probably already locks out most other applications. Yet lock escalation can prevent it from potentially acquiring another half million locks.
- If you alter a table space from LOCKSIZE PAGE or LOCKSIZE ANY to LOCKSIZE ROW, consider increasing LOCKMAX to allow for the increased number of locks that applications might require. **GUI**

### Specifying a default value for the LOCKMAX option

The LOCKS PER TABLE(SPACE) field of installation panel DSNTIPI becomes the default value (SYSTEM) for the LOCKMAX clause of the SQL statements CREATE TABLESPACE and ALTER TABLESPACE.

**PSPI** The default value of the LOCKS PER TABLE(SPACE) field is 1000.

- Use the default or, if you are migrating from a previous release of DB2, continue to use the existing value. The value should be less than the value for LOCKS PER USER, unless the value or LOCKS PER USER is 0.
- When you create or alter a table space, especially when you alter one to use row locks, use the LOCKMAX clause explicitly for that table space. **PSPI**

### Specifying lock modes for statements bound with ISOLATION RR or RS

The U LOCK FOR RR/RS option, on installation panel DSNTIPI, determines the mode of the lock first acquired on a row or page, table, partition, or table space for certain statements that are bound with RR or RS isolation.

**PSPI** Those statements include:

#### SELECT with FOR UPDATE OF

The following table shows which mode of lock is held on rows or pages when you specify the SELECT using the WITH RS or WITH RR isolation clause.

*Table 76. Which mode of lock is held on rows or pages when you specify the SELECT using the WITH RS or WITH RR isolation clause*

Option Value	Lock Mode
USE AND KEEP EXCLUSIVE LOCKS	X
USE AND KEEP UPDATE LOCKS	U
USE AND KEEP SHARE LOCKS	S

#### UPDATE and DELETE, without a cursor

The following table shows which mode of lock is held on rows or pages

when you specify an update or a delete without a cursor.

Table 77. Which mode of lock is held on rows or pages when you specify an update or a delete without a cursor

Option Value	Lock Mode
NO (default)	S
YES	U or X

The YES option can avoid deadlocks but it reduces concurrency. 


### Releasing locks for cursors defined WITH HOLD

You can use the RELEASE LOCKS field of installation panel DSNTIP4 to indicate Whether you want DB2 to release a data page or row lock after a COMMIT is issued for cursors defined WITH HOLD.

 This lock is not necessary for maintaining cursor position.


The default value for this option is YES. It causes DB2, at commit time, to release the data page or row lock for the row on which the cursor is positioned. This lock is unnecessary for maintaining cursor position.

To improve concurrency:

- Specify YES for the value of the RELEASE LOCKS field.
- Specify NO only for those cases in which existing applications rely on that particular data lock. 

### Disabling update locks for searched UPDATE and DELETE


You can use the XLKUPDLT option on the DSNTIPI installation panel to disable update lock (ULOCK) on searched UPDATE and DELETE statements.

 When you do that, you do not have to issue a second lock request to upgrade the lock from U to X (exclusive lock) for each updated row. This feature is primarily beneficial in a data sharing environment.

The default value of the XLKUPDLT field is NO


Specify the XLKUPDLT field when most or all searched UPDATE and DELETE statements use an index or can be evaluated by stage 1 processing.

- When you specify NO, DB2 might use lock avoidance when scanning for qualifying rows. When a qualifying row is found, an S lock or a U lock is acquired on the row. The lock on any qualifying row or page is then upgraded to an X lock before performing the update or delete. For stage one non-qualifying rows or pages, the lock is released if ISOLATION(CS) or ISOLATION(RS) is used. For ISOLATION(RR), an S lock is retained on the row or page until the next commit point. This option is best for achieving the highest rates of concurrency.
- When you specify YES, DB2 uses an X lock on rows or pages that qualify during stage 1 processing. With ISOLATION(CS), the lock is released if the row or page is not updated or deleted because it is rejected by stage 2 processing. With ISOLATION(RR) or ISOLATION(RS), DB2 acquires an X lock on all rows that fall within the range of the selection expression. Thus, a lock upgrade request is not needed for qualifying rows though the lock duration is changed from manual to commit. The lock duration change is not as costly as a lock upgrade.

- When you specify TARGET, DB2 treats the rows or pages of the specific table targeted by the insert or update as if the value of XLKUPDLT was YES, and treats rows or pages of other tables referenced by the query, such as those in referenced only in the WHERE clause, as if XLKUPDLT were set to NO. By specifying this blended processing, you can prevent time outs caused by strong lock acquisition of the read-only non-target objects referenced in the update or delete statement. 

## Avoiding locks during predicate evaluation

The EVALUATE UNCOMMITTED field of installation panel DSNTIP4 indicates if predicate evaluation can occur on uncommitted data of other transactions.

 The option applies only to stage 1 predicate processing that uses table access (table space scan, index-to-data access, and RID list processing) for queries with isolation level RS or CS.


Although this option influences whether predicate evaluation can occur on uncommitted data, it does not influence whether uncommitted data is returned to an application. Queries with isolation level RS or CS return only committed data. They never return the uncommitted data of other transactions, even if predicate evaluation occurs on such. If data satisfies the predicate during evaluation, the data is locked as needed, and the predicate is evaluated again as needed before the data is returned to the application.

A value of NO specifies that predicate evaluation occurs only on committed data (or on the uncommitted changes made by the application). NO ensures that all qualifying data is always included in the answer set.

A value of YES specifies that predicate evaluation can occur on uncommitted data of other transactions. With YES, data might be excluded from the answer set. Data that does not satisfy the predicate during evaluation but then, because of undo processing (ROLLBACK or statement failure), reverts to a state that does satisfy the predicate is missing from the answer set. A value of YES enables DB2 to take fewer locks during query processing. The number of locks avoided depends on the following factors:

- The query's access path
- The number of evaluated rows that do not satisfy the predicate
- The number of those rows that are on overflow pages

The default value for this field is NO.

Specify YES to improve concurrency if your applications can tolerate returned data to falsely exclude any data that would be included as the result of undo processing (ROLLBACK or statement failure). 

## Disregarding uncommitted inserts

The SKIP UNCOMMITTED INSERTS field on installation panel DSNTIP8 controls whether uncommitted inserts are ignored.

 DB2 can handle uncommitted inserts in the following ways:

- DB2 can wait until the INSERT transaction completes (commits or rolls back) and return data accordingly. This is the default option, NO.

- DB2 can ignore uncommitted inserts, which in many cases can improve concurrency. This behavior must be specified as YES.

For greater concurrency:

Specify YES for most applications. However, the following examples indicate some instances when the default option, NO, is preferred.

**One transaction creates another:** Suppose that an initial transaction produces a second transaction. The initial transaction passes information to the second transaction by inserting data into a table that the second transaction reads. In this case, NO should be used.

**Data modified by DELETE and INSERT:** Suppose that you frequently modify data by deleting the data and inserting the new image of the data. In such cases that avoid UPDATE statements, the default should be used.

PSPI

---

## Controlling DB2 locks for LOBs

You can take measures to control how DB2 uses locks to process LOB data.

### LOB locks

The purpose of LOB locks is different than that of regular transaction locks. A lock that is taken on a LOB value in a LOB table space is called a *LOB lock*.

#### Relationship between transaction locks and LOB locks

LOB column values are stored in a different table space, called a *LOB table space*, from the values in the base table.

PSPI

An application that reads or updates a row in a table that contains LOB columns obtains its normal transaction locks on the base table. The locks on the base table also control concurrency for the LOB table space. When locks are not acquired on the base table, such as for ISO(UR), DB2 maintains data consistency by using LOB locks.

#### ISOLATION (UR)

When an application is reading rows using uncommitted read, no page or row locks are taken on the base table. Therefore, these readers must take an S LOB lock to ensure that they are not reading a partial LOB or a LOB value that is inconsistent with the base row. This LOB lock is acquired and released immediately, which is sufficient for DB2 to ensure that a complete copy of the LOB

data is ready for subsequent reference. PSPI

#### Hierarchy of LOB locks

Just as page, row, and table space locks have a hierarchical relationship, LOB locks and locks on LOB table spaces have a hierarchical relationship.

PSPI

If the LOB table space is locked with a gross lock, then LOB locks are not acquired. In a data sharing environment, the lock on the LOB table space is used to determine whether the lock on the LOB must be propagated beyond the local

IRLM. PSPI

### Related concepts

“Hierarchy of lock sizes” on page 330

## LOB and LOB table space lock modes

This information describes the modes of LOB locks and LOB table space locks.


### Modes of LOB locks

 The following LOB lock modes are possible:

#### S (SHARE)

The lock owner and any concurrent processes can read, update, or delete the locked LOB. Concurrent processes can acquire an S lock on the LOB.

#### X (EXCLUSIVE)

The lock owner can read or change the locked LOB. Concurrent processes cannot access the LOB. 

### Modes of LOB table space locks

The following lock modes are possible on the LOB table space:

#### IS (INTENT SHARE)

The lock owner can update LOBs to null or zero-length, or read or delete LOBs in the LOB table space. Concurrent processes can both read and change LOBs in the same table space.

#### IX (INTENT EXCLUSIVE)

The lock owner and concurrent processes can read and change data in the LOB table space. The lock owner acquires a LOB lock on any data it accesses.

#### S (SHARE)

The lock owner and any concurrent processes can read and delete LOBs in the LOB table space. An S-lock is only acquired on a LOB in the case of an ISO(UR)

#### SIX (SHARE with INTENT EXCLUSIVE)

The lock owner can read and change data in the LOB table space. If the lock owner is inserting (INSERT or UPDATE), the lock owner obtains a LOB lock. Concurrent processes can read or delete data in the LOB table space (or update to a null or zero-length LOB).

#### X (EXCLUSIVE)


The lock owner can read or change LOBs in the LOB table space. The lock owner does not need LOB locks. Concurrent processes cannot access the data.




## LOB lock and LOB table space lock duration

This information describes the duration of LOB locks and LOB table space locks.

### The duration of LOB locks


 Locks on LOBs are taken when they are needed for an INSERT or UPDATE operations and released immediately at the completion of the operation.

LOB locks are not held for SELECT and DELETE operations. In the case of an application that uses the uncommitted read option, a LOB lock might be acquired, but only to test the LOB for completeness. The lock is released immediately after it is acquired. 

### The duration of LOB table space locks


Locks on LOB table spaces are acquired when they are needed; that is, the ACQUIRE option of BIND has no effect on when the table space lock on the LOB table space is taken. When the table space lock is released is determined by a combination of factors:


- The RELEASE option of bind
- Whether the SQL statement is static or dynamic
- Whether there are held cursors or held locators

When the release option is COMMIT, the lock is released at the next commit point, unless there are held cursors or held locators. If the release option is DEALLOCATE, the lock is released when the object is deallocated (the application ends). The BIND option has no effect on dynamic SQL statements, which always use RELEASE(COMMIT), unless you use dynamic statement caching. 

### When LOB table space locks are not taken



A lock might not be acquired on a LOB table space at all.

 For example, if a row is deleted from a table and the value of the LOB column is null, the LOB table space associated with that LOB column is not locked. DB2 does not access the LOB table space if the application:

- Selects a LOB that is null or zero length
- Deletes a row where the LOB is null or zero length
- Inserts a null or zero length LOB
- Updates a null or zero-length LOB to null or zero-length 

## Controlling the number of LOB locks

You can control the number of LOB locks that are taken.

 LOB locks are counted toward the total number of locks allowed per user. Control this number by the value you specify on the LOCKS PER USER field of installation panel DSNTIPJ. The number of LOB locks that are acquired during a unit of work is reported in IFCID 0020. As with any table space, use the LOCKMAX clause of the CREATE TABLESPACE or ALTER TABLESPACE statement to control the number of LOB locks that are acquired within a particular LOB table space. 

## Explicitly locking LOB tables

The reasons for using LOCK TABLE on an auxiliary table are somewhat different than that for regular tables.

 You might use the LOCK table statement for lob for any of the following reasons:

- You can use LOCK TABLE to control the number of locks acquired on the auxiliary table.
- You can use LOCK TABLE IN SHARE MODE to prevent other applications from inserting LOBs.


With auxiliary tables, LOCK TABLE IN SHARE MODE does not prevent any changes to the auxiliary table. The statement does prevent LOBs from being inserted into the auxiliary table, but it does not prevent deletes. Updates are generally restricted also, except where the LOB is updated to a null value or a zero-length string.

- You can use LOCK TABLE IN EXCLUSIVE MODE to prevent other applications from accessing LOBs.

With auxiliary tables, LOCK TABLE IN EXCLUSIVE MODE also prevents access from uncommitted readers.

- Either statement eliminates the need for lower-level LOB locks. 

## Controlling lock size for LOB table spaces

 The LOCKSIZE TABLE, PAGE, ROW, and XML options are not valid for LOB table spaces. The other options act as follows:


### LOCKSIZE TABLESPACE

A process acquires no LOB locks

### LOCKSIZE ANY

DB2 chooses the size of the lock. For a LOB table space, this is usually LOCKSIZE LOB.

### LOCKSIZE LOB

If LOBs are accessed, a process acquires the necessary LOB table space locks (IS or IX), and might acquire LOB locks. 

---

## Controlling DB2 locks for XML data


You can take measures to control how DB2 uses locks to process XML data.

### XML locks

This information describes the locking that occurs when XML data is accessed.

#### Locks that are acquired for operations on XML data

DB2 stores XML column values in a separate XML table space. An application that reads or updates a row in a table that contains XML columns might use lock avoidance or obtain transaction locks on the base table.

 If an XML column is updated or read, the application might also acquire transaction locks on the XML table space and XML values that are stored in the XML table space. A lock that is taken on an XML value in an XML table space is called an XML lock.

In addition to the XML locks, page locks are also acquired on pages in the XML table space. During insert, update, and delete operations, X page locks are acquired; during fetch processing, including uncommitted read and lock avoidance, S page locks are acquired.

In summary, the main purpose of XML locks is for managing the space used by XML data and to ensure that XML readers do not read partially updated XML data.

The following table shows the relationship between an operation that is performed on XML data and the associated XML table space and XML locks that are acquired.

*Table 78. Locks that are acquired for operations on XML data.* This table does not account for gross locks that can be taken because of the LOCKSIZE TABLESPACE option, the LOCK TABLE statement, or lock escalation.

Operation on XML value	XML table space lock	XML lock	Comment
Read (including UR)	IS	S	Prevents storage from being reused while the XML data is being read
Insert	IX	X	Prevents other processes from seeing partial XML data
Delete	IX	X	To hold space in case the delete is rolled back. Storage is not reusable until the delete is committed and no other readers of the XML data exist.
Update	IS->IX	Two XML locks: an X-lock for the delete and an X-lock for the insert.	Operation is a delete followed by an insert.

**ISOLATION(UR) or ISOLATION(CS):** When an application reads rows using uncommitted read or lock avoidance, no page or row locks are taken on the base table. Therefore, these readers must take an S XML lock to ensure that they are not reading a partial XML value or an XML value that is inconsistent with the base row. When a conditional XML lock cannot be acquired for a SQL statement with UR isolation, DB2 might return no rows and issue an SQL return code +100.

**PSPI**

## Hierarchy of XML locks

Just as page locks (or row locks) and table space locks have a hierarchical relationship, XML locks and locks on XML table spaces have a hierarchical relationship.

**PSPI**

If the XML table space is locked with a gross lock, then XML locks are not acquired. In a data sharing environment, the lock on the XML table space is used to determine whether the lock on the XML must be propagated beyond the local

IRLM. **PSPI**

### Related concepts

“Hierarchy of lock sizes” on page 330

## XML and XML table space lock modes

This information describes the modes of XML locks and XML table space locks

**PSPI**

### S (SHARE)

The lock owner and any concurrent processes can read the locked XML data. Concurrent processes can acquire an S lock on the XML data. The purpose of the S lock is to reserve the space used by the XML data.

### X (EXCLUSIVE)

The lock owner can read, update, or delete the locked XML data. Concurrent processes cannot access the XML data.

**PSPI**

## XML lock and XML table space lock duration

This information describes the duration of XML locks and XML table space locks.

### The duration of XML locks

**PSPI**

X-locks on XML data that are acquired for insert, update, and delete statements are usually released at commit. The duration of XML locks acquired for select statements varies depending upon isolation level, the setting of the CURRENTDATA parameter, and whether work files or multi-row fetch are used.

XML locks acquired for fetch are not normally held until commit and are either released at next fetch or at close cursor. Because XML locks for updating (INSERT, UPDATE, and DELETE) are held until commit and because locks are put on each XML column in both a source table and a target table, it is possible that a statement such as an INSERT with a fullselect that involves XML columns can accumulate many more locks than a similar statement that does not involve XML data. To prevent system problems caused by too many locks, you can:

- Ensure that you have lock escalation enabled for the XML table spaces that are involved in the INSERT. In other words, make sure that LOCKMAX is non-zero for those XML table spaces.
- Alter the XML table space to change the LOCKSIZE to TABLESPACE before executing the INSERT with fullselect.
- Increase the LOCKMAX value on the table spaces involved and ensure that the user lock limit is sufficient.
- Use LOCK TABLE statements to lock the XML table spaces. (Locking the auxiliary table that is contained in a partitioned XML table space locks the XML table space. In the case of segmented, but non-partitioned, XML table spaces, the LOCK TABLE statement locks the table with a gross lock and locks the table space with an intent lock.)

### The duration of XML table space locks

Locks on XML table spaces are acquired when they are needed; that is, the ACQUIRE option of BIND has no effect on when the table space lock on the XML table space is taken. The table space lock is released according to the value specified on the RELEASE option of BIND (except when a cursor is defined WITH HOLD).

**PSPI**

## When XML table space locks are not taken

A lock might not be acquired on an XML table space at all.

**PSPI**

For example, if a row is deleted from a table and the value of the XML column is null, the XML table space associated with that XML column is not locked. DB2 does not access the XML table space if the application:

- Selects an XML value that is null
- Deletes a row where the XML value is null
- Inserts a null XML value
- Updates an XML value to null

**PSPI**

## Controlling the number of XML locks

You can control the number of XML locks that are taken.

**PSPI**

XML locks are counted toward the total number of locks allowed per user.

**PSPI**

Control this number by the value you specify on the LOCKS PER USER field of installation panel DSNTIPJ. The number of XML locks that are acquired during a unit of work is reported in IFCID 0020.

## Controlling XML lock escalation

As with any table space, use the LOCKMAX clause of the ALTER TABLESPACE statement to control the number of locks that are acquired within a particular XML table space before the lock is escalated.

**PSPI**

When the total number of page, row, and XML locks reaches the maximum that you specify in the LOCKMAX clause, the XML locks escalate to a gross lock on the XML table space, and the page, row, and XML locks are released.

Information about XML locks and lock escalation is reported in IFCID 0020.

**PSPI**

## Explicitly locking XML data

You can use a LOCK TABLE statement to explicitly lock XML data

**PSPI**

The reasons for using LOCK TABLE on an auxiliary table are somewhat different than that for regular tables.

- You can use LOCK TABLE to control the number of locks acquired on the auxiliary table.
- You can use LOCK TABLE IN SHARE MODE to prevent other applications from inserting, updating, or deleting XML data.
- You can use LOCK TABLE IN EXCLUSIVE MODE to prevent other applications from accessing XML data.

With auxiliary tables, LOCK TABLE IN EXCLUSIVE MODE also prevents access from uncommitted readers.

- Either statement eliminates the need for lower-level XML locks.

**PSPI**

## Specifying the size of locks for XML data

The LOCKSIZE TABLE, PAGE, ROW, and LOB options are not valid for XML table spaces.

**PSPI** The other options act as follows:

### LOCKSIZE TABLESPACE

A process acquires no XML locks.

### LOCKSIZE ANY

DB2 chooses the size of the lock. For an XML table space, this is usually LOCKSIZE XML.

### LOCKSIZE XML

If XML data is accessed, a process acquires XML locks and the necessary XML table space locks (IS or IX). **PSPI**

---

## Claims and drains for concurrency control

DB2 utilities, commands, and some ALTER, CREATE, and DROP statements can take over access to some objects independently of any transaction locks that are held on the object.

## Objects that are subject to takeover

DB2 utilities, commands, and some ALTER, CREATE, and DROP statements can take over access to certain objects.

**PSPI** The following table and index spaces are subject to takeover by those operations:

- Simple and segmented table spaces
- Partitions of table spaces
- LOB table spaces
- XML table spaces
- Nonpartitioned index spaces
- Partitions of index spaces
- Logical partitions of nonpartitioned indexes **PSPI**

## Claims

A *claim* is a notification to DB2 that an object is being accessed.

**PSPI**

When an application first accesses an object, within a unit of work, it makes a claim on the object. It releases the claim at the next commit point. Unlike a transaction lock, a claim normally does not persist past the commit point. To access the object in the next unit of work, the application must make a new claim.

However, an exception exists. If a cursor defined with the clause WITH HOLD is positioned on the claimed object, the claim is not released at a commit point.

A claim indicates activity on or interest in a particular page set or partition to DB2. Claims prevent drains from occurring until the claim is released.


## Three classes of claims

The following table shows the three classes of claims and the actions that they allow.

*Table 79. Three classes of claims and the actions that they allow*

Claim class	Actions allowed
Write	Reading, updating, inserting, and deleting
Repeatable read	Reading only, with repeatable read (RR) isolation
Cursor stability read	Reading only, with read stability (RS), cursor stability (CS), or uncommitted read (UR) isolation

## Detecting long-running read claims

DB2 issues a warning message and generates a trace record for each time period that a task holds an uncommitted read claim. You can set the length of the period in minutes by using the LRDRTHLD subsystem parameter in the DSNTIPE thread management panel. 

## Drains

A *drain* is the action of taking over access to an object by preventing new claims and waiting for existing claims to be released.

 PSPI


A utility can drain a partition when applications are accessing it. The drain quiesces the applications by allowing each one to reach a commit point, but preventing any of them, or any other applications, from making a new claim. When no more claims exist, the process that drains (the drainer) controls access to the drained object. The applications that were drained can still hold transaction locks on the drained object, but they cannot make new claims until the drainer has finished.

## Drained claim classes

A drainer does not always need complete control. It could drain the following combinations of claim classes:

- Only the write claim class
- Only the repeatable read claim class
- All claim classes

## Example

The CHECK INDEX utility needs to drain only writers from an index space and its associated table space. RECOVER, however, must drain all claim classes from its table space. The REORG utility can drain either writers (with DRAIN WRITERS) or all claim classes (with DRAIN ALL). 

## How DB2 uses drain locks

A *drain lock* prevents conflicting processes from trying to drain the same object at the same time.

**PSPI** Processes that drain only writers can run concurrently; but a process that drains all claim classes cannot drain an object concurrently with any other process. To drain an object, a drainer first acquires one or more drain locks on the object, one for each claim class that it needs to drain. When the locks are in place, the drainer can begin after all processes with claims on the object have released their claims.

A drain lock also prevents new claimers from accessing an object while a drainer has control of it.

## Types of drain locks

Three types of drain locks on an object correspond to the three claim classes:

- Write
- Repeatable read
- Cursor stability read

In general, after an initial claim has been made on an object by a user, no other user in the system needs a drain lock. When the drain lock is granted, no drains on the object are in process for the claim class needed, and the claimer can proceed.

## Exception

The claimer of an object requests a drain lock in two exceptional cases:

- A drain on the object is in process for the claim class needed. In this case, the claimer waits for the drain lock.
- The claim is the first claim on an object before its data set has been physically opened. Here, acquiring the drain lock ensures that no exception states prohibit allocating the data set.

When the claimer gets the drain lock, it makes its claim and releases the lock before beginning its processing. **PSPI**

## Utility locks on the catalog and directory

When the target of a utility is an object in the catalog or directory, such as a catalog table, the utility either drains or claims the object.

**PSPI**

When the target is a user-defined object, the utility claims or drains it but also uses the directory and, perhaps, the catalog; for example, to check authorization. In those cases, the utility uses transaction locks on catalog and directory tables. It acquires those locks in the same way as an SQL transaction does. For information about the SQL statements that require locks on the catalog, see “Contention on the DB2 catalog” on page 337.

## The UTSERIAL lock

Access to the SYSUTILX table space in the directory is controlled by a unique lock called UTSERIAL. A utility must acquire the UTSERIAL lock to read or write in SYSUTILX, whether SYSUTILX is the target of the utility or is used only incidentally. **PSPI**

## Compatibility of utilities

Two utilities are considered *compatible* if they do not need access to the same object at the same time in incompatible modes.

**PSPI** The concurrent operation of two utilities is not typically controlled by either drain locks or transaction locks, but merely by a set of compatibility rules.

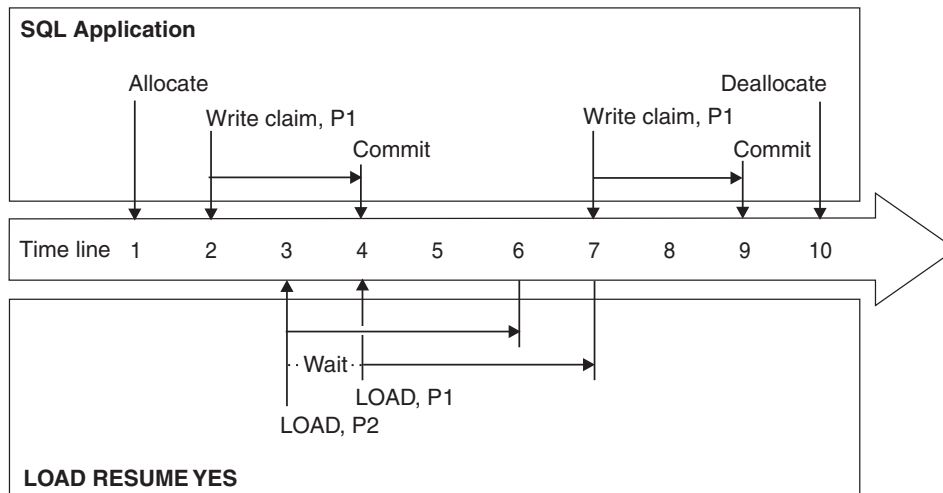
Before a utility starts, it is checked against all other utilities running on the same target object. The utility starts only if all the others are compatible.

The check for compatibility obeys the following rules:

- The check is made for each target object, but only for target objects. Typical utilities access one or more table spaces or indexes, but if two utility jobs use none of the same target objects, the jobs are always compatible.  
An exception is a case in which one utility must update a catalog or directory table space that is not the direct target of the utility. For example, the LOAD utility on a user table space updates DSNDB06.SYSCOPY. Therefore, other utilities that have DSNDB06.SYSCOPY as a target might not be compatible.
- Individual data and index partitions are treated as distinct target objects. Utilities operating on different partitions in the same table or index space are compatible.
- When two utilities access the same target object, their most restrictive access modes determine whether they are compatible. For example, if utility job 1 reads a table space during one phase and writes during the next, it is considered a writer. It cannot start concurrently with utility 2, which allows only readers on the table space. (Without this restriction, utility 1 might start and run concurrently with utility 2 for one phase; but then it would fail in the second phase, because it could not become a writer concurrently with utility 2.)

For details on which utilities are compatible, refer to each utility's description in *DB2 Utility Guide and Reference*.

The following figure illustrates how SQL applications and DB2 utilities can operate concurrently on separate partitions of the same table space.



Time	Event
t1	An SQL application obtains a transaction lock on every partition in the table space. The duration of the locks extends until the table space is deallocated.
t2	The SQL application makes a write claim on data partition 1 and index partition 1.
t3	The LOAD jobs begin draining all claim classes on data partitions 1 and 2 and index partitions 1 and 2. LOAD on partition 2 operates concurrently with the SQL application on partition 1. LOAD on partition 1 waits.
t4	The SQL application commits, releasing its write claims on partition 1. LOAD on partition 1 can begin.
t6	LOAD on partition 2 completes.
t7	LOAD on partition 1 completes, releasing its drain locks. The SQL application (if it has not timed out) makes another write claim on data partition 1.
t10	The SQL application deallocates the table space and releases its transaction locks.

PSPI

Figure 49. SQL and utility concurrency. Two LOAD jobs execute concurrently on two partitions of a table space

## Concurrency during REORG

**PSPI** If you get timeouts or deadlocks when you use REORG with the SHRLEVEL CHANGE option, run the REORG utility with the DRAIN ALL option. The default is DRAIN WRITERS, which is done in the log phase. The specification of DRAIN ALL indicates that both writers and readers are drained when the MAXRO threshold is reached. Consider the DRAIN ALL option in environments where a lot of update activity occurs during the log phase. With this specification, no subsequent drain is required in the switch phase. **PSPI**

## Utility operations with nonpartitioned indexes

In a nonpartitioned index, either a partitioning index or a secondary index, an entry can refer to any partition in the underlying table space.

**PSPI** DB2 can process a set of entries of a nonpartitioned index that all refer to a single partition and achieve the same results as for a partition of a partitioned index. (Such a set of entries is called a *logical partition* of the nonpartitioned index.)

Suppose that two LOAD jobs execute concurrently on different partitions of the same table space. When the jobs proceed to build a partitioned index, either a partitioning index or a secondary index, they operate on different partitions of the index and can operate concurrently. Concurrent operations on different partitions are possible because the index entries in an index partition refer only to data in the corresponding data partition for the table.

Utility processing can be more efficient with partitioned indexes because, with the correspondence of index partitions to data partitions, they promote partition-level independence. For example, the REORG utility with the PART option can run faster and with more concurrency when the indexes are partitioned. REORG rebuilds the parts for each partitioned index during the BUILD phase, which can increase parallel processing and reduce the lock contention of nonpartitioned indexes.

Similarly, for the LOAD PART and REBUILD INDEX PART utilities, the parts for each partitioned index can be built in parallel during the BUILD phase, which reduces lock contention and improves concurrency. The LOAD PART utility also processes partitioned indexes with append logic, instead of the insert logic that it uses to process nonpartitioned indexes, which also improves performance. **PSPI**

---

## Chapter 16. Programming for parallel processing

You can significantly reduce the response time for data or processor-intensive queries by taking advantage of the ability of DB2 to initiate multiple parallel operations when it accesses data from a table or index in a partitioned table space.

---

### Parallel processing

DB2 can initiate multiple parallel operations when it accesses data from a table or index in a partitioned table space.

Query *I/O parallelism* manages concurrent I/O requests for a single query, fetching pages into the buffer pool in parallel. This processing can significantly improve the performance of I/O-bound queries. I/O parallelism is used only when one of the other parallelism modes cannot be used.

Query *CP parallelism* enables true multitasking within a query. A large query can be broken into multiple smaller queries. These smaller queries run simultaneously on multiple processors accessing data in parallel. This reduces the elapsed time for a query.

To expand even farther the processing capacity available for processor-intensive queries, DB2 can split a large query across different DB2 members in a data sharing group. This is known as *Sysplex query parallelism*.

DB2 can use parallel operations for processing the following types of operations:

- Static and dynamic queries
- Local and remote data access
- Queries using single table scans and multi-table joins
- Access through an index, by table space scan or by list prefetch
- Sort

### Parallelism for partitioned and nonpartitioned table spaces

Parallel operations usually involve at least one table in a partitioned table space. Scans of large partitioned table spaces have the greatest performance improvements where both I/O and central processor (CP) operations can be carried out in parallel.

Both partitioned, nonpartitioned, and partition-by-growth table spaces can take advantage of query parallelism. Parallelism is enabled to include non-clustering indexes. Thus, table access can be run in parallel when the application is bound with DEGREE (ANY) and the table is accessed through a non-clustering index.

### Methods of parallel processing

The figures in this topic show how the parallel methods compare with sequential prefetch and with each other.

All three techniques assume access to a table space with three partitions, P1, P2, and P3. The notations P1, P2, and P3 are partitions of a table space. R1, R2, R3, and so on, are requests for sequential prefetch. The combination P2R1, for example, means the first request from partition 2.

## Sequential processing

The figure below shows sequential processing. With *sequential processing*, DB2 takes the 3 partitions in order, completing partition 1 before starting to process partition 2, and completing 2 before starting 3. Sequential prefetch allows overlap of CP processing with I/O operations, but I/O operations do not overlap with each other. In the example in the following figure, a prefetch request takes longer than the time to process it. The processor is frequently waiting for I/O.

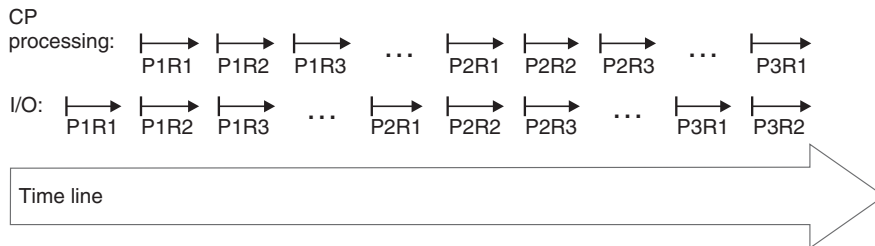


Figure 50. CP and I/O processing techniques. Sequential processing.

## Parallel I/O

The following figure shows parallel I/O operations. With *parallel I/O*, DB2 prefetches data from the 3 partitions at one time. The processor processes the first request from each partition, then the second request from each partition, and so on. The processor is not waiting for I/O, but there is still only one processing task.

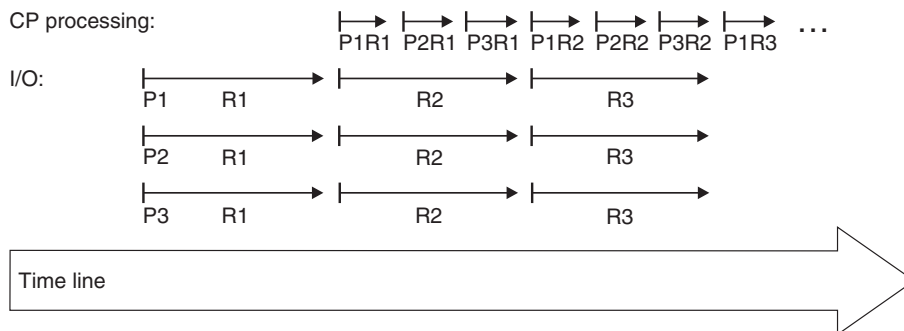


Figure 51. CP and I/O processing techniques. Parallel I/O processing.

## Parallel CP processing and sysplex query parallelism

The following figure shows parallel CP processing. With *parallel CP processing*, DB2 can use multiple parallel tasks to process the query. Three tasks working concurrently can greatly reduce the overall elapsed time for data-intensive and processor-intensive queries. The same principle applies for *Sysplex query parallelism*, except that the work can cross the boundaries of a single CPC.

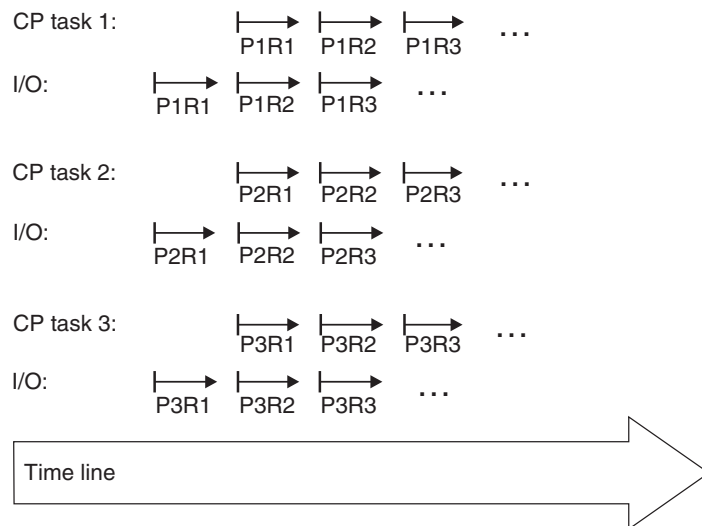


Figure 52. CP and I/O processing techniques. Query processing using CP parallelism. The tasks can be contained within a single CPC or can be spread out among the members of a data sharing group.

## Queries that are most likely to take advantage of parallel operations

### GUIP

Queries that can take advantage of parallel processing are those in which:

- DB2 spends most of the time fetching pages—an I/O-intensive query  
A typical I/O-intensive query is something like the following query, assuming that a table space scan is used on many pages:

```
SELECT COUNT(*) FROM ACCOUNTS
WHERE BALANCE > 0 AND
DAYS_OVERDUE > 30;
```

### GUIP

- DB2 spends a lot of processor time and also, perhaps, I/O time, to process rows for certain types of queries. Those queries include:

#### Queries with intensive data scans and high selectivity

Those queries involve large volumes of data to be scanned but relatively few rows that meet the search criteria.

#### Queries that contain aggregate functions

Column functions (such as MIN, MAX, SUM, AVG, and COUNT) usually involve large amounts of data to be scanned but return only a single aggregate result.

#### Queries that access long data rows

Those queries access tables with long data rows, and the ratio of rows per page is very low (one row per page, for example).

#### Queries that require large amounts of central processor time

Those queries might be read-only queries that are complex, data-intensive, or that involve a sort.

A typical processor-intensive query is something like:

```
SELECT MAX(QTY_ON_HAND) AS MAX_ON_HAND,
AVG(PRICE) AS AVG_PRICE,
AVG(DISCOUNTED_PRICE) AS DISC_PRICE,
SUM(TAX) AS SUM_TAX,
SUM(QTY_SOLD) AS SUM_QTY_SOLD,
```

```

SUM(QTY_ON_HAND - QTY_BROKEN) AS QTY_GOOD,
AVG(DISCOUNT) AS AVG_DISCOUNT,
ORDERSTATUS,
COUNT(*) AS COUNT_ORDERS
FROM ORDER_TABLE
WHERE SHIPPER = 'OVERNIGHT' AND
SHIP_DATE < DATE('2006-01-01')
GROUP BY ORDERSTATUS
ORDER BY ORDERSTATUS;

```

## Terminology

When the term *task* is used with information about parallel processing, the context should be considered. For parallel query CP processing or Sysplex query parallelism, a task is an actual z/OS execution unit used to process a query. For parallel I/O processing, a task simply refers to the processing of one of the concurrent I/O streams.

A *parallel group* is the term used to name a particular set of parallel operations (parallel tasks or parallel I/O operations). A query can have more than one parallel group, but each parallel group within the query is identified by its own unique ID number.

The *degree of parallelism* is the number of parallel tasks or I/O operations that DB2 determines can be used for the operations on the parallel group. The maximum of parallel operations that DB2 can generate is 254. However, for most queries and DB2 environments, DB2 chooses a lower number.

You might need to limit the maximum number further because more parallel operations consume processor, real storage, and I/O resources. If resource consumption is high in your parallelism environment, use the MAX DEGREE field on installation panel DSNTIP4 to explicitly limit the maximum number of parallel operations that DB2 generates, as explain in “Enabling parallel processing” on page 398.

In a parallel group, an *originating task* is the TCB (SRB for distributed requests) that coordinates the work of all the *parallel tasks*. Parallel tasks are executable units composed of special SRBs, which are called *preemptable* SRBs.

With preemptable SRBs, the z/OS dispatcher can interrupt a task at any time to run other work at the same or higher dispatching priority. For non-distributed parallel work, parallel tasks run under a type of preemptable SRB called a *client* SRB, which lets the parallel task inherit the importance of the originating address space. For distributed requests, the parallel tasks run under a preemptable SRB called an *enclave* SRB.

---

## Partitioning for optimal parallel performance

The following are general considerations for how to partition data for the best performance when using parallel processing. Bear in mind that DB2 does not always choose parallelism, even if you partition the data.

This exercise assumes the following:

- You have narrowed the focus to a few, critical queries that are running sequentially. It is best to include a mix of I/O-intensive and processor-intensive queries into this initial set. You know how long those queries take now and

what your performance objectives for those queries are. Although tuning for one set of queries might not work for all queries, overall performance and throughput can be improved.

- You are optimizing for query-at-a-time operations, and you want a query to make use of all the processor and I/O resources available to it.

When running many queries at the same time, you might need to increase the number of partitions and the amount of processing power to achieve similar elapsed times.

This information guides you through the following analyses:

1. Determining the nature of the query (what balance of processing and I/O resources it needs)
2. Determining how many partitions the table space should have to meet your performance objective, number based on the nature of the query and on the processor and I/O configuration at your site

## Determining if a query is I/O- or processor-intensive

How DB2 can best take advantage of parallel processing for a particular query depends upon whether the query is I/O or processor intensive.

To determine if your sequential queries are I/O or processor-intensive:

Examine the DB2 accounting reports:

- If the “other read I/O time” is close to the total query elapsed time, then the query is I/O-intensive. “Other read I/O time” is the time that DB2 is waiting for pages to be read in to the buffer pools.
- If “CPU time” is close to the total query elapsed time, then the query is processor-intensive.
- If the processor time is somewhere between 30 and 70 percent of the elapsed time, then the query is pretty well-balanced in terms of CPU and I/O.

## Determining the number of partitions for parallel processing

You can calculate the number of partitions that will enable your queries to best take advantage of parallel processing.

This information provides general guidance for determining the number of partitions. However, you must take into account the I/O subsystem, the nature of the queries that you run, and plan for the data to grow.

If your physical and logical design are not closely tied together, and you can specify any number of partitions, immediately specifying more partitions than you need causes no harm. However, you should start with a reasonable number of partitions because you can always add more partitions later with the ALTER TABLESPACE statement.

You can also create *partition-by-growth* table spaces, which begin as a single-partition table spaces and automatically add partitions as needed to accommodate data growth. Consider creating a partition by growth table space in cases such as a table space with a single table that is expected to become larger than 64 GB, and which does not include a suitable partitioning key.

Consider too the operational complexity of managing many partitions. This complexity might not be as much of an issue at sites that use tools, such as the DB2 Automated Utilities Generator and job scheduler.

In general, the number of partitions falls in a range between the number of CPs and the maximum number of I/O paths to the data. When determining the number of partitions that use a mixed set of processor- and I/O-intensive queries, always choose the largest number of partitions in the range you determine.

- For processor-intensive queries, specify, at a minimum, a number that is equal to the number of CPs in the system that you want to use for parallelism, whether you have a single CPC or multiple CPCs in a data sharing group. If the query is processor-intensive, it can use all CPs available in the system. If you plan to use Sysplex query parallelism, then choose a number that is close to the total number of CPs (including partial allocation of CPs) that you plan to allocate for decision support processing across the data sharing group. Do not include processing resources that are dedicated to other, higher priority, work.
- For I/O-intensive queries:
  1. Calculate the ratio of elapsed time to processor time.
  2. Multiply that ratio by the number of processors allocated for decision support processing.
  3. Round up the resulting number to determine how many partitions you can use to the best advantage, assuming that these partitions can be on separate devices and have adequate paths to the data.

This calculation also assumes that you have adequate processing power to handle the increase in partitions. (Which might not be much of an issue with an extremely I/O-intensive query.)

By partitioning the amount indicated previously, the query is brought into balance by reducing the I/O wait time. If the number of partitions is less than the number of CPs available on your system, increase this number close to the number of CPs available. By doing so, other queries that read this same table, but that are more processor-intensive, can take advantage of the additional processing power.

**Example:** Suppose that you have a 10-way CPC and the calculated number of partitions is five. Instead of limiting the table space to five partitions, use 10, to equal the number of CPs in the CPC.

## Example configurations for an I/O-intensive query

If the I/O cost of your queries is about twice as much as the processing cost, the optimal number of partitions when run on a 10-way processor is 20 (2 \* number of processors). The figure below shows an I/O configuration that minimizes the elapsed time and allows the CPC to run at 100% busy. It assumes the suggested guideline of four devices per control unit and four channels per control unit.<sup>1</sup>

---

1. A lower-cost configuration could use as few as two to three channels per control unit shared among all controllers using an ESCON director. However, using four paths minimizes contention and provides the best performance. Paths might also need to be taken offline for service.

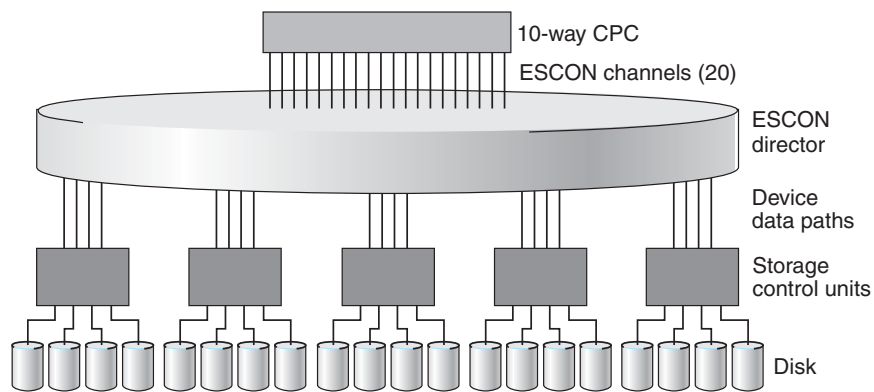


Figure 53. I/O configuration that maximizes performance for an I/O-intensive query

## Working with a table space that is already partitioned

You can examine an existing partitioned table space to determine whether parallel processing can be improved.

Assume that a table space already has 10 partitions and a particular query uses CP parallelism on a 10-way CPC. When you add “other read I/O wait time” (from accounting class 3) and processing time (from accounting class 2), you determine that I/O cost is three times more than the processing cost. In this case, the optimal number of partitions is 30 (three times more I/O paths). However, if you can run on a data sharing group and you add another DB2 subsystem to the group that is running on a 10-way CPC, the I/O configuration that minimizes the elapsed time and allows both CPCs to run at 100% would be 60 partitions.

## Making the partitions the same size

The degree of parallelism is influenced by the size of the largest physical partition.

In most cases, DB2 divides the table space into logical pieces, called *work ranges* to differentiate these from physical pieces, based on the size of the largest physical partition of a given table. Suppose that a table consists of 10 000 pages and 10 physical partitions, the largest of which is 5000 pages. DB2 is most likely to create only two work ranges, and the degree of parallelism would be 2. If the same table has evenly sized partitions of 1000 pages each and the query is I/O-intensive, then ten logical work ranges might be created. This example would result in a degree of parallelism of 10 and reduced elapsed time.

DB2 tries to create equal work ranges by dividing the total cost of running the work by the logical partition cost. This division often has some left over work. In this case, DB2 creates an additional task to handle the extra work, rather than making all the work ranges larger, which would reduce the degree of parallelism.

To rebalance partitions that have become skewed:

Reorganize the table space, and specify the REBALANCE keyword on the REORG utility statement.

## Working with partitioned indexes

The degree of parallelism for accessing partitioned indexes depends on the nature of the query and on the processor and I/O configuration at your site.

For an I/O-intensive query, the degree of a parallelism for access to a partitioned index depends on the number of index partitions that are referenced, whereas the degree of parallelism for access to a nonpartitioned index depends on the number of CPs in the system. For a processor-intensive query, the degree of parallelism for both partitioned and nonpartitioned indexes is influenced by the number of CPs in the system.

---

## Enabling parallel processing

Queries cannot take advantage of parallelism unless you enable parallel processing.

**Prerequisite:** DB2 must be running on a central processor complex that contains two or more tightly coupled processors (sometimes called central processors, or CPs). If only one CP is online when the query is bound, DB2 considers only parallel I/O operations.

DB2 also considers only parallel I/O operations if you declare a cursor WITH HOLD and bind with isolation RR or RS.

**PSPI** To enable parallel processing:


- For static SQL, specify DEGREE(ANY) on BIND or REBIND.
  - **GUPI** You can set the special register with the following SQL statement:  
`SET CURRENT DEGREE='ANY';`

**GUPI**

- You can also change the special register default from 1 to ANY for the entire DB2 subsystem by modifying the CURRENT DEGREE field on installation panel DSNTIP4.

This bind option affects static SQL only and does not enable parallelism for dynamic statements.

- For dynamic SQL, set the CURRENT DEGREE special register to 'ANY'. Setting the special register affects dynamic statements only. It has no effect on your static SQL statements. You should also make sure that parallelism is not disabled for your plan, package, or authorization ID in the RLST.
- If you bind with isolation CS, choose also the option CURRENTDATA(NO), if possible. This option can improve performance in general, but it also ensures that DB2 considers parallelism for ambiguous cursors. If you bind with CURRENTDATA(YES) and DB2 cannot tell if the cursor is read-only, DB2 does not consider parallelism. When a cursor is read-only, it is recommended that you specify the FOR FETCH ONLY or FOR READ ONLY clause on the DECLARE CURSOR statement to explicitly indicate that the cursor is read-only.
- Specify a virtual buffer pool parallel sequential threshold (VPPSEQT) value that is large enough to provide adequate buffer pool space for parallel processing. If you enable parallel processing when DB2 estimates a given query's I/O and central processor cost is high, multiple parallel tasks can be activated if DB2 estimates that elapsed time can be reduced by doing so.

- For parallel sorts, allocate sufficient work files to maintain performance. DB2 also considers only parallel I/O operations if you declare a cursor WITH HOLD and bind with isolation RR or RS.
- For complex queries, run the query in parallel within a member of a data sharing group. With Sysplex query parallelism, use the power of the data sharing group to process individual complex queries on many members of the data sharing group.
- Limit the degree of parallelism. If you want to limit the maximum number of parallel tasks that DB2 generates, you can use the MAX DEGREE field on installation panel DSNTIP4. Changing MAX DEGREE, however, is not the way to turn parallelism off. You use the DEGREE bind parameter or CURRENT DEGREE special register to turn parallelism off. 

## When parallelism is not used

Parallelism is not used for all queries; for some access paths, incurring parallelism overhead makes no sense.

For example, if you are selecting from a temporary table, parallelism is not used. Check the following table to determine whether your query uses any of the access paths that do not allow parallelism.

Table 80. Checklist of parallel modes and query restrictions

If query uses this...	Is parallelism allowed?			Comments
	I/O	CP	Sysplex	
Access via RID list (list prefetch and multiple index access)	Yes	Yes	No	Indicated by an "L" in the PREFETCH column of PLAN_TABLE, or an M, MX, MI, or MQ in the ACESSTYPE column of PLAN_TABLE.
Queries that return LOB values	Yes	Yes	No	
Merge scan join on more than one column	Yes	Yes	Yes	
Queries that qualify for direct row access	No	No	No	Indicated by D in the PRIMARY_ACCESS_TYPE column of PLAN_TABLE
Materialized views or materialized nested table expressions at reference time	No	No	No	
EXISTS within WHERE predicate	No	No	No	
Security label column on table	Yes	Yes	No	
An XML data type	No	No	No	
Multiple index access to return a DOCID list	No	No	No	Indicated by a DX, DI, or DU in the ACESSTYPE column of PLAN_TABLE

## DB2 avoids certain hybrid joins when parallelism is enabled

To ensure that you can take advantage of parallelism, DB2 does not pick one type of hybrid join (SORTN\_JOIN=Y) when the plan or package is bound with CURRENT DEGREE=ANY or if the CURRENT DEGREE special register is set to 'ANY'.



---

## Chapter 17. Tuning and monitoring in a distributed environment

These topics describe some ways that you can tune your systems and applications that use DRDA or DB2 private protocol for distributed data access.

---

### Remote access types: DRDA and private protocol

DB2 supports two different types of remote access between the requesting relational database management system (DBMS) and the serving relational database management system. The two types of access are *DRDA access* and *DB2 private protocol*.

When three-part named objects are referenced (or aliases for three-part name objects are referenced), DB2 chooses between the two connection types based on the bind option that you choose (or the default protocol set at your site).

**Important:** Use DRDA for new applications, and migrate existing private protocol applications to DRDA. No enhancements are planned for private protocol.

#### Characteristics of DRDA

With DRDA, the application can remotely bind packages and can execute packages of static or dynamic SQL that have previously been bound at that location. DRDA has the following characteristics and benefits:

- With DRDA access, an application can access data at any server that supports DRDA, not just a DB2 server on a z/OS operating system.
- DRDA supports all SQL features, including user-defined functions, LOBs, stored procedures, and XML data.
- DRDA can avoid multiple binds and minimize the number of binds that are required.
- DRDA supports multiple-row FETCH.

DRDA is the preferred method for remote access with DB2.

#### Characteristics of DB2 private protocol

Private protocol is an older method for remote access. It can be used only between DB2 subsystems and only over a SNA network. Private protocol has not been enhanced to support many new SQL features. Because of these limitations, it is highly recommended that you migrate to DRDA protocol for communicating between DB2 subsystems.



---

## Chapter 18. Tuning distributed applications

A query that is sent to a remote system can sometimes take longer to execute than the same query, accessing tables of the same size, on the local DB2 subsystem.

The principal reasons for this potential increase in execution time are:

- The time required to send messages across the network
- Overhead processing, including startup and communication subsystem session management

Some aspects of overhead processing, for instance, network processing, are not under DB2 control.

Monitoring and tuning performance in a distributed environment is a complex task that requires knowledge of several products. Some guidelines follow for improving the performance of distributed applications.

---

### Application and requesting systems

Minimizing the number of messages sent between the requester and the server is a primary way to improve performance.

#### **BIND options for distributed applications**

In many cases, certain bind options can improve the performance of SQL statements that run as part of distributed applications.

Consider using the following bind options to improve performance:

- Use the DEFER(PREPARE) bind option, which can reduce the number of messages that must be sent back and forth across the network.
- Bind application plans and packages with ISOLATION(CS) to reduce contention and message overhead.

#### **SQL statement options for distributed applications**

In many cases, you can use certain strategies to improve the performance of SQL statements that run on distributed systems.

Such strategies include:

##### **Committing frequently**

Commit frequently to avoid holding resources at the server.

##### **Using fewer SQL statements**

Avoid using several SQL statements when one well-tuned SQL statement can retrieve the desired results. Alternatively, put your SQL statements in a stored procedure, issue your SQL statements at the server through the stored procedure, and return the result. Using a stored procedure creates only one send and receive operation (for the CALL statement) instead of a potential send and receive operation for each SQL statement.

Depending on how many SQL statements are in your application, using stored procedures can significantly decrease your elapsed time and might decrease your processor costs.

### Using the RELEASE statement and the DISCONNECT(EXPLICIT) bind option

The RELEASE statement minimizes the network traffic that is needed to release a remote connection at commit time. For example, if the application has connections to several different servers, specify the RELEASE statement when the application has completed processing for each server. The RELEASE statement does not close cursors, release any resources, or prevent further use of the connection until the COMMIT is issued. It just makes the processing at COMMIT time more efficient.

The bind option DISCONNECT(EXPLICIT) destroys all remote connections for which RELEASE was specified.

### Using the COMMIT ON RETURN YES clause

**GUIP** Consider using the COMMIT ON RETURN YES clause of the CREATE PROCEDURE statement to indicate that DB2 should issue an implicit COMMIT on behalf of the stored procedure upon return from the CALL statement. Using the clause can reduce the length of time locks are held and can reduce network traffic. With COMMIT ON RETURN YES, any updates made by the client before calling the stored procedure are committed with the stored procedure changes. **GUIP**

### Setting CURRENT RULES special register to DB2

**GUIP** When requesting LOB data, set the CURRENT RULES special register to DB2 instead of to STD before performing a CONNECT. A value of DB2, which is the default, can offer performance advantages. When a DB2 for z/OS server receives an OPEN request for a cursor, the server uses the value in the CURRENT RULES special register to determine whether the application intends to switch between LOB values and LOB locator values when fetching different rows in the cursor. If you specify a value of DB2 for CURRENT RULES, the application indicates that the first FETCH request specifies the format for each LOB column in the answer set and that the format does not change in a subsequent FETCH request. However, if you set the value of CURRENT RULES to STD, the application intends to fetch a LOB column into either a LOB locator host variable or a LOB host variable.

Although a value of STD for CURRENT RULES gives you more programming flexibility when you retrieve LOB data, you can get better performance if you use a value of DB2. With the STD option, the server does not block the cursor, while with the DB2 option it might block the cursor where it is possible to do so. For more information, see “LOB and

XML data and its effect on block fetch for DRDA” on page 406. **GUIP**

## Block fetching result sets

*Block fetch* can significantly decrease the number of messages sent across the network. Block fetch is used only with cursors that do not update or delete data.

With block fetch, DB2 groups the rows that are retrieved by an SQL query into as large a “block” of rows as can fit in a message buffer. DB2 then transmits the block over the network, without requiring a separate message for each row.

DB2 can use two different types of block fetch:

- Limited block fetch
- Continuous block fetch

Both types of block fetch are used for DRDA and private protocol, but the implementation of continuous block fetch for DRDA is slightly different than that for private protocol.

### Continuous block fetch

In terms of response times, continuous block fetch is most efficient for larger result sets because fewer messages are transmitted and because overlapped processing is performed at the requester and the server.

However, continuous block fetch uses more networking resources than limited block fetch. When networking resources are critical, use limited block fetch to run applications.

The requester can use both forms of blocking at the same time and with different servers.

If an application is doing read-only processing and can use continuous block fetch, the sequence goes like this:

1. The requester sends a message to open a cursor and begins fetching the block of rows at the server.
2. The server sends back a block of rows and the requester begins processing the first row.
3. The server continues to send blocks of rows to the requester, without further prompting. The requester processes the second and later rows as usual, but fetches them from a buffer on the requester's system.

For private protocol, continuous block fetch uses one conversation for each open cursor. Having a dedicated conversation for each cursor allows the server to continue sending until all the rows are returned.

For DRDA, only one conversation is used, and it must be made available to the other SQL statements that are in the application. Thus, the server usually sends back a subset of all the rows. The number of rows that the server sends depends on the following factors:

- The size of each row
- The number of extra blocks that are requested by the requesting system compared to the number of extra blocks that the server returns

For a DB2 for z/OS requester, the EXTRA BLOCKS REQ field on installation panel DSNTIP5 determines the maximum number of extra blocks requested. For a DB2 for z/OS server, the EXTRA BLOCKS SRV field on installation panel DSNTIP5 determines the maximum number of extra blocks allowed.

**Example:** Suppose that the requester asks for 100 extra query blocks and that the server allows only 50. The server returns no more than 50 extra query blocks. The server might choose to return fewer than 50 extra query blocks for any number of reasons that DRDA allows.

- Whether continuous block fetch is enabled, and the number of extra rows that the server can return if it regulates that number.

To enable continuous block fetch for DRDA and to regulate the number of extra rows sent by a DB2 for z/OS server, you must use the OPTIMIZE FOR n ROWS clause on your SELECT statement. See "Optimizing for very large results sets for DRDA" on page 409 for more information.

If you want to use continuous block fetch for DRDA, have the application fetch all the rows of the cursor before doing any other SQL. Fetching all the rows first

prevents the requester from having to buffer the data, which can consume a lot of storage. Choose carefully which applications should use continuous block fetch for DRDA.

### Limited block fetch

Limited block fetch guarantees the transfer of a minimum amount of data in response to each request from the requesting system.

With limited block fetch, a single conversation is used to transfer messages and data between the requester and server for multiple cursors. Processing at the requester and server is synchronous. The requester sends a request to the server, which causes the server to send a response back to the requester. The server must then wait for another request to tell it what should be done next.

### Block fetch with scrollable cursors for DRDA

When a DB2 for z/OS requester uses a scrollable cursor to retrieve data from a DB2 for z/OS server, the following conditions are true.

- The requester never requests more than 64 rows in a query block, even if more rows fit in the query block. In addition, the requester never requests extra query blocks. This is true even if the setting of field EXTRA BLOCKS REQ in the DISTRIBUTED DATA FACILITY PANEL 2 installation panel on the requester allows extra query blocks to be requested.
- The requester discards rows of the result table if the application does not use those rows.

**GUPI** Example: If the application fetches row  $n$  and then fetches row  $n+2$ , the requester discards row  $n+1$ . **GUPI**

The application gets better performance for a blocked scrollable cursor if it mostly scrolls forward, fetches most of the rows in a query block, and avoids frequent switching between FETCH ABSOLUTE statements with negative and positive values.

- If the scrollable cursor does not use block fetch, the server returns one row for each FETCH statement.

### LOB and XML data and its effect on block fetch for DRDA

For a non-scrollable blocked cursor, the server sends all the non-LOB and non-XML data columns for a block of rows in one message, including LOB locator values.

As each row is fetched by the application, the requester obtains the non-LOB data columns directly from the query block. If the row contains non-null and non-zero length LOB values, those values are retrieved from the server at that time. This behavior limits the impact to the network by pacing the amount of data that is returned at any one time. If all LOB data columns are retrieved into LOB locator host variables or if the row does not contain any non-null or non-zero length LOB columns, then the whole row can be retrieved directly from the query block.

For a scrollable blocked cursor, the LOB data columns are returned at the same time as the non-LOB and non XML data columns. When the application fetches a row that is in the block, a separate message is not required to get the LOB columns.

### Ensuring block fetch

To use either limited or continuous block fetch, DB2 must determine that the cursor is not used for updating or deleting.



The easiest way to indicate that the cursor does not modify data is to add the FOR FETCH ONLY or FOR READ ONLY clause to the query in the DECLARE CURSOR statement as in the following example:

```
EXEC SQL
  DECLARE THISEMP CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT, JOB
    FROM DSN8910.EMP
    WHERE WORKDEPT = 'D11'
    FOR FETCH ONLY
END-EXEC.
```

If you do not use FOR FETCH ONLY or FOR READ ONLY, DB2 still uses block fetch for the query if the following conditions are true:

- The cursor is a non-scrollable cursor, and the result table of the cursor is read-only. This applies to static and dynamic cursors except for read-only views.
- The cursor is a scrollable cursor that is declared as INSENSITIVE, and the result table of the cursor is read-only.
- The cursor is a scrollable cursor that is declared as SENSITIVE, the result table of the cursor is read-only, and the value of bind option CURRENTDATA is NO.
- The result table of the cursor is not read-only, but the cursor is ambiguous, and the value of bind option CURRENTDATA is NO. A cursor is ambiguous when:
  - It is not defined with the clauses FOR FETCH ONLY, FOR READ ONLY, or FOR UPDATE OF.
  - It is not defined on a read-only result table.
  - It is not the target of a WHERE CURRENT clause on an SQL UPDATE or DELETE statement.
  - It is in a plan or package that contains the SQL statements PREPARE or EXECUTE IMMEDIATE.

DB2 triggers block fetch for static SQL only when it can detect that no updates or deletes are in the application. For dynamic statements, because DB2 cannot detect what follows in the program, the decision to use block fetch is based on the declaration of the cursor.

DB2 does not use continuous block fetch if the following conditions are true:

- The cursor is referred to in the statement DELETE WHERE CURRENT OF elsewhere in the program.
- The cursor statement appears that it can be updated at the requesting system. (DB2 does not check whether the cursor references a view at the server that cannot be updated.)

The following three tables summarize the conditions under which a DB2 server uses block fetch.

Table 81 shows the conditions for a non-scrollable cursor.

*Table 81. Effect of CURRENTDATA and cursor type on block fetch for a non-scrollable cursor*

Isolation level	CURRENTDATA	Cursor type	Block fetch
CS, RR, or RS	Yes	Read-only	Yes
		Updatable	No
		Ambiguous	No
	No	Read-only	Yes
		Updatable	No
		Ambiguous	Yes
UR	Yes	Read-only	Yes
	No	Read-only	Yes

Table 82 shows the conditions for a scrollable cursor that is not used to retrieve a stored procedure result set.

*Table 82. Effect of CURRENTDATA and isolation level on block fetch for a scrollable cursor that is not used for a stored procedure result set*

Isolation level	Cursor sensitivity	CURRENTDATA	Cursor type	Block fetch
CS, RR, or RS	INSENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes
	SENSITIVE	Yes	Read-only	No
			Updatable	No
			Ambiguous	No
		No	Read-only	Yes
			Updatable	No
			Ambiguous	Yes
UR	INSENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes
	SENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes

Table 83 shows the conditions for a scrollable cursor that is used to retrieve a stored procedure result set.

*Table 83. Effect of CURRENTDATA and isolation level on block fetch for a scrollable cursor that is used for a stored procedure result set*

Isolation level	Cursor sensitivity	CURRENTDATA	Cursor type	Block fetch
CS, RR, or RS	INSENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes
	SENSITIVE	Yes	Read-only	No
		No	Read-only	Yes

Table 83. Effect of CURRENTDATA and isolation level on block fetch for a scrollable cursor that is used for a stored procedure result set (continued)

Isolation level	Cursor sensitivity	CURRENTDATA	Cursor type	Block fetch
UR	INSENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes
	SENSITIVE	Yes	Read-only	Yes
		No	Read-only	Yes



## Optimizing for very large results sets for DRDA

Enabling a DB2 client to request multiple query blocks on each transmission can reduce network activity and improve performance significantly for applications that use DRDA access to download large amounts of data.

You can specify a large value of *n* in the OPTIMIZE FOR *n* ROWS clause of a SELECT statement to increase the number of DRDA query blocks that a DB2 server returns in each network transmission for a non-scrollable cursor. If *n* is greater than the number of rows that fit in a DRDA query block, OPTIMIZE FOR *n* ROWS lets the DRDA client request multiple blocks of query data on each network transmission instead of requesting a new block when the first block is full. This use of OPTIMIZE FOR *n* ROWS is intended only for applications in which the application opens a cursor and downloads great amounts of data. The OPTIMIZE FOR *n* ROWS clause has no effect on scrollable cursors.

**Recommendation:** Because the application SQL uses only one conversation, do not try to do other SQL work until the entire answer set is processed. If the application issues another SQL statement before the previous statement's answer set has been received, DDF must buffer them in its address space. You can buffer up to 10 MB in this way.

Because specifying a large number of network blocks can saturate the network, limit the number of blocks according to what your network can handle. You can limit the number of blocks used for these large download operations. When the client supports extra query blocks, DB2 chooses the **smallest** of the following values when determining the number of query blocks to send:

- The number of blocks into which the number of rows (*n*) on the OPTIMIZE clause can fit. For example, assume you specify 10000 rows for *n*, and the size of each row that is returned is approximately 100 bytes. If the block size used is 32 KB (32768 bytes), the calculation is as follows:  

$$(10000 * 100) / 32768 = 31 \text{ blocks}$$
- The DB2 server value for the EXTRA BLOCKS SRV field on installation panel DSNTIP5. The maximum value that you can specify is 100.
- The client's extra query block limit, which is obtained from the DRDA MAXBLKEXT parameter received from the client. When DB2 for z/OS acts as a DRDA client, you set this parameter at installation time with the EXTRA BLOCKS REQ field on installation panel DSNTIP5. The maximum value that you can specify is 100. DB2 Connect sets the MAXBLKEXT parameter to -1 (unlimited).

If the client does not support extra query blocks, the DB2 server on z/OS automatically reduces the value of  $n$  to match the number of rows that fit within a DRDA query block.

**Recommendation for cursors that are defined WITH HOLD:** Do not set a large number of query blocks for cursors that are defined WITH HOLD. If the application commits while there are still a lot of blocks in the network, DB2 buffers the blocks in the requester's memory (the *ssnmDIST* address space if the requester is a DB2 for z/OS) before the commit can be sent to the server.

## Optimizing for small results sets for DRDA

When a client does not need all the rows from a potentially large result set, preventing the DB2 server from returning all the rows for a query can reduce network activity and improve performance significantly for DRDA applications.

**GUIP** You can use either the OPTIMIZE FOR  $n$  ROWS clause or the FETCH FIRST  $n$  ROWS ONLY clause of a SELECT statement to limit the number of rows returned to a client program.

**Using OPTIMIZE FOR  $n$  ROWS:** When you specify OPTIMIZE FOR  $n$  ROWS and  $n$  is less than the number of rows that fit in the DRDA query block (default size on z/OS is 32 KB), the DB2 server prefetches and returns only as many rows as fit into the query block. For example, if the client application is interested in seeing only one screen of data, specify OPTIMIZE FOR  $n$  ROWS, choosing a small number for  $n$ , such as 3 or 4. The OPTIMIZE FOR  $n$  ROWS clause has no effect on scrollable cursors.

**Using FETCH FIRST  $n$  ROWS ONLY:** The FETCH FIRST  $n$  ROWS ONLY clause does not affect network blocking. If FETCH FIRST  $n$  ROWS ONLY is specified and OPTIMIZE FOR  $n$  ROWS is not specified, DB2 uses the FETCH FIRST value to optimize the access path. However, DRDA does not consider this value when it determines network blocking.

When both the FETCH FIRST  $n$  ROWS ONLY clause and the OPTIMIZE FOR  $n$  ROWS clause are specified, the value for the OPTIMIZE FOR  $n$  ROWS clause is used for access path selection.

**Example:** Suppose that you submit the following SELECT statement:

```
SELECT * FROM EMP
FETCH FIRST 5 ROWS ONLY
OPTIMIZE FOR 20 ROWS;
```


The OPTIMIZE FOR value of 20 rows is used for network blocking and access path selection.

When you use FETCH FIRST  $n$  ROWS ONLY, DB2 might use a *fast implicit close*. Fast implicit close means that during a distributed query, the DB2 server automatically closes the cursor when it prefetches the  $n$ th row if FETCH FIRST  $n$  ROWS ONLY is specified or when there are no more rows to return. Fast implicit close can improve performance because it can save an additional network transmission between the client and the server.

DB2 uses fast implicit close when the following conditions are true:

- The query uses limited block fetch.
- The query retrieves no LOBs.

- The query retrieves no XML data.
- The cursor is not a scrollable cursor.
- Either of the following conditions is true:
  - The cursor is declared WITH HOLD, and the package or plan that contains the cursor is bound with the KEEP DYNAMIC(YES) option.
  - The cursor is declared WITH HOLD and the DRDA client passes the QRYCLSIMP parameter set to SERVER MUST CLOSE, SERVER DECIDES, or SERVER MUST NOT CLOSE.
  - The cursor is not defined WITH HOLD.

When you use FETCH FIRST *n* ROWS ONLY and DB2 does a fast implicit close, the DB2 server closes the cursor after it prefetches *n* rows, or when there are no more rows. 

## Data encryption security options

Data encryption security options provide added security for the security-sensitive data that an application requests from the system. However, the encryption options can also have a negative impact on performance.

The following encryption options have a larger performance cost than other options:

- Encrypted user ID and encrypted security-sensitive data
- Encrypted user ID, encrypted password, and encrypted security-sensitive data

**Recommendation:** To maximize performance of requester systems, use the minimum level of security that is required by the sensitivity of the data.

---

## Serving system

For access that uses DRDA, the *serving system* is the system on which your remotely bound package executes. For access that uses DB2 private protocol, the serving system is the DB2 system on which the SQL is dynamically executed. .

If you are executing a package on a remote DBMS, then improving performance on the server depends on the nature of the server. If the remote DBMS on which the package executes is another DB2, then the information in “Investigating SQL performance with EXPLAIN” on page 421 is appropriate for access path considerations.

Other considerations that could affect performance on a DB2 server are:

- The maximum number of database access threads that the server allows to be allocated concurrently. (This is the MAX REMOTE ACTIVE field on installation panel DSNTIPE.) A request can be queued while waiting for an available thread. Making sure that requesters commit frequently can let threads be used by other requesters. See “Setting thread limits for database access threads” on page 143 for more information.
- The priority of database access threads on the remote system. A low priority could impede your application’s distributed performance. See “Using z/OS workload management to set performance objectives” on page 147 for more information.
- You can manage IDs through DB2 to avoid RACF calls at the server

When DB2 is the server, it is a good idea to activate accounting trace class 7. This provides accounting information at the package level, which can be very useful in determining performance problems.

---

## **Part 4. Monitoring DB2 for z/OS performance**

Proactive performance monitoring is a key element of maintaining the health of your system.



---

## Chapter 19. Planning for performance monitoring

When you plan to monitor DB2 performance you should consider how to monitor performance continuously, how and when to perform periodic monitoring, how you will monitor exceptions, and the costs associated with monitoring.

Your plan for monitoring DB2 performance should include:

- A master schedule of monitoring. Large batch jobs or utility runs can cause activity peaks. Coordinate monitoring with other operations so that it need not conflict with unusual peaks, unless that is what you want to monitor.
- The kinds of analysis to be performed and the tools to be used. Document the data that is extracted from the monitoring output. These reports can be produced using Tivoli Decision Support for z/OS, IBM Tivoli OMEGAMON XE, other reporting tools, manual reduction, or a program of your own that extracts information from standard reports.
- A list of people who should review the results. The results of monitoring and the conclusions based on them should be available to the user support group and to system performance specialists.
- A strategy for tuning DB2 describes how often changes are permitted and standards for testing their effects. Include the tuning strategy in regular system management procedures.

Tuning recommendations might include generic database and application design changes. You should update development standards and guidelines to reflect your experience and to avoid repeating mistakes.

### Cost factors of performance monitoring

You should consider the following cost factors when planning for performance monitoring and tuning.

- Trace overhead for global, accounting, statistics, audit, and performance traces
- Trace data reduction and reporting times
- Time spent on report analysis and tuning action

#### Related concepts

Chapter 20, “Using tools to monitor performance,” on page 419

#### Related tasks

“Minimizing the use of DB2 traces” on page 46

---

## Continuous performance monitoring

Continuous monitoring watches system throughput, resource usage (processor, I/Os, and storage), changes to the system, and significant exceptions that might affect system performance.

- Try to continually run classes 1, 3, 4, and 6 of the DB2 statistics trace and classes 1 and 3 of the DB2 accounting trace.
- In the data that you collect, look for statistics or counts that differ from past records.
- Pay special attention to peak periods of activity, both of any new application and of the system as a whole

- Run accounting class 2 as well as class 1 to separate DB2 times from application times.

Running with CICS without the open transaction environment (OTE), entails less need to run with accounting class 2. Application and non-DB2 processing take place under the CICS main TCB. Because SQL activity takes place under the SQL TCB, the class 1 and class 2 times are generally close. The CICS attachment work is spread across class 1, class 2, and time spent processing outside of DB2. Class 1 time thus reports on the SQL TCB time and some of the CICS attachment. If you are concerned about class 2 overhead and you use CICS, you can generally run without turning on accounting class 2.

- Statistics and accounting information can be very helpful for application and database designers. Consider putting this information into a performance warehouse so that the data can be analyzed more easily by all the personnel who need the information.

IBM Tivoli OMEGAMON/OMEGAMON XE includes a performance warehouse that allows you to define, schedule, and run processes that:

The data in the performance warehouse can be accessed by any member of the DB2 family or by any product that supports Distributed Relational Database Architecture (DRDA).

---

## Planning for periodic monitoring

Periodic monitoring serves to check system throughput, utilized resources (processor, I/Os, and storage), changes to the system, and significant exceptions that might affect system performance during peak periods when constraints and response-time problems are more pronounced.

A typical periodic monitoring interval of about ten minutes provides information on the workload achieved, resources used, and significant changes to the system. In effect, you are taking “snapshots” at peak loads and under normal conditions.

The current peak is also a good indicator of the future average. You might have to monitor more frequently at first to confirm that expected peaks correspond with actual ones. Do not base conclusions on one or two monitoring periods, but on data from several days representing different periods.

You might notice that subsystem response is becoming increasingly sluggish, or that more applications fail from lack of resources (such as from locking contention or concurrency limits). You also might notice an increase in the processor time DB2 is using, even though subsystem responses seem normal. In any case, if the subsystem continues to perform acceptably and you are not having any problems, DB2 might not need additional tuning.

To monitor peak periods:

Gather information from the different parts of your system, including:

- DB2 for z/OS
- z/OS
- The transaction manager (IMS, CICS, or WebSphere)
- DB2 Connect™
- The network
- Distributed application platforms (such as Windows, UNIX, or Linux)

To compare the different results from each source, monitor each for the same period of time. Because the monitoring tools require resources, you need to consider the overhead for using these tools as well.

**Related tasks**

“Minimizing the use of DB2 traces” on page 46

---

## Detailed performance monitoring

You can add detailed monitoring to periodic monitoring when you discover or suspect a problem. Use it also to investigate areas not covered periodically. To keep the cost of the detailed monitoring low, limit the information to the specific application and data as much as possible.

- If you have a performance problem, first verify that it is not caused by faulty design of an application or database.
- If you believe that the problem is caused by the choice of system parameters, I/O device assignments, or other factors, begin monitoring DB2 to collect data about its internal activity.
- If you have access path problems, use IBM Optimization Service Center for DB2 for z/OS or DB2 EXPLAIN to locate and tune the problems.

**Related concepts**

Chapter 20, “Using tools to monitor performance,” on page 419

“Investigating SQL performance with EXPLAIN” on page 421

---

## Exception performance monitoring

Exception monitoring looks for specific exceptional values or events, such as very high response times or deadlocks. Perform exception monitoring for response-time and concurrency problems.

With IBM Tivoli OMEGAMON XE, exception monitoring is available in both batch reporting and online monitor. For information on how to use exception processing, set exception threshold limits in the threshold limit data set, and use the exception profiling function of OMEGAMON DB2 Performance Expert, see:

- *Using IBM Tivoli OMEGAMON XE on z/OS*
- *OMEGAMON Report Reference*
- *OMEGAMON Monitoring Performance from ISPF*
- *OMEGAMON Monitoring Performance from Performance Expert Client*

**Related concepts**

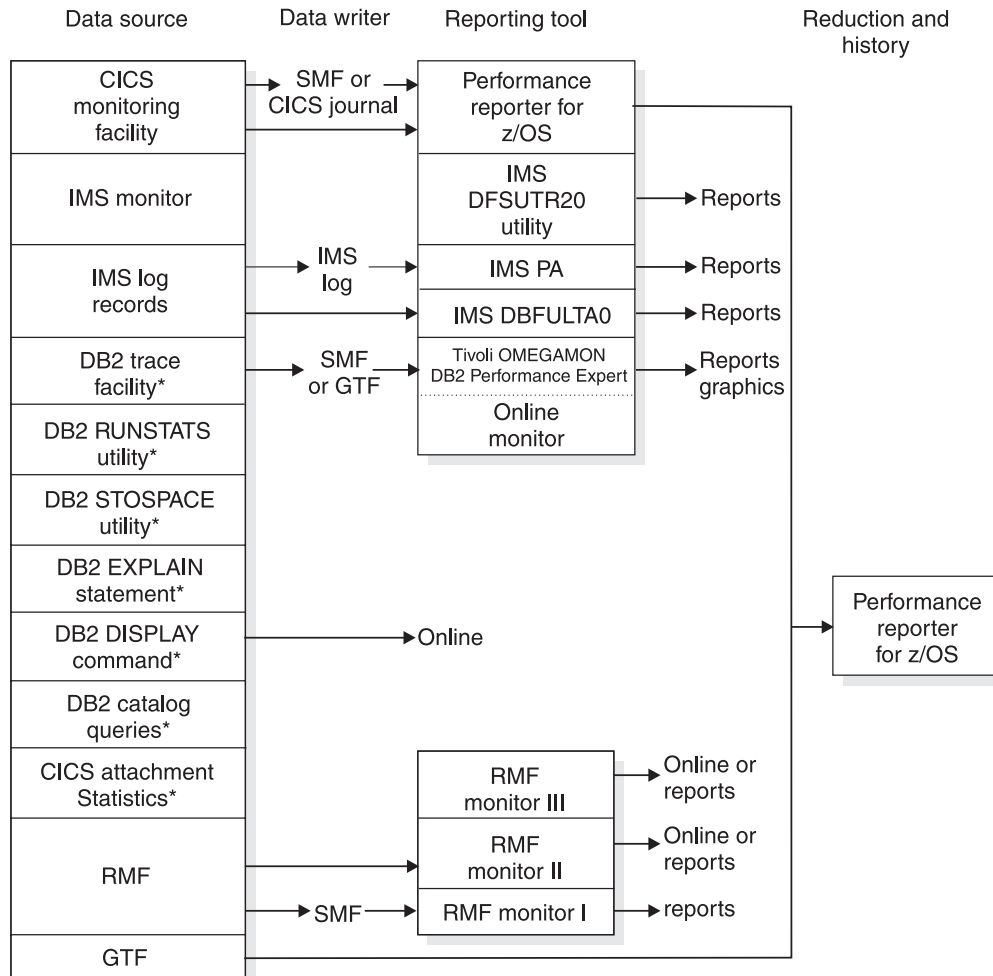
“Scenario for analyzing concurrency” on page 485



## Chapter 20. Using tools to monitor performance

These topics describe the various facilities for monitoring DB2 activity and performance.

The included information covers facilities within the DB2 product as well as tools that are available outside of DB2.



\*Facilities available with the DB2 product

Figure 54. Monitoring tools in a DB2 environment

Table 84 describes these monitoring tools.

Table 84. Monitoring tools in a DB2 environment

Monitoring tool	Description
CICS Attachment Facility statistics	Provide information about the use of CICS threads. This information can be displayed on a terminal or printed in a report.

Table 84. Monitoring tools in a DB2 environment (continued)

Monitoring tool	Description
OMEGAMON CICS Monitoring Facility (CMF)	Provides performance information about each CICS transaction executed. It can be used to investigate the resources used and the time spent processing transactions. Be aware that overhead is significant when CMF is used to gather performance information.
DB2 catalog queries	Help you determine when to reorganize table spaces and indexes. See Chapter 31, “When to reorganize indexes and table spaces,” on page 501.
DB2 Connect	Can monitor and report DB2 server-elapsed time for client applications that access DB2 data. See “Reporting server-elapsed time” on page 609.
DB2 DISPLAY command	Gives you information about the status of threads, databases, buffer pools, traces, allied subsystems, applications, and the allocation of tape units for the archive read process. For information about the DISPLAY BUFFERPOOL command, see “Monitoring and tuning buffer pools using online commands” on page 479. For information about using the DISPLAY command to monitor distributed data activity, see “The DISPLAY command” on page 604
DB2 EXPLAIN statement	Provides information about the access paths used by DB2. See “Investigating SQL performance with EXPLAIN” on page 421.
IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS	A licensed program that integrates the function of DB2 Buffer Pool Analyzer and DB2 Performance Monitor (DB2 PM). OMEGAMON provides performance monitoring, reporting, buffer pool analysis, and a performance warehouse, all in one tool. OMEGAMON monitors all subsystem instances across many different platforms in a consistent way. You can use OMEGAMON to analyze DB2 trace records and optimize buffer pool usage. See “IBM Tivoli OMEGAMON XE” on page 422 for more information.
IBM Tivoli OMEGAMON XE for DB2 Performance Monitor (DB2 PM)	An orderable feature of DB2 that you can use to analyze DB2 trace records. As indicated previously, OMEGAMON includes the function of DB2 PM. OMEGAMON is described under “IBM Tivoli OMEGAMON XE” on page 422.
DB2 RUNSTATS utility	Can report space use and access path statistics in the DB2 catalog. See Chapter 30, “Gathering monitor statistics and update statistics,” on page 499.
DB2 STOSPACE utility	Provides information about the actual space allocated for storage groups, table spaces, table space partitions, index spaces, and index space partitions.
DB2 trace facility	Provides DB2 performance and accounting information. It is described under Chapter 21, “Using DB2 Trace to monitor performance,” on page 429.
Generalized Trace Facility (GTF)	A z/OS service aid that collects information to analyze particular situations. GTF can also be used to analyze seek times and Supervisor Call instruction (SVC) usage, and for other services. See Chapter 23, “Recording GTF trace data,” on page 437 for more information.
IMS DFSUTR20 utility	A print utility for IMS Monitor reports.
IMS Fast Path Log Analysis utility (DBFULTA0)	An IMS utility that provides performance reports for IMS Fast Path transactions.

Table 84. Monitoring tools in a DB2 environment (continued)

Monitoring tool	Description
OMEGAMON IMS Performance Analyzer (IMS PA)	A separately licensed program that can be used to produce transit time information based on the IMS log data set. It can also be used to investigate response-time problems of IMS DB2 transactions.
Resource Measurement Facility (RMF)	An optional feature of z/OS that provides system-wide information on processor utilization, I/O activity, storage, and paging. RMF provides for three basic types of sessions: Monitor I, Monitor II, and Monitor III. Monitor I and Monitor II sessions collect and report data primarily about specific system activities. Monitor III sessions collect and report data about overall system activity in terms of work flow and delay.
System Management Facility (SMF)	A z/OS service aid used to collect information from various z/OS subsystems. This information is dumped and reported periodically, such as once a day. Refer to Chapter 22, "Recording SMF trace data," on page 435 for more information.
Tivoli Decision Support for z/OS	A licensed program that collects SMF data into a DB2 database and allows you to create reports on the data. See "Tivoli Decision Support for z/OS" on page 423.

## Investigating SQL performance with EXPLAIN

DB2 EXPLAIN is a monitoring tool that captures detailed information about the performance of SQL plans, packages, and statements.

**PSPI** By using DB2 EXPLAIN you can capture and analyze the following types of information:

- A plan, package, or SQL statement when it is bound. The output appears in a table that you create called PLAN\_TABLE, which is also called a *plan table*. For experienced users, you can use PLAN\_TABLE to give optimization hints to DB2.
- An estimated cost of executing an SQL SELECT, INSERT, UPDATE, or DELETE statement. The output appears in a table that you create called DSN\_STATEMENT\_TABLE, which is also called a *statement table*.
- User-defined functions referred to in the statement, including the specific name and schema. The output appears in a table that you create called DSN\_FUNCTION\_TABLE, which is also called a *function table*.
- Execution of SQL statements and groups of SQL statements. By creating a profile table, you can monitor statements, monitor exceptions such as RLF constraint violations, obtain snapshot execution reports, and specify statements to be monitored and explained later.

**Related tools and accessories:** The following tools can also help you to monitor, analyze, and tune SQL performance:

### Optimization Service Center for DB2 for z/OS

IBM Optimization Service Center for DB2 for z/OS, an order-able feature and part of IBM DB2 Accessories Suite for z/OS, is a workstation tool that helps you tune your queries and query workloads. You can quickly get customized tuning recommendations or perform your own in-depth

analysis by graphing the access plan for a query. You can perform the following key tasks from the Optimization Service Center:

- View query activity and identify problematic queries and query workloads
- Get tuning recommendations for statistics that could improve query performance
- Get tuning recommendations for statistics that could improve workload performance
- Graph the access plan for a query
- Graphically generate optimization hints
- Generate reports with information about tables and predicates that are associated with a particular query
- View the values of subsystem parameters
- View a query with relevant statistics displayed next to the appropriate objects

For information about using DB2 Optimization Service Center, which is a separately packaged function provided with your DB2 for z/OS license.

## OMEGAMON

IBM Tivoli OMEGAMON XE for DB2 on z/OS is a performance monitoring tool that formats performance data. OMEGAMON combines information from EXPLAIN and from the DB2 catalog. It displays access paths, indexes, tables, table spaces, plans, packages, DBRMs, host variable definitions, ordering, table access and join sequences, and lock types. Output is presented in a dialog rather than as a table, making the information easy to read and understand. DB2 Performance Monitor (DB2 PM) performs some of the functions of OMEGAMON.

### DB2-supplied EXPLAIN stored procedure

Users with authority to run EXPLAIN directly can obtain access path information by calling the DB2-supplied EXPLAIN stored procedure.



---

## IBM Tivoli OMEGAMON XE

OMEGAMON provides performance monitoring, reporting, buffer pool analysis, and a performance warehouse all in one tool.

- OMEGAMON XE for DB2 Performance Expert on z/OS includes the function of OMEGAMON DB2 Performance Monitor (DB2 PM), which is also available as a stand-alone product. Both products report DB2 instrumentation in a form that is easy to understand and analyze. The instrumentation data is presented in the following ways:

- The Batch report sets present the data you select in comprehensive reports or graphs containing system-wide and application-related information for both single DB2 subsystems and DB2 members of a data sharing group. You can combine instrumentation data from several different DB2 locations into one report.

Batch reports can be used to examine performance problems and trends over a period of time.

- The Online Monitor gives a current “snapshot” view of a running DB2 subsystem, including applications that are running. Its history function displays information about subsystem and application activity in the recent past.

Both a host-based and Workstation Online Monitor are provided. The Workstation Online Monitor substantially improves usability, simplifies online monitoring and problem analysis, and offers significant advantages. For example, from Workstation Online Monitor, you can launch Visual Explain so you can examine the access paths and processing methods chosen by DB2 for the currently executing SQL statement.

For more information about the Workstation Online Monitor, see *OMEGAMON Monitoring Performance from Performance Expert Client or DB2 Performance Expert for z/OS and Multiplatforms Monitoring Performance from Workstation for z/OS and Multiplatforms*.

In addition, OMEGAMON contains a Performance Warehouse function that lets you:

- Save DB2 trace and report data in a performance database for further investigation and trend analysis
- Configure and schedule the report and load process from the workstation interface
- Define and apply analysis functions to identify performance bottlenecks.
- OMEGAMON for DB2 Performance Expert also includes the function of DB2 Buffer Pool Analyzer, which is also available as a stand-alone product. Both products help you optimize buffer pool usage by offering comprehensive reporting of buffer pool activity, including:
  - Ordering by various identifiers such as buffer pool, plan, object, and primary authorization ID
  - Sorting by getpage, sequential prefetch, and synchronous read
  - Filtering capability

In addition, you can simulate buffer pool usage for varying buffer pool sizes and analyze the results of the simulation reports to determine the impact of any changes before making those changes to your current system.

---

## Tivoli Decision Support for z/OS

Tivoli Decision Support for z/OS, formerly known as Tivoli Performance Reporter for z/OS, collects data into a DB2 database and allows you to create graphical and tabular reports to use in managing systems performance.

The data can come from different sources, including SMF, the IMS log, the CICS journal, RMF, and DB2

When considering the use of Tivoli Decision Support for z/OS, consider the following:

- Tivoli Decision Support data collection and reporting are based on user specifications. Therefore, an experienced user can produce more suitable reports than the predefined reports produced by other tools.
- Tivoli Decision Support provides historical performance data that you can use to compare a current situation with previous data.
- Tivoli Decision Support can be used very effectively for reports based on the DB2 statistics and accounting records. When using it for the performance trace consider that:

- Because of the large number of different DB2 performance records, a substantial effort is required to define their formats to Tivoli Decision Support. Changes in the records require review of the definitions.
- Tivoli Decision Support does not handle information from paired records, such as “start event” and “end event.” These record pairs are used by OMEGAMON to calculate elapsed times, such as the elapsed time of I/Os and lock suspensions.

The general recommendation for Tivoli Decision Support and OMEGAMON use in a DB2 subsystem is:

- If Tivoli Decision Support is already used or you plan to use it:
  - Extend Tivoli Decision Support usage to the DB2 accounting and statistics records.
  - Use OMEGAMON for the DB2 performance trace.
- If Tivoli Decision Support is not used and you do not plan to use it:
  - Use OMEGAMON for the statistics, accounting, and performance trace.
  - Consider extending OMEGAMON with user applications based on DB2 and DB2 QMF, to provide historical performance data.

---

## Response time reporting

To correctly monitor response time, you must understand how it is reported. Response time can be measured in several different ways.

The figure below shows how some of the main measures relate to the flow of a transaction.

### End user response time

This is the time from the moment the end user presses the enter key until he or she receives the first response back at the terminal.

### DB2 accounting elapsed times

These times are collected in the records from the accounting trace and can be found in the OMEGAMON accounting reports. They are taken over the accounting interval between the point where DB2 starts to execute the first SQL statement, and the point preceding thread termination or reuse by a different user (sign-on).

This interval excludes the time spent creating a thread, and it includes a portion of the time spent terminating a thread.

Parallelism requires special considerations for doing accounting. See “Monitoring parallel operations” on page 601 for more information.

Elapsed times for stored procedures or user-defined functions separate the time spent in the allied address space and the time spent in the stored procedures address space.

Two types of elapsed times are provided:

- Class 1 elapsed time
 

This time is always presented in the accounting record and shows the duration of the accounting interval. It includes time spent in DB2 as well as time spent in the front end. In the accounting reports, it is referred to as “application time.”
- Class 2 elapsed time
 

Class 2 elapsed time, produced only if the accounting class 2 is active, counts only the time spent in the DB2 address space during the

accounting interval. It represents the sum of the times from any entry into DB2 until the corresponding exit from DB2. It is also referred to as the time spent in DB2. If class 2 is not active for the duration of the thread, the class 2 elapsed time does not reflect the entire DB2 time for the thread, but only the time when the class was active.

#### DB2 total transit time

In the particular case of an SQL transaction or query, the “total transit time” is the elapsed time from the beginning of create thread, or sign-on of another authorization ID when reusing the thread, until either the end of the thread termination, or the sign-on of another authorization ID.

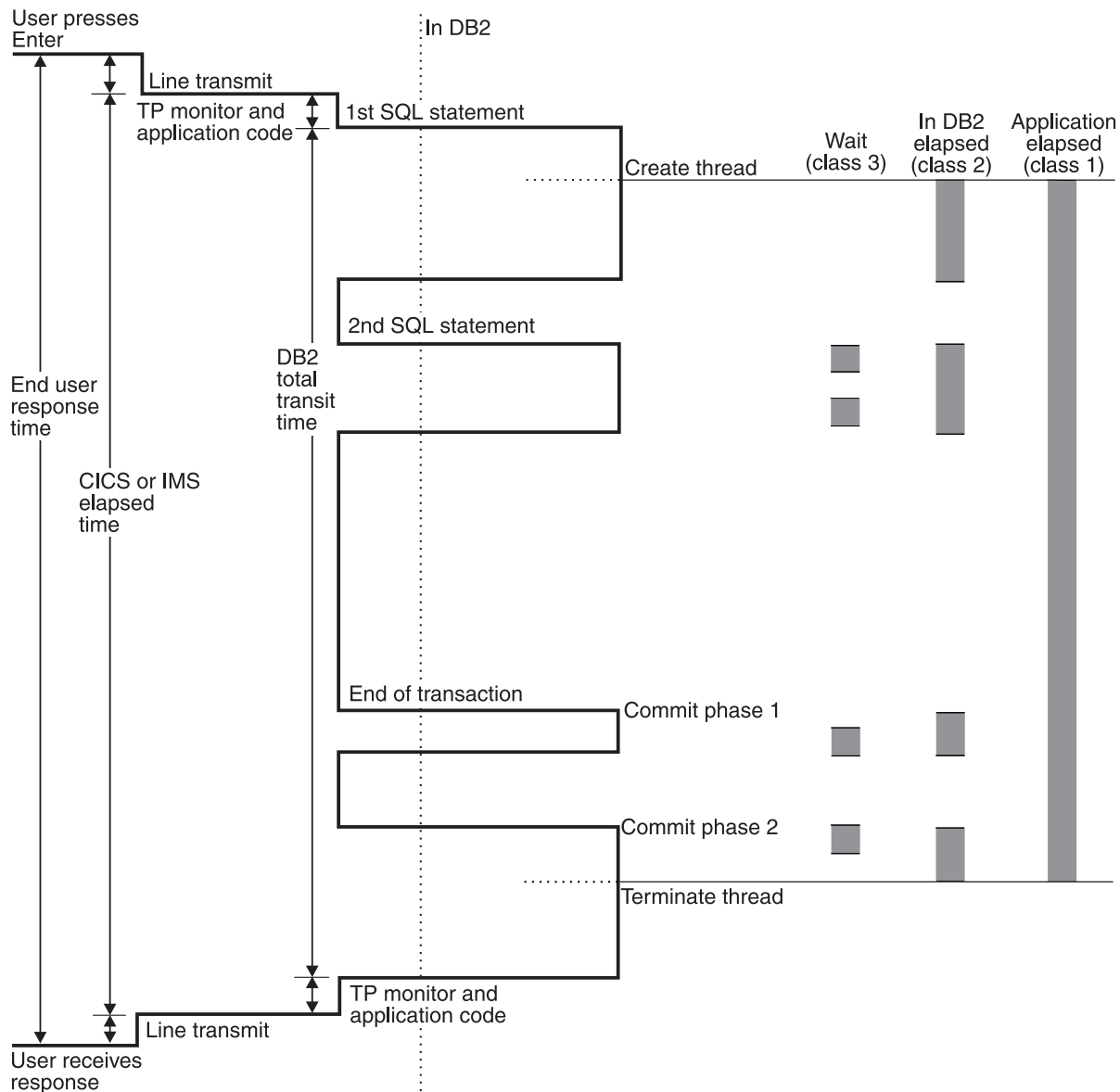


Figure 55. Transaction response times. Class 1 is standard accounting data. Class 2 is elapsed and processor time in DB2. Class 3 is elapsed wait time in DB2. Standard accounting data is provided in IFCID 0003, which is turned on with accounting class 1. When accounting classes 2 and 3 are turned on as well, IFCID 0003 contains additional information about DB2 times and wait times.

---

## Using z/OS, CICS, and IMS tools

Certain facilities enable you to monitor the performance of DB2 in conjunction with CICS and IMS.

### DB2 and CICS

To monitor DB2 and CICS, you can use the following facilities.

- RMF Monitor II for physical resource utilizations
- GTF for detailed I/O monitoring when needed
- Tivoli Decision Support for z/OS for application processor utilization, transaction performance, and system statistics.

You can use RMF Monitor II to dynamically monitor system-wide physical resource utilizations, which can show queuing delays in the I/O subsystem.

In addition, the CICS attachment facility DSNC DISPLAY command allows any authorized CICS user to dynamically display statistical information related to thread usage and situations when all threads are busy.

Be sure that the number of threads reserved for specific transactions or for the pool is large enough to handle the actual load. You can dynamically modify the value specified in the CICS resource definition online (RDO) attribute ACCOUNTREC with the DSNC MODIFY TRANSACTION command. You might also need to modify the maximum number of threads specified for the MAX USERS field on installation panel DSNTIPE.

### DB2 and IMS

To monitor DB2 and IMS, you can use the following facilities.

- RMF Monitor II for physical resource utilizations
- GTF for detailed I/O monitoring when needed
- OMEGAMON IMS Performance Analyzer, or its equivalent, for response-time analysis and tracking all IMS-generated requests to DB2
- IMS Fast Path Log Analysis Utility (DBFULTA0) for performance reports for IMS Fast Path transactions.

The DB2 IMS attachment facility also allows you to use the DB2 command DISPLAY THREAD command to dynamically observe DB2 performance.

## Monitoring system resources

You can monitor system resources to: detect resource constraints (processor, I/O, storage); to determine how resources are consumed; check processor, I/O, and paging rate to detect bottlenecks in the system; and to detect changes in resource use over comparable periods.

The following figure shows an example of a suggested system resources report.

SYSTEM RESOURCES REPORT		DATE xx/xx/xx
		FROM xx:xx:xx
		TO xx:xx:xx
TOTAL CPU Busy	74.3 %	
DB2 & IRLM	9.3 %	
IMS/CICS	45.3 %	
QMF Users	8.2 %	

DB2 Batch & Util	2.3 %			
OTHERS	9.2 %			
SYSTEM AVAILABLE	98.0 %			
TOTAL I/Os/sec.	75.5			
TOTAL Paging/sec.	6.8			
		Short Transaction	Medium Transaction	Long Transaction
Average Response Time	3.2 secs	8.6 secs	15.0 secs	
MAJOR CHANGES:				
	DB2 application DEST07 moved to production			

Figure 56. User-created system resources report

The RMF reports used to produce the information in the preceding figure were:

- The RMF CPU activity report, which lists TOTAL CPU busy and the TOTAL I/Os per second.
- RMF paging activity report, which lists the TOTAL paging rate per second for real storage.
- The RMF work load activity report, which is used to estimate where resources are spent. Each address space or group of address spaces to be reported on separately must have different SRM reporting or performance groups. The following SRM reporting groups are considered:
  - DB2 address spaces:
    - DB2 database address space (*ssnmDBM1*)
    - DB2 system services address space (*ssnmMSTR*)
    - Distributed data facility (*ssnmDIST*)
    - IRLM (IRLMPROC)
  - IMS or CICS
  - TSO-QMF
  - DB2 batch and utility jobs

The CPU for each group is obtained using the ratio  $(A/B) \times C$ , where:

A is the sum of CPU and service request block (SRB) service units for the specific group

B is the sum of CPU and SRB service units for all the groups

C is the total processor utilization.

The CPU and SRB service units must have the same coefficient.

You can use a similar approach for an I/O rate distribution.

MAJOR CHANGES shows the important environment changes, such as:

- DB2 or any related software-level change
- DB2 changes in the load module for system parameters
- New applications put into production
- Increase in the number of DB2 QMF users
- Increase in batch and utility jobs
- Hardware changes

MAJOR CHANGES is also useful for discovering the reason behind different monitoring results.

## Monitoring transaction manager throughput

You can use IMS or CICS monitoring facilities to determine throughput, in terms of transactions processed, and transaction response times.

Depending on the transaction manager, you can use the following reports:

- OMEGAMON IMS Performance Analyzer
- Fast Path Log Analysis Utility (DBFULTA0)
- Tivoli Decision Support for z/OS

In these reports:

- The transactions processed include DB2 and non-DB2 transactions.
- The transaction processor time includes the DB2 processor time for IMS but not for CICS.
- The transaction transit response time includes the DB2 transit time.

A historical database is useful for saving monitoring data from different periods. Such data can help you track the evolution of your system. You can use Tivoli Decision Support for z/OS or write your own application based on DB2 and DB2 QMF when creating this database.

---

## Chapter 21. Using DB2 Trace to monitor performance

The DB2 instrumentation facility component (IFC) provides a trace facility that you can use to record DB2 data and events.

**GUPI** With the IFC, however, analysis and reporting of the trace records must take place outside of DB2. You can use OMEGAMON to format, print, and interpret DB2 trace output. You can view an online snapshot from trace records by using OMEGAMON or other online monitors. For more information on OMEGAMON, see *Using IBM Tivoli OMEGAMON XE on z/OS*.

If you do not have OMEGAMON, or if you want to do your own analysis of the DB2 trace output, refer to Chapter 36, “Interpreting DB2 trace output,” on page 621. Also, consider writing your own program using the instrumentation facility interface (IFI). Refer to Chapter 24, “Programming for the instrumentation facility interface (IFI),” on page 439 for more information on using IFI.

Each *trace class* captures information on several subsystem events. These events are identified by many instrumentation facility component identifiers (IFCIDs). The IFCIDs are described by the comments in their mapping macros, contained in *prefix.SDSNMACS*, which is shipped to you with DB2. **GUPI**

---

### Minimizing the effects of traces on DB2 performance


The volume of data DB2 trace collects can be quite large. Consequently, the number of trace records that you request affect system performance.

In particular, when you activate a performance trace, you should qualify the START TRACE command with the particular classes, plans, authorization IDs, and IFCIDs you want to trace.

To minimize the effects of traces on DB2 performance:

- When starting a performance trace, be sure that you know what you want to report. I/O only or SQL only, for example. See OMEGAMON for examples of which classes produce which reports. Otherwise, you might have incomplete reports and have to rerun or collect too much data, overloading the data collector.
- Use the default statistics frequency of 30 minutes. When the statistics trace is active, statistics are collected by SMF at all times.
- Decide if the continuous collection of accounting data is needed. If a transaction manager provides enough accounting information, DB2 accounting might not be needed. In environments where the processor is heavily loaded, consider not running accounting on a continuous basis.
- When using accounting on a continuous basis, start classes 1 and 3 to SMF (SMF ACCOUNTING on installation panel DSNTIPN). You might also want to start accounting class 2 because it provides additional information that can be useful in resolving problems. Accounting class 2 does introduce some additional processor cost.
- To capture minimal package accounting information, start accounting trace classes 7 and 8. If you need more detailed accounting information for packages,

such as SQL statements and locking. Start accounting trace classes 7, 8, and 10. Package accounting introduces additional CPU cost and increases the amount of SMF accounting trace records.

- If you need only minimal accounting information for packages, start account accounting.
- Use the performance trace for short periods of time (START/STOP TRACE) and restrict it to certain users, applications, and classes. Use the default destination GTF to allow immediate analysis of the trace information.
- Start the global trace only if a problem is under investigation, and IBM service personnel have requested a trace. 

## Performance overhead from traces

Traces have their own associated performance overhead.

 For example, you can expect the following trace costs.

### Statistics traces

Negligible, because the statistics trace record is written only at the statistics time interval specified in minutes as a DB2 systems parameter.

### Accounting traces

Accounting traces typically incur the following overhead:

- Class 1: Typically, less than 5% transaction overhead, except for fetch-intensive applications that do not use multi-row fetch
- Class 2: CPU overhead can be 20% for fetch-intensive applications,
- Class 3: Less than 1% CPU overhead.
- Class 7 and 8: Less than 5% CPU overhead.

### Audit Traces


Audit traces typically incur less than 10% transaction overhead with all classes on.

### Performance traces class 1, 2, and 3

Performance might incur the following overhead:

- 5% to 100% CPU overhead
- Performance class 3 CPU overhead can be as high 100% for fetch-intensive applications

### Global Trace

Global trace can incur 10 to 150% CPU overhead. 

---

## Types of traces

DB2 trace can record six types of data: statistics, accounting, audit, performance, monitor, and global.

**GUIP** The description of the START TRACE command indicates which IFCIDs are activated for the different types of trace and the classes within those trace types. For details on what information each IFCID returns, see the mapping macros in *prefix.SDSNMACS*.

The trace records are written using GTF or SMF records. Trace records can also be written to storage, if you are using the monitor trace class. **GUIP**

## Statistics trace

The statistics trace reports information about how much the DB2 system services and database services are used.

**GUIP** Statistics trace is a system-wide trace and should not be used for chargeback accounting. Use the information the statistics trace provides to plan DB2 capacity, or to tune the entire set of active DB2 programs.

Statistics trace classes 1, 3, 4, 5, and 6 are the default classes for the statistics trace if statistics is specified YES in panel DSNTIPN. If the statistics trace is started using the START TRACE command, then class 1 is the default class.

- Class 1 provides information about system services and database statistics. It also includes the system parameters that were in effect when the trace was started.
- Class 3 provides information about deadlocks and timeouts.
- Class 4 provides information about exceptional conditions.
- Class 5 provides information about data sharing.
- Class 6 provides storage statistics for the DBM1 address space.

If you specified YES in the SMF STATISTICS field on installation panel DSNTIPN, the statistics trace starts automatically when you start DB2, sending class 1, 3, 4 and 5 statistics data to SMF. SMF records statistics data in both SMF type 100 and 102 records. IFCIDs 0001, 0002, 0202, and 0230 are of SMF type 100. All other IFCIDs in statistics trace classes are of SMF type 102. From installation panel DSNTIPN, you can also control the statistics collection interval (STATISTICS TIME field).

The statistics trace is written on an interval basis, and you can control the exact time that statistics traces are taken. **GUIP**

## Accounting trace

The DB2 accounting trace provides information related to application programs.

**GUIP** The information provided by accounting trace includes the following items:

- Start and stop times
- Number of commits and aborts
- The number of times certain SQL statements are issued
- Number of buffer pool requests
- Counts of certain locking events
- Processor resources consumed

- Thread wait times for various events
- RID pool processing
- Distributed processing
- Resource limit facility statistics

DB2 trace begins collecting this data at successful thread allocation to DB2, and writes a completed record when the thread terminates or when the authorization ID changes.

During CICS thread reuse, a change in the authid or transaction code initiates the sign-on process, which terminates the accounting interval and creates the accounting record. TXIDSO=NO eliminates the sign-on process when only the transaction code changes. When a thread is reused without initiating sign-on, several transactions are accumulated into the same accounting record, which can make it very difficult to analyze a specific transaction occurrence and correlate DB2 accounting with CICS accounting. However, applications that use ACCOUNTREC(UOW) or ACCOUNTREC(TASK) in the DBENTRY RDO definition initiate a “partial sign-on”, which creates an accounting record for each transaction. You can use this data to perform program-related tuning and assess and charge DB2 costs.

Accounting data for class 1 (the default) is accumulated by several DB2 components during normal execution. This data is then collected at the end of the accounting period; it does not involve as much overhead as individual event tracing.

On the other hand, when you start class 2, 3, 7, or 8, many additional trace points are activated. Every occurrence of these events is traced internally by DB2 trace, but these traces are not written to any external destination. Rather, the accounting facility uses these traces to compute the additional total statistics that appear in the accounting record, IFCID 003, when class 2 or class 3 is activated. Accounting class 1 must be active to externalize the information.

Before you can turn on accounting for packages and DBRMs, accounting trace classes 1 and 7 must be active. Though you can turn on class 7 while a plan is being executed, accounting trace information is only gathered for packages or DBRMs executed after class 7 is activated. Activate accounting trace class 8 with class 1 to collect information about the amount of time an agent was suspended in DB2 for each executed package. If accounting trace classes 2 and 3 are activated, activating accounting trace classes 7 and 8 incurs minimal additional performance cost.

If you want information from either, or both, accounting class 2 and 3, be sure to activate class 2, class 3, or both classes before your application starts. If these classes are activated while the application is running, the times gathered by DB2 trace are only from the time the class was activated.


Accounting trace class 5 provides information on the amount of elapsed time and TCB time that an agent spent in DB2 processing instrumentation facility interface (IFI) requests. If an agent did not issue any IFI requests, these fields are not included in the accounting record.


If you specified YES for SMF ACCOUNTING on installation panel DSNTIPN, the accounting trace starts automatically when you start DB2, and sends IFCIDs that are of SMF type 101 to SMF. The accounting record IFCID 0003 is of SMF type 101.

 **GUIP**

## Audit trace


The audit trace collects information about DB2 security controls and is used to ensure that data access is allowed only for authorized purposes.

 **GUIP** On the CREATE TABLE or ALTER TABLE statements, you can specify whether or not a table is to be audited, and in what manner; you can also audit security information such as any access denials, grants, or revokes for the table. The default causes no auditing to take place. .

If you specified YES for AUDIT TRACE on installation panel DSNTIPN, audit trace class 1 starts automatically when you start DB2. By default, DB2 sends audit data to SMF. SMF records audit data in type 102 records. When you invoke the -START TRACE command, you can also specify GTF as a destination for audit data.  **GUIP**

## Performance trace

The performance trace provides information about a variety of DB2 events, including events related to distributed data processing.


 **GUIP** You can use this information to further identify a suspected problem, or to tune DB2 programs and resources for individual users or for DB2 as a whole.

You cannot automatically start collecting performance data when you install or migrate DB2. To trace performance data, you must use the -START TRACE(PERFM) command.

The performance trace defaults to GTF.  **GUIP**

## Monitor trace

The monitor trace records data for online monitoring with user-written programs.

 **GUIP** This trace type has several predefined classes; those that are used explicitly for monitoring are listed here:

- Class 1 (the default) allows any application program to issue an instrumentation facility interface (IFI) READS request to the IFI facility. If monitor class 1 is inactive, a READS request is denied. Activating class 1 has a minimal impact on performance.
- Class 2 collects processor and elapsed time information. The information can be obtained by issuing a READS request for IFCID 0147 or 0148. In addition, monitor trace class 2 information is available in the accounting record, IFCID 0003. Monitor class 2 is equivalent to accounting class 2 and results in equivalent overhead. Monitor class 2 times appear in IFCIDs 0147, 0148, and 0003 if either monitor trace class 2 or accounting class 2 is active.
- Class 3 activates DB2 wait timing and saves information about the resource causing the wait. The information can be obtained by issuing a READS request for IFCID 0147 or 0148. In addition, monitor trace class 3 information is available

in the accounting record, IFCID 0003. As with monitor class 2, monitor class 3 overhead is equivalent to accounting class 3 overhead.

When monitor trace class 3 is active, DB2 can calculate the duration of a class 3 event, such as when an agent is suspended due to an unavailable lock. Monitor class 3 times appear in IFCIDs 0147, 0148, and 0003, if either monitor class 3 or accounting class 3 is active.

- Class 5 traces the amount of time spent processing IFI requests.
- Class 7 traces the amount of time an agent spent in DB2 to process each package. If monitor trace class 2 is active, activating class 7 has minimal performance impact.
- Class 8 traces the amount of time an agent was suspended in DB2 for each package executed. If monitor trace class 3 is active, activating class 8 has minimal performance impact.



---

## Chapter 22. Recording SMF trace data

Each location is responsible for processing the SMF records produced by DB2 trace.

For example, during DB2 execution, you can use the z/OS operator command SETSMF or SS to alter SMF parameters that you specified previously. The following command records statistics (record type 100), accounting (record type 101), and performance (record type 102) data to SMF. To execute this command, specify PROMPT(ALL) or PROMPT(LIST) in the SMFPRMxx member used from SYS1.PARMLIB.

```
SETSMF SYS(TYPE(100:102))
```

If you are not using measured usage licensing, do not specify type 89 records or you incur the overhead of collecting that data.

You can use the SMF program IFASMFDP to dump these records to a sequential data set. You might want to develop an application or use OMEGAMON to process these records. For a sample DB2 trace record sent to SMF, see Figure 95 on page 623. For more information about SMF, refer to *z/OS JES2 Initialization and Tuning Guide*.

---

### Activating SMF

SMF must be running before you can send data to it.

To make SMF operational:

- Specify the ACTIVE parameter and the proper TYPE subparameter for SYS and SUBSYS to update member SMFPRMxx of SYS1.PARMLIB. member SMFPRMxx indicates whether SMF is active and which types of records SMF accepts. For member SMFPRMxx, xx are two user-defined alphanumeric characters appended to 'SMFPRM' to form the name of an SMFPRMxx member.
- Optional: You can also code an IEFU84 SMF exit to process the records that are produced.

---

### Allocating SMF buffers

When you specify a performance trace type, the volume of data that DB2 can collect can be quite large. If you are sending this data to SMF, you must allocate adequate SMF buffers; the default buffer settings are probably insufficient.

If an SMF buffer shortage occurs, SMF rejects any trace records sent to it. DB2 sends a message (DSNW133I) to the MVS operator when this occurs. DB2 treats the error as temporary and remains active even though data could be lost. DB2 sends another message (DSNW123I) to the z/OS operator when the shortage has been alleviated and trace recording has resumed.

You can determine if trace data has been lost by examining the DB2 statistics records with an IFCID of 0001, as mapped by macro DSNDQWST. These records show:

- The number of trace records successfully written
- The number of trace records that could not be written
- The reason for the failure

If your location uses SMF for performance data or global trace data, be sure that:

- Your SMF data sets are large enough to hold the data.
- SMF is set up to accept record type 102. (Specify member SMFPRMxx, for which 'xx' are two user-defined alphanumeric characters.)
- Your SMF buffers are large enough.

To allocate SMF buffers:

- Specify SMF buffering on the VSAM BUFSP parameter of the access method services DEFINE CLUSTER statement. Do not use the default settings if DB2 performance or global trace data is sent to SMF.
- Specify CISZ(4096) and BUFSP(81920) on the DEFINE CLUSTER statement for each SMF VSAM data set. These values are the minimum required for DB2; you might have to increase them, depending on your z/OS environment.  
DB2 runs above the 16MB line of virtual storage in a cross-memory environment.

---

## Reporting data in SMF

You can send report trace records to SMF in several ways.

By using any of the following tools, you can compare any report for a current day, week, or month with an equivalent sample, as far back as you want to go. The samples become more widely spaced but are still available for analysis.

To send reporting data to SMF:

- Use Tivoli Decision Support for z/OS to collect the data and create graphical or tabular reports.
- Write an application program to read and report information from the SMF data set. You can tailor it to fit your exact needs.
- Use OMEGAMON See "IBM Tivoli OMEGAMON XE" on page 422 for a discussion of OMEGAMON's capabilities.

---

## Chapter 23. Recording GTF trace data

The default destination for the performance trace classes is the generalized trace facility (GTF). The z/OS operator must start GTF before you can send data to it.

**Prerequisite:** Be sure that no GTF member exists in SYS1.

When starting GTF, if you use the JOBNAMEP option to obtain only those trace records written for a specific job, trace records written for other agents are not written to the GTF data set. This means that a trace record that is written by a system agent that is processing for an allied agent is discarded if the JOBNAMEP option is used. For example, after a DB2 system agent performs an IDENTIFY request for an allied agent, an IFCID record is written. If the JOBNAMEP keyword is used to collect trace data for a specific job, however, the record for the IDENTIFY request is not written to GTF, even if the IDENTIFY request was performed for the job named on the JOBNAMEP keyword.

You can record DB2 trace data in GTF using a GTF event ID of X'FB9'.

Trace records longer than the GTF limit of 256 bytes are spanned by DB2. For instructions on how to process GTF records, refer to Chapter 36, "Interpreting DB2 trace output," on page 621.

To start GTF:

Start GTF as indicated in the following table to ensure that offsets map correctly. When starting GTF, specify TIME=YES, and then TRACE=USRP.

*Table 85. Recording GTF trace data*

You enter...	System responds...
S GTF,,, (TIME=YES)	AHL100A SPECIFY TRACE OPTIONS
TRACE=USRP	AHL101A SPECIFY TRACE EVENT KEYWORDS --USR=
USR=(FB9)	AHL102A CONTINUE TRACE DEFINITION OR REPLY END
END	AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
U	AHL031I GTF INITIALIZATION COMPLETE

**Note:** To make stopping GTF easier, you can name the GTF session when you start it. For example, you could specify S GTF.GTF,,, (TIME=YES).

If a GTF member exists in SYS1.PARMLIB, the GTF trace option USR might not be in effect. When no other member exists in SYS1.PARMLIB, you are sure to have only the USR option activated, and no other options that might add unwanted data to the GTF trace.



---

## Chapter 24. Programming for the instrumentation facility interface (IFI)

The DB2 instrumentation facility gathers trace data that can be written to one or more destinations that you specify.

**PSPI** The instrumentation facility interface (IFI) is designed for a programs that need online trace information. IFI can be accessed through any of the DB2 attachment facilities.

IFI uses the standard security mechanisms that DB2 uses: connection authorization, plan authorization, and so forth. Security checks specifically related to IFI are included in the descriptions of the functions.

Before using IFI, you should be familiar with the material in Chapter 21, “Using DB2 Trace to monitor performance,” on page 429, which includes information on the DB2 trace facility and instrumentation facility component identifiers (IFCIDs).

Note that where the trace output indicates a particular release level, you see ‘xx’ to show that this information varies according to the actual release of DB2 that you are using.

You can use IFI in a monitor program (a program or function outside of DB2 that receives information about DB2).

When a DB2 trace is active, internal events trigger the creation of trace records. The records, identified by *instrumentation facility component identifiers* (IFCIDs), can be written to buffers, and you can read them later with the IFI READA function. This means you are collecting the data *asynchronously*; you are not reading the data at the time it was written.

You can trigger the creation of certain types of trace records by using the IFI READS function. Because the records are identified by IFCIDs, they do not require an internal buffer; they are passed immediately to your monitor program through IFI. This means you are collecting the data *synchronously*. The data is collected at the time of the request for the data. **PSPI**

### Related tasks

Chapter 21, “Using DB2 Trace to monitor performance,” on page 429

---


## Submitting DB2 commands through IFI

You can submit any DB2 command through IFI, but this capability is most useful for submitting DB2 trace commands to start, stop, display, and modify traces.

**PSPI**

Using specified trace classes and IFCIDs, a monitor program can control the amount and type of its data. You can design your monitor program to:

- Activate and deactivate pre-defined trace classes.
- Activate and deactivate a trace record or group of records (identified by IFCIDs).

- Activate and deactivate predefined trace classes and trace records (identified by IFCIDs) restricting tracing to a set of DB2 identifiers (plan name, authorization ID, resource manager identifier (RMID), and so on). 

## Example DB2 command through IFI

 This example issues a DB2 START TRACE command for MONITOR Class 1.

Figure 57. Starting a trace using IFI

```
CALL DSNWLI,('COMMAND ',IFCAAREA,RETAREA,OUTAREA,BUFAREA),VL
:
:
COMMAND DC CL8 'COMMAND '
*****
* Function parameter declaration                                *
*****
* Storage of LENGTH(IFCA) and properly initialized              *
*****
IFCAAREA DS      0CL180
:
*****
* Storage for length and returned info.                        *
*****
RETAREA DS      CL608
*****
* Storage for length and DB2 Command                          *
*****
OUTAREA DS      0CL42
OUTLEN DC      X'002A0000'
OUTCMD DC      CL37'-STA TRAC(MON) DEST(OPX) BUFSIZE(256)'
*****
* Storage of LENGTH(WBUF) and properly initialized            *
*****
BUFAREA DS      0CL16
:
:
```



## Obtaining trace data through IFI

You might want to collect trace data from DB2

 You can use DB2 trace information for the following purposes:

- To obtain accounting information for online billing.
- To periodically obtain system-wide information about DB2, highlight exceptional conditions, or provide throughput information.

The following illustrates the logic flow:

1. Initialize.
2. Set a timer.
3. Wait for the timer to expire.
4. Call IFI to obtain statistics data via a READS request.
5. Do delta calculations to determine activity.

This step is not necessary for IFCID 0199 because DB2 resets statistics at the beginning of every collection interval.

6. Display the information on a terminal.
  7. Loop back to the timer.
- To learn which processes have been connected to DB2 the longest, or which processes have used the most CPU time in DB2.
  - To obtain accounting records as transactions terminate.
  - To determine the access and processing methods for an SQL statement. Start a trace, issue a PREPARE statement, and then use the resulting trace data as an alternative to using EXPLAIN.
  - To capture log buffers online for use in remote recovery.
  - To retrieve SQL changes synchronously from the log for processing in an application.

PSPI

---

## Passing data to DB2 through IFI

You can use IFI to pass data to the destination of a DB2 trace.

PSPI

For example, you can:

- Extend accounting data collected within DB2. For example, a monitor program can collect batch file I/O counts, store them in a user-defined trace record, and process them along with standard DB2 accounting data.
- Include accounting data from DB2 QMF, IMS, or CICS
- Permit CICS users to write the CICS accounting token and task number into the DB2 trace, assuming ACCOUNTREC in the DB2ENTRY RDO definition is neither UOW nor TASK .

PSPI

---

## IFI functions

A monitor program can use the following IFI functions:

PSPI

### COMMAND

To submit DB2 commands. For more information, see “COMMAND: Syntax and usage with IFI” on page 443.

### READS

To obtain monitor trace records synchronously. The READS request causes those records to be returned immediately to the monitor program.

### READA

To obtain trace records of any trace type asynchronously. DB2 records trace events as they occur and places that information into a buffer; a READA request moves the buffered data to the monitor program.

### WRITE

To write information to a DB2 trace destination that was previously activated by a START TRACE command.

PSPI

### Related tasks

“Using READA requests through IFI” on page 462

“Using READS requests through IFI” on page 445

“Using WRITE requests through IFI” on page 465

---

## Invoking IFI from your program

IFI can be used by assembler and PL/I programs.



To use IFI, include a call to DSNWLI in your monitor program.

The following example depicts an IFI call in an assembler program. All IFI-related examples are given for assembler.

```
CALL DSNWLI,(function,ifca,parm-1,...parm-n),VL
```

The parameters that are passed on the call indicate the desired function (as described in “IFI functions” on page 441), point to communication areas used by the function, and provide other information that depends on the function specified. Because the parameter list might vary in length, the high-order bit of the last parameter must be on to signal that it is the last parameter in the list.

**Example:** To turn on the bit in assembler, use the VL option to signal a variable length parameter list.

The communication areas that are used by IFI are described in “Common communication areas for IFI calls” on page 466.

After you insert this call in your monitor program, you must link-edit the program with the correct language interface. Each of the following language interface modules has an entry point of DSNWLI for IFI:


- CAF DSNALI
- TSO DSNELI
- CICS DSNCLI
- IMS DFSLI000
- RRSF DSNRLI

CAF DSNALI, the CAF (call attachment facility) language interface module, includes a second entry point of DSNWLI2. The monitor program that link-edits DSNALI with the program can make IFI calls directly to DSNWLI. The monitor program that loads DSNALI must also load DSNWLI2 and remember its address. When the monitor program calls DSNWLI, the program must have a dummy entry point to handle the call to DSNWLI and then call the real DSNWLI2 routine.

**Considerations for writing a monitor program:** A monitor program issuing IFI requests must be connected to DB2 at the thread level. If the program contains SQL statements, you must precompile the program and create a DB2 plan using the BIND process. If the monitor program does not contain any SQL statements, it does not have to be precompiled. However, as is the case in all the attachment environments, even though an IFI only program (one with no SQL statements) does not have a plan of its own, it can use any plan to get the thread level connection to DB2.

The monitor program can run in either 24- or 31-bit mode.

**Monitor trace classes:** Monitor trace classes 2 through 8 can be used to collect information related to DB2 resource usage. Use monitor trace class 5, for example, to find out how much time is spent processing IFI requests. Monitor trace classes 2, 3, 5, 7, and 8 are identical to accounting trace classes 2, 3, 5, 7, and 8. For more information about these traces, see “Monitor trace” on page 433.


**Monitor authorization:** On the first READA or READS call from a user, an authorization is checked to determine if the primary authorization ID or one of the secondary authorization IDs of the plan executor has MONITOR1 or MONITOR2 privilege. If your installation uses the access control authorization exit routine, that exit routine might control the privileges that can use the monitor trace. If you have an authorization failure, an audit trace (class 1) record is generated that contains the return and reason codes from the exit. This is included in IFCID 0140. 

---

## Using IFI from stored procedures

You can use the IFI interface from a stored procedure, and the output of the trace can be returned to the client.



You can also issue DB2 commands, such as “DISPLAY THREAD”, from a stored procedure and return the results to the client. 


---

## COMMAND: Syntax and usage with IFI

A DB2 command resides in the output area; a monitor program can submit that command by issuing a COMMAND request to IFI.




The DB2 command is processed and the output messages are returned to the monitor program in the return area.

You can submit any DB2 command, including START TRACE, STOP TRACE, DISPLAY TRACE, and MODIFY TRACE. 

## Authorization for DB2 commands through IFI

For an application program to submit a command, the primary authorization ID or one of the secondary authorization IDs of the process must have the appropriate DB2 command authorization.



Otherwise the request is denied. An application program might have the authorization to issue DB2 commands, but not the authorization to issue READA requests. 

## Syntax for DB2 commands through IFI

DB2 commands that are issued through IFI have the following syntax.



```
CALL DSNWLI,('COMMAND ',ifca,return-area,output-area,buffer-info .),VL  
ifca
```

IFCA (instrumentation facility communication area) is an area of storage that contains the return code and reason code. IFCA indicates the following information:

- The success or failure of the request
- Diagnostic information from the DB2 component that executed the command
- The number of bytes moved to the return area
- The number of bytes of the message segments that did not fit in the return area

Some commands might return valid information despite a non-zero return code or reason code. For example, the DISPLAY DATABASE command might indicate that more information could have been returned than was allowed.

If multiple errors occur, the last error is returned to the caller. For example, if the command is in error and the error message does not fit in the area, the error return code and reason code indicate that the return area is too small.

If a monitor program issues START TRACE, the ownership token (IFCAOWNER) in the IFCA determines the owner of the asynchronous buffer. The owner of the buffer is the only process that can obtain data through a subsequent READA request.

#### *return-area*

When the issued command finishes processing, it places messages (if any) in the return area. The messages are stored as varying-length records, and the total number of bytes in the records is placed in the IFCABM (bytes moved) field of the IFCA. If the return area is too small, as many message records as can fit are placed into the return area.

The monitor program should analyze messages that are returned by the command function.

#### *output-area*

Contains the varying-length command.

#### *buffer-info*

This parameter is required for starting traces to an OP buffer. Otherwise, it is not needed. This parameter is used only on COMMAND requests. It points to an area that contains information about processing options when a trace is started by an IFI call to an unassigned OP $n$  destination buffer. An OP $n$  destination buffer is considered unassigned if it is not owned by a monitor program.

If the OP $n$  destination buffer is assigned, then the buffer information area is not used on a later START or MODIFY TRACE command to that OP $n$  destination. For more information about using OP $n$  buffers.

When you use *buffer-info* on START TRACE, you can specify the number of bytes that can be buffered before the monitor program ECB is posted. The ECB is posted when the amount of trace data collected has reached the value that is specified in the byte count field. The byte count field is also specified in the buffer information area.

Table 86. Buffer information area fields. This area is mapped by assembler mapping macro DSNDWBUF.

Name	Hex offset	Data type	Description
WBUFLEN	0	Signed two-byte integer	Length of the buffer information area, plus 4. A zero indicates the area does not exist.
	2	Signed two-byte integer	Reserved.
WBUFEYE	4	Character, 4 bytes	Eye catcher for block, WBUF.
WBUFECB	8	Address	The ECB address to post when the buffer has reached the byte count specification (WBUFBC). The ECB must reside in monitor key storage.
			A zero indicates not to post the monitor program. In this case, the monitor program should use its own timer to determine when to issue a READA request.
WBUFBC	C	Signed four-byte integer	The records placed into the instrumentation facility must reach this value before the ECB can be posted. If the number is zero, and an ECB exists, posting occurs when the buffer is full.

## Example of DB2 command through IFI

This example issues a DB2 START TRACE command for MONITOR Class 1.

```
CALL DSNWLI,('COMMAND ',IFCAAREA,RETAREA,OUTAREA,BUFAREA),VL
:
:
COMMAND DC CL8 'COMMAND '
*****
* Function parameter declaration                                     *
*****
* Storage of LENGTH(IFCA) and properly initialized                 *
*****
IFCAAREA DS      0CL180
:
*****
* Storage for length and returned info.                             *
*****
RETAREA  DS      CL608
*****
* Storage for length and DB2 Command                               *
*****
OUTAREA  DS      0CL42
OUTLEN   DC      X'002A0000'
OUTCMD   DC      CL37'-STA TRAC(MON) DEST(OPX) BUFSIZE(256)'
*****
* Storage of LENGTH(WBUF) and properly initialized                 *
*****
BUFAREA  DS      0CL16
:
:
```

Figure 58. Starting a trace using IFI



## Using READS requests through IFI

READS allows your monitor program to read DB2 status information that is collected at the time of the IFI call.

**Prerequisite:** Monitor class 1 must be activated prior to any READS requests.

**PSPI** The records available are for the following IFCIDs


0001	0199
0002	0202
0106	0217
0124	0225
0129	0230
0147	0234
0148	0254
0149	0306
0150	0316
0185	0317

The following IFCID records can be obtained only through the IFI READS interface.

0124  
0129  
0147  
0148  
0149  
0150  
0199  
0217  
0225  
0234  
0254  
0316  
0317


Due to performance considerations, the majority of data that is obtained by a monitor program probably comes over the synchronous interface. Summarized DB2 information is easier for a monitor program to process, and the monitor program logic is simpler because a smaller number of records are processed.

- Start monitor classes 2, 3, 5, 7, and 8 to collect summary and status information for later probing. In this case, an instrumentation facility trace is started and information is summarized by the instrumentation facility, but not returned to the caller until it is requested by a READS call.
- Consider reasonability tests for data that is obtained through READS. The READS request can reference data that is updated during the retrieval process. Because the READS function does not suspend activity that takes place under referenced structures, an abend can occur. If an abend occurs, the READS function is terminated without a dump and the monitor program is notified through the return code and reason code information in the IFCA. However, the return area can contain valid trace records, even if an abend occurred; therefore, your monitor program should check for a non-zero value in the IFCABM (bytes moved) field of the IFCA.
- When you use a READS request with a query parallelism task, remember that each parallel task is a separate thread. Each parallel thread has a separate READS output for more information on tracing the parallel tasks. A READS request might return thread information for parallel tasks on a DB2 data sharing member without the thread information for the originating task in a Sysplex query parallelism case.


- Use the qual-area parameter, mapped by DSNDWQAL, to qualify the trace records to be returned on IFI READS requests. 

## Authorization for READS requests through IFI

The primary authorization ID or one of the secondary authorization IDs of the process running the application program must have MONITOR1 or MONITOR2 privilege.

 If neither the primary authorization ID nor one of the secondary authorization IDs has authorization, the request is denied.

READS requests are checked for authorization once for each user (ownership token) of the thread. (Several users can use the same thread, but an authorization check is performed each time the user of the thread changes.)

If you use READS to obtain your own data (IFCID 0124, 0147, 0148, or 0150 not qualified), no authorization check is performed. 

## Syntax for READS requests through IFI

DB2 commands that are issued through IFI have the following syntax.



CALL DSNWLI, ('READS ' , *ifca*, *return-area*, *ifcid-area*, *qual-area*), VL

*ifca*

Contains information about the status of a READS call. See “Instrument facility communications area (IFCA)” on page 466 for a description of the IFCA.

*return-area*

Contains the varying-length records that are returned by the instrumentation facility. IFI monitor programs might require large enough READS return areas to accommodate the following information:

- Larger IFCID 0147 and 0148 records that contain distributed thread data (both allied and database access).
- Additional records that are returned when database access threads exist that satisfy the specified qualifications on the READS request.
- Log record control intervals with IFCID 129. For more information about using IFI to return log records.
- Log records based on user-specified criteria with IFCID 306. For example, the user can retrieve compressed or decompressed log records.
- Data descriptions and changed data returned with IFCID 185.

If the return area is too small to hold all the records returned, it contains as many records as can fit. The monitor program obtains the return area for READS requests in its private address space. See “Return area” on page 470 for a description of the return area.

*ifcid-area*

Contains the IFCIDs of the desired information. The number of IFCIDs can be variable. If the length specification of the IFCID area is exceeded or an IFCID of X'FFFF' is encountered, the list is terminated. If an invalid IFCID is specified no data is retrieved. See “IFCID area” on page 471 for a description of the IFCID area.

### *qual-area*

This parameter is optional, and is used only on READS requests. It points to the qualification area, where a monitor program can specify constraints on the data that is to be returned. If the qualification area does not exist (length of binary zero), information is obtained from all active allied threads and database access threads. Information is not obtained for any inactive database access threads that might exist.

The length constants for the qualification area are provided in the DSNDWQAL mapping macro. If the length is not equal to the value of one of these constants, IFI considers the call invalid.

The following trace records, identified by IFCID, cannot be qualified: 0001, 0002, 0106, 0202, 217, 225, 0230. If you attempt to qualify them, the qualification is ignored.

The rest of the synchronous records can be qualified. See “Synchronous data and READS requests through IFI” on page 459 for information about these records. However, not all the qualifications in the qualification area can be used for these IFCIDs. See “Which qualifications are used for READS requests issued through IFI?” on page 458 for qualification restrictions. Unless the qualification area has a length of binary zero (in which case the area does not exist), the address of *qual-area* supplied by the monitor program points to an area that is formatted by the monitor program, as shown in Table 87.

Table 87. Qualification area fields. This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALLEN	0	Signed two-byte integer	Length of the qualification area, plus 4. The following constants set the qualification area length field:  <b>WQALLN4</b> When specified, the location name qualifications (WQALLOCN and WQALLUWI), the group buffer pool qualifier (WQALGBPN) and the read log fields are used.  <b>WQALLN5</b> When specified, the dynamic statement cache fields (WQALFFLD, WQALFVAL, WQALSTNM, and WQALSTID) are used for READS calls for IFCID 0316 and 0317.  <b>WQALLN6</b> When specified, the end-user identification fields (WQALEUID, WQALEUTX, and WQALEUWS) are used for READS calls for IFCID 0124, 0147, 0148, 0149, and 0150.  <b>WQALLN21</b> When specified, the location name qualifications (WQALLOCN and WQALLUWI) are ignored.  <b>WQALLN22</b> When specified, the location name qualifications (WQALLOCN and WQALLUWI) are used.  <b>WQALLN23</b> When specified, the log data access fields (WQALLTYP, WQALLMOD, WQALLRBA, and WQALLNUM) are used for READS calls that use IFCID 129.
	2	Signed two-byte integer	Reserved.

Table 87. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALEYE	4	Character, 4 bytes	Eye catcher for block, WQAL.
WQALACE	8	Address	Thread identification token value. This value indicates the specific thread wanted; binary zero if it is not to be used.
WQALAIT2	C	Address	Reserved.
WQALPLAN	10	Character, 8 bytes	Plan name; binary zero if it is not to be used.
WQALAUTH	18	Character, 8 bytes	The current primary authorization ID; binary zero if it is not to be used.
WQALOPID	20	Character, 8 bytes	The original authorization ID; binary zero if it is not to be used.
WQALCONN	28	Character, 8 bytes	Connection name; binary zero if it is not to be used.
WQALCORR	30	Character, 12 bytes	Correlation ID; binary zero if it is not to be used.
WQALREST	3C	Character, 32 bytes	Resource token for a specific lock request when IFCID 0149 is specified. The field must be set by the monitor program. The monitor program can obtain the information from a previous READS request for IFCID 0150 or from a READS request for IFCID 0147 or 0148.
WQALHASH	5C	Hex, 4 bytes	Resource hash value that specifies the resource token for a specific lock request when IFCID 0149 is specified. The field must be set by the monitor program. The monitor program can obtain the information from a previous READS request for IFCID 0150 or possibly from a READS request for IFCID 0147 or 0148.
WQALASID	60	Hex, 2 bytes	ASID that specifies the address space of the desired process.
WQALFOPT	62	Hex, 1 byte	Filtering options for IFCID 0150: <div> <div> </div> <div> </div> <div> </div> <div> </div> <div> </div> <div> </div> </div> <div> <div>X'20'</div> <div>X'40'</div> <div>X'80'</div> </div> <div> <div>Return lock information for resources that have local or global waiters. (WQALLCON)</div> <div>Return lock information only for resources that have one or more interested agents. (WQALMAGN)</div> <div>Return lock information only for resources that have waiters. (WQALWAIT)</div> </div>
WQALFLGS	63	Hex, 1 byte	Options for 147/148 records: <div> <div>X'40'</div> <div>X'80'</div> </div> <div> <div>Active allied agent 147/148 records are <b>not</b> written for this READS request. DDF/RRSFA rollup records are written if WQAL148NR = OFF. (WQAL148NA)</div> <div>DDF/RRSAF rollup 147/148 records are <b>not</b> written for this READS request. Active allied agent records are written if WQAL148NA = OFF. <sup>1</sup> (WQAL148NR)</div> </div>
WQALLUWI	64	Character, 24 bytes	LUWID (logical unit of work ID) of the thread wanted; binary zero if it is not to be used

Table 87. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
	7C	Character, 16 bytes	<p>Location name. If specified, data is returned only for distributed agents that originate at the specified location.</p> <p><b>Example:</b> If site A is located where the IFI program is running and SITE A is specified in the WQALLOCN, database access threads and distributed allied agents that execute at SITE A are reported. Local non-distributed agents are not reported.</p> <p><b>Example:</b> If site B is specified in the WQALLOCN and the IFI program is still executing at site A, information on database access threads that execute in support of a distributed allied agent at site B are reported.</p> <p><b>Example:</b> If WQALLOCN is not specified, information on all threads that execute at SITE A (the site where the IFI program executes) is returned. This includes local non-distributed threads, local database access agents, and local distributed allied agents.</p>
WQALLTYP	8C	Character, 3 bytes	Specifies the type of log data access. 'CI ' must be specified to obtain log record control intervals (CIs).

Table 87. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALLMOD	8F	Character, 1 byte	The mode of log data access:
			'D' Return the direct log record specified in WQALLRBA if the IFCID is 0306. (WQALMODD)
			'F' Access the first log CI of the restarted DB2 system if the IFCID is 0129. One CI is returned, and the WQALLNUM and WQALLRBA fields are ignored. It indicates to return the first set of qualified log records if the IFCID is 0306. (WQALMODF)
			'H' Return the highest LRSN or log RBA in the active log. The value is returned in the field IFCAHLRS in the IFCA. (WQALMODH)
			'N' Return the next set of qualified log records. (WQALMODN)
			'P' the last partial CI written to the active log is given to the Log Capture Exit. If the last CI written to the log was not full, the RBA of the log CI given to the Log Exit is returned in the IFCAHLRS field of the IFI communication area (IFCA). Otherwise, an RBA of zero is returned in IFCAHLRS. This option ignores WQALLRBA and WQALLNUM. (WQALMODP)
			'R' Access the CIs specified by the value in the WQALLRBA field: <ul style="list-style-type: none"> <li>• If the requested number of complete CIs (as specified in WQALLNUM) are currently available, those CIs are returned. If fewer than the requested number of complete CIs are available, IFI returns as many complete CIs as are available.</li> <li>• If the WQALLRBA value is beyond the end of the active log, IFI returns a return code of X'0000000C' and a reason code of X'00E60855'. No records are returned.</li> <li>• If no complete CIs exist beyond the WQALLRBA value, IFI returns a return code of X'0000000C' and a reason code of X'00E60856'. No records are returned.</li> </ul> (WQALMODR)
			'T' Terminate the log position that is held to anticipate a future mode 'N' call. (WQALMODT)
WQALLNUM	90	Hex, 2 bytes	The number of log CIs to be returned. The valid range is X'0001' to X'0007'.
WQALCDCD	92	Character, 1 byte	Data description request flag:
			'A' Indicates that a data description can only be returned the first time a DATA request is issued from the region or when it was changed for a given table. This is the default.
			'N' Indicates that a data description is not returned.
			'Y' Indicates that a data description is returned for each table in the list for every new request.
	93	Hex, 1 byte	Reserved.

Table 87. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALLRBA	94	Hex, 8 bytes	<p>If the IFCID is 0129, this is the starting log RBA of the CI to be returned. The CI starting log RBA value must end in X'000'. The RBA value must be right-justified.</p> <p>If the IFCID is 0306, this is the log RBA or LRSN to be used in mode 'F'.</p>
WQALGBPN	9C	Character, 8 bytes	<p>Group buffer pool name for IFCID 0254. Buffer pool name for IFCID 0199. To specify a single buffer pool or group buffer pool, specify the buffer pool name in hexadecimal, followed by hexadecimal blanks.</p> <p><b>Example:</b> To specify buffer pool BP1, put X'C2D7F140404040' in this field.</p> <p>To specify more than one buffer pool or group buffer pool, use the pattern-matching character X'00' in any position in the buffer pool name. X'00' indicates that any character can appear in that position, and in all positions that follow.</p> <p><b>Example:</b> If you put X'C2D7F10000000000' in this field, you indicate that you want data for all buffer pools whose names begin with BP1, so IFI collects data for BP1, BP10 through BP19, and BP16K0 through BP16K9.</p> <p><b>Example:</b> If you put X'C2D700F100000000' in this field, you indicate that you want data for all buffer pools whose names begin with BP, so IFI collects data for all buffer pools. IFI ignores X'F1' in position four because it occurs after the first X'00'.</p>
WQALLCRI	A4	Hex, 1 byte	<p>Log Record Selection Criteria:</p> <p>X'00' Indicates the return DB2CDC and UR control log records. (WQALLCR0)</p> <p>X'FF' (WQALLCRA)</p>
WQALLOPT	A5	Hex, 1 byte	<p>Processing Options relating to decompression:</p> <p>X'00' Indicates that decompression should not occur. (WQALLOP0)</p> <p>X'01' Indicates to decompress the log records if they are compressed. (WQALLOP1)</p>

Table 87. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALFLTR	A6	Hex, 1 byte	<p>For an IFCID 0316 request, WQALFLTR identifies the filter method:</p> <p><b>X'00'</b> Indicates no filtering. This value tells DB2 to return information for as many cached statements as fit in the return area. (WQALFLT0)</p> <p><b>X'01'</b> Indicates that DB2 returns information about the cached statements that have the highest values for a particular statistics field. The statistics field is specified in WQALFFLD. DB2 returns information for as many statements as fit in the return area. (WQALFLT1)</p> <p><b>Example:</b> If the return is large enough for information about 10 statements, the statements with the ten highest values for the specified statistics field are reported.</p> <p><b>X'02'</b> Indicates that DB2 returns information about the cached statements that exceed a threshold value for a particular statistics field. The name of the statistics field is specified in WQALFFLD. The threshold value is specified in WQALFVAL. DB2 returns information for as many qualifying statements as fit in the return area.</p> <p><b>X'04'</b> Indicates that DB2 returns information about a single cached statement. The application provides the four-byte cached statement identifier in field WQALSTID. An IFCID 0316 request with this qualifier is intended for use with IFCID 0172 or IFCID 0196, to obtain information about the statements that are involved in a timeout or deadlock.</p> <p>For an IFCID 0317 request, WQALFLTR identifies the filter method:</p> <p><b>X'04'</b> Indicates that DB2 returns information about a single cached statement. The application provides the four-byte cached statement identifier in field WQALSTID. An IFCID 0317 request with this qualifier is intended for use with IFCID 0172 or IFCID 0196, to obtain information about the statements that are involved in a timeout or deadlock.</p> <p>For an IFCID 0306 request, WQALFLTR indicates whether DB2 merges log records in a data sharing environment:</p> <p><b>X'00'</b> Indicates that DB2 merges log records from data sharing members. (WQALMERG)</p> <p><b>X'03'</b> Indicates that DB2 does not merge log records from data sharing members. (WQALNOMR)</p>

Table 87. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALFFLD	A7	Character, 1 byte	<p>For an IFCID 0316 request, when WQALFLTR is X'01' or X'02', this field specifies the statistics field that is used to determine the cached statements about which DB2 reports. You can enter the following values:</p> <p>'A' The accumulated elapsed time (QW0316AE). This option is valid only when QWALFLTR=X'01'. (WQALFFLA)</p> <p>'B' The number of buffer reads (QW0316NB). (WQALFFLB)</p> <p>'C' The accumulated CPU time (QW0316CT). This option is valid only when QWALFLTR=X'01'. (WQALFFLC)</p> <p>'E' The number of executions of the statement (QW0316NE). (WQALFFLE)</p> <p>'G' The number of GETPAGE requests (QW0316NG). (WQALFFG)</p> <p>'I' The number of index scans (QW0316NI). (WQALFFLI)</p> <p>'L' The number of parallel groups (QW0316NL). (WQALFFLL)</p> <p>'P' The number of rows processed (QW0316NP). (WQALFFLP)</p> <p>'R' The number of rows examined (QW0316NR). (WQALFFLR)</p> <p>'S' The number of sorts performed (QW0316NS). (WQALFFLS)</p> <p>'T' The number of table space scans (QW0316NT). (WQALFFLT)</p> <p>(Continued in the following row.)</p>

Table 87. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALFFLD (continued)	A7	Character, 1 byte	<p>(Continued from the previous row.)</p> <p>'W' The number of buffer writes (QW0316NW). (WQALFFLW)</p> <p>'X' The number of times that a RID list was not used because the number of RIDs would have exceeded one or more internal DB2 limits (QW0316RT).</p> <p>'Y' The number of times that a RID list was not used because not enough storage was available (QW0316RS). (WQALFFLY)</p> <p>'1' The accumulated wait time for synchronous I/O (QW0316W1). This option is valid only when QWALFLTR=X'01'. (WQALFFL1)</p> <p>'2' The accumulated wait time for lock and latch requests (QW0316W2). This option is valid only when QWALFLTR=X'01'. (WQALFFL6).</p> <p>'3' The accumulated wait time for a synchronous execution unit switch (QW0316W3). This option is valid only when QWALFLTR=X'01'. (WQALFFL3)</p> <p>'4' The accumulated wait time for global locks (QW0316W4). This option is valid only when QWALFLTR=X'01'. (WQALFFL4)</p> <p>'5' The accumulated wait time for read activity by another thread (QW0316W5). This option is valid only when QWALFLTR=X'01'. (WQALFFL5)</p> <p>'6' The accumulated wait time for write activity by another thread (QW0316W6). This option is valid only when QWALFLTR=X'01'. (WQALFFL6)</p>
WQALFVAL	A8	Signed 4-byte integer	<p>For an IFCID 0316 request, when WQALFLTR is X'02', this field and WQALFFLD determine the cached statements about which DB2 reports.</p> <p>To be eligible for reporting, a cached statement must have a value for WQALFFLD that is no smaller than the value that you specify in WQALFVAL. DB2 reports information on as many eligible statements as fit in the return area.</p>
WQALSTNM	AC	Character, 16 bytes	<p>For an IFCID 0317 request, when WQALFLTR is not X'04', this field specifies the name of a cached statement about which DB2 reports. This is a name that DB2 generates when it caches the statement. To obtain this name, issue a READS request for IFCID 0316. The name is in field QW0316NM. This field and WQALSTID uniquely identify a cached statement.</p>

Table 87. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.


Name	Hex offset	Data type	Description
WQALSTID	BC	Unsigned 4-byte integer	<p>For an IFCID 0316 or IFCID 0317 request, this field specifies the ID of a cached statement about which DB2 reports. DB2 generates this ID when it caches the statement.</p> <p>To obtain the ID, use the following options:</p> <ul style="list-style-type: none"> <li>For an IFCID 0317 request, when WQALFLTR is not X'04', obtain this ID by issuing a READS request for IFCID 0316. The ID is in field QW0316TK. This field and WQALSTNM uniquely identify a cached statement.</li> <li>For an IFCID 0316 or IFCID 0317 request, when WQALFLTR is X'04', obtain this ID by issuing a READS request for IFCID 0172 or IFCID 0196. The ID is in field QW0172H9 (cached statement ID for the holder in a deadlock), QW0172W9 (cached statement ID for the waiter in a deadlock), or QW0196H9 (cached statement ID of the holder in a timeout). This field uniquely identifies a cached statement.</li> </ul>
WQALEUID	C0	Character, 16 bytes	The end user's workstation user ID. This value can be different from the authorization ID that is used to connect to DB2.
WQALEUTX	D0	Character, 32 bytes	The name of the transaction or application that the end user is running. This value identifies the application that is currently running, not the product that is used to run the application.
WQALEUWN	F0	Character, 18 bytes	The end user's workstation name. This value can be different from the authorization ID used to connect to DB2.
WQALPLOC	102	Character, 128 bytes	The package location name.
WQALPCOL	182	Character, 128 bytes	The collection name of the package that the end user is running.
WQALPPNM	202	Character, 128 bytes	The program name of the package that the end user is running.
WQALROLE	282	Character, 128 bytes	The connection role of the end user. This field contains binary zeros if the client does not supply this information.

Table 87. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
WQALEXCD	382	Character, 4 bytes	This field specifies the level of exclude filtering to be performed, according to the following values:
			X'80000000'
			Specifies exclude filtering by plan name (WQALPLAN)
			X'40000000'
			Specifies exclude filtering by current authorization ID (WQALAUTH)
			X'20000000'
			Specifies exclude filtering by original authorization ID (WQALOPID)
			X'10000000'
			Specifies exclude filtering by connection name (WQALCONN)
			X'08000000'
			Specifies exclude filtering by correlation name (WQALCORR)
			X'04000000'
			Specifies exclude filtering by cached statement ID (WQALASID)
			X'02000000'
			Specifies exclude filtering by location name (WQALLUWI)
			X'01000000'
			Specifies exclude filtering by location name (WQALLOCN)
			X'00800000'
			Specifies exclude filtering by workstation ID (WQALEUID)
			X'00400000'
			Specifies exclude filtering by transaction or application name (WQALEUTX)
			X'00200000'
			Specifies exclude filtering by workstation name (WQALEUWN)
			X'00100000'
			Specifies exclude filtering by package location name (WQALEPLOC)
			X'00080000'
			Specifies exclude filtering by package collection name (WQALEPCOL)
			X'00040000'
			Specifies exclude filtering by package program name (WQALEPPNM)
			X'00002000'
			Specifies exclude filtering by connection role (WQALEROLE)

Table 87. Qualification area fields (continued). This area is mapped by the assembler mapping macro DSNDWQAL.

Name	Hex offset	Data type	Description
<b>Note:</b>			
1. The only valid filters for DDF/RRSAF 147/148 rollup records are WQALEUID, WQALEUTX, and WQALEUWN. For a 147/148 request, DDF/RRSAF records are not processed if any of the following WQAL fields are not X'00':			
• WQALACE			
• WQALAUTH			
• WQALOPID			
• WQALPLAN			
• WQALCORR			
• WQALLUWI			
• WQALLOCN			
• WQALASID			
• WQALCONN			
• WQALPLOC			
• WQALPCOL			
• WQALPPNM			
• WQALROLE			

If your monitor program does not initialize the qualification area, the READS request is denied. 

## Which qualifications are used for READS requests issued through IFI?

Not all qualifications are used for all IFCIDs.

 Table 88 lists the qualification fields that are used for each IFCID.

Table 88. Qualification fields for IFCIDs

These IFCIDs...	Are allowed to use these qualification fields
0124, 0147, 0148, 0150	WQALACE WQALAIT2 WQALPLAN <sup>1</sup> WQALAUTH <sup>1</sup> WQALOPID <sup>1</sup> WQALCONN <sup>1</sup> WQALCORR <sup>1</sup> WQALASID WQALLUWI <sup>1</sup> WQALLOCN <sup>1</sup> WQALEUID WQALEUTX WQALEUWN WQALROLE <sup>1</sup>
0129	WQALLTYP WQALLMOD WQALLRBA WQALLNUM
0149	WQALREST WQALHASH
0150	WQALFOPT
0185	WQALCDCD

Table 88. Qualification fields for IFCIDs (continued)

These IFCIDs...	Are allowed to use these qualification fields
0199, 0254	WQALGBP <sup>N2</sup>
0306	WQALFLTR WQALLMOD WQALLRBA WQALLCRI WQALLOPT
0316	WQALFLTR WQALFFLD WQALFVAL WQALSTID
0317	WQALFLTR WQALSTNM WQALSTID

**Note:**

1. DB2 allows you to partially qualify a field and fill the rest of the field with binary zero. For example, the 12-byte correlation value for a CICS thread contains the 4-character CICS transaction code in positions 5-8. Assuming a CICS transaction code of AAAA, the following hexadecimal *qual-area* correlation qualification can be used to find the first transaction with a correlation value of AAAA in positions 5-8:  
X'00000000C1C1C1C100000000'.
2. X'00' in this field indicates a pattern-matching character. X'00' in any position of the field indicates that IFI collects data for buffer pools whose names contain any character in that position and all following positions.

**PSPI**

## Synchronous data and READS requests through IFI

You can read certain types of IFI records synchronously.

**PSPI**

Identified by IFCID, these records are:

- 0001** Statistical data on the systems services address space, including task control block (TCB) and service request block (SRB) times for system services, database services, including DDF statistics, and Internal Resource Lock Manager (IRLM) address spaces.
- 0002** Statistical data on the database services address space.
- 0104** Information on the current active log data sets.
- 0106** Static system parameters.
- 0124** An active SQL snapshot that provides status information about:
  - The process
  - The SQL statement text
  - The relational data system input parameter list (RDI) block
  - Certain bind and locking information

You can obtain a varying amount of data because the request requires the process to be connected to DB2, have a cursor table allocated (RDI and status information is provided), and be active in DB2 (SQL text is provided if available). The SQL text that is provided does not include the SQL host variables, and is truncated at 4000 bytes.


For dynamic SQL, IFI provides the original SQL statement. The RDISTYPE field contains the actual SQL function taking place. For example, for a SELECT statement, the RDISTYPE field can indicate that an open cursor, fetch, or other function occurred. For static SQL, you can see the DECLARE CURSOR statement, and the RDISTYPE indicates the function. The RDISTYPE field is mapped by mapping macro DSNXRDI.

- 0129 Returns one or more VSAM control intervals (CIs) that contain DB2 recovery log records. You can use IFI to return these records for use in remote site recovery.
- 0147 An active thread snapshot that provides a status summary of processes at a DB2 thread or non-thread level.
- 0148 An active thread snapshot that provides more detailed status of processes at a DB2 thread or non-thread level.
- 0149 Information that indicates who (the thread identification token) is holding locks and waiting for locks on a particular resource and hash token. The data is in the same format as IFCID 0150.
- 0150 All the locks held and waited on by a given user or owner (thread identification token).
- 0185 Data descriptions for each table for which captured data is returned on this DATA request. IFCID 0185 data is only available through a propagation exit routine that is triggered by DB2.
- 0199 Information about buffer pool usage by DB2 data sets. DB2 reports this information for an interval that you specify in the DATASET STATS TIME field of installation panel DSNTIPN. At the beginning of each interval, DB2 resets these statistics to 0.
- 0202 Dynamic system parameters.
- 0217 Storage detail record for the DBM1 address space.
- 0225 Storage summary record for the DBM1 address space.
- 0230 Global statistics for data sharing.
- 0234 User authorization information.
- 0254 Group buffer pool usage in the data sharing group.
- 0306 Returns compressed or decompressed log records in both a data sharing or non data-sharing environment. For IFCID 306 requests, your program's return area must reside in ECSA key 7 storage with the IFI application program running in key 0 supervisor state. The IFI application program must set the eye catcher to "I306" before making the IFCID 306 call. See "Instrument facility communications area (IFCA)" on page 466 for more information about the instrumentation facility communication area (IFCA) and what is expected of the monitor program.
- 0316 Returns information about the contents of the dynamic statement cache. The IFI application can request information for all statements in the cache, or provide qualification parameters to limit the data returned. DB2 reports the following information about a cached statement:
  - A statement name and ID that uniquely identify the statement
  - If IFCID 0318 is active, performance statistics for the statement
  - The first 60 bytes of the statement text
- 0317 Returns the complete text of an SQL statement in the dynamic statement

cache and the PREPARE attributes string. You must provide the statement name and statement ID from IFCID 0316 output. For more information about using IFI to obtain information about the dynamic statement cache, see “Using READS calls to monitor the dynamic statement cache.”


**0345** Provides the name of the DSNHDECP module that is being used by the subsystem.

**0346** Monitors all invoked packages for a DB2 thread.

For more information about IFCID field descriptions, see the mapping macros in *prefix.SDSNMACS*. See also Chapter 21, “Using DB2 Trace to monitor performance,” on page 429 and Chapter 36, “Interpreting DB2 trace output,” on page 621 for additional information. 

## Using READS calls to monitor the dynamic statement cache

You can use READS requests from an IFI application to monitor the contents of the dynamic statement cache, and optionally, to see some accumulated statistics for those statements.

 This strategy can help you detect and diagnose performance problems for those cached dynamic SQL statements.

An IFI program that monitors the dynamic statement cache should include these steps:

1. Acquire and initialize storage areas for common IFI communication areas.
2. Issue an IFI COMMAND call to start performance trace class 30, 31, or 32 for IFCID 0318. This step enables statistics collection for statements in the dynamic statement cache.
3. Put the IFI program into a wait state. During this time, SQL applications in the subsystem execute dynamic SQL statements by using the dynamic statement cache.
4. Resume the IFI program after enough time has elapsed for a reasonable amount of activity to occur in the dynamic statement cache.
5. Set up the qualification area for a READS call for IFCID 0316
6. Set up the IFCID area to request data for IFCID 0316.
7. Issue an IFI READS call to retrieve the qualifying cached SQL statements.
8. Examine the contents of the return area.

For a statement with unexpected statistics values:

- a. Obtain the statement name and statement ID from the IFCID 0316 data.
  - b. Set up the qualification area for a READS call for IFCID 0317.
  - c. Set up the IFCID area to request data for IFCID 0317.
  - d. Issue a READS call for IFCID 0317 to get the entire text of the statement.
  - e. Obtain the statement text from the return area.
  - f. Use the statement text to execute an SQL EXPLAIN statement.
  - g. Fetch the EXPLAIN results from the PLAN\_TABLE.
9. Issue an IFI COMMAND call to stop performance trace class 30 for IFCID 0318.

An IFI program that monitors deadlocks and timeouts of cached statements should include these steps:

1. Acquire and initialize storage areas for common IFI communication areas.
2. Issue an IFI COMMAND call to start performance trace class 30 for IFCID 0318. This step enables statistics collection for statements in the dynamic statement cache.
3. Start statistics trace class 3, or performance trace class 6, for IFCID 0172 to monitor deadlocks, or for IFCID 0196 to monitor timeouts.
4. Put the IFI program into a wait state. During this time, SQL applications in the subsystem execute dynamic SQL statements by using the dynamic statement cache.
5. Resume the IFI program when a deadlock or timeout occurs.
6. Issue a READA request to obtain IFCID 0172 or IFCID 0196 trace data.
7. Obtain the cached statement ID of the statement that was involved in the deadlock or timeout from the IFCID 0172 or IFCID 0196 trace data. Using the statement ID, set up the qualification area for a READS call for IFCID 0316 or IFCID 0317..
8. Set up the IFCID area to request data for IFCID 0316 or IFCID 0317.
9. Issue an IFI READS call to retrieve the qualifying cached SQL statement.
10. Examine the contents of the return area.
11. Issue an IFI COMMAND call to stop monitor trace class 1.
12. Issue an IFI COMMAND call to stop performance trace class 30 for IFCID 0318 and performance trace class 3 for IFCID 0172 or IFCID 0196.

**PSPI**

## Controlling collection of dynamic statement cache statistics with IFCID 0318

The collection of statistics for statements in the dynamic statement cache can increase the processing cost for those statements.

**PSPI**

To minimize this increase, use IFCID 0318 to enable and disable the collection of dynamic statement cache statistics. When IFCID 0318 is inactive, DB2 does not collect those statistics. DB2 tracks the statements in the dynamic statement cache, but does not accumulate the statistics as those statements are used. When you are not actively monitoring the cache, you should turn off the trace for IFCID 0318.

If you issue a READS call for IFCID 0316 while IFCID 0318 is inactive, DB2 returns identifying information for all statements in the cache, but returns 0 in all the IFCID 0316 statistics counters.

When you stop or start the trace for IFCID 0318, DB2 resets the IFCID 0316

statistics counters for all statements in the cache to 0.

**PSPI**

---

## Using READA requests through IFI

The READA function allows a monitor program to asynchronously read data that has accumulated in an OP $n$  buffer.

**PSPI**

You can use a monitor trace that uses any one of eight online performance monitor destinations, OP $n$ , (where  $n$  is equal to a value from 1 to 8). Typically, the

destination of  $OP_n$  is only used with commands issued from a monitor program. For example, the monitor program can pass a specific online performance monitor destination ( $OP_1$ , for example) on the START TRACE command to start asynchronous trace data collection.

If the monitor program passes a generic destination of  $OP_n$ , the instrumentation facility assigns the next available buffer destination slot and returns the  $OP_n$  destination name to the monitor program. To avoid conflict with another trace or program that might be using an  $OP$  buffer, you should use the generic  $OPX$  specification when you start tracing. You can then direct the data to the destination specified by the instrumentation facility with the START or MODIFY TRACE commands.

Sometimes, however, you should use a specific  $OP_n$  destination initially:

- When you plan to start numerous asynchronous traces to the same  $OP_n$  destination. To do this, you must specify the  $OP_n$  destination in your monitor program. The  $OP_n$  destination started is returned in the IFCA.
- When the monitor program specifies that a particular monitor class (defined as available) together with a particular destination (for example  $OP_7$ ) indicates that certain IFCIDs are started. An operator can use the DISPLAY TRACE command to determine which monitors are active and what events are being traced.

**Buffering data:** To have trace data go to the  $OP_n$  buffer, you must start the trace from within the monitor program. After the trace is started, DB2 collects and buffers the information as it occurs. The monitor program can then issue a read asynchronous (READA) request to move the buffered data to the monitor program. The buffering technique ensures that the data is not being updated by other users while the buffer is being read by the READA caller. For more information, see “Data integrity and IFI” on page 473.



**Possible data loss:** You can activate all traces and have the trace data buffered. However, this plan is definitely not recommended because performance might suffer and data might be lost.

Data loss occurs when the buffer fills before the monitor program can obtain the data. DB2 does not wait for the buffer to be emptied, but, instead, informs the monitor program on the next READA request (in the IFCARLC field of the IFCA) that the data has been lost. The user must have a high enough dispatching priority that the application can be posted and then issue the READA request before

significant data is lost. 

## Authorization for READA requests through IFI

On a READA request the application program must own the specified destination buffer, or the request is denied.

 You can obtain ownership of a storage buffer by issuing a START TRACE to an  $OP_n$  destination. If the primary authorization ID or one of the secondary authorization IDs of the process does not have MONITOR1 or MONITOR2 privilege, the request is denied. READA requests are checked for authorization once for each user of the thread. (Several users can use the same thread, but an authorization check is performed each time the user of the thread changes.) 

## Syntax for READA requests through IFI

READA requests that are issued through IFI have the following syntax.

### PSPI

```
CALL DSNWLI,('READA ',ifca,return-area),VL
```


#### *ifca*

Contains information about the OP<sub>n</sub> destination and the ownership token value (IFCAOWNER) at call initiation. After the READA call completes, the IFCA contains the return code, reason code, the number of bytes moved to the return area, the number of bytes not moved to the return area if the area was too small, and the number of records lost. See “Common communication areas for IFI calls” on page 466 for a description of the IFCA.

#### *return-area*

Contains the varying-length records that are returned by the instrumentation facility. If the return area is too small, as much of the output as can fit is placed into the area (a complete varying-length record). Reason code 00E60802 is returned in cases where the monitor program's return area is not large enough to hold the returned data. See “Return area” on page 470 for a description of the return area.

IFI allocates up to eight OP buffers upon request from storage above the line in extended CSA. IFI uses these buffers to store trace data until the owning application performs a READA request to transfer the data from the OP buffer to the application's return area. An application becomes the owner of an OP buffer when it issues a START TRACE command and specifies a destination of OP<sub>n</sub> or OPX. Each buffer can be of size 256 KB to 16 MB. IFI allocates a maximum of 16 MB of storage for each of the eight OP buffers. The default monitor buffer size is determined by the MONSIZE parameter in the

DSNZPARM module. 

## Asynchronous data and READA requests through IFI

DB2 buffers all IFCID data that is activated by the START TRACE command and passes it to a monitor program on a READA request.


### PSPI

The IFCID events include all of the following:

- Serviceability
- Statistics
- Accounting
- Performance
- Audit data
- IFCIDs defined for the IFI write function

IFCID events are discussed in Chapter 21, “Using DB2 Trace to monitor performance,” on page 429.

Your monitor program can request an asynchronous buffer, which records trace data as trace events occur. The monitor program is then responsible for unloading the buffer on a timely basis. One method is to set a timer to wake up and process the data. Another method is to use the buffer information area on a START TRACE command request, shown in Table 86 on page 445, to specify an ECB address to

post when a specified number of bytes have been buffered. 

## How DB2 processes READA requests through IFI

You can monitor DB2 accounting and display the information on a terminal.

**PSPI** The following depicts the logic flow for monitoring DB2 accounting and for displaying the information on a terminal:

1. Initialize.
2. Use GETMAIN to obtain a storage area equal to BUFSIZE.
3. Start an accounting trace by issuing a DB2 START TRACE=ACCTG DEST=OPX command through IFI indicating to wake up this routine by a POST whenever the buffer is 20% full.
4. Check the status in the IFCA to determine if the command request was successful.
5. WAIT for the buffer to be posted.
6. Clear the post flag.
7. Call IFI to obtain the buffer data via a READA request.
8. Check the status of the IFCA to determine if the READA request was successful.
9. De-block the information provided.
10. Display the information on a terminal.
11. Loop back to the WAIT.

**PSPI**

---

## Using WRITE requests through IFI

A monitor program can write information to a DB2 trace destination by issuing a write (WRITE) request for a specific IFCID.

**PSPI** A WRITE request is written to a destination that is activated by a START TRACE command.

### Recommendations:

- If your site uses the IFI WRITE function, establish usage procedures and standards. Procedures ensure that the correct IFCIDs are active when DB2 is performing the WRITE function. Standards determine the records and record formats that a monitor program sends to DB2.
- Because your site can use one IFCID to contain many different records, place your site's record type and sub-type in the first fields in the data record. **PSPI**

## Authorization for WRITE requests through IFI

WRITE requests are not checked for authorization, but a DB2 trace must be active for the IFCID being written.

**PSPI** If the IFCID is not active, the request is denied. For a WRITE request, no other authorization checks are made. **PSPI**

## Syntax for WRITE requests through IFI

WRITE requests that are issued through IFI have the following syntax.

### PSPI

```
CALL DSNWLI,('WRITE ',ifca,output-area,ifcid-area),VL
```

The write function must specify an IFCID area. The data that is written is defined and interpreted by your site.

#### *ifca*

Contains information regarding the success of the call. See “Instrument facility communications area (IFCA)” for a description of the IFCA.

#### *output-area*

Contains the varying-length of the monitor program’s data record to be written. See “Output area” on page 471 for a description of the output area.

#### *ifcid-area*

Contains the IFCID of the record to be written. Only the IFCIDs that are defined to the write function (see Table 89) are allowed. If an invalid IFCID is specified or the IFCID is not active (not started by a TRACE command), no data is written. See Table 89 for IFCIDs that can be used by the write function.

Table 89. Valid IFCIDs for WRITE function

IFCID (decimal)	IFCID (hex)	Trace type	Class	Comment
0146	0092	Auditing	9	Write to IFCID 146
0151	0097	Accounting	4	Write to IFCID 151
0152	0098	Statistics	2	Write to IFCID 152
0153	0099	Performance	1	Background events and write to IFCID 153
0154	009A	Performance	15	Write to IFCID 154
0155	009B	Monitoring	4	Write to IFCID 155
0156	009C	Serviceability	6	Reserved for user-defined serviceability trace

See “IFCID area” on page 471 for a description of the IFCID area.

### PSPI

## Common communication areas for IFI calls

The following communication areas are used on all IFI calls.

### Instrument facility communications area (IFCA)

The program’s instrumentation facility communication area (IFCA) is a communications area between the monitor program and IFI. A required parameter on all IFI requests, the IFCA contains information about the success of the call in its return code and reason code fields.

### PSPI

**The monitor program is responsible for allocating storage for the IFCA and initializing it.** The IFCA must be initialized to binary zeros and the eye catcher, 4-byte owner field, and length field must be set by the monitor program. Failure to properly initialize the IFCA results in denying any IFI requests.

The monitor program is also responsible for checking the IFCA return code and reason code fields to determine the status of the request.

The IFCA fields are described in the following table.

*Table 90. Instrumentation facility communication area.* The IFCA is mapped by assembler mapping macro DSNDIFCA.

Name	Hex offset	Data type	Description
IFCALEN	0	Hex, 2 bytes	Length of IFCA.
IFCAFLGS	2	Hex, 1 byte	Processing flags. <ul style="list-style-type: none"> <li>IFCAGLBL, X'80'</li> </ul> This bit is on if an IFI request is to be processed on all members of a data sharing group.
	3	Hex, 1 byte	Reserved.
IFCAID	4	Character, 4 bytes	Eye catcher for block, IFCA.
IFCAOWNR	8	Character, 4 bytes	Owner field, provided by the monitor program. This value is used to establish ownership of an OP $n$ destination and to verify that a requester can obtain data from the OP $n$ destination. This is <i>not</i> the same as the owner ID of a plan.
IFCARC1	C	4-byte signed integer	Return code for the IFI call. Binary zero indicates a successful call. For a return code of 8 from a COMMAND request, the IFCAR0 and IFCAR15 values contain more information.
IFCARC2	10	4-byte signed integer	Reason code for the IFI call. Binary zero indicates a successful call.
IFCABM	14	4-byte signed integer	Number of bytes moved to the return area. A non-zero value in this field indicates information was returned from the call. Only complete records are moved to the monitor program area.
IFCABNM	18	4-byte signed integer	Number of bytes that did not fit in the return area and still remain in the buffer. Another READA request retrieves that data. Certain IFI requests return a known quantity of information. Other requests terminate when the return area is full.
	1C	4-byte signed integer	Reserved.
IFCARLC	20	4-byte signed integer	Indicates the number of records lost prior to a READA call. Records are lost when the OP buffer storage is exhausted before the contents of the buffer are transferred to the application program via an IFI READA request. Records that do not fit in the OP buffer are not written and are counted as records lost.
IFCAOPN	24	Character, 4 bytes	Destination name used on a READA request. This field identifies the buffer requested, and is required on a READA request. Your monitor program must set this field. The instrumentation facility fills in this field on START TRACE to an OP $n$ destination from an monitor program. If your monitor program started multiple OP $n$ destination traces, the first one is placed in this field. If your monitor program did not start an OP $n$ destination trace, the field is not modified. The OP $n$ destination and owner ID are used on subsequent READA calls to find the asynchronous buffer.
IFCAOPNL	28	2-byte signed integer	Length of the OP $n$ destinations started. On any command entered by IFI, the value is set to X'0004'. If an OP $n$ destination is started, the length is incremented to include all OP $n$ destinations started.
	2A	2-byte signed integer	Reserved.
IFCAOPNR	2C	Character, 8 fields of 4 bytes each	Space to return 8 OP $n$ destination values.

Table 90. Instrumentation facility communication area (continued). The IFCA is mapped by assembler mapping macro DSNDIFCA.

Name	Hex offset	Data type	Description
IFCATNOL	4C	2-byte signed integer	Length of the trace numbers plus 4. On any command entered by IFI the value is set to X'0004'. If a trace is started, the length is incremented to include all trace numbers started.
	4E	2-byte signed integer	Reserved.
IFCATNOR	50	Character, 8 fields of 2 bytes each.	Space to hold up to eight EBCDIC trace numbers that were started. The trace number is required if the MODIFY TRACE command is used on a subsequent call.
IFCADL	60	Hex, 2 bytes	Length of diagnostic information.
	62	Hex, 2 bytes	Reserved.

Table 90. Instrumentation facility communication area (continued). The IFCA is mapped by assembler mapping macro DSNDIFCA.

Name	Hex offset	Data type	Description
IFCADD	64	Character, 80 bytes	<p>Diagnostic information.</p> <ul style="list-style-type: none"> <li>• IFCAFCI, offset 64, 6 bytes This contains the RBA of the first CI in the active log if IFCARC2 is 00E60854.</li> <li>• IFCAR0, offset 6C, 4 bytes For COMMAND requests, this field contains -1 or the return code from the component that executed the command.</li> <li>• IFCAR15, offset 70, 4 bytes For COMMAND requests, this field contains one of the following values: <ul style="list-style-type: none"> <li>0 The command completed successfully.</li> <li>4 Internal error.</li> <li>8 The command was not processed because of errors in the command.</li> <li>12 The component that executed the command returned the return code in IFCAR0.</li> <li>16 An abend occurred during command processing. Command processing might be incomplete, depending on when the error occurred. See IFCAR0 for more information.</li> <li>20 Response buffer storage was not available. The command completed, but no response messages are available. See IFCAR0 for more information.</li> <li>24 Storage was not available in the DSNMSTR address space. The command was not processed.</li> <li>28 CSA storage was not available. If a response buffer is available, the command might have partially completed. See IFCAR0 for more information.</li> <li>32 The user is not authorized to issue the command. The command was not processed.</li> </ul> </li> <li>• IFCAGBPN, offset 74, 8 bytes This is the group buffer pool name in error if IFCARC2 is 00E60838 or 00E60860</li> <li>• IFCABSRQ, offset 88, 4 bytes This is the size of the return area required when the reason code is 00E60864.</li> <li>• IFC AHLRS, offset 8C, 6 bytes This field can contain the highest LRSN or log RBA in the active log (when WQALLMOD is 'H'). Or, it can contain the RBA of the log CI given to the Log Exit when the last CI written to the log was not full, or an RBA of zero (when WQALLMOD is 'P').</li> </ul>
IFCAGRSN	98	Four-byte signed integer	Reason code for the situation in which an IFI calls requests data from members of a data sharing group, and not all the data is returned from group members.
IFCAGBM	9C	Four-byte signed integer	Total length of data that was returned from other data sharing group members and fit in the return area.

Table 90. Instrumentation facility communication area (continued). The IFCA is mapped by assembler mapping macro DSNDIFCA.

Name	Hex offset	Data type	Description
IFCAGBNM	A0	Four-byte signed integer	Total length of data that was returned from other data sharing group members and did not fit in the return area..
IFCADMBR	A4	Character, 8 bytes	Name of a single data sharing group member on which an IFI request is to be executed. Otherwise, this field is blank. If this field contains a member name, DB2 ignores field IFCAGLBL.
IFCARMBR	AC	Character, 8 bytes	Name of the data sharing group member from which data is being returned. DB2 sets this field in each copy of the IFCA that it places in the return area, not in the IFCA of the application that makes the IFI request.



## Return area

You must specify a return area on all READA, READS, and COMMAND requests.



IFI uses the return area to return command responses, synchronous data, and asynchronous data to the monitor program. Table 91 describes the return area.

Table 91. Return area

Hex offset	Data type	Description
0	Signed 4-byte integer	The length of the return area, plus 4. This must be set by the monitor program. The valid range for READA requests is 100 to 1048576 (X'00000064' to X'00100000'). The valid range for READS requests is 100 to 2147483647 (X'00000064' to X'7FFFFFFF').
4	Character, varying-length	DB2 places as many varying-length records as it can fit into the area following the length field. The monitor program's length field is not modified by DB2. Each varying-length trace record has either a 2-byte or 4-byte length field, depending on the high-order bit. If the high-order bit is on, the length field is 4 bytes. If the high-order bit is off, the length field is 2 bytes. In this case, the third byte indicates whether the record is spanned, and the fourth byte is reserved.  After a COMMAND request, the last character in the return area is a new-line character (X'15').

The following table shows the return area for IFCID 0306.


Table 92. Return area using IFCID 0306

Hex offset	Data type	Description
0	Signed four-byte integer	The length of the return area.
4	Character, 4 bytes	The eye catcher, a constant, I306. Beginning of QW0306OF mapping.
8	Character, 60 bytes	Reserved.
44	Signed four-byte integer	The length of the returned data.

The destination header for data that is returned on a READA or READS request is mapped by macro DSNDQWIW or the header QW0306OF for IFCID 306 requests. Please refer to *prefix.SDSNIVPD(DSNWMSGs)* for the format of the trace record

and its header. The size of the return area for READA calls should be as large the size specified with the BUFSIZE keyword on the START TRACE command.

Data returned on a COMMAND request consists of varying-length segments (X'xxxxrrrr' where the length is 2 bytes and the next 2 bytes are reserved), followed by the message text. More than one record can be returned. The last character in the return area is a new-line character (X'15').

The monitor program must compare the number of bytes moved (IFCABM in the IFCA) to the sum of the record lengths to determine when all records have been processed. 

IFCID area

You must specify the IFCID area on READS and WRITE requests.

 The IFCID area contains the IFCIDs to process. The following table shows the IFCID area.


Table 93. IFCID area

Hex Offset	Data type	Description
0	Signed two-byte integer	Length of the IFCID area, plus 4. The length can range from X'0006' to X'0044'. For WRITE requests, only one IFCID is allowed, so the length must be set to X'0006'.  For READS requests, you can specify multiple IFCIDs. If so, you must be aware that the returned records can be in a different sequence than requested and some records can be missing.
2	Signed two-byte integer	Reserved.
4	Hex, <i>n</i> fields of 2 bytes each	The IFCIDs to be processed. Each IFCID is placed contiguous to the previous IFCID for a READS request. The IFCIDs start at X'0000' and progress upward. You can use X'FFFF' to signify the last IFCID in the area to process.



Output area

The output area is used on command and WRITE requests

 . The first two bytes contain the length of the monitor program's record to write or the DB2 command to be issued, plus 4 additional bytes. The next two bytes are reserved. You can specify any length from 10 to 4096 (X'000A0000' to X'10000000'). The rest of the area is the actual command or record text.

**Example:** In an assembler program, a START TRACE command is formatted in the following way:

```
DC X'002A0000'          LENGTH INCLUDING LL00 + COMMAND
DC CL37'-STA TRACE(MON) DEST(OPX) BUFSIZE(256)'
```



---

## Using IFI in a data sharing group

You can use IFI READA and READS calls in an application that runs on one member of a data sharing group to gather trace data from other members of the data sharing group.

**PSPI** You can also use an IFI COMMAND call to execute a command at another member of a data sharing group. In addition to the IFCA fields that you use for an IFI request for a single subsystem, you need to set or read the following fields for an IFI request for a data sharing group:

### IFCAGLBL

Set this flag on to indicate that the READS or READA request should be sent to all members of the data sharing group.

### IFCADMBR

If you want an IFI READS, READA, or COMMAND request to be executed at a single member of the data sharing group, assign the name of the group member to this field. If you specify a name in this field, DB2 ignores IFCAGLBL. If the name that you specify is not active when DB2 executes the IFI request, DB2 returns an error.

**Recommendation:** To issue a DB2 command that does not support SCOPE(GROUP) at another member of a data sharing group, set the IFCADMBR field and issue an IFI COMMAND.

### IFCARMBR

The name of the data sharing member that generated the data that follows the IFCA. DB2 sets this value in the copy of the IFCA that it places in the requesting program's return area.

### IFCAGRSN

A reason code that DB2 sets when not all data is returned from other data sharing group members.

### IFCAGBM

The number of bytes of data that other members of the data sharing group return and that the requesting program's return area can contain.

### IFCAGBNM

The number of bytes of data that members of the data sharing group return but that the requesting program's return area cannot contain.

As with READA or READS requests for single DB2 subsystems, you need to issue a START TRACE command before you issue the READA or READS request. You can issue START TRACE with the parameter SCOPE(GROUP) to start the trace at all members of the data sharing group. For READA requests, specify DEST(OPX) in the START TRACE command. DB2 collects data from all data sharing members and returns it to the OPX buffer for the member from which you issue the READA request.

If a new member joins a data sharing group while a trace with SCOPE(GROUP) is active, the trace starts at the new member.


After you issue a READS or READA call for all members of a data sharing group, DB2 returns data from all members in the requesting program's return area. Data from the local member is first, followed by the IFCA and data for all other members.

## Example

If the local DB2 is called DB2A, and the other two members in the group are DB2B and DB2C, the return area looks like this:

```
Data for DB2A
IFCA for DB2B (DB2 sets IFCARMBR to DB2B)
Data for DB2B
IFCA for DB2C (DB2 sets IFCARMBR to DB2C)
Data for DB2C
```


If an IFI application requests data from a single other member of a data sharing group (IFCADMBR contains a member name), the requesting program's return area contains the data for that member but no IFCA for the member. All information about the request is in the requesting program's IFCA.

Because a READA or READS request for a data sharing group can generate much more data than a READA or READS request for a single DB2, you need to increase the size of your return area to accommodate the additional data. 


---

## Data integrity and IFI

Although IFI displays DB2 statistics, agent status, and resource status data, it does not change or display DB2 database data.

 When a process retrieves data, information is moved from DB2 fetch-protected storage to the user's address space, or from the address space to DB2 storage, in the storage key of the requester. Data that is moved by the READA request is serialized so that only *clean data* is moved to the address space of the requester.



The serialization techniques used to obtain data for a given READA request might minimally degrade performance on processes that simultaneously store data into the instrumentation facility buffer. Failures during the serialization process are handled by DB2.

The DB2 structures that are searched on a READS request are validated before they are used. If the DB2 structures are updated while being searched, inconsistent data might be returned. If the structures are deleted while being searched, users might access invalid storage areas, causing an abend. If an abend does occur, the functional recovery routine of the instrumentation facility traps the abend and returns information about it to the application program's IFCA. 

---

## Auditing data and IFI

Starting, stopping, or modifying trace through IFI might cause changes to the events being traced for audit.

 Each time these trace commands are processed a record is sent to the destination processing the trace type. In the case of audit, the audit destination receives a record indicating a trace status has been changed. These records are IFCID 0004 and 0005. 

---

## Locking considerations for IFI

When you design your application to use IFI, consider the potential for locking delays, deadlocks, and time-out conflicts.

**PSPI** Locks are obtained for IFI in the following situations:

- When READS and READA requests are checked for authorization, short duration locks on the DB2 catalog are obtained. When the check is made, subsequent READS or READA requests are not checked for authorization. Remember, if you are using the access control exit routine, then that routine might be controlling the privileges that the monitor trace can use.
- When DB2 commands are submitted, each command is checked for authorization. DB2 database commands obtain additional locks on DB2 objects.

A program can issue SQL statements through an attachment facility and DB2 commands through IFI. This environment creates the potential for an application to deadlock or time-out with itself over DB2 locks acquired during the execution of SQL statements and DB2 database commands. You should ensure that all DB2 locks acquired by preceding SQL statements are no longer held when the DB2 database command is issued. You can do this by:

- Binding the DB2 plan with ACQUIRE(USE) and RELEASE(COMMIT) bind parameters
- Initiating a commit or rollback to free any locks your application is holding, before issuing the DB2 command

If you use SQL in your application, the time between commit operations should be short. For more information on locking, see Chapter 15, “Programming for concurrency,” on page 319. **PSPI**

---

## Recovery considerations for IFI

When an application program issues an IFI call, the requested function is immediately performed. However, if the application program subsequently abends, the IFI request is not backed out.

**PSPI** In contrast, requests that do not use IFI are committed and abended as usual. For example, if an IFI application program also issues SQL calls, a program abend causes the SQL activity to be backed out. **PSPI**

---

## Errors and IFI

While using IFI, you might encounter certain errors.

**PSPI**


- Connection failure, because the user is not authorized to connect to DB2
- Authorization failure, because the process is not authorized to access the DB2 resources specified

### IFI request failures

Requests sent through IFI can fail for a variety of reasons, including:

- One or more parameters are invalid.

- The IFCA area is invalid.
- The specified  $OP_n$  is in error.
- The requested information is not available.
- The return area is too small.

Return code and reason code information is stored in the IFCA in fields IFCARC1 and IFCARC2. 



---

## Chapter 25. Monitoring the use of IBM zIIPs

You can use various facilities to monitor the use of IBM System z9 Integrated Information Processors (IBM zIIPs) by DB2.

To monitor IBM zIIP usage:

- Use DB2 trace.

The DB2 accounting trace records provide information related to application programs including processor resources consumed by the application.

Accumulated zIIP CPU time is not accumulated in the existing class 1, class 2, and class 7 accounting fields. New accounting fields are defined to let you know how much time is spent on IBM zIIPs, as well as how much time zIIP eligible work overflowed to standard processors. CPU time eligible for offload to a zIIP processor is provided in the accounting records even if a zIIP is not installed or DB2 is not running on a z9™ mainframe.

- Use RMF.

The Resource Measurement Facility (RMF) provides information on zIIP usage to help you identify when to consider purchasing a zIIP or adding more zIIPs.

Also, SMF Type 72 records contain information on zIIP usage and fields in SMF Type 30 records let you know how much time is spent on zIIPs, as well as how much time was spent executing zIIP eligible work on standard processors. For more details about the new RMF support for zIIPs, refer to the *z/OS MVS Initialization and Tuning Reference*.

---

### IBM System z9 Integrated Information Processor

The *IBM zIIP* is a high-speed engine that better enables you to centralize your data on the mainframe, breaking down traditional walls between transactional data stores on the mainframe and the BI, ERP, and CRM applications that run on distributed computers.

The System z9 Integrated Information Processor (IBM zIIP) is a specialty engine that runs eligible database workloads. The IBM zIIP is designed to help free-up general computing capacity and lower software costs for select DB2 workloads such as business intelligence (BI), enterprise resource planning (ERP), and customer relationship management (CRM) on the mainframe.

With the zIIP capability, the System z9 mainframe helps minimize the need to maintain duplicate copies of data and provides better security between applications and data. New accounting fields and improvements to RMF also permit you to project the amount of CPU time eligible to run on an IBM zIIP without installing any IBM zIIPs.

Portions of DDF server thread processing, utility processing, and star join parallel child processing can be directed to an IBM zIIP. The amount of general-purpose processor savings will vary based on the amount of workload executed by the zIIP, among other factors.

#### Processes that can use IBM zIIP

- DDF server threads that process SQL requests from applications that access DB2 by TCP/IP.

- Parallel child processes. A portion of each child process executes under a dependent enclave SRB if it processes on behalf of an application that originated from an allied address space, or under an independent enclave SRB if the processing is performed on behalf of a remote application that accesses DB2 by TCP/IP. The enclave priority is inherited from the invoking allied address space for a dependent enclave or from the main DDF server thread enclave classification for an independent enclave.
- Utility index build and maintenance processes for the LOAD, REORG, and REBUILD INDEX utilities. A portion of the index build/maintenance runs under a dependent enclave SRB.

### **Processes that cannot use IBM zIIP**

- External stored procedures.
- Triggers or functions that join the enclave SRB using a TCB.
- DDF server threads that use SNA to connect to DB2.

### **Restrictions for data sharing**

In a data sharing environment, where DB2 Version 8 is part of the data sharing group, DRDA work running in the Version 7 member cannot run on the zIIP. Work running on the Version 8 member will run on the zIIP. IBM zIIP capacity is not included in the sysplex routing information returned to DB2 Connect Server. Accordingly, members without zIIPs might be favored over members with zIIPs. However, if work is routed to the zIIP-enabled system, the work will run on the zIIP.

---

## Chapter 26. Monitoring storage

---

### Monitoring I/O activity of data sets

The best way to monitor your I/O activity against database data sets is through IFCID 0199 (statistics class 8).

This IFCID can give you information such as the average write I/O delay or the maximum delay for a particular data set over the last statistics reporting interval. The same information is also reported in the DISPLAY BUFFERPOOL command with the LSTATS option. Furthermore, OMEGAMON reports IFCID 0199 in batch reports and in online monitoring.

More detailed information is available, with more overhead, with the I/O trace (performance class 4). If you want information about I/O activity to the log and BSDS data sets, use performance class 5.

You can also use RMF to monitor data set activity. SMF record type 42-6 provides activity information and response information for a data set over a time interval. These time intervals include the components of I/O time, such as IOS queue time. Using RMF incurs about the same overhead as statistics class 8.

For information on how to tune your environment to improve I/O performance, see “Reducing the time required for I/O operations” on page 18 and Synchronous I/O suspension time.

---

### Buffer Pool Analyzer

You can use the Buffer Pool Analyzer for z/OS to recommend buffer pool allocation changes and to do “what-if” analysis of your buffer pools.

Specifically, the “what if” analysis can help you compare the performance of using only the default buffer pools and multiple buffer pools. Additionally, the Buffer Pool Analyzer can recommend how to split buffer pools and where to place table space and index space objects.

---

### Monitoring and tuning buffer pools using online commands

The DISPLAY BUFFERPOOL and ALTER BUFFERPOOL commands allow you to monitor and tune buffer pools on line, while DB2 is running, without the overhead of running traces.

**GUPI** To monitor and tune your buffer pools with online commands:

- Use the DISPLAY BUFFERPOOL command to display the current status of one or more active or inactive buffer pools.

```
DISPLAY BUFFERPOOL(BP0) DETAIL
```

```
+DISPLAY BPOOL(BP0) DETAIL
```

```
DSNB401I + BUFFERPOOL NAME BP0, BUFFERPOOL ID 0, USE COUNT 47
```

```
DSNB402I + BUFFERPOOL SIZE = 2000 BUFFERS AUTOSIZE = NO
```

```
ALLOCATED = 2000 TO BE DELETED = 0
```

```
IN-USE/UPDATED = 0
```

```
DSNB404I + THRESHOLDS -
```

```
VP SEQUENTIAL = 80
```

```

DEFERRED WRITE          = 85    VERTICAL DEFERRED WRT = 80, 0
PARALLEL SEQUENTIAL     = 50    ASSISTING PARALLEL SEQ = 0
DSNB406I + PGFIX ATTRIBUTE -
CURRENT = NO
PENDING = NO                PAGE STEALING METHOD = LRU
DSNB409I + INCREMENTAL STATISTICS SINCE 14:57:55 JAN 22, yyyy
DSNB411I + RANDOM GETPAGE   = 491222 SYNC READ I/O (R) = A 18193
SEQ. GETPAGE               = 1378500 SYNC READ I/O (S) = B 0
DMTH HIT                   = 0 PAGE-INS REQUIRED = 460400
DSNB412I + SEQUENTIAL PREFETCH
REQUESTS D = 41800    PREFETCH I/O C = 14473
PAGES READ E = 444030
DSNB413I + LIST PREFETCH -
REQUESTS = 9046    PREFETCH I/O = 2263
PAGES READ = 3046
DSNB414I + DYNAMIC PREFETCH -
REQUESTS = 6680    PREFETCH I/O = 142
PAGES READ = 1333
DSNB415I + PREFETCH DISABLED -
NO BUFFER = 0    NO READ ENGINE = 0
DSNB420I + SYS PAGE UPDATES = F 220425 SYS PAGES WRITTEN = G 35169
ASYNC WRITE I/O = 5084 SYNC WRITE I/O = 3
PAGE-INS REQUIRED = 45
DSNB421I + DWT HIT H = 2 VERTICAL DWT HIT = I 0
NO WRITE ENGINE = 0
DSNB440I + PARALLEL ACTIVITY -
PARALLEL REQUEST = 0 DEGRADED PARALLEL = 0
DSNB441I + LPL ACTIVITY -
PAGES ADDED = 0
DSN9022I + DSNB1CMD '+DISPLAY BPOOL' NORMAL COMPLETION

```

Figure 59. Sample output from the DISPLAY BUFFERPOOL command

In figure above, find the following fields:

- SYNC READ I/O (R) (**A**) shows the number of random synchronous read I/O operations. SYNC READ I/O (S) (**B**) shows the number of sequential synchronous read I/O operations. Sequential synchronous read I/Os occur when prefetch is disabled.

To determine the total number of synchronous read I/Os, add SYNC READ I/O (S) and SYNC READ I/O (R).

- In message DSNB412I, REQUESTS (**C**) shows the number of times that sequential prefetch was triggered, and PREFETCH I/O (**D**) shows the number of times that sequential prefetch occurred. PAGES READ (**E**) shows the number of pages read using sequential prefetch.
- SYS PAGE UPDATES (**F**) corresponds to the number of buffer updates.
- SYS PAGES WRITTEN (**G**) is the number of pages written to disk.
- DWT HIT (**H**) is the number of times the deferred write threshold (DWQT) was reached. This number is workload dependent.
- VERTICAL DWT HIT (**I**) is the number of times the vertical deferred write threshold (VDWQT) was reached. This value is per data set, and it is related to the number of asynchronous writes.
- Use the LSTATS option of the DISPLAY BUFFERPOOL command to obtain buffer pool information on a specific data set. For example, you can use the LSTATS option to:
  - Provide page count statistics for a certain index. With this information, you could determine whether a query used the index in question, and perhaps drop the index if it was not used.

- Monitor the response times on a particular data set. If you determine that I/O contention is occurring, you could redistribute the data sets across your available disks.

This same information is available with IFCID 0199 (statistics class 8).

- Use the ALTER BUFFERPOOL command to change the following attributes:
  - Size
  - Thresholds
  - Page-stealing algorithm
  - Page fix attribute
  - Automatic adjustment attribute

## Example

Because the number of synchronous read I/Os ( **A** ) and the number of SYS PAGE UPDATES ( **F** ) shown in the preceding figure are relatively high, you would want to tune the buffer pools by changing the buffer pool specifications. For example, you could increase the buffer pool size to reduce the amount of unnecessary I/O, which would make buffer operations more efficient. To do that, you would enter the following command:

```
-ALTER BUFFERPOOL(BP0) VPSIZE(6000)
```




---

## Using OMEGAMON to monitor buffer pool statistics

You can find information about buffer pools in the OMEGAMON statistics report.



To analyze your buffer pools with OMEGAMON:

1. Examine the OMEGAMON statistics report.
2. Increase the size of the buffer pool in the following situations:
  - Sequential prefetch is inhibited. PREF.DISABLED-NO BUFFER ( **A** ) shows how many times sequential prefetch is disabled because the sequential prefetch threshold (90% of the pages in the buffer pool are unavailable) has been reached.
  - You detect poor update efficiency. You can determine update efficiency by checking the values in both of the following fields:
    - BUFF.UPDATES/PAGES WRITTEN ( **C** )
    - PAGES WRITTEN PER WRITE I/O ( **E** )

In evaluating the values you see in these fields, remember that no values are absolutely acceptable or absolutely unacceptable. Each installation's workload is a special case. To assess the update efficiency of your system, monitor for overall trends rather than for absolute high values for these ratios.

The following factors impact buffer updates per pages written and pages written per write I/O:

- Sequential nature of updates
- Number of rows per page
- Row update frequency

For example, a batch program that processes a table in skip sequential mode with a high row update frequency in a dedicated environment can achieve

very good update efficiency. In contrast, update efficiency tends to be lower for transaction processing applications, because transaction processing tends to be random.

The following factors affect the ratio of pages written per write I/O:

#### **Checkpoint frequency**

The CHECKPOINT FREQ field on installation panel DSNTIPN specifies either the number of consecutive log records written between DB2 system checkpoints or the number of minutes between DB2 system checkpoints. You can use a large value to specify CHECKPOINT FREQ in number of log records; the default value is 500 000. You can use a small value to specify CHECKPOINT FREQ in minutes. If you specify CHECKPOINT FREQ in minutes, the recommended setting is 2 to 5 minutes. At checkpoint time, I/Os are scheduled to write all updated pages on the deferred write queue to disk. If system checkpoints occur too frequently, the deferred write queue does not grow large enough to achieve a high ratio of pages written per write I/O.

#### **Frequency of active log switch**

DB2 takes a system checkpoint each time the active log is switched. If the active log data sets are too small, checkpoints occur often, which prevents the deferred write queue from growing large enough to achieve a high ratio of pages written per write I/O.

#### **Buffer pool size**

The deferred write thresholds (VDWQT and DWQT) are a function of buffer pool size. If the buffer pool size is decreased, these thresholds are reached more frequently, causing I/Os to be scheduled more often to write some of the pages on the deferred write queue to disk. This prevents the deferred write queue from growing large enough to achieve a high ratio of pages written per write I/O.

#### **Number of data sets, and the spread of updated pages across them**

The maximum number of pages written per write I/O is 32, subject to a limiting scope of 180 pages (roughly one cylinder).

**Example:** If your application updates page 2 and page 179 in a series of pages, the two changed pages could potentially be written with one write I/O. But if your application updates page 2 and page 185 within a series of pages, writing the two changed pages would require two write I/Os because of the 180-page limit. Updated pages are placed in a deferred write queue based on the data set. For batch processing it is possible to achieve a high ratio of pages written per write I/O, but for transaction processing the ratio is typically lower.

For LOAD, REORG, and RECOVER, the maximum number of pages written per write I/O is 64, and the scope is typically unlimited. However, in some cases, such as loading a partitioned table space with nonpartitioned indexes, a 180-page limit scope exists.

- **SYNCHRONOUS WRITES (D)** is a high value. This field counts the number of immediate writes. However, immediate writes are not the only type of synchronous write. Therefore, providing a monitoring value for the number of immediate writes can be difficult.

Ignore SYNCHRONOUS WRITES when DM THRESHOLD is zero.

- DM THRESHOLD ( **F** ) is reached. This field shows how many times a page was immediately released because the data management threshold was reached. The quantity listed for this field should be zero.

Also note the following fields:

**WRITE ENGINE NOT AVAILABLE ( **B** )**

Records the number of times that asynchronous writes were deferred because DB2 reached its maximum number of concurrent writes. You cannot change this maximum value. This field has a nonzero value occasionally.

**PREF.DISABLED-NO READ ENG ( **G** )**

Records the number of times that a sequential prefetch was not performed because the maximum number of concurrent sequential prefetches was reached. Instead, normal reads were done. You cannot change this maximum value.

**PAGE-INS REQUIRED FOR WRITE ( **H** )**

Records the number of page-ins that are required for a read or write I/O. When the buffer pools are first allocated, the count might be high. After the first allocations are complete, the count should be close to zero.

## Example

The following figure shows where you can find important values in the statistics report.

TOT4K READ OPERATIONS	QUANTITY	TOT4K WRITE OPERATIONS	QUANTITY
-----	-----	-----	-----
BPOOL HIT RATIO (%)	73.12	BUFFER UPDATES	220.4K
GETPAGE REQUEST	1869.7K	PAGES WRITTEN	35169.00
GETPAGE REQUEST-SEQUENTIAL	1378.5K	BUFF.UPDATES/PAGES WRITTEN <b>C</b>	6.27
GETPAGE REQUEST-RANDOM	491.2K	SYNCHRONOUS WRITES <b>D</b>	3.00
SYNCHRONOUS READS	54187.00	ASYNCHRONOUS WRITES	5084.00
SYNCHRON. READS-SEQUENTIAL	35994.00	PAGES WRITTEN PER WRITE I/O <b>E</b>	5.78
SYNCHRON. READS-RANDOM	18193.00	HORIZ.DEF.WRITE THRESHOLD	2.00
GETPAGE PER SYN.READ-RANDOM	27.00	VERTI.DEF.WRITE THRESHOLD	0.00
SEQUENTIAL PREFETCH REQUEST	41800.00	DM THRESHOLD <b>F</b>	0.00
SEQUENTIAL PREFETCH READS	14473.00	WRITE ENGINE NOT AVAILABLE <b>G</b>	0.00
PAGES READ VIA SEQ.PREFETCH	444.0K	PAGE-INS REQUIRED FOR WRITE <b>H</b>	45.00
S.PRF.PAGES READ/S.PRF.READ	30.68		
LIST PREFETCH REQUESTS	9046.00		
LIST PREFETCH READS	2263.00		
PAGES READ VIA LST PREFETCH	3046.00		
L.PRF.PAGES READ/L.PRF.READ	1.35		
DYNAMIC PREFETCH REQUESTED	6680.00		
DYNAMIC PREFETCH READS	142.00		
PAGES READ VIA DYN.PREFETCH	1333.00		
D.PRF.PAGES READ/D.PRF.READ	9.39		
PREF.DISABLED-NO BUFFER <b>A</b>	0.00		
PREF.DISABLED-NO READ ENG <b>B</b>	0.00		
PAGE-INS REQUIRED FOR READ	460.4K		

Figure 60. OMEGAMON database buffer pool statistics (modified)

---

## Monitoring work file data sets

You should monitor how work files use devices, both in terms of space use and I/O response times.

Work file data sets are used for sorting, for materializing views and nested table expressions, for temporary tables, and for other activities. DB2 does not distinguish or place priorities on these uses of the work file data sets. Excessive activity from one type of use can interfere and detract from the performance of others.

When a temporary work file result table is populated using an INSERT statement, it uses work file space.

No other process can use the same work file space as that temporary work file table until the table goes away. The space is reclaimed when the application process commits or rolls back, or when it is deallocated, depending which RELEASE option was used when the plan or package was bound.

To best monitor these result tables:

- Keep work files in a separate buffer pool.
- Use IFCID 0311 in performance trace class 8 to distinguish these tables from other uses of the work file.

---

## Chapter 27. Monitoring of DB2 locking

If you have problems with suspensions, timeouts, or deadlocks, you should monitor the use of locks by DB2.

**PSPI** **Recommendation:** Always have statistics classes 1, 3, and 4 and accounting classes 1 and 3 running. The statistics reports provide counters for timeouts, deadlocks and suspensions. Statistics class 3 includes IFCID 0172 (deadlocks) and IFCID 0196 (deadlocks). If deadlocks or timeouts are a concern, print these detail records to investigate the situation during exceptional situations.

The accounting reports show the locking activity under the heading of LOCKING. The other key indication of locking problems are the class 3 suspensions LOCK/LATCH(DB2+IRLM). If locking and latching are increasing the elapsed time of your transactions or batch work, you want to investigate further.

Use the statistics trace to monitor the system-wide use of locks, the accounting trace to monitor locks used by a particular application process.

Use EXPLAIN to monitor the locks required by a particular SQL statement, or all the SQL in a particular plan or package. **PSPI**.

---

### Scenario for analyzing concurrency

In OMEGAMON, a report titled "Trace" corresponds to a single DB2 trace record; a report titled "Report" summarizes information from several trace records.

**PSPI** The concurrency problem analyzed in this information follows the CONCURRENCY PROBLEM branch of OMEGAMON reports that are used for problem analysis and makes use of OMEGAMON reports. This scenario makes use of:

- Accounting report - long
- Locking suspension report
- Lockout report
- Lockout trace

**PSPI**

### Scenario description


An application, which has recently been moved into production, is experiencing timeouts. Other applications have not been significantly affected in this example.

**PSPI** To investigate the problem, determine a period when the transaction is likely to time out. When that period begins:

1. Start the GTF.
2. Start the DB2 accounting classes 1, 2, and 3 to GTF to allow for the production of OMEGAMON accounting reports.
3. Stop GTF and the traces after about 15 minutes.
4. Produce and analyze the OMEGAMON accounting report - long.



5. Use the DB2 performance trace selectively for detailed problem analysis.

In some cases, the initial and detailed stages of tracing and analysis presented in this information can be consolidated into one. In other cases, the detailed analysis might not be required at all.

To analyze the problem, generally start with the accounting report - long. (If you have enough information from program and system messages, you can skip this first step.) 

### Concurrency in the accounting report

The class 3 LOCK/LATCH time and number of occurrences in the Accounting Report are key statistics to watch if you are concerned about locking problems.

 The figure below shows a portion of the accounting report - long. To determine the effect of lock suspensions on your applications, examine the class 3 LOCK/LATCH time and number of occurrences in the Accounting Report (  in the following figure). This If locking and latching are increasing the elapsed time of your transactions or batch work, you want to investigate further.

AVERAGE	APPL (CL.1)	DB2 (CL.2)	IFI (CL.5)	CLASS 3 SUSPENSIONS	AVERAGE TIME	AV.EVENT	HIGHLIGHTS	D			
	A	B			C						
ELAPSED TIME	0.123229	0.079743	N/P	LOCK/LATCH(DB2+IRLM)	0.050311	0.04	#OCCURRENCES	193			
NONNESTED	0.123229	0.079743	N/A	SYNCHRON. I/O	0.010170	9.16	#ALLIEDS	0			
STORED PROC	0.000000	0.000000	N/A	DATABASE I/O	0.006325	8.16	#ALLIEDS DISTRIB:	0			
UDF	0.000000	0.000000	N/A	LOG WRITE I/O	0.003845	1.00	#DBATS	193			
TRIGGER	0.000000	0.000000	N/A	OTHER READ I/O	0.000000	0.00	#DBATS DISTRIB.:	0			
				OTHER WRTE I/O	0.000148	0.04	#NO PROGRAM DATA:	0			
CP CPU TIME	0.026978	0.018994	N/P	SER.TASK SWTCH	0.000115	0.04	#NORMAL TERMINAT:	193			
AGENT	0.026978	0.018994	N/A	UPDATE COMMIT	0.000000	0.00	#ABNORMAL TERMIN:	0			
NONNESTED	0.026978	0.018994	N/P	OPEN/CLOSE	0.000000	0.00	#CP/X PARALLEL.:	0			
STORED PRC	0.000000	0.000000	N/A	SYSLGRNG REC	0.000115	0.04	#IO PARALLELISM:	0			
UDF	0.000000	0.000000	N/A	EXT/DEL/DEF	0.000000	0.00	#INCREMENT. BIND:	0			
TRIGGER	0.000000	0.000000	N/A	OTHER SERVICE	0.000000	0.00	#COMMITTS	193			
PAR.TASKS	0.000000	0.000000	N/A	ARC.LOG(QUIES)	0.000000	0.00	#ROLLBACKS	0			
IIPCP CPU	0.000000	N/A	N/A	DRAIN LOCK	0.000000	0.00	#SVPT RELEASE	0			
				CLAIM RELEASE	0.000000	0.00	#SVPT ROLLBACK	0			
IIP CPU TIME	0.000000	0.000000	N/A	PAGE LATCH	0.000000	0.00	MAX SQL CASC LVL:	0			
				ARC.LOG READ	0.000000	0.00	#SVPT REQUESTS	0			
SUSPEND TIME	0.000000	0.060744	N/A	DRAIN LOCK	0.000000	0.00	#SVPT RELEASE	0			
AGENT	N/A	0.060744	N/A	CLAIM RELEASE	0.000000	0.00	#SVPT ROLLBACK	0			
PAR.TASKS	N/A	0.000000	N/A	PAGE LATCH	0.000000	0.00	MAX SQL CASC LVL:	0			
STORED PROC	0.000000	N/A	N/A	NOTIFY MSGS	0.000000	0.00	UPDATE/COMMIT	40.00			
UDF	0.000000	N/A	N/A	GLOBAL CONTENTION	0.000000	0.00	SYNCH I/O AVG.:	0.001110			
				COMMIT PHI WRITE I/O	0.000000	0.00					
NOT ACCOUNT.	N/A	0.000004	N/A	ASYNCH CF REQUESTS	0.000000	0.00	0.00DB2				
ENT/EXIT	N/A	182.00	N/A	TOTAL CLASS 3	0.060744	9.28					
EN/EX-STPROC	N/A	0.00	N/A								
EN/EX-UDF	N/A	0.00	N/A								
DCAPT.DESCR.	N/A	N/A	N/P								
LOG EXTRACT.	N/A	N/A	N/P								
GLOBAL	CONTENTION	L-LOCKS	AVERAGE TIME	AV.EVENT	GLOBAL	CONTENTION	P-LOCKS	AVERAGE TIME	AV.EVENT		
L-LOCKS			0.000000	0.00	P-LOCKS			0.000000	0.00		
PARENT (DB,TS,TAB,PART)			0.000000	0.00	PAGESET/PARTITION			0.000000	0.00		
CHILD (PAGE,ROW)			0.000000	0.00	PAGE			0.000000	0.00		
OTHER			0.000000	0.00	OTHER			0.000000	0.00		
SQL DML	AVERAGE	TOTAL	SQL DCL	TOTAL	SQL DDL	CREATE	DROP	ALTER	LOCKING	AVERAGE	TOTAL
SELECT	20.00	3860	LOCK TABLE	0	TABLE	0	0	0	TIMEOUTS	0.00	0
INSERT	0.00	0	GRANT	0	CRT TTABLE	0	N/A	N/A	DEADLOCKS	0.00	0
UPDATE	30.00	5790	REVOKE	0	DCL TTABLE	0	N/A	N/A	ESCAL.(SHARED)	0.00	0
MERGE	0.00	0	SET CURR.SQLID	0	AUX TABLE	0	N/A	N/A	ESCAL.(EXCLUS)	0.00	0
DELETE	10.00	1930	SET CURR.SQLID	0	AUX TABLE	0	N/A	N/A	ESCAL.(EXCLUS)	0.00	0
			SET HOST VAR.	0	INDEX	0	0	0	MAX PG/ROW LOCKS HELD	43.34	47
DESCRIBE	0.00	0	SET CUR.DEGREE	0	TABLESPACE	0	0	0	LOCK REQUEST	63.82	12318
DESC.TBL	0.00	0	SET RULES	0	DATABASE	0	0	0	UNLOCK REQUEST	14.48	2794
PREPARE	0.00	0	SET CURR.PATH	0	STOGROUP	0	0	0	QUERY REQUEST	0.00	0
OPEN	10.00	1930	SET CURR.PREC.	0	SYNONYM	0	0	N/A	CHANGE REQUEST	33.35	6436
FETCH	10.00	1930	CONNECT TYPE 1	0	VIEW	0	0	N/A	OTHER REQUEST	0.00	0
CLOSE	10.00	1930	CONNECT TYPE 2	0	canceled	0	0	N/A	LOCK SUSPENSIONS	0.00	0
			SET CONNECTION	0	PACKAGE	N/A	0	N/A	IRLM LATCH SUSPENSIONS	0.03	5
			RELEASE	0	PROCEDURE	0	0	0	OTHER SUSPENSIONS	0.00	0
			CALL	0	FUNCTION	0	0	0	TOTAL SUSPENSIONS	0.03	5
			ASSOC LOCATORS	0	TRIGGER	0	0	N/A			
			ALLOC CURSOR	0	DIST TYPE	0	0	N/A			
			HOLD LOCATOR	0	SEQUENCE	0	0	0			
			FREE LOCATOR	0							
			DCL-ALL	0	TOTAL	0	0	0			
					RENAME TBL	0					
					COMMENT ON	0					
					LABEL ON	0					

Figure 61. Excerpt from accounting report - long

The accounting report - long shows the average elapsed times and the average number of suspensions per plan execution. In Figure 61:

- The class 1 average elapsed time **A** (AET) is 0.072929 seconds. The class 2 times show that 0.029443 seconds **B** of that are spent in DB2; the rest is spent in the application.
- The class 2 AET is spent mostly in lock or latch suspensions (LOCK/LATCH **C** is 0.050311 seconds).
- The HIGHLIGHTS section **D** of the report (upper right) shows #OCCURRENCES as 193; that is the number of accounting (IFCID 3) records.

## PSPI

## Lock suspension report

You can start DB2 performance class 6 to GTF to prepare a lock suspension report.

**PSPI** Because that class 6 traces only suspensions, it does not significantly reduce performance. The following figure shows the OMEGAMON lock suspension report.

```

:
-----
PRIMAUTH  --- L O C K   R E S O U R C E --- TOTAL  LOCAL  --SUSPEND REASONS-- ----- R E S U M E   R E A S O N S -----
PLANNAME  TYPE      NAME                                SUSPENDS LATCH  GLOB.  S.NFY  ---- NORMAL ---- TIMEOUT/CANCEL --- DEADLOCK ---
                                     SUSPENDS IRLMQ  OTHER  NMBR      AET  NMBR      AET  NMBR      AET
-----
FPB
PARALLEL  PARTITION DB  =PARADABA                                2      2      0      0      0      N/C      2  303.277805  0      N/C
                                     OB  =TAB1TS
                                     PART= 1
LOCKING REPORT COMPLETE

```

Figure 62. Portion of the OMEGAMON Lock suspension report

The lock suspension report shows:

- Which plans are suspended, by plan name within primary authorization ID. For statements bound to a package, see the information about the plan that executes the package.
- What IRLM requests and which lock types are causing suspensions.
- Whether suspensions are normally resumed or end in timeouts or deadlocks.
- What the average elapsed time (AET) per suspension is.

The report also shows the reason for the suspensions, as described in Table 94.

Table 94. Reasons for suspensions

Reason	Includes
LOCAL	Contention for a local resource
LATCH	Contention for latches within IRLM (with brief suspension)
GLOB.	Contention for a global resource
IRLMQ	An IRLM queued request
S.NFY	Intersystem message sending
OTHER	Page latch or drain suspensions, suspensions because of incompatible retained locks in data sharing, or a value for service use

The preceding list shows only the first reason for a suspension. When the original reason is resolved, the request could remain suspended for a second reason.

Each suspension results in either a normal resume, a timeout, or a deadlock.

The report shows that the suspension causing the delay involves access to partition 1 of table space PARADABA.TAB1TS by plan PARALLEL. Two LOCAL suspensions time out after an average of 5 minutes, 3.278 seconds (303.278 seconds). **PSPI**

## Lockout report

The lockout report summarizes information from several DB2 trace records.

**PSPI** The following figure shows the OMEGAMON lockout report. This report shows that plan PARALLEL contends with the plan DSNESPRR. It also shows that contention is occurring on partition 1 of table space PARADABA.TAB1TS.

PRIMAUTH PLANNAME	--- L O C K   R E S O U R C E ---				----- A G E N T S -----				HOLDER WAITER	
	TYPE	NAME	TIMEOUTS	DEADLOCKS	MEMBER	PLANNAME	CONNECT	CORRID		
FPB	PARALLEL	PARTITION DB =PARADABA OB =TABITS PART= 1	2	0	N/P	DSNESP RR	TSO	EOA	2	0
		** LOCKOUTS FOR PARALLEL **	2	0						

Figure 63. Portion of the OMEGAMON lockout report



## Lockout trace

The lockout trace contains information about contention from a single DB2 trace record.



The following figure shows the OMEGAMON lockout trace.

For each contender, this report shows the database object, lock state (mode), and duration for each contention for a transaction lock.

:													
PRIMAUTH	CORRNAME	CONNTYPE			--- L O C K   R E S O U R C E ---								
ORIGAUTH	CORRNMBR	INSTANCE	EVENT TIMESTAMP				TYPE		NAME		EVENT SPECIFIC DATA		
PLANNAME	CONNECT		RELATED TIMESTAMP		EVENT								
-----													
FPB	FPBPARAL	TSO	15:25:27.23692350		TIMEOUT		PARTITION		DB =PARADABA		REQUEST =LOCK UNCONDITIONAL		
FPB	'BLANK'	AB09C533F92E	N/P						OB =TABITS		STATE =S ZPARM INTERVAL= 300		
PARALLEL	BATCH								PART= 1		DURATION=COMMIT INTERV.COUNTER= 1		
										HASH =X'000020E0'			
										----- HOLDERS/WAITERS -----			
										HOLDER			
										LUW='BLANK'.IPSAQ421.AB09C51F32CB			
										MEMBER =N/P		CONNECT =TSO	
										PLANNAME=DSNESP RR		CORRID=EOA	
										DURATION=COMMIT		PRIMAUTH=KARELLE	
										STATE =X			
KARL	KARL	TSO	15:30:32.97267562		TIMEOUT		PARTITION		DB =PARADABA		REQUEST =LOCK UNCONDITIONAL		
KARL	'BLANK'	AB09C65528E6	N/P						OB =TABITS		STATE =IS ZPARM INTERVAL= 300		
PARALLEL	TSO								PART= 1		DURATION=COMMIT INTERV.COUNTER= 1		
										HASH =X'000020E0'			
										----- HOLDERS/WAITERS -----			
										HOLDER			
										LUW='BLANK'.IPSAQ421.AB09C51F32CB			
										MEMBER =N/P		CONNECT =TSO	
										PLANNAME=DSNESP RR		CORRID=EOA	
										DURATION=COMMIT		PRIMAUTH=DAVE	
										STATE =X			
										ENDUSER =DAVEUSER			
										WSNAME =DAVEWS			
										TRANS =DAVES TRANSACTION			
LOCKING TRACE COMPLETE													

Figure 64. Portion of the OMEGAMON lockout trace

At this point in the investigation, the following information is known:

- The applications that contend for resources
- The page sets for which contention occurs
- The impact, frequency, and type of the contentions

The application or data design must be reviewed to reduce the contention.



## Corrective decisions

When lock suspensions are unacceptably long or timeouts occur, you can use the DB2 performance trace for locking and the OMEGAMON reports to isolate the resource that causes the suspensions.

**PSPI** The lockout report identifies the resources involved. The lockout trace tells what contending process (agent) caused the timeout.

In Figure 62 on page 488, the number of suspensions is low (only 2) and both have ended in a timeout. Rather than use the DB2 performance trace for locking, use the preferred option, DB2 statistics class 3 and DB2 performance trace class 1. Then produce the OMEGAMON locking timeout report to obtain the information necessary to reduce overheads.

For specific information about OMEGAMON reports and their usage, see *OMEGAMON Report Reference* and *Using IBM Tivoli OMEGAMON XE on z/OS*.

**PSPI**

## OMEGAMON online locking conflict display

You can monitor locking conflicts by using the OMEGAMON online locking conflict display. The display shows what is waiting for resources and what is holding resources when a thread is suspended.

---

## Using the statistics and accounting traces to monitor locking

The statistics and accounting trace records contain information on locking.

**PSPI** OMEGAMON provides one way to view the trace results. The OMEGAMON reports for the Accounting Trace and Statistics Trace each corresponds to a single DB2 trace record. (Details of those reports are subject to change without notification from DB2 and are available in the appropriate OMEGAMON documentation).

The following figure shows a portion of the Statistics Trace, which tells how many suspensions (**A**), timeouts (**B**), deadlocks (**C**), and lock escalations (**D**) occur in the trace record. You can also use the statistics trace to monitor each occurrence of lock escalation.

LOCKING ACTIVITY	QUANTITY	/SECOND	/THREAD	/COMMIT
-----	-----	-----	-----	-----
SUSPENSIONS (ALL) <b>A</b>	2	0.02	1.00	0.40
SUSPENSIONS (LOCK ONLY)	2	0.02	1.00	0.40
SUSPENSIONS (IRLM LATCH)	0	0.00	0.00	0.00
SUSPENSIONS (OTHER)	0	0.00	0.00	0.00
TIMEOUTS <b>B</b>	0	0.00	0.00	0.00
DEADLOCKS <b>C</b>	1	0.01	0.50	0.20
LOCK REQUESTS	17	0.18	8.50	3.40
UNLOCK REQUESTS	12	0.13	6.00	2.40
QUERY REQUESTS	0	0.00	0.00	0.00
CHANGE REQUESTS	5	0.05	2.50	1.00
OTHER REQUESTS	0	0.00	0.00	0.00
LOCK ESCALATION (SHARED) <b>D</b>	0	0.00	0.00	0.00
LOCK ESCALATION (EXCLUSIVE)	0	0.00	0.00	0.00
DRAIN REQUESTS	0	0.00	0.00	0.00
DRAIN REQUESTS FAILED	0	0.00	0.00	0.00
CLAIM REQUESTS	7	0.08	3.50	1.40
CLAIM REQUESTS FAILED	0	0.00	0.00	0.00

Figure 65. Locking activity blocks from statistics trace

The figure below shows a portion of the Accounting Trace, which gives the same information for a particular application (suspensions **A**, timeouts **B**, deadlocks **C**, lock escalations **D**). It also shows the maximum number of concurrent page locks held and acquired during the trace (E and F). Review applications with a large number to see if this value can be lowered. This number is the basis for the proper setting of LOCKS PER USER and, indirectly, LOCKS PER TABLE or TABLESPACE.

LOCKING	TOTAL
-----	-----
TIMEOUTS <b>B</b>	0
DEADLOCKS <b>C</b>	0
ESCAL.(SHAR) <b>D</b>	0
ESCAL.(EXCL)	0
MAX PG/ROW LCK HELD <b>E</b>	2
LOCK REQUEST <b>F</b>	8
UNLOCK REQST	2
QUERY REQST	0
CHANGE REQST	5
OTHER REQST	0
LOCK SUSPENS.	1
IRLM LATCH SUSPENS.	0
OTHER SUSPENS.	0
TOTAL SUSPENS. <b>A</b>	1

DRAIN/CLAIM	TOTAL
-----	-----
DRAIN REQST	0
DRAIN FAILED	0
CLAIM REQST	4
CLAIM FAILED	0

Figure 66. Locking activity blocks from accounting trace



## Using EXPLAIN to tell which locks DB2 chooses

You can use DB2 EXPLAIN to determine what kind of locks DB2 uses to process an SQL statement.



1. Use the EXPLAIN statement, or the EXPLAIN option of the BIND and REBIND subcommands, to determine which modes of table, partition, and table space locks DB2 initially assigns for an SQL statement. Follow the instructions under "Using EXPLAIN to capture information about SQL statements." on page 557. (EXPLAIN does not return information about the locks acquired on LOB table spaces.)
2. EXPLAIN stores its results in a table called PLAN\_TABLE. To review the results, query PLAN\_TABLE. After running EXPLAIN, each row of PLAN\_TABLE describes the processing for a single table, either one named explicitly in the SQL statement that is being explained or an intermediate table that DB2 has to create. The column TSLOCKMODE of PLAN\_TABLE shows an

initial lock mode for that table. The lock mode applies to the table or the table space, depending on the value of LOCKSIZE and whether the table space is segmented or nonsegmented.

3. In Table 95, find what table or table space lock is used and whether page or row locks are used also, for the particular combination of lock mode and LOCKSIZE you are interested in.

**For statements executed remotely:** EXPLAIN gathers information only about data access in the DBMS where the statement is run or the bind operation is carried out. To analyze the locks obtained at a remote DB2 location, you must run EXPLAIN at that location. For more information on running EXPLAIN, and a fuller description of PLAN\_TABLE, see “Investigating SQL performance with EXPLAIN” on page 421.

**Table 95. Which locks DB2 chooses.** N/A = Not applicable; Yes = Page or row locks are acquired; No = No page or row locks are acquired.

Table space structure	Lock mode from EXPLAIN				
	IS	S	IX	U	X
For non-segmented and universal table spaces:					
Table space lock acquired is:	IS	S	IX	U	X
Page or row locks acquired?	Yes	No	Yes	No	No
<b>Note:</b> For partitioned and universal table spaces, the lock mode applies only to those partitions that are locked. Lock modes for LOB and XML table spaces are not reported with EXPLAIN.					
For segmented table spaces with:					
LOCKSIZE ANY, ROW, or PAGE					
Table space lock acquired is:	IS	IS	IX	n/a	IX
Table lock acquired is:	IS	S	IX	n/a	X
Page or row locks acquired?	Yes	No	Yes	n/a	No
LOCKSIZE TABLE					
Table space lock acquired is:	n/a	IS	n/a	IX	IX
Table lock acquired is:	n/a	S	n/a	U	X
Page or row locks acquired?	n/a	No	n/a	No	No
LOCKSIZE TABLESPACE					
Table space lock acquired is:	n/a	S	n/a	U	X
Table lock acquired is:	n/a	n/a	n/a	n/a	n/a
Page or row locks acquired?	n/a	No	n/a	No	No



---

## Chapter 28. Deadlock detection scenarios

The following information includes an examination of two different deadlock scenarios and an explanation of the use of the OMEGAMON deadlock detail report to determine the cause of the deadlock.

**PSPI** The OMEGAMON Locking Trace - Deadlock report formats the information contained in trace record IFCID 0172 (statistics class 3). The report outlines all the resources and agents involved in a deadlock and the significant locking parameters, such as lock state and duration, related to their requests.

These examples assume that statistics class 3 and performance class 1 are activated. Performance class 1 is activated to get IFCID 105 records, which contain the translated names for the database ID and the page set OBID.

The scenarios that follow use three of the DB2 sample tables, DEPT, PROJ, and ACT. They are all defined with LOCKSIZE ANY. Type 2 indexes are used to access all three tables. As a result, contention for locks is only on data pages. **PSPI**

---

### Scenario 1: Two-way deadlock with two resources

In this classic deadlock example, two agents contend for resources; the result is a deadlock in which one of the agents is rolled back.

**PSPI** Two transactions and two resources are involved. First, transaction LOC2A acquires a lock on one resource while transaction LOC2B acquires a lock on another. Next, the two transactions each request locks on the resource held by the other.

The transactions execute as follows:

#### LOC2A

1. Declare and open a cursor for update on DEPT and fetch from page 2.
2. Declare and open a cursor for update on PROJ and fetch from page 8.
3. Update page 2.
4. Update page 8.
5. Close both cursors and commit.

#### LOC2B

1. Declare and open a cursor for update on PROJ and fetch from page 8.
2. Declare and open a cursor for update on DEPT and fetch from page 2.
3. Update page 8.
4. Update page 2.
5. Close both cursors and commit.

Events take place in the following sequence:

1. LOC2A obtains a U lock on page 2 in table DEPT, to open its cursor for update.
2. LOC2B obtains a U lock on a page 8 in table PROJ, to open its cursor for update.
3. LOC2A attempts to access page 8, to open its cursor but cannot proceed because of the lock held by LOC2B.

- LOC2B attempts to access page 2, to open its cursor but cannot proceed because of the lock held by LOC2A.

DB2 selects one of the transactions and rolls it back, releasing its locks. That allows the other transaction to proceed to completion and release its locks also.

The following figure shows the OMEGAMON Locking Trace - Deadlock report that is produced for this situation.

The report shows that the only transactions involved came from plans LOC2A and LOC2B. Both transactions came in from BATCH.

```

:
PRIMAUTH CORRNAME CONNTYPE
ORIGAUTH CORRNMBR INSTANCE
PLANNAME CONNECT
-----
SYSADM RUNLOC2A TSO
SYSADM 'BLANK' AADD32FD8A8C N/P
LOC2A BATCH
A

--- LOCK RESOURCE ---
EVENT TIMESTAMP
RELATED TIMESTAMP EVENT
TYPE NAME
-----
DATAPAGE DB =DSN8D42A
OB =DEPT
PAGE=X'000002'

EVENT SPECIFIC DATA
COUNTER = 2 WAITERS = 2
TSTAMP =04/02/95 20:32:30.68
HASH =X'01060304'
----- BLOCKER IS HOLDER -----
LWM='BLANK'.EGTVLU2.AADD32FD8A8C
MEMBER =DB1A CONNECT =BATCH
PLANNAME=LOC2A CORRID=RUNLOC2A
DURATION=MANUAL PRIMAUTH=SYSADM
STATE =U
----- WAITER -----
LWM='BLANK'.EGTVLU2.AA65FEDC1022
MEMBER =DB1A CONNECT =BATCH
PLANNAME=LOC2B CORRID=RUNLOC2B
DURATION=MANUAL PRIMAUTH=KATHY
REQUEST =LOCK WORTH = 18
STATE =U

DATAPAGE DB =DSN8D42A
OB =PROJ
PAGE=X'000008'

HASH =X'01060312'
----- BLOCKER IS HOLDER -----
LWM='BLANK'.EGTVLU2.AA65FEDC1022
MEMBER =DB1A CONNECT =BATCH
PLANNAME=LOC2B CORRID=RUNLOC2B
DURATION=MANUAL PRIMAUTH=KATHY
STATE =U
----- WAITER -----*VICTIM*-
LWM='BLANK'.EGTVLU2.AADD32FD8A8C
MEMBER =DB1A CONNECT =BATCH
PLANNAME=LOC2A CORRID=RUNLOC2A
DURATION=MANUAL PRIMAUTH=SYSADM
REQUEST =LOCK WORTH = 17
STATE =U

```

Figure 67. Deadlock scenario 1: Two transactions and two resources

The lock held by transaction 1 (LOC2A) is a data page lock on the DEPT table and is held in U state. (The value of MANUAL for duration means that, if the plan was bound with isolation level CS and the page was not updated, then DB2 is free to release the lock before the next commit point.)

Transaction 2 (LOC2B) was requesting a lock on the same resource, also of mode U and hence incompatible.

The specifications of the lock held by transaction 2 (LOC2B) are the same. Transaction 1 was requesting an incompatible lock on the same resource. Hence, the deadlock.

Finally, note that the entry in the trace, identified at **A**, is LOC2A. That is the selected thread (the “victim”) whose work is rolled back to let the other proceed.



---

## Scenario 2: Three-way deadlock with three resources

In this scenario, three agents contend for resources and the result is a deadlock in which one of the agents is rolled back. Three transactions and three resources are involved.

### PSPI

First, the three transactions each acquire a lock on a different resource. LOC3A then requests a lock on the resource held by LOC3B, LOC3B requests a lock on the resource held by LOC3C, and LOC3C requests a lock on the resource held by LOC3A.

The transactions execute as follows:

#### LOC3A

1. Declare and open a cursor for update on DEPT and fetch from page 2.
2. Declare and open a cursor for update on PROJ and fetch from page 8.
3. Update page 2.
4. Update page 8.
5. Close both cursors and commit.

#### LOC3B

1. Declare and open a cursor for update on PROJ and fetch from page 8.
2. Declare and open a cursor for update on ACT and fetch from page 6.
3. Update page 6.
4. Update page 8.
5. Close both cursors and commit.

#### LOC3C

1. Declare and open a cursor for update on ACT and fetch from page 6.
2. Declare and open a cursor for update on DEPT and fetch from page 2.
3. Update page 6.
4. Update page 2.
5. Close both cursors and commit.

Events take place in the following sequence:

1. LOC3A obtains a U lock on page 2 in DEPT, to open its cursor for update.
2. LOC3B obtains a U lock on page 8 in PROJ, to open its cursor for update.
3. LOC3C obtains a U lock on page 6 in ACT, to open its cursor for update.
4. LOC3A attempts to access page 8 in PROJ but cannot proceed because of the lock held by LOC3B.
5. LOC3B attempts to access page 6 in ACT cannot proceed because of the lock held by LOC3C.
6. LOC3C attempts to access page 2 in DEPT but cannot proceed, because of the lock held by LOC3A.

DB2 rolls back LOC3C and releases its locks. That allows LOC3B to complete and release the lock on PROJ so that LOC3A can complete. LOC3C can then retry.

The following figure shows the OMEGAMON Locking Trace - Deadlock report produced for this situation.

```

:
PRIMAUTH CORRNAME CONNTYPE
ORIGAUTH CORRNMBR INSTANCE
PLANNAME CONNECT
EVENT TIMESTAMP
RELATED TIMESTAMP EVENT
--- L O C K   R E S O U R C E ---
TYPE      NAME
EVENT SPECIFIC DATA
-----
SYSADM    RUNLOC3C TSO      15:10:39.33061694 DEADLOCK
SYSADM    'BLANK'  AADE2CF16F34 N/P
LOC3C     BATCH
                                DATAPAGE DB =DSN8D42A
                                OB  =PROJ
                                PAGE=X'000008'
                                COUNTER = 3      WAITERS = 3
                                TSTAMP  =04/03/95 15:10:39.31
                                HASH    =X'01060312'
                                ----- BLOCKER IS HOLDER-----
                                LUW='BLANK'.EGTVLU2.AAD15D373533
                                MEMBER  =DB1A      CONNECT =BATCH
                                PLANNAME=LOC3B      CORRID=RUNLOC3B
                                DURATION=MANUAL      PRIMAUTH=JULIE
                                STATE    =U
                                ----- WAITER -----
                                LUW='BLANK'.EGTVLU2.AB33745CE357
                                MEMBER  =DB1A      CONNECT =BATCH
                                PLANNAME=LOC3A      CORRID=RUNLOC3A
                                DURATION=MANUAL      PRIMAUTH=BOB
                                REQUEST  =LOCK      WORTH  = 18
                                STATE    =U
                                ----- BLOCKER IS HOLDER --*VICTIM*-
                                LUW='BLANK'.EGTVLU2.AAD15D373533
                                MEMBER  =DB1A      CONNECT =BATCH
                                PLANNAME=LOC3C      CORRID =RUNLOC3C
                                DURATION=MANUAL      PRIMAUTH=SYSADM
                                STATE    =U
                                ----- WAITER -----
                                LUW='BLANK'.EGTVLU2.AB33745CE357
                                MEMBER  =DB1A      CONNECT =BATCH
                                PLANNAME=LOC3B      CORRID =RUNLOC3B
                                DURATION=MANUAL      PRIMAUTH=JULIE
                                REQUEST  =LOCK      WORTH  = 18
                                STATE    =U
                                ----- BLOCKER IS HOLDER -----
                                LUW='BLANK'.EGTVLU2.AAD15D373533
                                MEMBER  =DB1A      CONNECT =BATCH
                                PLANNAME=LOC3A      CORRID =RUNLOC3A
                                DURATION=MANUAL      PRIMAUTH=BOB
                                STATE    =U
                                ----- WAITER -----*VICTIM*-
                                LUW='BLANK'.EGTVLU2.AB33745CE357
                                MEMBER  =DB1A      CONNECT =BATCH
                                PLANNAME=LOC3C      CORRID =RUNLOC3C
                                DURATION=MANUAL      PRIMAUTH=SYSADM
                                REQUEST  =LOCK      WORTH  = 18
                                STATE    =U

```

Figure 68. Deadlock scenario 2: Three transactions and three resources



---

## Chapter 29. Querying the catalog for statistics

You can issue `SELECT` statements to retrieve some of the important statistics that DB2 uses for access path selection.

The catalog queries shown here are included in `DSNTESP` in `SDSNSAMP` and can be used as input to `SPUFI`. See “Investigating SQL performance with `EXPLAIN`” on page 421 for more information about how these statistics are used in access path selection.

**PSPI** To access information about your data and how it is organized, use the following queries:

```
SELECT CREATOR, NAME, CARDF, NPAGES, PCTPAGES
  FROM SYSIBM.SYSTABLES
 WHERE DBNAME = 'xxx'
 AND TYPE = 'T';

SELECT NAME, UNIQUERULE, CLUSTERRATIOF, FIRSTKEYCARDF, FULLKEYCARDF,
       NLEAF, NLEVELS, PGSIZE
  FROM SYSIBM.SYSINDEXES
 WHERE DBNAME = 'xxx';

SELECT NAME, DBNAME, NACTIVE, CLOSERULE, LOCKRULE
  FROM SYSIBM.SYSTABLESPACE
 WHERE DBNAME = 'xxx';

SELECT NAME, TBNAME, COLCARDF, HIGH2KEY, LOW2KEY, HEX(HIGH2KEY),
       HEX(LOW2KEY)
  FROM SYSIBM.SYSCOLUMNS
 WHERE TBCREATOR = 'xxx' AND COLCARDF <> -1;

SELECT NAME, FREQUENCYF, COLVALUE, HEX(COLVALUE), CARDF,
       COLGROUPCOLNO, HEX(COLGROUPCOLNO), NUMCOLUMNS, TYPE
  FROM SYSIBM.SYSCOLDIST
 WHERE TBNAME = 'ttt'
 ORDER BY NUMCOLUMNS, NAME, COLGROUPCOLNO, TYPE, FREQUENCYF DESC;

SELECT NAME, TSNAME, CARD, NPAGES
  FROM SYSIBM.SYSTABSTATS
 WHERE DBNAME='xxx';
```

**PSPI**

If the statistics in the DB2 catalog no longer correspond to the true organization of your data, you should reorganize the necessary tables, run `RUNSTATS`, and rebind the plans or packages that contain any affected queries. This includes the DB2 catalog table spaces as well as user table spaces. Then DB2 has accurate information to choose appropriate access paths for your queries. Use the `EXPLAIN` statement to verify the chosen access paths for your queries.

### Related concepts

Chapter 31, “When to reorganize indexes and table spaces,” on page 501



---

## Chapter 30. Gathering monitor statistics and update statistics

The DB2 utility RUNSTATS can update the DB2 catalog tables with statistical information about data and indexes.

For a list of the catalog columns for which RUNSTATS collects statistics, see Table 28 on page 153.

You can choose which DB2 catalog tables you want RUNSTATS to update: those used to optimize the performance of SQL statements or those used by database administrators to assess the status of a particular table space or index. You can monitor these catalog statistics in conjunction with EXPLAIN to make sure that your queries access data efficiently.

After you use the LOAD, REBUILD INDEX, or REORG utilities, you can gather statistics inline with those utilities by using the STATISTICS option.

**Why gather statistics:** Maintaining your statistics is a critical part of performance monitoring and tuning. DB2 must have correct statistical information to make the best choices for the access path.

**When to gather statistics:** To ensure that information in the catalog is current, gather statistics in situations in which the data or index changes significantly, such as in the following situations:

- After loading a table and before binding application plans and packages that access the table.
- After creating an index with the CREATE INDEX statement, to update catalog statistics related to the new index. (Before an application can use a new index, you must rebind the application plan or package.)
- After reorganizing a table space or an index. Then rebind plans or packages for which performance remains a concern. See “Whether to rebind after gathering statistics” on page 736 for more information. (It is not necessary to rebind after reorganizing a LOB table space, because those statistics are not used for access path selection.)
- After heavy insert, update, and delete activity. Again, rebind plans or packages for which performance is critical.
- Periodically. By comparing the output of one execution with previous executions, you can detect a performance problem early.
- Against the DB2 catalog to provide DB2 with more accurate information for access path selection of users’ catalog queries.

To obtain information from the catalog tables, use a SELECT statement, or specify REPORT YES when you invoke RUNSTATS. When used routinely, RUNSTATS provides data about table spaces and indexes over a period of time. For example, when you create or drop tables or indexes or insert many rows, run RUNSTATS to update the catalog. Then rebind your applications so that DB2 can choose the most efficient access paths.

**Collecting statistics by partition:** You can collect statistics for a single data partition or index partition. This information allows you to avoid the cost of running utilities against unchanged partitions. When you run utilities by partition,

DB2 uses the results to update the aggregate statistics for the entire table space or index. If statistics do not exist for each separate partition, DB2 can calculate the aggregate statistics only if the utilities are executed with the FORCEROLLUP YES keyword (or FORCEROLLUP keyword is omitted and the value of the STATISTICS ROLLUP field on installation panel DSNTIPO is YES). If you do not use the keyword or installation panel field setting to force the roll up of the aggregate statistics, you must run utilities once on the entire object before running utilities on separate partitions.

**Collecting history statistics:** When you collect statistics with RUNSTATS or gather them inline with the LOAD, REBUILD, or REORG utilities, you can use the HISTORY option to collect history statistics. With the HISTORY option, the utility stores the statistics that were updated in the catalog tables in history records in the corresponding catalog history tables. (For information on the catalog data that is collected for history statistics, see Table 30 on page 167.)

To remove old statistics that are no longer needed in the catalog history tables, use the MODIFY STATISTICS utility or the SQL DELETE statement. Deleting outdated information from the catalog history tables can help improve the performance of processes that access the data in these tables.

***Recommendations for performance:***

- To reduce the processor consumption WHEN collecting column statistics, use the SAMPLE option. The SAMPLE option allows you to specify a percentage of the rows to examine for column statistics. Consider the effect on access path selection before choosing sampling. Little or no effect is likely on access path selection if the access path has a matching index scan and very few predicates. However, if the access path joins of many tables with matching index scans and many predicates, the amount of sampling can affect the access path. In these cases, start with 25 percent sampling and see if a negative effect occurs on access path selection. If not, you could consider reducing the sampling percent until you find the percent that gives you the best reduction in processing time without negatively affecting the access path.
- To reduce the elapsed time of gathering statistics immediately after a LOAD, REBUILD INDEX, or REORG, gather statistics inline with those utilities by using the STATISTICS option.

---

## Chapter 31. When to reorganize indexes and table spaces

Data that is organized well physically can improve the performance of access paths that rely on index or data scans. Well-organized data can also help reduce the amount of disk storage used by the index or table space.

If your main reason for reorganizing is performance, the best way to determine when to reorganize is to watch your statistics for increased I/O, getpages, and processor consumption. When performance degrades to an unacceptable level, analyze the statistics described in the guidelines in this information to help you develop your own rules for when to reorganize in your particular environment. Consider the following general guidelines for when to run the REORG utility:

- **Using useful catalog queries:** Catalog queries you can use to help you determine when to reorganize are included in DSNTESP in SDSNSAMP and can be used as input to SPUFI.
- **Using the REORG utility to determine whether to reorganize:** The REORG utility imbeds the function of catalog queries. If a query returns a certain result (you can use the default or supply your own), REORG either reorganizes or does not reorganize. Optionally, you can have REORG run a report instead of actually doing the reorganization.

The REORG options that imbed these catalog queries are:

- OFFPOSLIMIT and INDREFLIMIT options of REORG TABLESPACE
- LEAFDISTLIMIT of REORG INDEX

The REORG utility does not imbed any function to help you determine when to reorganize LOB table spaces.

- **Reorganizing after table definition changes:** Another time to consider reorganizing data to improve performance is when ALTER TABLE statements have been used to add a column to the table or to change the data types or the lengths of existing columns. Such changes cause the table space to be placed in advisory REORG-pending (AREO\*) status.

In the case of changing the definition of existing column, the table space is placed in AREO\* status because the existing data is not immediately converted to its new definition. Reorganizing the table space causes the rows to be reloaded with the data converted to the new definition. Until the table space is reorganized, the changes must be tracked and applied as the data is accessed, possibly degrading performance. For example, depending on the number of changes, you might see decreased performance for dynamic SQL queries, updates and deletes, other ALTER statements (especially those that are run concurrently). In addition, running multiple REORG and LOAD utilities concurrently might perform slower or create timeouts. Unloading a table might also take longer if the table that has had many changes prior to it being reorganized.

### Related concepts

Chapter 31, “When to reorganize indexes and table spaces”



---

## Chapter 32. Monitoring SQL performance

You can monitor the performance of the SQL statements that are run by your application programs.

---

### Monitoring SQL performance with Optimization Service Center for DB2 for z/OS

IBM Optimization Service Center for DB2 for z/OS enables you to connect to a DB2 for z/OS subsystem or group member from a workstation computer and create monitor profiles to monitor the performance of the SQL workloads and statements that run on that system.

#### Tables that are used by optimization tools

Optimization tools, such as IBM Optimization Service Center for DB2 for z/OS, IBM DB2 Optimization Expert for z/OS, and the DB2 EXPLAIN function might use one or more of the following types of tables.

Refer to the documentation for each optimization tool or the EXPLAIN function to determine which tables might be used by a specific tool or function.

#### EXPLAIN tables that are used by optimization tools

Optimization tools such as IBM Optimization Service Center for DB2 for z/OS, IBM DB2 Optimization Expert for z/OS, and the DB2 EXPLAIN function might create and use any of the following EXPLAIN tables to store performance data.

##### DSN\_PREDICAT\_TABLE:

The predicate table, DSN\_PREDICAT\_TABLE, contains information about all of the predicates in a query.

**Recommendation:** Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

##### CREATE TABLE statement

The following figure shows the current format of DSN\_PREDICAT\_TABLE

```
CREATE TABLE userid.DSN_PREDICAT_TABLE
( QUERYNO          INTEGER      NOT NULL,
  QBLOCKNO         SMALLINT    NOT NULL,
  APPLNAME         VARCHAR(24)  NOT NULL,
  PROGNAME         VARCHAR(128) NOT NULL,
  PREDNO           INTEGER      NOT NULL,
  TYPE             CHAR(8)      NOT NULL,
  LEFT_HAND_SIDE   VARCHAR(128) NOT NULL,
  LEFT_HAND_PNO    INTEGER      NOT NULL,
  LHS_TABNO        SMALLINT     NOT NULL,
  LHS_QBNO         SMALLINT     NOT NULL,
  RIGHT_HAND_SIDE  VARCHAR(128) NOT NULL,
  RIGHT_HAND_PNO   INTEGER      NOT NULL,
  RHS_TABNO        SMALLINT     NOT NULL,
  RHS_QBNO         SMALLINT     NOT NULL,
  FILTER_FACTOR    FLOAT        NOT NULL,
```

```

        BOOLEAN_TERM      CHAR(1)      NOT NULL,
        SEARCHARG         CHAR(1)      NOT NULL,
        JOIN              CHAR(1)      NOT NULL,
        AFTER_JOIN        CHAR(1)      NOT NULL,
        ADDED_PRED         CHAR(1)      NOT NULL,
        REDUNDANT_PRED     CHAR(1)      NOT NULL,
        DIRECT_ACCESS      CHAR(1)      NOT NULL,
        KEYFIELD           CHAR(1)      NOT NULL,
        EXPLAIN_TIME       TIMESTAMP    NOT NULL,
        CATEGORY           SMALLINT     NOT NULL,
        CATEGORY_B         SMALLINT     NOT NULL,
        TEXT               VARCHAR(2000) NOT NULL,
        PRED_ENCODE        CHAR(1)      NOT NULL WITH DEFAULT,
        PRED_CCSID         SMALLINT     NOT NULL WITH DEFAULT,
        PRED_MCCSID        SMALLINT     NOT NULL WITH DEFAULT,
        MARKER            CHAR(1)      NOT NULL WITH DEFAULT,
        PARENT_PNO         INTEGER      NOT NULL,
        NEGATION           CHAR(1)      NOT NULL,
        LITERALS           VARCHAR(128) NOT NULL,
        CLAUSE             CHAR(8)      NOT NULL,
        GROUP_MEMBER       VARCHAR(24)   NOT NULL
    ) in database-name.table-space-name
    CCSID UNICODE;

```

Figure 69. The CREATE TABLE statement for userid.DSN\_PREDICAT\_TABLE

### Column descriptions

The following table describes the columns of the DSN\_PREDICAT\_TABLE

Table 96. DSN\_PREDICAT\_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	<p>The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are:</p> <ol style="list-style-type: none"> <li>1 The statement line number in the program</li> <li>2 QUERYNO clause</li> <li>3 The EXPLAIN statement</li> <li>4 EDM unique token in the statement cache</li> </ol>
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
APPLNAME	VARCHAR(24) NOT NULL	The application plan name.
PROGNAME	VARCHAR(128) NOT NULL	The program name (binding an application) or the package name (binding a package).
PREDNO	INTEGER NOT NULL	The predicate number, a number used to identify a predicate within a query.

Table 96. DSN\_PREDICAT\_TABLE description (continued)

Column name	Data type	Description
TYPE	CHAR(8) NOT NULL	<p>A string used to indicate the type or the operation of the predicate. The possible values are:</p> <ul style="list-style-type: none"> <li>• 'AND'</li> <li>• 'OR'</li> <li>• 'EQUAL'</li> <li>• 'RANGE'</li> <li>• 'BETWEEN'</li> <li>• 'IN'</li> <li>• 'LIKE'</li> <li>• 'NOT LIKE'</li> <li>• 'EXISTS'</li> <li>• 'NOTEXIST'</li> <li>• 'SUBQUERY'</li> <li>• 'HAVING'</li> <li>• 'OTHERS'</li> </ul>
LEFT_HAND_SIDE	VARCHAR(128) NOT NULL	<p>If the LHS of the predicate is a table column (LHS_TABNO &gt; 0), then this column indicates the column name. Other possible values are:</p> <ul style="list-style-type: none"> <li>• 'VALUE'</li> <li>• 'COLEXP'</li> <li>• 'NONCOLEXP'</li> <li>• 'CORSUB'</li> <li>• 'NONCORSUB'</li> <li>• 'SUBQUERY'</li> <li>• 'EXPRESSION'</li> <li>• Blanks</li> </ul>
LEFT_HAND_SIDE	VARCHAR(128)	<p>If the LHS of the predicate is a table column (LHS_TABNO &gt; 0), then this column indicates the column name. Other possible values are:</p> <ul style="list-style-type: none"> <li>• 'VALUE'</li> <li>• 'COLEXP'</li> <li>• 'NONCOLEXP'</li> <li>• 'CORSUB'</li> <li>• 'NONCORSUB'</li> <li>• 'SUBQUERY'</li> <li>• 'EXPRESSION'</li> <li>• Blanks</li> </ul>

Table 96. DSN\_PREDICAT\_TABLE description (continued)

Column name	Data type	Description
LEFT_HAND_PNO	INTEGER NOT NULL	If the LHS of the predicate is a table column (LHS_TABNO > 0), then this column indicates the column name. Other possible values are: <ul style="list-style-type: none"> <li>• 'VALUE'</li> <li>• 'COLEXP'</li> <li>• 'NONCOLEXP'</li> <li>• 'CORSUB'</li> <li>• 'NONCORSUB'</li> <li>• 'SUBQUERY'</li> <li>• 'EXPRESSION'</li> <li>• Blanks</li> </ul>
LHS_TABNO	SMALLINT NOT NULL	If the LHS of the predicate is a table column, then this column indicates a number which uniquely identifies the corresponding table reference within a query.
LHS_QBNO	SMALLINT NOT NULL	If the LHS of the predicate is a table column, then this column indicates a number which uniquely identifies the corresponding table reference within a query.
RIGHT_HAND_SIDE	VARCHAR(128) NOT NULL	If the RHS of the predicate is a table column (RHS_TABNO > 0), then this column indicates the column name. Other possible values are: <ul style="list-style-type: none"> <li>• 'VALUE'</li> <li>• 'COLEXP'</li> <li>• 'NONCOLEXP'</li> <li>• 'CORSUB'</li> <li>• 'NONCORSUB'</li> <li>• 'SUBQUERY'</li> <li>• 'EXPRESSION'</li> <li>• Blanks</li> </ul>
RIGHT_HAND_PNO	INTEGER NOT NULL	If the predicate is a compound predicate (AND/OR), then this column indicates the second child predicate. However, this column is not reliable when the predicate tree consolidation happens. Use PARENT_PNO instead to reconstruct the predicate tree.
RHS_TABNO	CHAR(1) NOT NULL	If the RHS of the predicate is a table column, then this column indicates a number which uniquely identifies the corresponding table reference within a query.
RHS_QBNO	CHAR(1) NOT NULL	If the RHS of the predicate is a subquery, then this column indicates a number which uniquely identifies the corresponding query block within a query.
FILTER_FACTOR	CHAR(1) NOT NULL	The estimated filter factor.
FILTER_FACTOR	CHAR(1) NOT NULL	Whether this predicate can be used to determine the truth value of the whole WHERE clause.
BOOLEAN_TERM	CHAR(1) NOT NULL	Whether this predicate can be used to determine the truth value of the whole WHERE clause.
SEARCHARG	CHAR(1) NOT NULL	Whether this predicate can be processed by data manager (DM). If it is not, then the relational data service (RDS) needs to be used to take care of it, which is more costly.

Table 96. DSN\_PREDICAT\_TABLE description (continued)

Column name	Data type	Description
AFTER_JOIN	CHAR(1) NOT NULL	Indicates the predicate evaluation phase: 'A'      After join 'D'      During join blank    Not applicable
ADDED_PRED	CHAR(1) NOT NULL	Whether it is generated by transitive closure, which means DB2 can generate additional predicates to provide more information for access path selection, when the set of predicates that belong to a query logically imply other predicates.
REDUNDANT_PRED	CHAR(1) NOT NULL	Whether it is a redundant predicate, which means evaluation of other predicates in the query already determines the result that the predicate provides.
DIRECT_ACCESS	CHAR(1) NOT NULL	Whether the predicate is direct access, which means one can navigate directly to the row through ROWID.
KEYFIELD	CHAR(1) NOT NULL	Whether the predicate includes the index key column of the involved table.
EXPLAIN_TIME	TIMESTAMP	The EXPLAIN timestamp.
CATEGORY	SMALLINT	IBM internal use only.
CATEGORY_B	SMALLINT	IBM internal use only.
TEXT	VARCHAR(2000)	The transformed predicate text; truncated if exceeds 2000 characters.
PRED_ENCODE	CHAR(1)	IBM internal use only.
PRED_CCSID	SMALLINT	IBM internal use only.
PRED_MCCSID	SMALLINT	IBM internal use only.
MARKER	CHAR(1)	Whether this predicate includes host variables, parameter markers, or special registers.
PARENT_PNO	INTEGER	The parent predicate number. If this predicate is a root predicate within a query block, then this column is 0.
NEGATION	CHAR(1)	Whether this predicate is negated via NOT.
LITERALS	VARCHAR(128)	This column indicates the literal value or literal values separated by colon symbols.
CLAUSE	CHAR(8)	The clause where the predicate exists: 'HAVING ' The HAVING clause 'ON '    The ON clause 'WHERE ' The WHERE clause
GROUP_MEMBER	CHAR(8)	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

#### DSN\_STRUCT\_TABLE:

The structure table, DSN\_STRUCT\_TABLE, contains information about all of the query blocks in a query.

**Recommendation:** Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

### CREATE TABLE statement

The following figure shows the CREATE TABLE statement for DSN\_STRUCT\_TABLE.

```
CREATE TABLE userid.DSN_STRUCT_TABLE
( QUERYNO          INTEGER      NOT NULL,
  QBLOCKNO         SMALLINT     NOT NULL,
  APPLNAME          VARCHAR(24)  NOT NULL,
  PROGNAME          VARCHAR(128) NOT NULL,
  PARENT            SMALLINT     NOT NULL,
  TIMES             FLOAT        NOT NULL,
  ROWCOUNT         INTEGER      NOT NULL,
  ATOPEN            CHAR(1)      NOT NULL,
  CONTEXT           CHAR(10)     NOT NULL,
  ORDERNO           SMALLINT     NOT NULL,
  DOATOPEN_PARENT   SMALLINT     NOT NULL,
  QBLOCK_TYPE       CHAR(6)      NOT NULL WITH DEFAULT,
  EXPLAIN_TIME      TIMESTAMP    NOT NULL,
  QUERY_STAGE       CHAR(8)      NOT NULL,
  GROUP_MEMBER      VARCHAR(24)  NOT NULL
)
  IN database-name.table-space-name
CCSID UNICODE;
```

Figure 70. The CREATE TABLE statement for userid.DSN\_STRUCT\_TABLE

### Column descriptions

The following table describes the columns of DSN\_STRUCT\_TABLE

Table 97. DSN\_STRUCT\_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: <ol style="list-style-type: none"> <li>1 The statement line number in the program</li> <li>2 QUERYNO clause</li> <li>3 The EXPLAIN statement</li> <li>4 EDM unique token in the statement cache</li> </ol>
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
APPLNAME	VARCHAR(24) NOT NULL	The application plan name.
PROGNAME	VARCHAR(128) NOT NULL	The program name (binding an application) or the package name (binding a package).
PARENT	SMALLINT NOT NULL	The parent query block number of the current query block in the structure of SQL text; this is the same as the PARENT_QBLOCKNO in PLAN_TABLE.
TIMES	FLOAT NOT NULL	The estimated number of rows returned by Data Manager; also the estimated number of times this query block is executed.

Table 97. DSN\_STRUCT\_TABLE description (continued)

Column name	Data type	Description
ROWCOUNT	INTEGER NOT NULL	The estimated number of rows returned by RDS (Query Cardinality).
ATOPEN	CHAR(1) NOT NULL	Whether the query block is moved up for do-at-open processing; 'Y' if done-at-open; 'N': otherwise.
CONTEXT	CHAR(10) NOT NULL	This column indicates what the context of the current query block is. The possible values are: <ul style="list-style-type: none"> <li>• 'TOP LEVEL'</li> <li>• 'UNION'</li> <li>• 'UNION ALL'</li> <li>• 'PREDICATE'</li> <li>• 'TABLE EXP'</li> <li>• 'UNKNOWN'</li> </ul>
ORDERNO	SMALLINT NOT NULL	Not currently used.
DOATOPEN_PARENT	SMALLINT NOT NULL	The parent query block number of the current query block; Do-at-open parent if the query block is done-at-open, this may be different from the PARENT_QBLOCKNO in PLAN_TABLE.
QBLOCK_TYPE	CHAR(6) NOT NULL WITH DEFAULT	This column indicates the type of the current query block. The possible values are <ul style="list-style-type: none"> <li>• 'SELECT'</li> <li>• 'INSERT'</li> <li>• 'UPDATE'</li> <li>• 'DELETE'</li> <li>• 'SELUPD'</li> <li>• 'DELCUR'</li> <li>• 'UPDCUR'</li> <li>• 'CORSUB'</li> <li>• 'NCOSUB'</li> <li>• 'TABLEX'</li> <li>• 'TRIGGR'</li> <li>• 'UNION'</li> <li>• 'UNIONA'</li> <li>• 'CTE'</li> </ul> It is equivalent to QBLOCK_TYPE column in PLAN_TABLE, except for CTE.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The EXPLAIN timestamp.
QUERY_STAGE	CHAR(8) NOT NULL	IBM internal use only.
GROUP_MEMBER	CHAR(8) NOT NULL	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

#### DSN\_PGROUP\_TABLE:

The parallel group table, PGROUP\_TABLE, contains information about the parallel groups in a query.

**Recommendation:** Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

### CREATE TABLE statement

The following figure shows the format of DSN\_PGROUP\_TABLE

```
CREATE TABLE userid.DSN_PGROUP_TABLE
( QUERYNO          INTEGER          NOT NULL,
  QBLOCKNO         SMALLINT         NOT NULL,
  PLANNAME         VARCHAR(24)      NOT NULL,
  COLLID           VARCHAR(128)     NOT NULL,
  PROGNAME         VARCHAR(128)     NOT NULL,
  EXPLAIN_TIME     TIMESTAMP        NOT NULL,
  VERSION          VARCHAR(122)     NOT NULL,
  GROUPID          SMALLINT         NOT NULL,
  FIRSTPLAN        SMALLINT         NOT NULL,
  LASTPLAN         SMALLINT         NOT NULL,
  CPUCOST          REAL             NOT NULL,
  IOCAST          REAL             NOT NULL,
  BESTTIME         REAL             NOT NULL,
  DEGREE           SMALLINT         NOT NULL,
  MODE             CHAR(1)          NOT NULL,
  REASON           SMALLINT         NOT NULL,
  LOCALCPU         SMALLINT         NOT NULL,
  TOTALCPU         SMALLINT         NOT NULL,
  FIRSTBASE        SMALLINT,
  LARGETS          CHAR(1),
  PARTKIND         CHAR(1),
  GROUPTYPE        CHAR(3),
  ORDER            CHAR(1),
  STYLE            CHAR(4),
  RANGEKIND        CHAR(1),
  NKEYCOLS         SMALLINT,
  LOWBOUND         VARCHAR(40),
  HIGHBOUND        VARCHAR(40),
  LOWKEY           VARCHAR(40),
  HIGHKEY          VARCHAR(40),
  FIRSTPAGE        CHAR(4),
  LASTPAGE         CHAR(4),
  GROUP_MEMBER     VARCHAR(24)      NOT NULL,
  HOST_REASON      SMALLINT,
  PARA_TYPE        CHAR(4),
  PART_INNER       CHAR(1),
  GRNU_KEYRNG      CHAR(1),
  OPEN_KEYRNG      CHAR(1)
) IN database-name.table-space-name

CCSID UNICODE;
```

Figure 71. The CREATE TABLE statement for *userid.DSN\_PGROUP\_TABLE*

### Column descriptions

The following table describes the columns of DSN\_PGROUP\_TABLE

Table 98. DSN\_PGROUP\_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: <ol style="list-style-type: none"> <li>1 The statement line number in the program</li> <li>2 QUERYNO clause</li> <li>3 The EXPLAIN statement</li> <li>4 EDM unique token in the statement cache</li> </ol>
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
PLANNAME	VARCHAR(24) NOT NULL	The application plan name.
COLLID	VARCHAR(128) NOT NULL	The collection ID for the package.
PROGNAME	VARCHAR(128) NOT NULL	The program name (binding an application) or the package name (binding a package).
EXPLAIN_TIME	TIMESTAMP NOT NULL	The explain timestamp.
VERSION	VARCHAR(122) NOT NULL	The version identifier for the package.
GROUPID	SMALLINT NOT NULL	The parallel group identifier within the current query block.
FIRSTPLAN	SMALLINT NOT NULL	The plan number of the first contributing mini-plan associated within this parallel group.
LASTPLAN	SMALLINT NOT NULL	The plan number of the last mini-plan associated within this parallel group.
CPUCOST	REAL NOT NULL	The estimated total CPU cost of this parallel group in milliseconds.
IO COST	REAL NOT NULL	The estimated total I/O cost of this parallel group in milliseconds.
BESTTIME	REAL NOT NULL	The estimated elapsed time for each parallel task for this parallel group.
DEGREE	SMALLINT NOT NULL	The degree of parallelism for this parallel group determined at bind time. Max parallelism degree if the Table space is large is 255, otherwise 64.
MODE	CHAR(1) NOT NULL	The parallel mode: 'T' IO parallelism 'C' CPU parallelism 'X' multiple CPU SyspleX parallelism (highest level) 'N' No parallelism
REASON	SMALLINT NOT NULL	The reason code for downgrading parallelism mode.
LOCALCPU	SMALLINT NOT NULL	The number of CPUs currently online when preparing the query.
TOTALCPU	SMALLINT NOT NULL	The total number of CPUs in Sysplex. LOCALCPU and TOTALCPU are different only for the DB2 coordinator in a Sysplex.
FIRSTBASE	SMALLINT	The table number of the table that partitioning is performed on.
TARGETS	CHAR(1)	'Y' if the TableSpace is large in this group.

Table 98. DSN\_PGROUP\_TABLE description (continued)

Column name	Data type	Description
PARTKIND	CHAR(1)	The partitioning type: <b>'L'</b> Logical partitioning <b>'P'</b> Physical partitioning
GROUPTYPE	CHAR(3)	Determines what operations this parallel group contains: table Access, Join, or Sort <b>'A'</b> <b>'AJ'</b> <b>'AJS'</b>
ORDER	CHAR(1)	The ordering requirement of this parallel group : <b>'N'</b> No order. Results need no ordering. <b>'T'</b> Natural Order. Ordering is required but results already ordered if accessed via index. <b>'K'</b> Key Order. Ordering achieved by sort. Results ordered by sort key. This value applies only to parallel sort.
STYLE	CHAR(4)	The Input/Output format style of this parallel group. Blank for IO Parallelism. For other modes: <b>'RIRO'</b> Records IN, Records OUT <b>'WIRO'</b> Work file IN, Records OUT <b>'WIWO'</b> Work file IN, Work file OUT
RANGEKIND	CHAR(1)	The range type: <b>'K'</b> Key range <b>'P'</b> Page range
NKEYCOLS	SMALLINT	The number of interesting key columns, that is, the number of columns that will participate in the key operation for this parallel group.
LOWBOUND	VARCHAR(40)	The low bound of parallel group.
HIGHBOUND	VARCHAR(40)	The high bound of parallel group.
LOWKEY	VARCHAR(40)	The low key of range if partitioned by key range.
HIGHKEY	VARCHAR(40)	The high key of range if partitioned by key range.
FIRSTPAGE	CHAR(4)	The first page in range if partitioned by page range.
LASTPAGE	CHAR(4)	The last page in range if partitioned by page range.
GROUP_MEMBER	CHAR(8) NOT NULL	IBM internal use only.
HOST_REASON	SMALLINT	IBM internal use only.
PARA_TYPE	CHAR(4)	IBM internal use only.
PART_INNER	CHAR(1)	IBM internal use only.
GRNU_KEYRNG	CHAR(1)	IBM internal use only.
OPEN_KEYRNG	CHAR(1)	IBM internal use only.

#### DSN\_PTASK\_TABLE:

The parallel tasks table, DSN\_PTASK\_TABLE, contains information about all of the parallel tasks in a query.

**Recommendation:** Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

### CREATE TABLE statement

The following figure shows the format of DSN\_PTASK\_TABLE.

```
CREATE TABLE userid.DSN_PTASK_TABLE
( QUERYNO          INTEGER          NOT NULL,
  QBLOCKNO         SMALLINT        NOT NULL,
  PGDNO            SMALLINT        NOT NULL,
  APPLNAME         VARCHAR(24)     NOT NULL,
  PROGNAME         VARCHAR(128)    NOT NULL,
  LPTNO            SMALLINT        NOT NULL,
  KEYCOLID         SMALLINT        ,
  DPSI             CHAR(1)         NOT NULL,
  LPTLOKEY         VARCHAR(40)     ,
  LPTHIKEY         VARCHAR(40)     ,
  LPTLOPAG         CHAR(4)         ,
  LPTHIPAG         CHAR(4)         ,
  LPTLOPG#         CHAR(4)         ,
  LPTHIPG#         CHAR(4)         ,
  LPTLOPT#         SMALLINT        ,
  LPTHIPT#         SMALLINT        ,
  KEYCOLDT         SMALLINT        ,
  KEYCOLPREC       SMALLINT        ,
  KEYCOLSCAL       SMALLINT        ,
  EXPLAIN_TIME     TIMESTAMP       NOT NULL,
  GROUP_MEMBER     VARCHAR(24)     NOT NULL
) IN database-name.table-space-name
CCSID UNICODE;
```

Figure 72. The CREATE TABLE statement for userid.DSN\_PTASK\_TABLE

### Column descriptions

The following table describes the columns of DSN\_PTASK\_TABLE.

Table 99. DSN\_PTASK\_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: <ol style="list-style-type: none"> <li>1 The statement line number in the program</li> <li>2 QUERYNO clause</li> <li>3 The EXPLAIN statement</li> <li>4 EDM unique token in the statement cache</li> </ol>
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
APPLNAME	VARCHAR(24) NOT NULL	The application plan name.
PROGNAME	VARCHAR(128) NOT NULL	The program name (binding an application) or the package name (binding a package).
LPTNO	SMALLINT NOT NULL	The parallel task number.

Table 99. DSN\_PTASK\_TABLE description (continued)

Column name	Data type	Description
KEYCOLID	SMALLINT	The key column ID (KEY range only).
DPSI	CHAR(1) NOT NULL	Indicates if a data partition secondary index (DPSI) is used.
LPTLOKEY	VARCHAR(40)	The low key value for this key column for this parallel task (KEY range only).
LPTHIKEY	VARCHAR(40)	The high key value for this key column for this parallel task (KEY range only).
LPTLOPAG	CHAR(4)	The low page information if partitioned by page range.
LPTLHIPAG	CHAR(4)	The high page information if partitioned by page range.
LPTLOPG#	CHAR(4)	The lower bound page number for this parallel task (Page range or DPSI enabled only).
LPTHIPG#	CHAR(4)	The upper bound page number for this parallel task (Page range or DPSI enabled only).
LPTLOPT#	SMALLINT	The lower bound partition number for this parallel task (Page range or DPSI enabled only).
KEYCOLDT	SMALLINT	The data type for this key column (KEY range only).
KEYCOLPREC	SMALLINT	The precision/length for this key column (KEY range only).
KEYCOLSCAL	SMALLINT	The scale for this key column (KEY range with Decimal datatype only).
EXPLAIN_TIME	TIMESTAMP NOT NULL	The EXPLAIN timestamp.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

#### DSN\_FILTER\_TABLE:

The filter table, DSN\_FILTER\_TABLE, contains information about how predicates are used during query processing.

**Recommendation:** Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

#### CREATE TABLE statement

The following figure shows the format of DSN\_FILTER\_TABLE.

```
CREATE TABLE userid.DSN_FILTER_TABLE
(
  QUERYNO           INTEGER      NOT NULL,
  QBLOCKNO          SMALLINT     NOT NULL,
  PLANNO            SMALLINT     NOT NULL,
  APPLNAME           VARCHAR(24)  NOT NULL,
  PROGNAME           VARCHAR(128) NOT NULL,
  COLLID             VARCHAR(128) NOT NULL WITH DEFAULT,
  ORDERNO            INTEGER      NOT NULL,
  PREDNO             INTEGER      NOT NULL,
  STAGE              CHAR(9)      NOT NULL,
  ORDERCLASS         INTEGER      NOT NULL,
  EXPLAIN_TIME       TIMESTAMP    NOT NULL,
  MIXOPSEQNO         SMALLINT     NOT NULL,
```

```

        REEVAL          CHAR(1)      NOT NULL,
        GROUP_MEMBER    VARCHAR(24)   NOT NULL
    )
    IN database-name.table-space-name

CCSID UNICODE;

```

Figure 73. The CREATE TABLE statement for userid.DSN\_FILTER\_TABLE

### Column descriptions

The following table describes the columns of DSN\_FILTER\_TABLE.

Table 100. DSN\_FILTER\_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: <ol style="list-style-type: none"> <li>1 The statement line number in the program</li> <li>2 QUERYNO clause</li> <li>3 The EXPLAIN statement</li> <li>4 EDM unique token in the statement cache.</li> </ol>
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
PLANNO	SMALLINT	The plan number, a number used to identify each miniplan with a query block.
APPLNAME	VARCHAR(24) NOT NULL	The application plan name.
PROGNAME	VARCHAR(128) NOT NULL WITH DEFAULT	The program name (binding an application) or the package name (binding a package).
COLLID	INTEGER NOT NULL	The collection ID for the package.
ORDERNO	INTEGER NOT NULL	The sequence number of evaluation. Indicates the order in which the predicate is applied within each stage
PREDNO	INTEGER NOT NULL	The predicate number, a number used to identify a predicate within a query.
STAGE	CHAR(9) NOT NULL	Indicates at which stage the Predicate is evaluated. The possible values are: <ul style="list-style-type: none"> <li>• 'Matching',</li> <li>• 'Screening',</li> <li>• 'Stage 1'</li> <li>• 'Stage 2'</li> </ul>
ORDERCLASS	INTEGER NOT NULL	IBM internal use only.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The explain timestamp.
MIXOPSEQ	SMALLINT NOT NULL	IBM internal use only.

Table 100. DSN\_FILTER\_TABLE description (continued)

Column name	Data type	Description
REEVAL	CHAR(1) NOT NULL	IBM internal use only.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

#### DSN\_DETCOST\_TABLE:

The detailed cost table, DSN\_DETCOST\_TABLE, contains information about detailed cost estimation of the mini-plans in a query.

**Recommendation:** Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

#### CREATE TABLE statement

The following figure shows the format of DSN\_DETCOST\_TABLE.

```
CREATE TABLE userid.DSN_DETCOST_TABLE
(
  QUERYNO          INTEGER      NOT NULL,
  QBLOCKNO         SMALLINT    NOT NULL,
  APPLNAME         VARCHAR(24)  NOT NULL,
  PROGNAME         VARCHAR(128) NOT NULL,
  PLANNO           SMALLINT    NOT NULL,
  OPENIO           FLOAT(4)     NOT NULL,
  OPENCPU          FLOAT(4)     NOT NULL,
  OPENCOST         FLOAT(4)     NOT NULL,
  DMIO             FLOAT(4)     NOT NULL,
  DMCPU            FLOAT(4)     NOT NULL,
  DMTOT            FLOAT(4)     NOT NULL,
  SUBQIO           FLOAT(4)     NOT NULL,
  SUBQCPU          FLOAT(4)     NOT NULL,
  SUBQCOST         FLOAT(4)     NOT NULL,
  BASEIO           FLOAT(4)     NOT NULL,
  BASECPU          FLOAT(4)     NOT NULL,
  BASETOT          FLOAT(4)     NOT NULL,
  ONECOMPROWS     FLOAT(4)     NOT NULL,
  IMLEAF           FLOAT(4)     NOT NULL,
  IMIO             FLOAT(4)     NOT NULL,
  IMPREFH          CHAR(2)      NOT NULL,
  IMMPRED          INTEGER      NOT NULL,
  IMFF             FLOAT(4)     NOT NULL,
  IMSRPRED         INTEGER      NOT NULL,
  IMFFADJ          FLOAT(4)     NOT NULL,
  IMSCANCST        FLOAT(4)     NOT NULL,
  IMROWCST         FLOAT(4)     NOT NULL,
  IMPAGECST        FLOAT(4)     NOT NULL,
  IMRIDSORT        FLOAT(4)     NOT NULL,
  IMMERGCST        FLOAT(4)     NOT NULL,
  IMCPU            FLOAT(4)     NOT NULL,
  IMTOT            FLOAT(4)     NOT NULL,
  IMSEQNO          SMALLINT    NOT NULL,
  DMPREFH          CHAR(2)      NOT NULL,
  DMCLUDIO         FLOAT(4)     NOT NULL,
  DMNCLUDIO        FLOAT(4)     NOT NULL,
  DMPREDS          INTEGER      NOT NULL,
  DMSROWS          FLOAT(4)     NOT NULL,
  DMSCANCST        FLOAT(4)     NOT NULL,
```

Figure 74. The CREATE TABLE statement for userid.DSN\_DSN\_DETCOST\_TABLE\_TABLE (part 1 of 3)

DMCOLS	SMALLINT	NOT NULL,
DMROWS	FLOAT(4)	NOT NULL,
RDSROWCST	FLOAT(4)	NOT NULL,
DMPAGECST	FLOAT(4)	NOT NULL,
DMDATAIO	FLOAT(4)	NOT NULL,
DMDATACPU	FLOAT(4)	NOT NULL,
DMDATATOT	FLOAT(4)	NOT NULL,
RDSROW	FLOAT(4)	NOT NULL,
SNCOLS	SMALLINT	NOT NULL,
SNROWS	FLOAT(4)	NOT NULL,
SNRECSZ	INTEGER	NOT NULL,
SNPAGES	FLOAT(4)	NOT NULL,
SNRUNS	FLOAT(4)	NOT NULL,
SNMERGES	FLOAT(4)	NOT NULL,
SNIOCOST	FLOAT(4)	NOT NULL,
SNCPUCOST	FLOAT(4)	NOT NULL,
SNCOST	FLOAT(4)	NOT NULL,
SNSCANIO	FLOAT(4)	NOT NULL,
SNSCANCPU	FLOAT(4)	NOT NULL,
SNSCANCOST	FLOAT(4)	NOT NULL,
SCCOLS	SMALLINT	NOT NULL,
SCROWS	FLOAT(4)	NOT NULL,
SCRECSZ	INTEGER	NOT NULL,
SCPAGES	FLOAT(4)	NOT NULL,
SCRUNS	FLOAT(4)	NOT NULL,
SCMERGES	FLOAT(4)	NOT NULL,
SCIOCOST	FLOAT(4)	NOT NULL,
SCCPUCOST	FLOAT(4)	NOT NULL,
SCCOST	FLOAT(4)	NOT NULL,
SCSCANIO	FLOAT(4)	NOT NULL,
SCSCANCPU	FLOAT(4)	NOT NULL,
SCSCANCOST	FLOAT(4)	NOT NULL,
COMPCARD	FLOAT(4)	NOT NULL,
COMPIOCOST	FLOAT(4)	NOT NULL,
COMPCPUCOST	FLOAT(4)	NOT NULL,
COMPCOST	FLOAT(4)	NOT NULL,
JOINCOLS	SMALLINT	NOT NULL,
EXPLAIN_TIME	TIMESTAMP	NOT NULL,
COSTBLK	INTEGER	NOT NULL,
COSTSTOR	INTEGER	NOT NULL,

Figure 75. The CREATE TABLE statement for userid.DSN\_DSN\_DETCOST\_TABLE\_TABLE (part 2 of 3)

MPBLK	INTEGER	NOT NULL,
MPSTOR	INTEGER	NOT NULL,
COMPOSITES	INTEGER	NOT NULL,
CLIPPED	INTEGER	NOT NULL,
PARTITION	INTEGER	NOT NULL,
TABREF	VARCHAR(64)	NOT NULL,
MAX_COMPOSITES	INTEGER	NOT NULL,
MAX_STOR	INTEGER	NOT NULL,
MAX_CPU	INTEGER	NOT NULL,
MAX_ELAP	INTEGER	NOT NULL,
TBL_JOINED_THRESH	INTEGER	NOT NULL,
STOR_USED	INTEGER	NOT NULL,
CPU_USED	INTEGER	NOT NULL,
ELAPSED	INTEGER	NOT NULL,
MIN_CARD_KEEP	FLOAT(4)	NOT NULL,
MAX_CARD_KEEP	FLOAT(4)	NOT NULL,
MIN_COST_KEEP	FLOAT(4)	NOT NULL,
MAX_COST_KEEP	FLOAT(4)	NOT NULL,

```

MIN_VALUE_KEEP          FLOAT(4)      NOT NULL,
MIN_VALUE_CARD_KEEP     FLOAT(4)      NOT NULL,
MIN_VALUE_COST_KEEP     FLOAT(4)      NOT NULL,
MAX_VALUE_KEEP          FLOAT(4)      NOT NULL,
MAX_VALUE_CARD_KEEP     FLOAT(4)      NOT NULL,
MAX_VALUE_COST_KEEP     FLOAT(4)      NOT NULL,
MIN_CARD_CLIP           FLOAT(4)      NOT NULL,
MAX_CARD_CLIP           FLOAT(4)      NOT NULL,
MIN_COST_CLIP           FLOAT(4)      NOT NULL,
MAX_COST_CLIP           FLOAT(4)      NOT NULL,
MIN_VALUE_CLIP          FLOAT(4)      NOT NULL,
MIN_VALUE_CARD_CLIP     FLOAT(4)      NOT NULL,
MIN_VALUE_COST_CLIP     FLOAT(4)      NOT NULL,
MAX_VALUE_CLIP          FLOAT(4)      NOT NULL,
MAX_VALUE_CARD_CLIP     FLOAT(4)      NOT NULL,
MAX_VALUE_COST_CLIP     FLOAT(4)      NOT NULL,
GROUP_MEMBER            VARCHAR(24)   NOT NULL,
PSEQIOCOST              FLOAT(4)      NOT NULL,
PSEQCPCUCOST            FLOAT(4)      NOT NULL,
PSEQCOST                FLOAT(4)      NOT NULL,
PADJIOCOST              FLOAT(4)      NOT NULL,
PADJCPUCOST             FLOAT(4)      NOT NULL,
PADJCOST                FLOAT(4)      NOT NULL)
IN database-name.table-space-name
CCSID UNICODE;

```

Figure 76. The CREATE TABLE statement for userid.DSN\_DSN\_DETCOST\_TABLE\_TABLE (part 3 of 3)

### Column descriptions

The following table describes the columns of DSN\_DETCOST\_TABLE.

Table 101. DSN\_DETCOST\_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: <ol style="list-style-type: none"> <li>1 The statement line number in the program</li> <li>2 QUERYNO clause</li> <li>3 The EXPLAIN statement</li> <li>4 EDM unique token in the statement cache.</li> </ol>
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
APPLNAME	VARCHAR(24) NOT NULL	The application plan name.
PROGNAME	VARCHAR(128) NOT NULL	The program name (binding an application) or the package name (binding a package).
PLANNO	SMALLINT NOT NULL	The plan number, a number used to identify each mini-plan with a query block.

Table 101. DSN\_DETCOST\_TABLE description (continued)

Column name	Data type	Description
OPENIO	FLOAT(4) NOT NULL	The Do-at-open IO cost for non-correlated subquery.
OPENCPU	FLOAT(4) NOT NULL	The Do-at-open CPU cost for non-correlated subquery.
OPENCOST	FLOAT(4) NOT NULL	The Do-at-open total cost for non-correlated subquery.
DMIO	FLOAT(4) NOT NULL	IBM internal use only.
DMCPU	FLOAT(4) NOT NULL	IBM internal use only.
DMTOT	FLOAT(4) NOT NULL	IBM internal use only.
SUBQIO	FLOAT(4) NOT NULL	IBM internal use only.
SUBQCOST	FLOAT(4) NOT NULL	IBM internal use only.
BASEIO	FLOAT(4) NOT NULL	IBM internal use only.
BASECPU	FLOAT(4) NOT NULL	IBM internal use only.
BASETOT	FLOAT(4) NOT NULL	IBM internal use only.
ONECOMPROWS	FLOAT(4) NOT NULL	The number of rows qualified after applying local predicates.
IMLEAF	FLOAT(4) NOT NULL	The number of index leaf pages scanned by Data Manager.
IMIO	FLOAT(4) NOT NULL	IBM internal use only.
IMPREFH	CHAR(2) NOT NULL	IBM internal use only.
IMMPRED	INTEGER NOT NULL	IBM internal use only.
IMFF	FLOAT(4) NOT NULL	The filter factor of matching predicates only.
IMSRPRED	INTEGER NOT NULL	IBM internal use only.
IMFFADJ	FLOAT(4) NOT NULL	The filter factor of matching and screening predicates.
IMSCANCST	FLOAT(4) NOT NULL	IBM internal use only.
IMROWCST	FLOAT(4) NOT NULL	IBM internal use only.
IMPAGECST	FLOAT(4) NOT NULL	IBM internal use only.
IMRIDSORT	FLOAT(4) NOT NULL	IBM internal use only.
IMMERGCST	FLOAT(4) NOT NULL	IBM internal use only.
IMCPU	FLOAT(4) NOT NULL	IBM internal use only.
IMTOT	FLOAT(4) NOT NULL	IBM internal use only.
IMSEQNO	SMALLINT NOT NULL	IBM internal use only.
DMPEREFH	FLOAT(4) NOT NULL	IBM internal use only.
DMCLUDIO	FLOAT(4) NOT NULL	IBM internal use only.
DMPREDS	INTEGER NOT NULL	IBM internal use only.
DMSROWS	FLOAT(4) NOT NULL	IBM internal use only.
DMSCANCST	FLOAT(4) NOT NULL	IBM internal use only.
DMCOLS	FLOAT(4) NOT NULL	The number of data manager columns.

Table 101. DSN\_DETCOST\_TABLE description (continued)

Column name	Data type	Description
DMROWS	FLOAT(4) NOT NULL	The number of data manager rows returned (after all stage 1 predicates are applied).
RDSROWCST	FLOAT(4) NOT NULL	IBM internal use only.
DMPAGECST	FLOAT(4) NOT NULL	IBM internal use only.
DMDATAIO	FLOAT(4) NOT NULL	IBM internal use only.
DMDATAIO	FLOAT(4) NOT NULL	IBM internal use only.
DMDATACPU	FLOAT(4) NOT NULL	IBM internal use only.
DMDATACPU	FLOAT(4) NOT NULL	IBM internal use only.
RDSROW	FLOAT(4) NOT NULL	The number of RDS rows returned (after all stage 1 and stage 2 predicates are applied).
SNCOLS	SMALLINT NOT NULL	The number of columns as sort input for new table.
SNROWS	FLOAT(4) NOT NULL	The number of rows as sort input for new table.
SNRECSZ	INTEGER NOT NULL	The record size for new table.
SNPAGES	FLOAT(4) NOT NULL	The page size for new table.
SNRUNS	FLOAT(4) NOT NULL	The number of runs generated for sort of new table.
SNMERGES	FLOAT(4) NOT NULL	The number of merges needed during sort.
SNIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
SNCPUCOST	FLOAT(4) NOT NULL	IBM internal use only.
SNCOST	FLOAT(4) NOT NULL	IBM internal use only.
SNSCANIO	FLOAT(4) NOT NULL	IBM internal use only.
SNSCANCPU	FLOAT(4) NOT NULL	IBM internal use only.
SNCCOLS	FLOAT(4) NOT NULL	The number of columns as sort input for Composite table.
SCROWS	FLOAT(4) NOT NULL	The number of rows as sort input for Composite Table.
SCRECSZ	FLOAT(4) NOT NULL	The record size for Composite table.
SCPAGES	FLOAT(4) NOT NULL	The page size for Composite table.
SCRUNS	FLOAT(4) NOT NULL	The number of runs generated during sort of composite.
SCMERGES	FLOAT(4) NOT NULL	The number of merges needed during sort of composite.
SCIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
SCCPUCOST	FLOAT(4) NOT NULL	IBM internal use only.
SCCOST	FLOAT(4) NOT NULL	IBM internal use only.
SCSCANIO	FLOAT(4) NOT NULL	IBM internal use only.
SCSCANCPU	FLOAT(4) NOT NULL	IBM internal use only.
SCSCANCOST	FLOAT(4) NOT NULL	IBM internal use only.

Table 101. DSN\_DETCOST\_TABLE description (continued)

Column name	Data type	Description
COMPCARD	FLOAT(4) NOT NULL	The total composite cardinality.
COMPIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
COMPCPUCOST	FLOAT(4) NOT NULL	IBM internal use only.
COMPCOST	FLOAT(4) NOT NULL	The total cost.
JOINCOLS	SMALLINT NOT NULL	IBM internal use only.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The EXPLAIN time stamp.
COSTBLK	INTEGER NOT NULL	IBM internal use only.
COSTSTOR	INTEGER NOT NULL	IBM internal use only.
MPBLK	INTEGER NOT NULL	IBM internal use only.
MPSTOR	INTEGER NOT NULL	IBM internal use only.
COMPOSITES	INTEGER NOT NULL	IBM internal use only.
CLIPPED	INTEGER NOT NULL	IBM internal use only.
TABREF	VARCHAR(64) NOT NULL	IBM internal use only.
MAX_COMPOSITES	INTEGER NOT NULL	IBM internal use only.
MAX_STOR	INTEGER NOT NULL	IBM internal use only.
MAX_CPU	INTEGER NOT NULL	IBM internal use only.
MAX_ELAP	INTEGER NOT NULL	IBM internal use only.
TBL_JOINED_THRESH	INTEGER NOT NULL	IBM internal use only.
STOR_USED	INTEGER NOT NULL	IBM internal use only.
CPU_USED	INTEGER NOT NULL	IBM internal use only.
ELAPSED	INTEGER NOT NULL	IBM internal use only.
MIN_CARD_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_CARD_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_COST_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_COST_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_CARD_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_COST_KEEP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_CARD_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_CARD_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_COST_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_COST_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_CARD_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MIN_VALUE_COST_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_VALUE_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_VALUE_CARD_CLIP	FLOAT(4) NOT NULL	IBM internal use only.
MAX_VALUE_COST_CLIP	FLOAT(4) NOT NULL	IBM internal use only.

Table 101. DSN\_DETCOST\_TABLE description (continued)

Column name	Data type	Description
GROUP_MEMBER	VARCHAR(240)	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
PSEQIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
PSEQIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
PSEQCPCUCOST	FLOAT(4) NOT NULL	IBM internal use only.
PSEQCOST	FLOAT(4) NOT NULL	IBM internal use only.
PADJIOCOST	FLOAT(4) NOT NULL	IBM internal use only.
PADJCPUCOST	FLOAT(4) NOT NULL	IBM internal use only.
PADJYCOST	FLOAT(4) NOT NULL	IBM internal use only.

#### DSN\_SORT\_TABLE:

The sort table, DSN\_SORT\_TABLE, contains information about the sort operations required by a query.

**Recommendation:** Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

#### CREATE TABLE statement

The following figure shows the format of DSN\_SORT\_TABLE.

Figure 77. The CREATE TABLE statement for userid.DSN\_SORT\_TABLE

```
CREATE TABLE userid.DSN_SORT_TABLE
(
  QUERYNO          INTEGER      NOT NULL,
  QBLOCKNO         SMALLINT    NOT NULL,
  PLANNO           SMALLINT    NOT NULL,
  APPLNAME          VARCHAR(24) NOT NULL,
  PROGNAME          VARCHAR(128) NOT NULL,
  COLLID           VARCHAR(128) NOT NULL WITH DEFAULT,
  SORTC            CHAR(5)      NOT NULL WITH DEFAULT,
  SORTN            CHAR(5)      NOT NULL WITH DEFAULT,
  SORTNO           SMALLINT    NOT NULL,
  KEYSIZE           SMALLINT    NOT NULL,
  ORDERCLASS        INTEGER     NOT NULL,
  EXPLAIN_TIME      TIMESTAMP   NOT NULL,
  GROUP_MEMBER      VARCHAR(24) NOT NULL
)
  IN database-name.table-space-name
  -name
  CCSID UNICODE;
```

Figure 78. The CREATE TABLE statement for userid.DSN\_SORT\_TABLE

## Column descriptions

The following table describes the columns of DSN\_SORT\_TABLE.

Table 102. DSN\_SORT\_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: <ol style="list-style-type: none"> <li>1 The statement line number in the program</li> <li>2 QUERYNO clause</li> <li>3 The EXPLAIN statement</li> <li>4 EDM unique token in the statement cache.</li> </ol>
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
PLANNO	SMALLINT NOT NULL	The plan number, a number used to identify each miniplan with a query block.
APPLNAME	VARCHAR(24) NOT NULL	The application name.
PROGNAME	VARCHAR(128) NOT NULL	The program name (binding an application) or the package name (binding a package).
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	The collection ID for the package.
SORTC	CHAR(5) NOT NULL WITH DEFAULT	Indicates the reasons for sort of the Composite table. Using a bitmap of 'G','J','O','U'. <p>'G' Group By</p> <p>'O' Order By</p> <p>'J' Join</p> <p>'U' Uniqueness</p>
SORTN	CHAR(5) NOT NULL WITH DEFAULT	Indicates the reasons for sort of the Composite table. Using a bitmap of 'G','J','O','U'. <p>'G' Group By</p> <p>'O' Order By</p> <p>'J' Join</p> <p>'U' Uniqueness</p>
SORTNO	SMALLINT NOT NULL	The sequence number of the sort.
KEYSIZE	SMALLINT NOT NULL	The sum of the lengths of the sort keys.
ORDERCLASS	INTEGER NOT NULL	IBM internal use only.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The EXPLAIN timestamp.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

**DSN\_SORTKEY\_TABLE:**

The sort key table, DSN\_SORTKEY\_TABLE, contains information about sort keys for all of the sorts required by a query.

**Recommendation:** Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

**CREATE TABLE statement**

The following figure shows the format of DSN\_SORTKEY\_TABLE.

```
CREATE TABLE userid.DSN_SORTKEY_TABLE
(
  QUERYNO           INTEGER      NOT NULL,
  QBLOCKNO          SMALLINT     NOT NULL,
  PLANNO            SMALLINT     NOT NULL,
  APPLNAME          VARCHAR(24)  NOT NULL,
  PROGNAME          VARCHAR(128) NOT NULL,
  COLLID            VARCHAR(128) NOT NULL WITH DEFAULT,
  SORTNO            SMALLINT     NOT NULL,
  ORDERNO           SMALLINT     NOT NULL,
  EXPTYPE           CHAR(3)      NOT NULL,
  TEXT              VARCHAR(128) NOT NULL,
  TABNO             SMALLINT     NOT NULL,
  COLNO             SMALLINT     NOT NULL,
  DATATYPE          CHAR(18)     NOT NULL,
  LENGTH            INTEGER      NOT NULL,
  CCSID              INTEGER      NOT NULL,
  ORDERCLASS        INTEGER      NOT NULL,
  EXPLAIN_TIME      TIMESTAMP    NOT NULL,
  GROUP_MEMBER      VARCHAR(24)  NOT NULL
)
  IN database-name.table-space-name

CCSID UNICODE;
```

Figure 79. The CREATE TABLE statement for userid.DSN\_SORTKEY\_TABLE

**Column descriptions**

The following table describes the columns of DSN\_SORTKEY\_TABLE.

Table 103. DSN\_SORTKEY\_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are:
		1        The statement line number in the program
		2        QUERYNO clause
		3        The EXPLAIN statement
		4        EDM unique token in the statement cache.
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
PLANNO	SMALLINT NOT NULL	The plan number, a number used to identify each miniplan with a query block.

Table 103. DSN\_SORTKEY\_TABLE description (continued)

Column name	Data type	Description
APPLNAME	VARCHAR(24) NOT NULL	The application name.
PROGNAME	VARCHAR(128) NOT NULL	The program name (binding an application) or the package name (binding a package).
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	The collection ID for the package.
SORTNO	SMALLINT NOT NULL	The sequence number of the sort
ORDERNO	SMALLINT NOT NULL	The sequence number of the sort key
EXPTYPE	CHAR(3) NOT NULL	The type of the sort key. The possible values are: <ul style="list-style-type: none"> <li>• 'COL'</li> <li>• 'EXP'</li> <li>• 'QRY'</li> </ul>
TEXT	VARCHAR(128) NOT NULL	The sort key text, can be a column name, an expression, or a scalar subquery, or 'Record ID'.
TABNO	SMALLINT NOT NULL	The table number, a number which uniquely identifies the corresponding table reference within a query.
COLNO	SMALLINT NOT NULL	The column number, a number which uniquely identifies the corresponding column within a query. Only applicable when the sort key is a column.
DATATYPE	CHAR(18)	The data type of sort key. The possible values are <ul style="list-style-type: none"> <li>• 'HEXADECIMAL'</li> <li>• 'CHARACTER'</li> <li>• 'PACKED FIELD '</li> <li>• 'FIXED(31)'</li> <li>• 'FIXED(15)'</li> <li>• 'DATE'</li> <li>• 'TIME'</li> <li>• 'VARCHAR'</li> <li>• 'PACKED FLD'</li> <li>• 'FLOAT'</li> <li>• 'TIMESTAMP'</li> <li>• 'UNKNOWN DATA TYPE'</li> </ul>
LENGTH	INTEGER NOT NULL	The length of sort key.
CCSID	INTEGER NOT NULL	IBM internal use only.
ORDERCLASS	INTEGER NOT NULL	IBM internal use only.
EXPLAIN_TIME	TIMESTAMP NOT NULL	The EXPLAIN timestamp.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

## DSN\_PGRANGE\_TABLE:

The page range table, DSN\_PGRANGE\_TABLE, contains information about qualified partitions for all page range scans in a query.

**Recommendation:** Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

### CREATE TABLE statement

The following figure shows the format of DSN\_PGRANGE\_TABLE.

```
CREATE TABLE userid.DSN_PGRANGE_TABLE
( QUERYNO          INTEGER          NOT NULL,
  QBLOCKNO         SMALLINT         NOT NULL,
  TABNO            SMALLINT         NOT NULL,
  RANGE            SMALLINT         NOT NULL,
  FIRSTPART        SMALLINT         NOT NULL,
  LASTPART         SMALLINT         NOT NULL,
  NUMPARTS         SMALLINT         NOT NULL,
  EXPLAIN_TIME     TIMESTAMP        NOT NULL,
  GROUP_MEMBER     VARCHAR(24)     NOT NULL
)
  IN database-name.table-space-name
CCSID UNICODE;
```

Figure 80. The CREATE TABLE statement for userid.DSN\_PGRANGE\_TABLE

### Column descriptions

The following table describes the columns of DSN\_PGRANGE\_TABLE.

Table 104. DSN\_PGRANGE\_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are:  1        The statement line number in the program 2        QUERYNO clause 3        The EXPLAIN statement 4        EDM unique token in the statement cache.
QBLOCKNO	SMALLINT NOT NULL	The query block number, a number used to identify each query block within a query.
TABNO	SMALLINT NOT NULL	The table number, a number which uniquely identifies the corresponding table reference within a query.
RANGE	SMALLINT NOT NULL	The sequence number of the current page range.
FIRSTPART	SMALLINT NOT NULL	The starting partition in the current page range.
LASTPART	SMALLINT NOT NULL	The ending partition in the current page range.
NUMPARTS	SMALLINT NOT NULL	The number of partitions in the current page range.
EXPLAIN_TIME	SMALLINT NOT NULL	The EXPLAIN timestamp.

Table 104. DSN\_PGRANGE\_TABLE description (continued)

Column name	Data type	Description
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

#### DSN\_VIEWREF\_TABLE:

The view reference table, DSN\_VIEWREF\_TABLE, contains information about all of the views and materialized query tables that are used to process a query.

**Recommendation:** Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

#### CREATE TABLE statement

The following figure shows the format of DSN\_VIEWREF\_TABLE.

```
CREATE TABLE userid.DSN_VIEWREF_TABLE (
  QUERYNO      INTEGER      NOT NULL WITH DEFAULT,
  APPLNAME     VARCHAR(24)  NOT NULL WITH DEFAULT,
  PROGNAME     VARCHAR(128) NOT NULL WITH DEFAULT,
  VERSION      VARCHAR(122) NOT NULL WITH DEFAULT,
  COLLID       VARCHAR(128) NOT NULL WITH DEFAULT,
  CREATOR      VARCHAR(128) NOT NULL WITH DEFAULT,
  NAME         VARCHAR(128) NOT NULL WITH DEFAULT,
  TYPE         CHAR(1)      NOT NULL WITH DEFAULT,
  MQTUSE       SMALLINT     NOT NULL WITH DEFAULT,
  EXPLAIN_TIME TIMESTAMP    NOT NULL WITH DEFAULT,
  GROUP_MEMBER VARCHAR(24)  NOT NULL
)
  IN database-name.table-space-name

CCSID UNICODE;
```

Figure 81. The CREATE TABLE statement for userid.DSN\_VIEWREF\_TABLE

#### Column descriptions

The following table describes the columns of DSN\_VIEWREF\_TABLE.

Table 105. DSN\_VIEWREF\_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are:
		1      The statement line number in the program
		2      QUERYNO clause
		3      The EXPLAIN statement
		4      EDM unique token in the statement cache.
APPLNAME	VARCHAR(24) NOT NULL	The application plan name.

Table 105. DSN\_VIEWREF\_TABLE description (continued)

Column name	Data type	Description
PROGNAME	VARCHAR(128) NOT NULL	The program name (binding an application) or the package name (binding a package).
VERSION	VARCHAR(128)	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. Blank if not applicable.
COLLID	VARCHAR(128)	The collection ID for the package. Applies only to an embedded EXPLAIN statement that is executed from a package or to a statement that is explained when binding a package. Blank if not applicable. The value DSN_DYNAMICSQLCACHE indicates that the row is for a cached statement.
CREATOR	VARCHAR(128)	Authorization ID of the owner of the object.
NAME	VARCHAR(128)	Name of the object.
TYPE	CHAR(1) NOT NULL WITH DEFAULT	The type of the object: <div> <div>'V'</div> <div>View</div> </div> <div> <div>'R'</div> <div>MQT that has been used to replace the base table for rewrite</div> </div> <div> <div>'M'</div> <div>MQT</div> </div>
MQTUSE	SMALLINT	IBM internal use only.
EXPLAIN TIME	TIMESTAMP	The EXPLAIN timestamp.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

#### DSN\_QUERY\_TABLE:

The query table, DSN\_QUERY\_TABLE, contains information about a SQL statement, and displays the statement before and after query transformation in XML.

**Recommendation:** Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

#### CREATE TABLE statement

The following figure shows the format of DSN\_QUERY\_TABLE.

```
CREATE TABLE userid.DSN_QUERY_TABLE(
  QUERYNO      INTEGER      NOT NULL,
  TYPE         CHAR(8)      NOT NULL,
  QUERY_STAGE  CHAR(8)      NOT NULL,
  SEQNO        INTEGER      NOT NULL,
  NODE_DATA    CLOB(2M)     NOT NULL,
  EXPLAIN_TIME TIMESTAMP    NOT NULL,
  QUERY_ROWID  ROWID        NOT NULL GENERATED ALWAYS,
  GROUP_MEMBER VARCHAR(24)  NOT NULL,
  HASHKEY      INTEGER      NOT NULL,
  HAS_PRED     CHAR(1)      NOT NULL
) IN database-name.table-space-name

CCSID UNICODE;
```

Figure 82. The CREATE TABLE statement for userid.DSN\_QUERY\_TABLE

### Column descriptions

The following table describes the columns of DSN\_QUERY\_TABLE.

Table 106. DSN\_QUERY\_TABLE description

Column name	Data type	Description
QUERYNO	INTEGER NOT NULL	The query number, a number used to help identify the query being explained. It is not a unique identifier. Using negative number will cause problems. The possible sources are: <ol style="list-style-type: none"> <li>1 The statement line number in the program</li> <li>2 QUERYNO clause</li> <li>3 The EXPLAIN statement</li> <li>4 EDM unique token in the statement cache.</li> </ol>
TYPE	CHAR(8) NOT NULL	The type of the data in the NODE_DATA column.
QUERY_STAGE	CHAR(8) NOT NULL WITH DEFAULT	The stage during query transformation when this row is populated.
SEQNO	NOT NULL	The sequence number for this row if NODE_DATA exceeds the size of its column.
NODE_DATA	CLOB(2M)	The XML data containing the SQL statement and its query block, table, and column information.
EXPLAIN_TIME	TIMESTAMP	The EXPLAIN timestamp.
QUERY_ROWID	ROWID NOT NULL GENERATED ALWAYS	The ROWID of the statement.
GROUP_MEMBER	VARCHAR(24) NOT NULL	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.
HASHKEY	INTEGER NOT NULL	The hash value of the contents in NODE_DATA
HAS_PRED	CHAR(1) NOT NULL	When NODE_DATA contains an SQL statement, this column indicates if the statement contains a parameter marker literal, non-parameter marker literal, or no predicates.

### Tables that are used exclusively by DB2 optimization tools

Optimization tools such as IBM Optimization Service Center for DB2 for z/OS and IBM DB2 Optimization Expert for z/OS use certain tables that are provided with DB2 to perform workload-level monitoring and tuning tasks.

**Restriction:** Do not directly manipulate the data that is stored in tables that are used exclusively by DB2 optimization tools. Doing so might cause Optimization Service Center and other optimization tools to malfunction

### Workload control center tables:

The Optimization Service Center for DB2 for z/OS workload control center maintains a set of tables to record the definition and content of workloads and monitor profiles.

The tables are maintained in the DB2OSC database. Do not directly manipulate the data that is stored in these tables. Doing so might cause Optimization Service Center and other optimization tools to malfunction. The workload control center uses the following tables:

*DB2OSC.DSN\_WCC\_WORKLOADS:*

The workload table records all of the workloads defined within the current subsystem, including the workload ID, name, status and description.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

*Table 107. DB2OSC.DSN\_WCC\_WORKLOADS description*

Column name	Data type	Description
WLID	INTEGER GENERATED BY DEFAULT AS IDENTITY	A unique identifier of the workload. Primary key.
NAME	VARCHAR(128)NOT NULL	Name of the workload.
DESCRIPTION	VARCHAR(2000)	Description of the workload.
STATUS	SMALLINT NOT NULL DEFAULT 1	Status of the workload: <ul style="list-style-type: none"> <li>1 The workload is defined, and there is no statement in the workload.</li> <li>2 The workload definition is being updated.</li> <li>3 The workload is being captured.\$5 – The workload is captured, there are statements captured into the workload.</li> <li>8 The workload is being explained.</li> <li>9 All of the statements in the workload have explain information.</li> <li>10 The workload is being analyzed.</li> <li>11 The workload is locked.</li> <li>12 Some sources in the workload are being captured.</li> <li>13 Some sources in the workload are not captured.</li> </ul>
ANALYZE_COUNT	SMALLINT NOT NULL DEFAULT 0	If the status is Analyzing, this column indicates how many workload based advisors are running on the current workload.
OWNER	VARCHAR(128) NOT NULL	Owner of a workload.

Table 107. DB2OSC.DSN\_WCC\_WORKLOADS description (continued)

Column name	Data type	Description
EXPLAIN_STATUS	SMALLINT DEFAULT 0	<p>The explain status of a workload:</p> <ol style="list-style-type: none"> <li>1 One or more statements in the workload do not have EXPLAIN information.</li> <li>2 Some statements in the workload have EXPLAIN information only in PLAN_TABLE</li> <li>3 Some statements in the workload have explain information only in the external EXPLAIN tables.</li> <li>4 Some statements in the workload have explain information in the internal EXPLAIN tables except DSN_QUERY_TABLE.</li> <li>5 All of the statements in the workload have EXPLAIN information in both external and internal EXPLAIN tables.</li> </ol>
CPUTIME_STATUS	CHAR(1) NOT NULL DEFAULT 'N'	Indicates whether the workload has CPU time statistics.

#### Related reference

“SYSIBM.DSN\_PROFILE\_ATTRIBUTES” on page 580

DB2OSC.DSN\_WCC\_WL\_SOURCES:

The workload source table contains all of the source information for workloads defined in the workload table, including the source ID, source name, source description.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 108. DB2OSC.DSN\_WCC\_WL\_SOURCES description

Column name	Data type	Description
SRCID	INTEGER GENERATED BY DEFAULT AS IDENTITY	A unique identifier for source. Primary key.
WLID	INTEGER NOT NULL	Foreign key. References DSN_WCC_WORKLOADS.WLID.
NAME	VARCHAR(128) NOT NULL	Name of a source.

Table 108. DB2OSC.DSN\_WCC\_WL\_SOURCES description (continued)

Column name	Data type	Description
TYPE	SMALLINT NOT NULL	Type of the source, the following source types exist in the workload control center: 1 PLAN 2 PACKAGE 3 CACHE 4 CATEGORY 5 FILEDIR 6 WORKLOAD 7 DB2 QMF 8 QMFHPO 9 MONITOR 10 INPUT
DESCRIPTION	VARCHAR(2000)	Description of the source
STATUS	SMALLINT NOT NULL WITH DEFAULT 1	Status of the source: 1 The source is defined, and there is no statement in the source. 2 The source definition is being updated. 3 The source is being captured. 5 The source is captured, there are statements captured into the source. 8 EXPLAIN is running for the source. 9 All of the statements in the source have EXPLAIN information. 10 The source is being analyzed. 11 The source is locked.
CONSOLIDATE_STATUS	SMALLINT NOT NULL DEFAULT 21	Consolidation status of the source: 21 Neither the access plans nor the literal values of the statements in the source are consolidated. 22 The access plans of the statements in the source are consolidated. 23 The literal values of the statements in the source are consolidated into parameter markers. 24 Both the access plans and the literal values of the statements in the source are consolidated.
LASTUPDATETS	TIMESTAMP NOT NULL	The timestamp when the source definition was updated the last time.
QUERY_COUNT	INTEGER NOT NULL DEFAULT 0	The number of queries captured in the source.
PROFILEID	INTEGER	The corresponding monitor profile id.

Table 108. DB2OSC.DSN\_WCC\_WL\_SOURCES description (continued)

Column name	Data type	Description
MONITOR_STATUS	SMALLINT NOT NULL DEFAULT 1	The monitor status of monitor sources: <ol style="list-style-type: none"> <li>1 The monitor is to be started.</li> <li>2 The monitor is being started.</li> <li>3 The monitor is started.</li> <li>4 The monitor is to be stopped.</li> <li>5 The monitor is being stopped.</li> <li>6 The monitor is stopped.</li> </ol>
NEW_QUERY_COUNT	INTEGER NOT NULL DEFAULT 0	The number of newly captured statements in the latest capturing.
LAST_CAPTURE_TS	TIMESTAMP	The timestamp of the latest capturing.
MONITOR_ENABLED	CHAR(1) NOT NULL DEFAULT 'Y'	Indicates whether the monitor is enabled for the source. Y if the monitor is enabled, otherwise N.

*DB2OSC.DSN\_WCC\_WL\_SOURCE\_DTL:*

The detailed source table records the filters of a source in DSN\_WCC\_WORKLOAD\_SOURCES. A filter can contain filter items.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 109. DB2OSC.DSN\_WCC\_SOURCE\_DTL description

Column name	Data type	Description
SRCID	INTEGER NOT NULL	Foreign key. References DSN_WCC_WL_SOURCES.SRCID.
SEQNO	SMALLINT NOT NULL DEFAULT 0	The sequence number of the filter.
CONDITION	VARCHAR(128)	Left hand side of a filter.
OPERATOR	CHAR(32)	=, >, <, etc.
VALUE	VARCHAR(512)	Right hand side of a filter.

*DB2OSC.DSN\_WCC\_WL\_AUTHIDS:*

The workload user relation table records which users can access a workload, and the privileges of each ID.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 110. DB2OSC.DSN\_WCC\_WL\_AUTHIDS description

Column name	Data type	Description
WLID	INTEGER	Foreign key. Workload ID references DSN_WCC_WORKLOADS.WLID.
AUTHID	VARCHAR(128) NOT NULL	SQLIDs of the users who can access the workload.

Table 110. DB2OSC.DSN\_WCC\_WL\_AUTHIDS description (continued)

Column name	Data type	Description
PRIVILEGE	CHAR(1)	Indicates the privileges that an AUTHID holds for the workload.  ‘R’ Read only  ‘W’ Update

DB2OSC.DSN\_WCC\_TB\_STATUS:

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 111. DB2OSC.DSN\_WCC\_TB\_STATUS description

Column name	Data type	Description
NAME	VARCHAR(128) NOT NULL	The name of the table to be locked.
TASKID	INTEGER	Foreign key references DSN_WCC_TASKS.TASKID

DB2OSC.DSN\_WCC\_EV\_HISTORY:

The event history table records all of the events related to workloads that are defined in the workload table.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 112. DB2OSC.DSN\_WCC\_EV\_HISTORY description

Column name	Data type	Description
WLID	INTEGER	Foreign key references to DSN_WCC_WORKLOADS.WLID. This column indicates the event is related to which workload.
SRCID	INTEGER	Foreign key references to DSN_WCC_WL_SOURCES.SRCID. This column indicates the event is related to which source.
TASKID	INTEGER	Foreign key references DSN_WCC_TASKS.TASKID. If the event is related a task defined for the workload.
OWNER	VARCHAR(128)	Event owner.
TYPE	SMALLINT NOT NULL	Indicate workload related event:  1 Create workload event.  2 Update workload event.  3 Capture workload event.  8 Explain workload event.  20 Analyze workload event.
BEGIN_TS	TIMESTAMP NOT NULL	When the event started.
END_TS	TIMESTAMP NOT NULL	When the event finished.
DESCRIPTION	VARCHAR(200)	Description of the event.

Table 112. DB2OSC.DSN\_WCC\_EV\_HISTORY description (continued)

Column name	Data type	Description
STATUS	CHAR(1) NOT NULL	Status of the event.\$F – The event is finished.\$C – The event is canceled.\$A – The event is abnormally terminated.

#### DB2OSC.DSN\_WCC\_EP\_HISTORY:

The explain history table records all of the statement instance IDs and their EXPLAIN timestamps.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 113. DB2OSC.DSN\_WCC\_EP\_HISTORY description

Column name	Data type	Description
INSTID	INTEGER NOT NULL	Foreign key reference DSN_WCC_STATEMENT_INSTS.INSTID
EXPLAIN_TIME	TIMESTAMP NOT NULL	The explain timestamp.

#### DB2OSC.DSN\_WCC\_MESSAGES:

The workload messages table records all of the error and warning messages that were issued while the monitor profile is active.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 114. DB2OSC.DSN\_WCC\_MESSAGES description

Column name	Data type	Description
INSERT_TS	TIMESTAMP NOT NULL	The timestamp when the message was inserted.
TYPE	SMALLINT NOT NULL	The message type, and possible values are: <ul style="list-style-type: none"> <li>• ERROR</li> <li>• WARNING</li> <li>• INFO</li> </ul>
CONFIG	VARCHAR(20)	For internal use.
MESSAGE	VARCHAR(2048)	The message.

#### DB2OSC.DSN\_WCC\_STMT\_TEXTS:

The statement text table records the text of SQL statements.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 115. DB2OSC.DSN\_WCC\_STATEMENT\_TEXTS description

Column name	Data type	Description
STMT_TEXT_ID	INTEGER NOT NULL	Unique identifier for a statement text. Primary key.
QUALIFIER	VARCHAR(128) NOT NULL	Default table qualifier for this statement.
STMT_TEXT	CLOB(2M) NOT NULL	Statement text.
STMT_ROWID	ROWID NOT NULL GENERATED ALWAYS	ROWID.
HASHKEY	INTEGER NOT NULL DEFAULT 0	The length of the statement
TRANSFORMED_HASKEY	INTEGER NOT NULL DEFAULT 0	The length of the query generated by query table for transformed statement text.

*DB2OSC.DSN\_WCC\_STMT\_INSTS:*

The statement instance table records all of the SQL statements that are captured by workloads, including their statement text ID, and bind options.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 116. DB2OSC.DSN\_WCC\_STATEMENT\_INSTS description

Column name	Data type	Description
INSTID	INTEGER NOT NULL	A unique identifier for statement instance. Primary key.
STMT_TEXT_ID	INTEGER	Foreign key references DSN_WCC_STMT_TEXTS.STMTID.
TRANSFORMED_TEXTID	INTEGER	Foreign key references DSN_WCC_STATEMENT_TEXT.STMT_TEXT_ID
WLID	INTEGER	Foreign key references DSN_WCC_WORKLOADS.WLID
SRCID	SMALLINT	Foreign key references DSN_WCC_SOURCES.SRCID
PRMAUTH	VARCHAR(128)	Primary authorization ID of the user that did the initial prepares.
CURSQLID	VARCHAR(128)	CURRENT SQLID of the user that did the initial prepares.
PLANNAME	VARCHAR(24)	Application plan name. For DB2 QMF OWNER
COLLID	VARCHAR(128)	Collection ID. For DB2 QMF: NAME
PKGNAME	VARCHAR(128)	Package name. For DB2 QMF: TYPE
VERSION	VARCHAR(122)	Version name. For DB2 QMF: SUBTYPE
SECTNOI	INTEGER	Section number.
REOPT	CHAR(5)	Re-optimize option.

Table 116. DB2OSC.DSN\_WCC\_STATEMENT\_INSTS description (continued)

Column name	Data type	Description
BIND_ISO	CHAR(2)	ISOLATION BIND option:  <b>'UR'</b> Uncommitted Read  <b>'CS'</b> Cursor Stability  <b>'RS'</b> Read Stability  <b>'RR'</b> Repeatable Read
BIND_CDATA	CHAR(1)	CURRENTDATA BIND option:  <b>'Y'</b> CURRENTDATA(YES)  <b>'N'</b> CURRENTDATA(NO) For DB2 QMF: RESTRICTED
BIND_DYNRL	CHAR(1)	DYNAMICRULES BIND option:  <b>'B'</b> DYNAMICRULES(BIND)  <b>'R'</b> DYNAMICRULES(RUN)
BIND_DEGRE	CHAR(1)	CURRENT DEGREE value:  <b>'A'</b> CURRENT DEGREE = ANY  <b>'1'</b> CURRENT DEGREE = 1
BIND_SQLRL	CHAR(1)	CURRENT RULES value:  <b>'D'</b> CURRENT RULES = DB2  <b>'S'</b> CURRENT RULES = SQL
BIND_CHOLD	CHAR(1)	Cursor WITH HOLD bind option:  <b>'Y'</b> Initial PREPARE was done for a cursor WITH HOLD  <b>'N'</b> Initial PREPARE was not done for cursor WITH HOLD
CACHED_TS	TIMESTAMP	For static SQL the statement's BIND_TIME, for dynamic SQL the statement's cached timestamp. For DB2 QMF: CREATED
LAST_UPDATE_TS	TIMESTAMP	Timestamp when the statement was captured last time. For DB2 QMF: LAST_USED
LAST_EXPLAIN_TS	TIMESTAMP	Latest explain timestamp.
LITERALS	VARCHAR(2000)	Contains the consolidated literals.
PERMANENT	CHAR(1) DEFAULT 'Y'	Indicates whether this instance is permanent or snapshot result. Default value is 'Y'.

Table 116. DB2OSC.DSN\_WCC\_STATEMENT\_INSTS description (continued)

Column name	Data type	Description
EXPLAIN_STATUS	SMALLINT NOT NULL DEFAULT 0	<p>The EXPLAIN status of a statement:</p> <ol style="list-style-type: none"> <li>1 The statement does not have explain information.</li> <li>2 The statement has explain information only in PLAN_TABLE</li> <li>3 The statement has EXPLAIN information only in the external EXPLAIN tables.</li> <li>4 The statement has EXPLAIN information in the internal explain tables except DSN_QUERY_TABLE.</li> <li>5 The statement has EXPLAIN information in both external and internal EXPLAIN tables.</li> </ol>

*DB2OSC.DSN\_WCC\_STATEMENT\_INST\_SUMMY:*

This table records the summary information of monitored statements, including the relationship between pushed-out and snapshot statements.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 117. DB2OSC.DSN\_WCC\_STATEMENT\_INST\_SUMMY description

Column name	Data type	Description
SEQNO	INTEGER NOT NULL	The sequence number of the statement instance. Primary key.
SRCID	INTEGER	Foreign key references DSN_WCC_WL_SOURCES.SRCID
PERMANENT_INSTID	INTEGER	The statement instance id of pushed-out statement.
SNAPSHOT_INSTID	INTEGER	The statement instance id of snapshot statement.
SUMMARY	SMALLINT	<p>Summary information indicates whether this is a pushed-out statement, a snapshot statement, or both.</p> <ol style="list-style-type: none"> <li>0 pushed-out statement</li> <li>1 snapshot statement</li> <li>2 both</li> </ol>

*DB2OSC.DSN\_WCC\_TASKS:*

The workload task table records all of the tasks scheduled for a workload, and indicates task attributes and status.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 118. DB2OSC.DSN\_WCC\_TASKS description

Column name	Data type	Description
TASKID	INTEGER GENERATED BY DEFAULT AS IDENTITY	A unique identifier for a task. Primary key.
NEXTTASK	INTEGER	The id of the next task.
SCHEDULED_ TASKID	INTEGER	The id of the task in Admin Scheduler.
WLID	INTEGER	Foreign key, which references DSN_WCC_WORKLOADS.WLID. It represents the task is defined for which workload.
SRCID	INTEGER	Foreign key, which references DSN_WCC_WL_SOURCES.SRCID. If this is a capture workload task, this column indicates which source is going to be captured.
TYPE	SMALLINT NOT NULL	Types of tasks: <ol style="list-style-type: none"> <li>1 Capture workload task.</li> <li>2 Consolidate literal values task.</li> <li>3 Consolidate access plan table.</li> <li>4 EXPLAIN workload task.</li> <li>5 Analyze workload task.</li> <li>6 Monitor task.</li> <li>7 Snapshot task.</li> </ol>
SUBTYPE	SMALLINT	For capture tasks, indicates capture type: <ol style="list-style-type: none"> <li>1 one-time capture</li> <li>2 multi-time sample.</li> <li>3 continuous monitoring</li> </ol> For EXPLAIN task, indicates the EXPLAIN type <ol style="list-style-type: none"> <li>1 EXPLAIN all</li> <li>2 explain partial</li> </ol>
CREATOR	VARCHAR(128) NOT NULL	The creator of the task.
START_TIME	TIMESTAMP	Time when this task is scheduled to start.
END_TIME	TIMESTAMP	Time when this task is scheduled to finish. (multiple time sampling)
CONSOLIDATION_ TIME	TIMESTAMP	<ul style="list-style-type: none"> <li>For TYPE='1', the time indicates the consolidation time.</li> <li>For TYPE='4', EXPLAIN runs only for statements that have an EXPLAIN timestamp that is earlier than the timestamp in this column.</li> <li>For blank, EXPLAIN runs for all statements that have no EXPLAIN information.</li> </ul>
INTERVAL	INTEGER	If this is a capture task, indicating the number of minutes between two samples.

Table 118. DB2OSC.DSN\_WCC\_TASKS description (continued)

Column name	Data type	Description
CONSOLIDATE_RTINFO	CHAR(1)	Indicates whether the retrieve runtime information during capture source from dynamic statement cache: 'Y' if retrieving runtime information, N if not retrieving runtime information.
CONSOLIDATE_EPINFO	SMALLINT	The type of access path consolidation: <ol style="list-style-type: none"> <li>1 The access path will not be consolidated.</li> <li>2 Keep only the most recent access path.</li> <li>3 Keep all access paths.</li> <li>4 Remove duplicate access paths.</li> </ol>
CONSOLIDATE_LITERA	CHAR(1)	Indicate whether consolidate literal values or not.
KEEP_STATEMENTS	CHAR(1)	For a capture workload task this column indicates whether keep the statements already captured in a workload or not. 'Y' if this is an incremental capture and statements already captured by the workload will not be overwritten.
STATUS	CHAR(1) NOT NULL	Status of the task.
START_TRACE	CHAR(1)	Indicates whether start statement cache trace or not before capture queries from cache.
STOP_TRACE	CHAR(1)	Indicates whether stop statement cache trace or not after capture queries from cache.
WARMUP_TIME	INTEGER	Indicates the number of minutes between start statement cache trace and the first sapling of statement cache.
LAST_FIRE_TIME	TIMESTAMP	Indicates the last fire timestamp for the task.
LAST_UPDATE_TS	TIMESTAMP	The time when this task was updated last time.
CONFIG	BLOB(2M)	For internal use.
CONFIG_ROWID	ROWID	ROWID for CONFIG.

#### DB2OSC.DSN\_WCC\_STMT\_RUNTM:

The statement runtime table records the runtime information of SQL statements.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 119. DB2OSC.DSN\_WCC\_STMT\_RUNTM description

Column name	Data type	Description
INSTID	INTEGER NOT NULL	Foreign key references DSN_WCC_STMT_INST.INSTID.
STAT_EXEC	INTEGER	Number of executions of statement.
STAT_GPAG	INTEGER	Number of get page operations performed for statement.
STAT_SYNR	INTEGER	Number of synchronous buffer reads performed for statement.

Table 119. DB2OSC.DSN\_WCC\_STMT\_RUNTM description (continued)

Column name	Data type	Description
STAT_WRIT	INTEGER	Number of buffer write operations performed for statement.
STAT_EROW	INTEGER	Number of rows examined for statement.
STAT_PROW	INTEGER	Number of rows processed for statement.
STAT_SORT	INTEGER	Number of sorts performed for statement.
STAT_INDX	INTEGER	Number of index scans performed for statement.
STAT_RSCN	INTEGER	Number of relation scans performed for statement.
STAT_PGRP	INTEGER	Number of parallel groups created for statement.
STAT_ELAP	FLOAT	Accumulated elapsed time used for statement.
STAT_CPU	FLOAT	Accumulated CPU time used for statement.
STAT_SUS_SYNIO	FLOAT	Accumulated wait time for synchronous I/O.
STAT_SUS_LOCK	FLOAT	Accumulated wait time for lock and latch request.
STAT_SUS_SWIT	FLOAT	Accumulated wait time for synchronous execution unit switch.
STAT_SUS_GLCK	FLOAT	Accumulated wait time for global locks.
STAT_SUS_OTHM	FLOAT	Accumulated wait time for read activity done by another thread.
STAT_SUS_OTHW	FLOAT	Accumulated wait time for write activity done by another thread.
STAT_RIDLMT	INTEGER	Number of times a RID list was not used because the number of RIDs would have exceeded one or more internal DB2 limits.
STAT_RIDSTOR	INTEGER	Number of times a RID list was not used because not enough storage was available to hold the list of RIDs.
AVG_STAT_GPAG	INTEGER	The average number of get page operations performed for statement.
AVG_STAT_SYNR	INTEGER	The average number of synchronous buffer reads performed for statement.
AVG_STAT_WRIT	INTEGER	The average number of buffer write operations performed for statement.
AVG_STAT_EROW	INTEGER	The average number of rows examined for statement.
AVG_STAT_PROW	INTEGER	The average number of rows processed for statement.
AVG_STAT_SORT	INTEGER	The average number of sorts performed for statement.
AVG_STAT_INDX	INTEGER	The average number of index scans performed for statement.
AVG_STAT_RSCN	INTEGER	The average number of relation scans performed for statement.
AVG_STAT_PGRP	INTEGER	The average number of parallel groups created for statement.
AVG_STAT_ELAP	FLOAT	The average elapsed time used for statement.
AVG_STAT_CPU	FLOAT	The average CPU time used for statement.

Table 119. DB2OSC.DSN\_WCC\_STMT\_RUNTM description (continued)

Column name	Data type	Description
AVG_STAT_SUS_SYNIO	FLOAT	The average wait time for synchronous I/O.
AVG_STAT_SUS_LOCK	FLOAT	The average wait time for lock and latch request.
AVG_STAT_SUS_SWIT	FLOAT	The average wait time for synchronous execution unit switch.
AVG_STAT_SUS_GLCK	FLOAT	The average wait time for global locks.
AVG_STAT_SUS_OTHR	FLOAT	The average wait time for read activity done by another thread.
AVG_STAT_SUS_OTHW	FLOAT	The average wait time for write activity done by another thread.
AVG_STAT_RIDLIMT	INTEGER	The average number of times a RID list was not used because the number of RIDs would have exceeded one or more internal DB2 limits.
AVG_STAT_RIDSTOR	INTEGER	The average number of times a RID list was not used because not enough storage was available to hold the list of RIDs.
EXCEPTION_COUNT	INTEGER NOT NULL DEFAULT 0	The number of exceptions that occurred for the statement during the monitoring period. This value is applicable only to the summary rows. For exception rows, the value is zero.
CAPTURED	CHAR(1) NOT NULL WITH DEFAULT 'Y'	Indicates whether the runtime information is captured from DB2 or manually added.
BEGIN_TS	TIMESTAMP	The start timestamp when the runtime information is collected.
END_TS	TIMESTAMP	The end timestamp when the runtime information is collected.
STAT_CPU_STMT	FLOAT	The CPU time for a statement having exception.

#### DB2OSC.DSN\_WCC\_OBJ\_RUNTM:

The object runtime table stores additional information pushed-out by the monitor exceptions function. It contains information that pertains to both tables and indexes that are accessed during the execution of an SQL statement.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 120. DB2OSC.DSN\_WCC\_OBJECT\_RUNTIME description

Column name	Data type	Description
INSTID	INTEGER NOT NULL	Foreign key references DSN_WCC_STATEMENT_INSTANCES.INSTID.
QBLKNO	SMALLINT NOT NULL WITH DEFAULT	A number that identifies each query block within a query. The values of the numbers are not in any particular order, nor are they necessarily consecutive.
PLANNO	SMALLINT NOT NULL WITH DEFAULT	The number of step in which the query indicated in QBLOCKNO was processed. This column indicates the order in which the steps were executed.

Table 120. DB2OSC.DSN\_WCC\_OBJECT\_RUNTIME description (continued)

Column name	Data type	Description
MIXOPSEQ	SMALLINT NOT NULL WITH DEFAULT	The sequence number of a step in a multiple index operation:  <b>1, 2,...n</b> For the steps of multiple index procedure (ACCESSTYPE is MX, MI, or MU)  <b>0</b> For any other rows (ACCESSTYPE is I, I1, M, N, R, or blank.)
PAR_QBLKNO	SMALLINT NOT NULL WITH DEFAULT	A number that indicates the QBLOCKNO of the parent query block.
PAR_PLANNO	SMALLINT NOT NULL WITH DEFAULT	The plan number of the parent query.
DMSROW_SET	FLOAT NOT NULL WITH DEFAULT	The estimate of rows scanned by Data Manager (after all indexed predicates are applied).
DMROWS_SET	FLOAT NOT NULL WITH DEFAULT	The estimate of rows returned by the data manager (after all stage 1 predicates are applied).
RDSROW_SET	FLOAT NOT NULL WITH DEFAULT	The estimate of rows qualified by RDS.
COMPCD_SET	FLOAT NOT NULL WITH DEFAULT	The estimate on composite cardinality.
PROCESSED_ROWS	FLOAT NOT NULL WITH DEFAULT	The total number of rows processed in all read-in pages.
LOOKAT_ROWS	FLOAT NOT NULL WITH DEFAULT	The total number of rows that DB2 reads in all read-in pages.
DMROWS	FLOAT NOT NULL WITH DEFAULT	The total number of rows that are qualified by the data manager component of DB2.
RDSROWS	FLOAT NOT NULL WITH DEFAULT	The total number of rows that are qualified by relational data system component of DB2.
INDEX_FF_EST	FLOAT NOT NULL WITH DEFAULT	The filter factor estimate (for index object).
INDEX_FFADJ_EST	FLOAT NOT NULL WITH DEFAULT	The adjusted filter factor (for index object).
IX_PROCESSED_ROWS	FLOAT NOT NULL WITH DEFAULT	The number of rows processed (for index object).
IX_LOOKAT_ROWS	FLOAT NOT NULL WITH DEFAULT	The number of rows looked at (for index object).
IX_DMROWS	FLOAT NOT NULL WITH DEFAULT	The number of rows qualified by DM (for index object).
IX_RDSROWS	FLOAT NOT NULL WITH DEFAULT	The rows qualified by RDS (for index object).
CREATION_TS	TIMESTAMP NOT NULL WITH DEFAULT	The timestamp when this record is created.

DB2OSC.DSN\_WCC\_WL\_INFO:

The workload information table records the analytical results of workload-based advisors.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 121. DB2OSC.DSN\_WCC\_WL\_INFO description

Column name	Data type	Description
WLID	INTEGER NOT NULL	Foreign key references DSN_WCC_WORKLOADS.WLID.
STATUS	SMALLINT	Status of the workload analysis result.
BEGIN_TS	TIMESTAMP	The begin timestamp when the workload-based advisor started to analyze a workload.
END_TS	TIMESTAMP	The end timestamp when the workload-based advisor finished analyzing a workload.
SEQNO	SMALLINT	The sequence number.
DETAIL	BLOB(2M) NOT NULL	The analytical results.
DETAIL_ROWID	ROWID	ROWID.
DESCRIPTION	VARCHAR(128)	The description for a workload-based advisor.

DB2OSC.DSN\_WCC\_STMT\_INFO:

The explain information table records the EXPLAIN information and parse information of a statement instance in XML format.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 122. DB2OSC.DSN\_WCC\_STMT\_INFO description

Column name	Data type	Description
INSTID	INTEGER NOT NULL	Foreign key references DSN_WCC_STMT_INST.INSTID
TYPE	SMALLINT	Indicates the type pf statement info.
SEQNO	SMALLINT	Sequence number for the XML.
EXPLAIN_TIME	TIMESTAMP	The explain timestamp.
DETAIL	BLOB(2M) NOT NULL	The XML containing explain information
DETAIL_ROWID	ROWID	ROWID

DB2OSC.DSN\_WCC\_CAP\_TMP\_ES:

This table records the queries that are captured by each turn of sample during a multi-turn sample operation (a capture task may contain several turns of sampling), including statement text and accumulated runtime information.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 123. DB2OSC.DSN\_WCC\_CAP\_TMP\_ES description

Column name	Data type	Description
SRCID	INTEGER	Foreign key references DSN_WCC_WL_SOURCES.SRCID
STMT_ID	INTEGER NOT NULL	Statement cache identifier.
STMT_TEXT_ID	INTEGER	Foreign key references DSN_WCC_STMT_TEXTS.STMT_TEXT_ID
INST_ID	INTEGER	Foreign key references DSN_WCC_STMT_INST.INSTID
CACHED_TS	TIMESTAMP NOT NULL	Timestamp when statement was cached.
EXPLAIN_TS	TIMESTAMP	Timestamp when this information is retrieved from cache.
STAT_TS	TIMESTAMP	Timestamp when the IFCID 318 is started.
PRIMAUTH	VARCHAR(128)	User ID - Primary authorization ID of the user that did the initial PREPARE.
CURSQLID	VARCHAR(128)	CURRENT SQLID of the user that did the initial prepare.
BIND_QUALIFIER	VARCHAR(128)	Bind Qualifier, object qualifier for unqualified table names.
BIND_ISO	CHAR(2)	ISOLATION BIND option.  <b>'UR'</b> Uncommitted Read  <b>'CS'</b> Cursor Stability  <b>'RS'</b> Read Stability  <b>'RR'</b> Repeatable Read
BIND_CDATA	CHAR(1)	CURRENTDATA BIND option:  <b>'Y'</b> CURRENTDATA(YES) <b>'N'</b> CURRENTDATA(NO)
BIND_DYNRL	CHAR(1)	DYNAMICRULES BIND option:  <b>'B'</b> DYNAMICRULES(BIND) <b>'R'</b> DYNAMICRULES(RUN)
BIND_DEGRE	CHAR(1)	CURRENT DEGREE value:  <b>'A'</b> CURRENT DEGREE = ANY <b>'1'</b> CURRENT DEGREE = 1
BIND_SQLRL	CHAR(1)	CURRENT RULES value:  <b>'D'</b> CURRENT RULES = DB2 <b>'S'</b> CURRENT RULES = SQL
BIND_CHOLD	CHAR(1)	Cursor WITH HOLD bind option:  <b>'Y'</b> Initial PREPARE was done for a cursor WITH HOLD <b>'N'</b> Initial PREPARE was not done for cursor WITH HOLD
SCHEMA	VARCHAR(128)	CURRENT SCHEMA value
STAT_EXEC	INTEGER	Number of executions of statement.
STAT_GPAG	INTEGER	Number of get page operations performed for statement.

Table 123. DB2OSC.DSN\_WCC\_CAP\_TMP\_ES description (continued)

Column name	Data type	Description
STAT_SYNR	INTEGER	Number of synchronous buffer reads performed for statement.
STAT_WRIT	INTEGER	Number of buffer write operations performed for statement.
STAT_EROW	INTEGER	Number of rows examined for statement.
STAT_PROW	INTEGER	Number of rows processed for statement.
STAT_SORT	INTEGER	Number of sorts performed for statement.
STAT_INDX	INTEGER	Number of index scans performed for statement.
STAT_RSCN	INTEGER	Number of relation scans performed for statement.
STAT_PGRP	INTEGER	Number of parallel groups created for statement.
STAT_ELAP	FLOAT	Accumulated elapsed time used for statement.
STAT_CPU	FLOAT	Accumulated CPU time used for statement.
STAT_SUS_SYNIO	FLOAT	Accumulated wait time for synchronous I/O.
STAT_SUS_LOCK	FLOAT	Accumulated wait time for lock and latch request.
STAT_SUS_SWIT	FLOAT	Accumulated wait time for synchronous execution unit switch.
STAT_SUS_GLCK	FLOAT	Accumulated wait time for global locks.
STAT_SUS_OTHR	FLOAT	Accumulated wait time for read activity done by another thread.
STAT_SUS_OTHW	FLOAT	Accumulated wait time for write activity done by another thread.
STAT_RIDLMT	FLOAT	Number of times a RID list was not used because the number of RIDs would have exceeded one or more internal DB2 limits.
STAT_RIDSTOR	FLOAT	Number of times a RID list was not used because not enough storage was available to hold the list of RIDs.
STMT_TEXT	CLOB(2M) NOT NULL	Statement text.
STMT_ROWID	ROWID NOT NULL GENERATED ALWAYS	Statement ROWID.
HASHKEY	INTEGER NOT NULL	A hash key generated on statement text and qualifier.

#### DB2OSC.DSN\_WCC\_RP\_TBS:

This table records the tables that workload references and their related statistics information.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 124. DB2OSC.DSN\_WCC\_RP\_TBS description

Column name	Data type	Description
WLID	INTEGER NOT NULL	The workload id references DSN_WCC_WORKLOADS.WLID N

Table 124. DB2OSC.DSN\_WCC\_RP\_TBS description (continued)

Column name	Data type	Description
NAME	VARCHAR(128) NOT NULL	Table name.
CREATOR	VARCHAR(128) NOT NULL	Authorization ID of the owner of the table.
DBNAME	VARCHAR(24) NOT NULL	The database name.
TSNAME	VARCHAR(24) NOT NULL	The table space name.
CARDF	FLOAT NOT NULL	Total number of rows in the table or total number of LOBs in an auxiliary table. The value is -1 if statistics have not been gathered.
NPAGESF	FLOAT(8) NOT NULL	Number of pages used by the table. The value is -1 if statistics have not been gathered.
STATSTIME	TIMESTAMP NOT NULL	If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. The default value is '0001-01-01.00.00.00.000000'. For a created temporary table, the value of STATSTIME is always the default value. This is an updatable column.
REPORT_TS	TIMESTAMP NOT NULL	The time when the report is inserted.

#### DB2OSC.DSN\_WCC\_REPORT\_TBS\_HIS:

This table records the history of tables that are referenced by a workload and their related statistics information.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

The same format as DSN\_WCC\_REPORT\_TABLES

#### DB2OSC.DSN\_WCC\_RP\_IDXES:

This table records the indexes that are referenced in a workload and their related statistics information .

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 125. DB2OSC.DSN\_WCC\_RP\_IDXES description

Column name	Data type	Description
WLID	INTEGER NOT NULL	The workload id references DSN_WCC_WORKLOADS.WLID
NAME	VARCHAR(128) NOT NULL	Index name.
CREATOR	VARCHAR(128) NOT NULL	Authorization ID of the owner of the index.
TBNAME	VARCHAR(128) NOT NULL	Table name.
TBCREATOR	VARCHAR(128) NOT NULL	Authorization ID of the owner of the table.
FIRSTKEYCARDF	FLOAT NOT NULL	Number of distinct values of the first key column. This number is an estimate if updated while collecting statistics on a single partition. The value is -1 if statistics have not been gathered.

Table 125. DB2OSC.DSN\_WCC\_RP\_IDXES description (continued)

Column name	Data type	Description
FULLKEYCARDF	FLOAT NOT NULL	Number of distinct values of the key. The value is -1 if statistics have not been gathered.
CLUSTERRATIOF	FLOAT NOT NULL	When multiplied by 100, the value of the column is the percentage of rows that are in clustering order. For example, a value of .9125 indicates 91.25%.  For a partitioning index, it is the weighted average of all index partitions in terms of the number of rows in the partition. Special values:  <b>0</b> Statistics have not been gathered.  <b>-2</b> The index is for an auxiliary table.
NLEAF	INTEGER NOT NULL	Number of active leaf pages in the index. The value is -1 if statistics have not been gathered.
NLEVELS	SMALLINT NOT NULL	Number of levels in the index tree. If the index is partitioned, the maximum of the number of levels in the index tree for all the partitions. The value is -1 if statistics have not been gathered.
KEYS	VARCHAR(1000) NOT NULL	The columns in the indexes, separated by comma. Only the first 1000 characters are recorded.
STATSTIME	TIMESTAMP NOT NULL	If RUNSTATS updated the statistics, the date and time when the last invocation of RUNSTATS updated the statistics. The default value is '0001-01-01.00.00.00.000000'.
REPORT_TS	TIMESTAMP NOT NULL	The time when the report is inserted.

*DB2OSC.DSN\_WCC\_RP\_IDX\_HIS:*

This table records the history of indexes referenced by a workload and their related statistics information.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

The same format as DSN\_WCC\_RP\_IDXES

*DB2OSC.DSN\_WCC\_RP\_STMTS:*

This table records the tables and indexes referenced by each statement in a workload.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 126. DB2OSC.DSN\_WCC\_RP\_STMTS description

Column name	Data type	Description
WLID	INTEGER NOT NULL	The workload ID; references DSN_WCC_WORKLOADS.WLID
INSTID	INTEGER NOT NULL	The statements instance ID; references DSN_WCC_STMT_INSTS.INSTID

Table 126. DB2OSC.DSN\_WCC\_RP\_STMTS description (continued)

Column name	Data type	Description
TYPE	SMALLINT NOT NULL	The type of object that the relation defines: 1 Table 2 Index
NAME	VARCHAR(128)	Object name
CREATOR	VARCHAR(128)	Object creator

#### Workload statistics advisor tables:

The Optimization Service Center for DB for z/OS workload statistics advisor maintains a set of tables to record workload statistics objects and analysis results.

The tables are maintained in the DB2OSC database. Do not directly manipulate the data that is stored in these tables. Doing so might cause Optimization Service Center and other optimization tools to malfunction. The workload statistics advisor uses the following tables:

#### DB2OSC.DSN\_WSA\_SESSIONS:

The workload statistics advisor sessions table records all the workloads that are analyzed by workload statistics advisor.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 127. DB2OSC.DSN\_WSA\_SESSIONS description

Column name	Data type	Description
SESSIONID	INTEGER GENERATED BY DEFAULT	Unique identifier of the session. Primary key.
WLNAME	VARCHAR(128)	Name of the workload.
BEGINTIME	TIMESTAMP	The time that the workload began to be processed..
ENDTIME	TIMESTAMP	The time that the workload processing ended.
STATUS	CHAR(1)	The status of the workload processing.

#### DB2OSC.DSN\_WSA\_DATABASES:

The workload statistics advisor databases table records all of the databases that are referenced in workloads that are analyzed by workload statistics advisor.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 128. DB2OSC.DSN\_WSA\_DATABASES description

Column name	Data type	Description
SESSIONID	INTEGER	Session ID.
NAME	VARCHAR(128)	Name of the database.

#### *DB2OSC.DSN\_WSA\_TSS:*

The workload statistics advisor table spaces table records the table spaces that are referenced in workloads that are analyzed by workload statistics advisor.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

*Table 129. DB2OSC.DSN\_WSA\_TSS description*

Column name	Data type	Description
SESSIONID	INTEGER	Unique identifier of the session.
DBNAME	VARCHAR(128)	Name of the database.
NAME	VARCHAR(128)	Name of the table space.

#### *DB2OSC.DSN\_WSA\_TABLES:*

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

*Table 130. DB2OSC.DSN\_WSA\_TABLES description*

Column name	Data type	Description
SESSIONID	INTEGER	Session ID.
DBNAME	VARCHAR(128)	Name of the database.
TSNAME	VARCHAR(128)	Name of the table space.
CREATOR	VARCHAR(128)	Creator of the table.
NAME	VARCHAR(128)	Name of the table.
REFCOUNT	INTEGER WITH DEFAULT 0	The reference count of the table.
WEIGHTEDREFCOUNT	DOUBLE WITH DEFAULT 0	The weighted reference count of the table.
CARDINALITY	DOUBLE	The cardinality of the table.
PAGES	DOUBLE	The count of active pages.
STATTIME	TIMESTAMP	The timestamp of the statistics information.
CONFLICTING	CHAR(1) WITH DEFAULT 'N'	Indicates whether the table is conflicting: <b>'Y'</b> Conflicting <b>'N'</b> Not conflicting
PARTITIONS	VARCHAR(512) FOR BIT DATA	The partitions.

#### *DB2OSC.DSN\_WSA\_COLUMNS:*

The workload statistics advisor columns table records all of the table columns that are referenced in workloads that are analyzed by workload statistics advisor.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 131. DB2OSC.DSN\_WSA\_COLUMNS description

Column name	Data type	Description
SESSIONID	INTEGER	Session ID.
TBCREATOR	VARCHAR(128)	Creator of the table.
TBNAME	VARCHAR(128)	Name of the table.
COLNO	INTEGER	Column number.
NAME	VARCHAR(128)	Name of the column.
COLTYPE	CHAR(32)	The data type.
LENGTH	INTEGER	The length of column.
SCALE	INTEGER	The scale.
NULLS	CHAR(1)	Indicates whether the column is nullable.  ‘Y’ Nulls are permitted. ‘N’ Not nullable.
CCSID	INTEGER	The CCSID.

DB2OSC.DSN\_WSA\_COLGROUPS:

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 132. DB2OSC.DSN\_WSA\_COLGROUPS description

Column name	Data type	Description
SESSIONID	INTEGER	Session ID.
TBCREATOR	VARCHAR(128)	Creator of the table.
TBNAME	VARCHAR(128)	Name of the table.
COLGROUPCOLNO	VARCHAR(254) FOR BIT DATA	The column group column list.
NAME	VARCHAR(500)	The name of the column group.
MEDIUMCONFREFCOUNT	INTEGER WITH DEFAULT 0	The medium confidence reference count of the column group.
HIGHCONFREFCOUNT	INTEGER WITH DEFAULT 0	The high confidence reference count of the column group.
HISTCONFREFCOUNT	INTEGER WITH DEFAULT 0	The histogram confidence reference count of the column group.
WMCONFREFCOUNT	REAL WITH DEFAULT 0	The weighted medium confidence reference count of the column group.
WHIGHTEHIGHCONFREFCOUNT	REAL WITH DEFAULT 0	The weighted high confidence reference count of the column group.
WHISTCONFREFCOUNT	REAL WITH DEFAULT 0	The weighted hist confidence reference count of the column group.
CARDINALITY	DOUBLE	The cardinality of the column group.
UNIFORMSTATSTIME	TIMESTAMP	The timestamp of the uniform statistics information.

Table 132. DB2OSC.DSN\_WSA\_COLGROUPS description (continued)

Column name	Data type	Description
FREQSTATTIME	TIMESTAMP	The timestamp of the frequency statistics information.
HISTSTATTIME	TIMESTAMP	The timestamp of the histogram statistics information.
POINTSKEW REASONS	SMALLINT WITH DEFAULT 0	Indicating the reason for the point skew: 1 COLCARD_FAR_LESS_THAN_TABCARD 2 COL_IS_NULL 3 COL_OP_DEFAULT_VALUE 4 COL_OP_BLANKS 5 FREQ_STATS
RANGESKEWREASONS	SMALLINT WITH DEFAULT 0	Indicating the reason for the range skew: 1 LIKE_PREDICATE 2 RANGE_PREDICATE
CORRELATION REASONS	SMALLINT WITH DEFAULT 0	Indicating the reasons for correlation: 1 MULTI_TABLE_JOIN 2 HIGH_COLCARD_LOCAL 3 LOCAL_FILTERING_FOR_JOIN 4 LOCAL_FILTERING_BY_INDEX
CUNIFORMSTATS	CHAR(1) WITH DEFAULT 'N'	Indicates whether the uniform statistics information is conflicting: 'Y' Conflicting 'N' Not conflicting
CFREQSTATS	CHAR(1) WITH DEFAULT 'N'	Indicates whether the frequency statistics information is conflicting: 'Y' Conflicting 'N' Not conflicting
CHISTSTATS	CHAR(1) WITH DEFAULT 'N'	Indicates whether the histogram statistics information is conflicting: 'Y' Conflicting 'N' Not conflicting
UNDERFLOWED	CHAR(1) WITH DEFAULT 'N'	Indicates whether the the related predicate is underflowed: 'Y' Underflowed 'N' Not underflowed

DB2OSC.DSN\_WSA\_CGFREQS:

The workload statistics advisor column group frequency table contains information about the frequencies of column groups.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

*Table 133. DB2OSC.DSN\_WSA\_CGFREQS description*

Column name	Data type	Description
SESSIONID	INTEGER	Session ID.
TBCREATOR	VARCHAR(128)	Creator of the table.
TBNAME	VARCHAR(128)	Name of the table.
COLGROUPCOLNO	VARCHAR(254) FOR BIT DATA	The column group column list.
VALUE	VARCHAR(2000) FOR BIT DATA	The value of the column group.
FREQUENCY	DOUBLE	The frequency of the value.

*DB2OSC.DSN\_WSA\_CGHISTS:*

The workload statistics advisor column group histogram table contains histogram statistics for column groups.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

*Table 134. DB2OSC.DSN\_WAS\_CGHISTS description*

Column name	Data type	Description
SESSIONID	INTEGER	Session ID.
TBCREATOR	VARCHAR(128)	Creator of the table.
TBNAME	VARCHAR(128)	Name of the table.
COLGROUPCOLNO	VARCHAR(254) FOR BIT DATA	The column group column list.
QUANTILENO	INTEGER	The number of the quantile.
LOWVALUE	VARCHAR(2000) FOR BIT DATA	The low value of the quantile.
HIGHVALUE	VARCHAR(2000) FOR BIT DATA	The high value of the quantile.
FREQUENCY	DOUBLE	The frequency of the quantile.
CARDINALITY	DOUBLE	The cardinality of the quantile.
BASECOLGROUP COLNO	DOUBLE FOR BIT DATA	The column group column list regardless of their order.

*DB2OSC.DSN\_WSA\_INDEXES:*

The workload statistics advisor indexes table records information about indexes that are referenced in workloads that are analyzed by workload statistics advisor.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 135. DB2OSC.DSN\_WSA\_INDEXES description

Column name	Data type	Description
SESSIONID	INTEGER	Session ID.
CREATOR	VARCHAR(128)	Creator of the index.
NAME	VARCHAR(128)	Name of the index.
TBCREATOR	VARCHAR(128)	Creator of the table.
TBNAME	VARCHAR(128)	Name of the table.
UNIQUERULE	CHAR(1)	The unique rule of the index.
EXTENSIONTYPE	CHAR(32)	The extension type of the index.
FIRSTKEYCARD	DOUBLE	The first key cardinality of the index.
FULLKEYCARD	DOUBLE	The full key cardinality of the index.
DRF	DOUBLE	The data repetition factor.
STATTIME	TIMESTAMP	The timestamp of the statistics information collected
CONFLICTING	CHAR(1) WITH DEFAULT 'N'	Indicates whether statistics information is conflicting. ' 'Y' Conflicting 'N' Not Conflicting
XMLCOLNAME	CHAR(128)	Name of the corresponding XML column if it is an XML index.

#### DB2OSC.DSN\_WSA\_KEYS:

The workload statistics advisor keys table records the keys of the indexes in DSN\_WSA\_INDEXES.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 136. DB2OSC.DSN\_WSA\_KEYS description

Column name	Data type	Description
SESSIONID	INTEGER	Session ID.
IXCREATOR	VARCHAR(128)	Creator of the index.
IXNAME	VARCHAR(128)	Name of the index.
KEYSEQ	INTEGER	The sequence of the key.
COLNO	INTEGER	The number of the corresponding column.
ORDERING	CHAR(32)	The ordering of the key.

#### DB2OSC.DSN\_WSA\_KEYTARGETS:

The workload statistics advisor key targets table contains information about the targets of keys in DSN\_WSA\_KEYS.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 137. DB2OSC.DSN\_WSA\_KEYTARGETS description

Column name	Data type	Description
SESSIONID	INTEGER	Session ID.
IXCREATOR	VARCHAR(128)	Creator of the index.
IXNAME	VARCHAR(128)	Name of the index.
KEYSEQ	INTEGER	The number of key-target.
DERIVEDFROM	VARCHAR(128)	The derived from expression.
COLNO	INTEGER	The number of the corresponding column.
COLTYPE	CHAR(32)	The data type.
LENGTH	INTEGER	The length of column.
SCALE	INTEGER	The scale.
NULLS	CHAR(1)	Indicates whether the key target is nullable. ‘Y’ Nulls are permitted. ‘N’ Not nullable.
CCSID	INTEGER	The CCSID.

DB2OSC.DSN\_WSA\_KTGS:

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 138. DB2OSC.DSN\_WAS\_KTGS description

Column name	Data type	Description
SESSIONID	INTEGER	Session ID.
IXCREATOR	VARCHAR(128)	Creator of the index.
IXNAME	VARCHAR(128)	Name of the index.
NUMKEYS	INTEGER	The count of the keys.
KEYGROUPKEYNO	VARCHAR(254) FOR BIT DATA	The key-target group key-target list.
CARDINALITY	DOUBLE	The cardinality of the key-target group.
UNIFORMSTATSTIME	TIMESTAMP	The timestamp of the uniform statistics information.
FREQSTATSTIME	TIMESTAMP	The timestamp of the frequency statistics information.
HISTSTATSTIME	TIMESTAMP	The timestamp of the histogram statistics information.
CUNIFORMSTATS	CHAR(1) WITH DEFAULT ‘N’	Indicating if its uniform statistics information is conflicting. ‘Y’ Conflicting ‘N’ Not conflicting
CFREQSTATS	CHAR(1) WITH DEFAULT ‘N’	Indicating if its frequency statistics information is conflicting. ‘Y’ Conflicting ‘N’ Not conflicting

Table 138. DB2OSC.DSN\_WAS\_KTGS description (continued)

Column name	Data type	Description
CHISTSTATS	CHAR(1) WITH DEFAULT 'N'	Indicating if its histogram statistics information is conflicting.  'Y' Conflicting  'N' Not conflicting

*DB2OSC.DSN\_WSA\_KTGFREQS:*

The workload statistics advisor key target frequencies table records frequency statistics for the key target groups in DSN\_WSA\_KTGS.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 139. DB2OSC.DSN\_WSA\_KTGFREQS description

Column name	Data type	Description
SESSIONID	INTEGER	Session ID.
IXCREATOR	VARCHAR(128)	Creator of the index.
IXNAME	VARCHAR(128)	Name of the index.
KEYGROUPKEYNO	VARCHAR(254) FOR BIT DATA	The key-target group key-target list.
VALUE	VARCHAR(2000) FOR BIT DATA	The value of the key-target group.
FREQUENCY	DOUBLE	The frequency of the value.

*DB2OSC.DSN\_WSA\_KTGHISTS:*

The workload statistics advisor key target histogram table records histogram statistics information for the target groups in DSN\_WSA\_KTGHISTS.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 140. DB2OSC.DSN\_WSA\_KTGHISTS description

Column name	Data type	Description
SESSIONID	INTEGER	Session ID.
IXCREATOR	VARCHAR(128)	Creator of the index.
IXNAME	VARCHAR(128)	Name of the index.
KEYGROUPKEYNO	VARCHAR(254) FOR BIT DATA	The key-target group key-target list.
QUANTILENO	INTEGER	The number of the quantile.
LOWVALUE	VARCHAR(2000) FOR BIT DATA	The low value of the quantile.
HIGHVALUE	VARCHAR(2000) FOR BIT DATA	The high value of the quantile.
FREQUENCY	DOUBLE	The frequency of the quantile.

Table 140. DB2OSC.DSN\_WSA\_KTGHISTS description (continued)

Column name	Data type	Description
CARDINALITY	DOUBLE	The cardinality of the quantile.

DB2OSC.DSN\_WSA\_LITERALS:

The workload statistics advisor literals table records all literals that exist in the predicates of queries that are analyzed by workload statistics advisor.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 141. DB2OSC.DSN\_WSA\_LITERALS description

Column name	Data type	Description
SESSIONID	INTEGER	Session ID.
TBCREATOR	VARCHAR(128)	The creator of the table.
TBNAME	VARCHAR(128)	The name of the table.
COLNO	INTEGER	The number of the column.
COLNAME	VARCHAR(500)	The name of the column.
VALUE	VARCHAR(1000)	The text value of the literal.
REFCOUNT	SMALLINT WITH DEFAULT 0	The reference count of this value in the workload.

DB2OSC.DSN\_WSA\_ADVICE:

The workload statistics advisor advice table contains the recommendations issued by workload statistics advisor.

**Restriction:** Do not directly manipulate the data that is stored in this table. Doing so might cause Optimization Service Center and other optimization tools to malfunction

Table 142. DB2OSC.DSN\_WSA\_ADVICE description

Column name	Data type	Description
SESSIONID	INTEGER	Session ID.
DBNAME	VARCHAR(128)	The name of the database.
TSNAME	VARCHAR(128)	The name of the table space.
SEQNO	INTEGER	The sequence number.
TEXT	VARCHAR(1000)	The text value of the advice.

## Using EXPLAIN to capture information about SQL statements.

You can use EXPLAIN to capture detailed information about the access paths that DB2 chooses to process a statement, the cost of processing statements, and which functions DB2 uses.



The information in EXPLAIN tables can help you to:

- Design databases, indexes, and application programs

- Determine when to rebind an application
- Determine the access path that DB2 chooses for a query

For each access to a single table, EXPLAIN tells you if an index access or table space scan is used. If indexes are used, EXPLAIN tells you how many indexes and index columns are used and what I/O methods are used to read the pages. For joins of tables, EXPLAIN tells you which join method and type are used, the order in which DB2 joins the tables, and when and why it sorts any rows.

The primary use of EXPLAIN is to observe the access paths for the SELECT parts of your statements. For UPDATE and DELETE WHERE CURRENT OF, and for INSERT, you receive somewhat less information in your plan table. EXPLAIN does not describe all or every type of access: for example, the access to LOB values, which are stored separately from the base table, and access to parent or dependent tables needed to enforce referential constraints.

The access paths shown for the example queries are intended only to illustrate those examples. If you execute the same queries on your system, DB2 might choose different access paths.



## Creating EXPLAIN tables

Before you can capture EXPLAIN information, you must create the appropriate EXPLAIN tables to hold the type of information that you want to capture.

You can create more than one of these tables on your subsystem or data sharing group, by using the following qualifiers:

### DB2OSC

SQL optimization tools, such as Optimization Service Center for DB2 for z/OS create EXPLAIN tables to collect information about SQL statements and workloads to enable analysis and tuning processes. You can find the SQL statement for creating this table in member DSNTIJOS of the SDSNSAMP library. You can also create this table from the Optimization Service Center interface on your workstation.

### SYSIBM

One table can be created with the qualifier SYSIBM. This table is used by SQL optimization tools. You can find the SQL statement for creating this table is in member DSNTESC of the SDSNSAMP library.

*userID* You can create additional instances of this table with user IDs as the qualifiers. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind or rebind a plan or package with the EXPLAIN(YES) option. These tables are also created and used by SQL optimization tools. You can find the SQL statement for creating an instance of this table qualified by user ID is in member DSNTESC of the SDSNSAMP library.

To create EXPLAIN tables:

Issue the CREATE TABLE statements for each of the tables according to the information that you require.

Option	Description
Access path information for SQL statements, plans, and packages	PLAN_TABLE
Cost estimates for SQL statements	DSN_STATEMNT_TABLE
Statements in the dynamic statement cache	DSN_STATEMENT_CACHE_TABLE
User defined functions in SQL statements	DSN_FUNCTION_TABLE

## Creating a plan table

You can use a plan table to collect information about SQL statements that run on DB2.

### PSPI

**Important:** If mixed data strings are allowed on a DB2 subsystem, EXPLAIN tables must be created with CCSID UNICODE. This includes, but is not limited to, mixed data strings that are used for tokens, SQL statements, application names, program names, correlation names, and collection IDs.

Before you can use EXPLAIN, you must create a plan table to hold the EXPLAIN statement results.

To create a plan table:

Issue the CREATE TABLE statement.

You can create more than one of these tables on your subsystem or data sharing group, by using the following qualifiers:

### DB2OSC

SQL optimization tools, such as Optimization Service Center for DB2 for z/OS create EXPLAIN tables to collect information about SQL statements and workloads to enable analysis and tuning processes. You can find the SQL statement for creating this table in member DSNTIJOS of the SDSNSAMP library. You can also create this table from the Optimization Service Center interface on your workstation.

### SYSIBM

One table can be created with the qualifier SYSIBM. This table is used by SQL optimization tools. You can find the SQL statement for creating this table is in member DSNTESC of the SDSNSAMP library.

*userid* You can create additional instances of this table with user IDs as the qualifiers. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind or rebind a plan or package with the EXPLAIN(YES) option. These tables are also created and used by SQL optimization tools. You can find the SQL statement for creating an instance of this table qualified by user ID is in member DSNTESC of the SDSNSAMP library.

The following figure shows the most current format for a plan table.

```
CREATE TABLE userid.PLAN_TABLE
  (QUERYNO          INTEGER          NOT NULL,
   QBLOCKNO         SMALLINT         NOT NULL,
   APPLNAME          VARCHAR(24)      NOT NULL,
   PROGNAME          VARCHAR(128)     NOT NULL,
```

```

|          PLANNO          SMALLINT      NOT NULL,
|          METHOD           SMALLINT      NOT NULL,
|          CREATOR          VARCHAR(128)  NOT NULL,
|          TNAME            VARCHAR(128)  NOT NULL,
|          TABNO            SMALLINT      NOT NULL,
|          ACESSTYPE        CHAR(2)       NOT NULL,
|          MATCHCOLS        SMALLINT      NOT NULL,
|          ACCESSCREATOR    VARCHAR(128)  NOT NULL,
|          ACCESSNAME        VARCHAR(128)  NOT NULL,
|          INDEXONLY        CHAR(1)       NOT NULL,
|          SORTN_UNIQ        CHAR(1)       NOT NULL,
|          SORTN_JOIN        CHAR(1)       NOT NULL,
|          SORTN_ORDERBY     CHAR(1)       NOT NULL,
|          SORTN_GROUPBY     CHAR(1)       NOT NULL,
|          SORTC_UNIQ        CHAR(1)       NOT NULL,
|          SORTC_JOIN        CHAR(1)       NOT NULL,
|          SORTC_ORDERBY     CHAR(1)       NOT NULL,
|          SORTC_GROUPBY     CHAR(1)       NOT NULL,
|          TSLOCKMODE        CHAR(3)       NOT NULL,
|          TIMESTAMP        CHAR(16)      NOT NULL,
|          REMARKS           VARCHAR(762)  NOT NULL,
|          PREFETCH          CHAR(1)       NOT NULL WITH DEFAULT,
|          COLUMN_FN_EVAL    CHAR(1)       NOT NULL WITH DEFAULT,
|          MIXOPSEQ          SMALLINT      NOT NULL WITH DEFAULT,
|          VERSION           VARCHAR(64)   NOT NULL WITH DEFAULT,
|          COLLID            VARCHAR(128)  NOT NULL WITH DEFAULT,
|          ACCESS_DEGREE     SMALLINT      ,
|          ACCESS_PGROUP_ID  SMALLINT      ,
|          JOIN_DEGREE       SMALLINT      ,
|          JOIN_PGROUP_ID    SMALLINT      ,
|          SORTC_PGROUP_ID   SMALLINT      ,
|          SORTN_PGROUP_ID   SMALLINT      ,
|          PARALLELISM_MODE  CHAR(1)       ,
|          MERGE_JOIN_COLS   SMALLINT      ,
|          CORRELATION_NAME  VARCHAR(128)  ,
|          PAGE_RANGE        CHAR(1)       NOT NULL WITH DEFAULT,
|          JOIN_TYPE         CHAR(1)       NOT NULL WITH DEFAULT,
|          GROUP_MEMBER      VARCHAR(24)   NOT NULL WITH DEFAULT,
|          IBM_SERVICE_DATA  VARCHAR(254)  FOR BIT DATA NOT NULL WITH DEFAULT,
|          WHEN_OPTIMIZE     CHAR(1)       NOT NULL WITH DEFAULT,
|          QBLOCK_TYPE        CHAR(6)       NOT NULL WITH DEFAULT,
|          BIND_TIME          TIMESTAMP     NOT NULL WITH DEFAULT,
|          OPTHINT            VARCHAR(128)  NOT NULL WITH DEFAULT,
|          HINT_USED          VARCHAR(128)  NOT NULL WITH DEFAULT,
|          PRIMARY_ACESSTYPE CHAR(1)       NOT NULL WITH DEFAULT,
|          PARENT_QBLOCKNO    SMALLINT      NOT NULL WITH DEFAULT,
|          TABLE_TYPE        CHAR(1)       ,
|          TABLE_ENCODE     CHAR(1)       NOT NULL WITH DEFAULT,
|          TABLE_SCCSID     SMALLINT      NOT NULL WITH DEFAULT,
|          TABLE_MCCSID     SMALLINT      NOT NULL WITH DEFAULT,
|          TABLE_DCCSID     SMALLINT      NOT NULL WITH DEFAULT,
|          ROUTINE_ID         INTEGER       NOT NULL WITH DEFAULT,
|          CTEREF             SMALLINT      NOT NULL WITH DEFAULT,
|          STMTOKEN           VARCHAR(240)... ,
|          PARENT_PLANNO      SMALLINT      NOT NULL WITH DEFAULT.)
|          IN database-name.table-space-name
|          CCSID UNICODE;

```

Figure 83. 59-column format of PLAN\_TABLE

## Creating a function table

You can use DSN\_FUNCTION\_TABLE to collect information about user-defined functions that are referenced in SQL statements.

Use DSN\_FUNCTION\_TABLE to collect information about user-defined functions that are referenced in SQL statements. You must create this table with your user ID as the qualifier. The table is then populated with information that is produced by the EXPLAIN function.

You do not need to create a function table, statement table, or statement cache table to use EXPLAIN, unless you want to capture the specific types of information that those tables collect.

**Important:** If mixed data strings are allowed on a DB2 subsystem, EXPLAIN tables must be created with CCSID UNICODE. This includes, but is not limited to, mixed data strings that are used for tokens, SQL statements, application names, program names, correlation names, and collection IDs.

Issue the CREATE TABLE statement.

You can create more than one of these tables on your subsystem or data sharing group, by using the following qualifiers:

#### DB2OSC

SQL optimization tools, such as Optimization Service Center for DB2 for z/OS create EXPLAIN tables to collect information about SQL statements and workloads to enable analysis and tuning processes. You can find the SQL statement for creating this table in member DSNTIJOS of the SDSNSAMP library. You can also create this table from the Optimization Service Center interface on your workstation.

#### SYSIBM

One table can be created with the qualifier SYSIBM. This table is used by SQL optimization tools. You can find the SQL statement for creating this table is in member DSNTESC of the SDSNSAMP library.

*userID* You can create additional instances of this table with user IDs as the qualifiers. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind or rebind a plan or package with the EXPLAIN(YES) option. These tables are also created and used by SQL optimization tools. You can find the SQL statement for creating an instance of this table qualified by user ID is in member DSNTESC of the SDSNSAMP library.

The following figure shows the format of a function table.

```
CREATE TABLE DSN_FUNCTION_TABLE
  (QUERYNO          INTEGER          NOT NULL WITH DEFAULT,
   QBLOCKNO         INTEGER          NOT NULL WITH DEFAULT,
   APPLNAME          VARCHAR(24)      NOT NULL WITH DEFAULT,
   PROGNAME          VARCHAR(128)     NOT NULL WITH DEFAULT,
   COLLID            VARCHAR(128)     NOT NULL WITH DEFAULT,
   GROUP_MEMBER      VARCHAR(24)      NOT NULL WITH DEFAULT,
   EXPLAIN_TIME      TIMESTAMP        NOT NULL WITH DEFAULT,
   SCHEMA_NAME       VARCHAR(128)     NOT NULL WITH DEFAULT,
   FUNCTION_NAME      VARCHAR(128)     NOT NULL WITH DEFAULT,
   SPEC_FUNC_NAME     VARCHAR(128)     NOT NULL WITH DEFAULT,
   FUNCTION_TYPE      CHAR(2)         NOT NULL WITH DEFAULT,
   VIEW_CREATOR       VARCHAR(128)     NOT NULL WITH DEFAULT,
   VIEW_NAME          VARCHAR(128)     NOT NULL WITH DEFAULT,
```

```

        PATH                VARCHAR(2048) NOT NULL WITH DEFAULT,
        FUNCTION_TEXT        VARCHAR(1500) NOT NULL WITH DEFAULT)
IN database-name.table-space-name
CCSID UNICODE;

```

Figure 84. Format of DSN\_FUNCTION\_TABLE

#### PSPI

### Creating a statement table

To collect information about the estimated cost of a statement, you create a table called DSN\_STATEMNT\_TABLE to hold the results of EXPLAIN statements.

#### PSPI

Your statement table can use an older format in which the STMT\_ENCODE column does not exist, PROGNAME has a data type of CHAR(8), and COLLID has a data type of CHAR(18). However, use the most current format because it gives you the most information. You can alter a statement table in the older format to a statement table in the current format.

You do not need to create a function table, statement table, or statement cache table to use EXPLAIN, unless you want to capture the specific types of information that those tables collect.

**Important:** If mixed data strings are allowed on a DB2 subsystem, EXPLAIN tables must be created with CCSID UNICODE. This includes, but is not limited to, mixed data strings that are used for tokens, SQL statements, application names, program names, correlation names, and collection IDs.

To create a statement table:

Issue the CREATE table statement.

You can create more than one of these tables on your subsystem or data sharing group, by using the following qualifiers:

#### DB2OSC

SQL optimization tools, such as Optimization Service Center for DB2 for z/OS create EXPLAIN tables to collect information about SQL statements and workloads to enable analysis and tuning processes. You can find the SQL statement for creating this table in member DSNTIJOS of the SDSNSAMP library. You can also create this table from the Optimization Service Center interface on your workstation.

#### SYSIBM

One table can be created with the qualifier SYSIBM. This table is used by SQL optimization tools. You can find the SQL statement for creating this table in member DSNTESC of the SDSNSAMP library.

*userID* You can create additional instances of this table with user IDs as the qualifiers. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind or rebind a plan or package with the EXPLAIN(YES) option. These tables are also created and used by SQL optimization tools. You can find the SQL statement for creating an instance of this table qualified by user ID in member DSNTESC of the SDSNSAMP library.

The following figure shows the current format of a statement table.

```
CREATE TABLE DSN_STATEMNT_TABLE
  (QUERYNO          INTEGER          NOT NULL WITH DEFAULT,
   APPLNAME         VARCHAR(24)      NOT NULL WITH DEFAULT,
   PROGNAME         VARCHAR(128)     NOT NULL WITH DEFAULT,
   COLLID           VARCHAR(128)     NOT NULL WITH DEFAULT,
   GROUP_MEMBER     VARCHAR(24)      NOT NULL WITH DEFAULT,
   EXPLAIN_TIME     TIMESTAMP        NOT NULL WITH DEFAULT,
   STMT_TYPE        CHAR(6)          NOT NULL WITH DEFAULT,
   COST_CATEGORY    CHAR(1)          NOT NULL WITH DEFAULT,
   PROCMS           INTEGER          NOT NULL WITH DEFAULT,
   PROCSU           INTEGER          NOT NULL WITH DEFAULT,
   REASON           VARCHAR(254)     NOT NULL WITH DEFAULT,
   STMT_ENCODE      CHAR(1)          NOT NULL WITH DEFAULT,
   TOTAL_COST       FLOAT            NOT NULL WITH DEFAULT);
IN database-name.table-space-name
CCSID UNICODE;
```

Figure 85. Current format of DSN\_STATEMNT\_TABLE

PSPI

## Creating a statement cache table

You can use a statement cache table to collect information about all of the statements in the cache and to concentrate the performance characteristics of specific statements.

PSPI

Use DSN\_STATEMENT\_CACHE\_TABLE table to collect information about statements in the cache. You must create this table with your user ID as the qualifier. The table is populated with information that is produced by the EXPLAIN STMTCACHE statement.

You do not need to create a function table, statement table, or statement cache table to use EXPLAIN, unless you want to capture the specific types of information that those tables collect.

**Important:** If mixed data strings are allowed on a DB2 subsystem, EXPLAIN tables must be created with CCSID UNICODE. This includes, but is not limited to, mixed data strings that are used for tokens, SQL statements, application names, program names, correlation names, and collection IDs.

To create a statement cache table:

Issue the CREATE TABLE statement.

You can create more than one of these tables on your subsystem or data sharing group, by using the following qualifiers:

### DB2OSC

SQL optimization tools, such as Optimization Service Center for DB2 for z/OS create EXPLAIN tables to collect information about SQL statements and workloads to enable analysis and tuning processes. You can find the SQL statement for creating this table in member DSNTIJOS of the

SDSNSAMP library. You can also create this table from the Optimization Service Center interface on your workstation.

## SYSIBM

One table can be created with the qualifier SYSIBM. This table is used by SQL optimization tools. You can find the SQL statement for creating this table is in member DSNDESC of the SDSNSAMP library.

*userID* You can create additional instances of this table with user IDs as the qualifiers. These tables are populated with statement cost information when you issue the EXPLAIN statement or bind or rebind a plan or package with the EXPLAIN(YES) option. These tables are also created and used by SQL optimization tools. You can find the SQL statement for creating an instance of this table qualified by user ID is in member DSNDESC of the SDSNSAMP library.

The following figure shows the format of a statement cache table.

```
CREATE TABLE DSN_STATEMENT_CACHE_TABLE
(STMT_ID          INTEGER          NOT NULL,
 STMT_TOKEN       VARCHAR(240)      ,
 COLLID           VARCHAR(128)     NOT NULL,
 PROGRAM_NAME     VARCHAR(128)     NOT NULL,
 INV_DROPALT      CHAR(1)          NOT NULL,
 INV_REVOKE       CHAR(1)          NOT NULL,
 INV_LRU          CHAR(1)          NOT NULL,
 INV_RUNSTATS     CHAR(1)          NOT NULL,
 CACHED_TS        TIMESTAMP        NOT NULL,
 USERS            INTEGER          NOT NULL,
 COPIES           INTEGER          NOT NULL,
 LINES            INTEGER          NOT NULL,
 PRIMAUTH         VARCHAR(128)     NOT NULL,
 CURSQLID         VARCHAR(128)     NOT NULL,
 BIND_QUALIFIER   VARCHAR(128)     NOT NULL,
 BIND_ISO         CHAR(2)          NOT NULL,
 BIND_CDATA       CHAR(1)          NOT NULL,
 BIND_DYNRL       CHAR(1)          NOT NULL,
 BIND_DEGRE       CHAR(1)          NOT NULL,
 BIND_SQLRL       CHAR(1)          NOT NULL,
 BIND_CHOLD       CHAR(1)          NOT NULL,
 STAT_TS          TIMESTAMP        NOT NULL,
 STAT_EXEC        INTEGER          NOT NULL,
 STAT_GPAG        INTEGER          NOT NULL,
 STAT_SYNR        INTEGER          NOT NULL,
 STAT_WRIT        INTEGER          NOT NULL,
 STAT_EROW        INTEGER          NOT NULL,
 STAT_PROW        INTEGER          NOT NULL,
 STAT_SORT        INTEGER          NOT NULL,
 STAT_INDX        INTEGER          NOT NULL,
 STAT_RSCN        INTEGER          NOT NULL,
 STAT_PGRP        INTEGER          NOT NULL,
 STAT_ELAP        FLOAT            NOT NULL,
 STAT_CPU         FLOAT            NOT NULL,
 STAT_SUS_SYNIO   FLOAT            NOT NULL,
 STAT_SUS_LOCK    FLOAT            NOT NULL,
 STAT_SUS_SWIT    FLOAT            NOT NULL,
 STAT_SUS_GLCK    FLOAT            NOT NULL,
 STAT_SUS_OTHR    FLOAT            NOT NULL,
 STAT_SUS_OTHW    FLOAT            NOT NULL,
 STAT_RIDLIMT     INTEGER          NOT NULL,
 STAT_RIDSTOR     INTEGER          NOT NULL,
 EXPLAIN_TS       TIMESTAMP        NOT NULL,
 SCHEMA           VARCHAR(128)     NOT NULL,
 STMT_TEXT        CLOB(2M)         NOT NULL,
 STMT_ROWID       ROWID            NOT NULL GENERATED ALWAYS
 BIND_RA_TOT      INTEGER          NOT NULL WITH DEFAULT,
```

```

|          BIND_RO_TYPE      CHAR(1)      NOT NULL WITH DEFAULT
|          )
|      IN database-name.table-space-name
|      CCSID UNICODE;

CREATE TYPE 2 INDEX DSN_STATEMENT_CACHE_IDX1
ON DSN_STATEMENT_CACHE_TABLE (STMT_ID ASC);

CREATE TYPE 2 INDEX DSN_STATEMENT_CACHE_IDX2
ON DSN_STATEMENT_CACHE_TABLE (STMT_TOKEN ASC)
CLUSTER;

CREATE TYPE 2 INDEX DSN_STATEMENT_CACHE_IDX3
ON DSN_STATEMENT_CACHE_TABLE (EXPLAIN_TS DESC) ;

CREATE AUX TABLE DSN_STATEMENT_CACHE_AUX IN
DSNSTMTC.DSNLOBTS
STORES DSN_STATEMENT_CACHE_TABLE COLUMN STMT_TEXT;

CREATE TYPE 2 INDEX DSN_STATEMENT_CACHE_AUXINX
ON DSN_STATEMENT_CACHE_AUX;

```

Figure 86. Format of DSN\_STATEMENT\_CACHE\_TABLE



## Updating the format of an existing PLAN\_TABLE

Your existing instances of PLAN\_TABLE, especially those created in earlier version of DB2 can use many other formats with fewer columns.

However, you should use the 59-column format because it gives you the most information.

To update existing instances of PLAN\_TABLE:

Issue ALTER TABLE statements.

1. Check whether the existing plan table has the following columns:
  - PROGNAME
  - CREATOR
  - TNAME
  - ACESSTYPE
  - ACCESSNAME
  - REMARKS
  - COLLID
  - CORRELATION\_NAME
  - IBM\_SERVICE\_DATA
  - OPTHINT
  - HINT\_USED
2. Change the data types of the existing columns above to match the columns shown in the following example.
3. Add any columns that were missing to the table and use the column definitions shown in the following example. For most of the columns that you add, specify NOT NULL WITH DEFAULT so that default values are included for the rows in the table. However, as the example shows, certain columns do allow nulls. Do not specify those columns as NOT NULL WITH DEFAULT.

QUERYNO	INTEGER	NOT NULL
QBLOCKNO	SMALLINT	NOT NULL
APPLNAME	CHAR(8)	NOT NULL
PROGNAME	CHAR(8)	NOT NULL
PLANNO	SMALLINT	NOT NULL
METHOD	SMALLINT	NOT NULL
CREATOR	CHAR(8)	NOT NULL
TNAME	CHAR(18)	NOT NULL
TABNO	SMALLINT	NOT NULL
ACCESSTYPE	CHAR(2)	NOT NULL
MATCHCOLS	SMALLINT	NOT NULL
ACCESSCREATOR	CHAR(8)	NOT NULL
ACCESSNAME	CHAR(18)	NOT NULL
INDEXONLY	CHAR(1)	NOT NULL
SORTN_UNIQ	CHAR(1)	NOT NULL
SORTN_JOIN	CHAR(1)	NOT NULL
SORTN_ORDERBY	CHAR(1)	NOT NULL
SORTN_GROUPBY	CHAR(1)	NOT NULL
SORTC_UNIQ	CHAR(1)	NOT NULL
SORTC_JOIN	CHAR(1)	NOT NULL
SORTC_ORDERBY	CHAR(1)	NOT NULL
SORTC_GROUPBY	CHAR(1)	NOT NULL
TSLOCKMODE	CHAR(3)	NOT NULL
TIMESTAMP	CHAR(16)	NOT NULL
REMARKS	VARCHAR(254)	NOT NULL
-----25 column format-----		
PREFETCH	CHAR(1)	NOT NULL WITH DEFAULT
COLUMN_FN_EVAL	CHAR(1)	NOT NULL WITH DEFAULT
MIXOPSEQ	SMALLINT	NOT NULL WITH DEFAULT
-----28 column format-----		
VERSION	VARCHAR(64)	NOT NULL WITH DEFAULT
COLLID	CHAR(18)	NOT NULL WITH DEFAULT
-----30 column format-----		
ACCESS_DEGREE	SMALLINT	
ACCESS_PGROUP_ID	SMALLINT	
JOIN_DEGREE	SMALLINT	
JOIN_PGROUP_ID	SMALLINT	
-----34 column format-----		
SORTC_PGROUP_ID	SMALLINT	
SORTN_PGROUP_ID	SMALLINT	
PARALLELISM_MODE	CHAR(1)	
MERGE_JOIN_COLS	SMALLINT	
CORRELATION_NAME	CHAR(18)	
PAGE_RANGE	CHAR(1)	NOT NULL WITH DEFAULT
JOIN_TYPE	CHAR(1)	NOT NULL WITH DEFAULT
GROUP_MEMBER	CHAR(8)	NOT NULL WITH DEFAULT
IBM_SERVICE_DATA	VARCHAR(254)	NOT NULL WITH DEFAULT
-----43 column format-----		
WHEN_OPTIMIZE	CHAR(1)	NOT NULL WITH DEFAULT
QBLOCK_TYPE	CHAR(6)	NOT NULL WITH DEFAULT
BIND_TIME	TIMESTAMP	NOT NULL WITH DEFAULT
-----46 column format-----		
OPTHINT	CHAR(8)	NOT NULL WITH DEFAULT
HINT_USED	CHAR(8)	NOT NULL WITH DEFAULT
PRIMARY_ACCESSTYPE	CHAR(1)	NOT NULL WITH DEFAULT
-----49 column format-----		
PARENT_QBLOCKNO	SMALLINT	NOT NULL WITH DEFAULT
TABLE_TYPE	CHAR(1)	
-----51 column format-----		

Figure 87. Formats of PLAN\_TABLE prior to the 59-column format

## Capturing EXPLAIN information

You can use DB2 EXPLAIN to populate explain tables with information about the access paths that DB2 uses to process your SQL queries.

### The DB2 EXPLAIN stored procedure

The DB2 EXPLAIN stored procedure, `DSNAEXP`, is a sample stored procedure that lets you execute an EXPLAIN statement on an SQL statement without having the authorization to execute that SQL statement.

PSPI

### Environment

`DSNAEXP` must run in a WLM-established stored procedure address space.

Before you can invoke `DSNAEXP`, table `sqlid.PLAN_TABLE` must exist. *sqlid* is the value that you specify for the *sqlid* input parameter when you call `DSNAEXP`.

Job `DSNTESC` in `DSN8910.SDSNSAMP` contains a sample CREATE TABLE statement for the `PLAN_TABLE`.

### Authorization required

To execute the `CALL DSN8.DSNAEXP` statement, the owner of the package or plan that contains the CALL statement must have one or more of the following privileges on each package that the stored procedure uses:

- The EXECUTE privilege on the package for `DSNAEXP`
- Ownership of the package
- PACKADM authority for the package collection
- SYSADM authority

In addition:

- The SQL authorization ID of the process in which `DSNAEXP` is called must have the authority to execute `SET CURRENT SQLID=sqlid`.
- The SQL authorization ID of the process must also have one of the following characteristics:
  - Be the owner of a plan table named `PLAN_TABLE`
  - Have an alias on a plan table named *owner*.`PLAN_TABLE` and have SELECT and INSERT privileges on the table

### DSNAEXP syntax diagram

The following syntax diagram shows the CALL statement for invoking `DSNAEXP`. Because the linkage convention for `DSNAEXP` is GENERAL, you cannot pass null values to the stored procedure.

```
►►—CALL—DSNAEXP—(—sqlid—,—queryno—,—sql-statement—,—parse—,—  
►—qualifier—,—sqlcode—,—sqlstate—,—error-message—)—►►
```

## DSNAEXP option descriptions

### *sqlid*

Specifies:

- The authorization ID under which the EXPLAIN statement is to be executed
- The qualifier for the table into which EXPLAIN output is written:  
*sqlid*.PLAN\_TABLE
- The implicit qualifier for unqualified objects in the SQL statement on which EXPLAIN is executed, if the value of *parse* is 'N'

*sqlid* is an input parameter of type CHAR(8).

### *queryno*

Specifies a number that is to be used to identify *sql-statement* in the EXPLAIN output. This number appears in the QUERYNO column in the PLAN\_TABLE.  
*queryno* is an input parameter of type INTEGER.

### *sql-statement*

Specifies the SQL statement on which EXPLAIN is to be executed. *sql-statement* is an input parameter of type CLOB(2M).

### *parse*

Specifies whether DSNAEXP adds a qualifier for unqualified table or view names in the input SQL statement. Valid values are 'Y' and 'N'. If the value of *parse* is 'Y', *qualifier* must contain a valid SQL qualifier name.

If *sql-statement* is insert-within-select and common table expressions, you need to disable the parsing functionality, and add the qualifier manually.

*parse* is an input parameter of type CHAR(1).

### *qualifier*

Specifies the qualifier that DSNAEXP adds to unqualified table or view names in the input SQL statement. If the value of *parse* is 'N', *qualifier* is ignored.

If the statement on which EXPLAIN is run contains an INSERT within a SELECT or a common table expression, *parse* must be 'N', and table and view qualifiers must be explicitly specified.

*qualifier* is an input parameter of type CHAR(8).

### *sqlcode*

Contains the SQLCODE from execution of the EXPLAIN statement. *sqlcode* is an output parameter of type INTEGER.

### *sqlstate*

Contains the SQLSTATE from execution of the EXPLAIN statement. *sqlstate* is an output parameter of type CHAR(5).

### *error-message*

Contains information about DSNAEXP execution. If the SQLCODE from execution of the EXPLAIN statement is not 0, *error-message* contains the error message for the SQLCODE. *error-message* is an output parameter of type VARCHAR(960).

## Example of DSNAEXP invocation

The following C example shows how to call DSNAEXP to execute an EXPLAIN statement.

```
EXEC SQL BEGIN DECLARE SECTION;
char      hvsqlid[9];          /* Qualifier for PLAN_TABLE */
long int  hvqueryno;          /* QUERYNO to give the SQL */
```

```

struct {
    short int hvsql_stmt_len;          /* Input SQL statement length */
    hvsql_stmt_txt SQL type is CLOB 2M;
    /* SQL statement text */
} hvsql_stmt;                        /* SQL statement to EXPLAIN */
char hvparse[1];                     /* Qualify object names */
/* indicator */
char hvqualifier[9];                 /* Qualifier for unqualified */
/* objects */
long int hvsqlcode;                  /* SQLCODE from CALL DSNAEXP */
char hvsqlstate[6];                  /* SQLSTATE from CALL DSNAEXP */
struct {
    short int hvmsg_len;              /* Error message length */
    char hvmsg_text[961];             /* Error message text */
} hvmsg;                             /* Error message from failed */
/* CALL DSNAEXP */

EXEC SQL END DECLARE SECTION;
short int i;
:
/* Set the input parameters */
strcpy(hvsqlid,"ADMF001 ");
hvqueryno=320;
strcpy(hvsql_stmt.hvsql_stmt_text,"SELECT CURRENT DATE FROM SYSDDUMMY1");
hvsql_stmt.hvsql_stmt_len=strlen(hvsql_stmt.hvsql_stmt_text);
hvparse[0]='Y';
strcpy(hvqualifier,"SYSIBM");
/* Initialize the output parameters */
hvsqlcode=0;
for (i = 0; i < 5; i++) hvsqlstate[i] = '0';
hvsqlstate[5]='\0';
hvmsg.hvmsg_len=0;
for (i = 0; i < 960; i++) hvmsg.hvmsg_text[i] = ' ';
hvmsg.hvmsg_text[960] = '\0';
/* Call DSNAEXP to do EXPLAIN and put output in ADMF001.PLAN_TABLE */
EXEC SQL
    CALL DSN8.DSNAEXP(:hvsqlid,
                      :hvqueryno,
                      :hvsql_stmt,
                      :hvparse,
                      :hvqualifier,
                      :hvsqlcode,
                      :hvsqlstate,
                      :hvmsg);

```

## DSNAEXP output

If DSNAEXP executes successfully, *sqlid*.PLAN\_TABLE contains the EXPLAIN output. A user with SELECT authority on *sqlid*.PLAN\_TABLE can obtain the results of the EXPLAIN that was executed by DSNAEXP by executing this query:

```
SELECT * FROM sqlid.PLAN_TABLE WHERE QUERYNO='queryno';
```

If DSNAEXP does not execute successfully, *sqlcode*, *sqlstate*, and *error-message* contain error information.

PSPI

## Executing the SQL statement EXPLAIN


You can populate a PLAN\_TABLE by executing the SQL statement EXPLAIN.

PSPI

If an alias is defined on a PLAN\_TABLE that was created with a different authorization ID, and you have the appropriate SELECT and INSERT privileges, you can populate that PLAN\_TABLE even if you do not own it.

To execute EXPLAIN for a SQL statement:

Issue the EXPLAIN statements and specify a single explainable SQL statement in the FOR clause. You can issue the EXPLAIN statement:

- Statically from an application program
- Dynamically by using, DB2 QMF, SPUFI, or the command line processor 

### Capturing EXPLAIN information at bind time


You can populate a plan table when you bind or rebind a plan or package.

EXPLAIN as a bind option should not be a performance concern. The same processing for access path selection is performed, regardless of whether you use EXPLAIN(YES) or EXPLAIN (NO). With EXPLAIN(YES), only a small amount of overhead processing is required to put the results in a plan table.

If a plan or package that was previously bound with EXPLAIN(YES) is automatically rebound, the value of the EXPLAIN PROCESSING field on installation panel DSNTIPO determines whether EXPLAIN is run again during the automatic rebind. Again, inserting the results into a plan table requires only a small amount of overhead.


 To capture EXPLAIN information when you bind a plan or package:


Specify the EXPLAIN(YES) option when you bind the plan or package. The information appears in table *package\_owner*.PLAN\_TABLE or *plan\_owner*.PLAN\_TABLE. For dynamically prepared SQL, the qualifier of PLAN\_TABLE is the current SQLID.

If the plan owner or the package owner has an alias on a PLAN\_TABLE that was created by another owner, *other\_owner*.PLAN\_TABLE is populated instead of *package\_owner*.PLAN\_TABLE or *plan\_owner*.PLAN\_TABLE. 

### Capturing EXPLAIN information for remote binds:

You can capture EXPLAIN information from a remotely bound package.

 A remote requester that accesses DB2 can specify EXPLAIN(YES) when binding a package at the DB2 server. You cannot get information about access paths for SQL statements that use private protocol.

- Specify EXPLAIN(YES) from the remote requester when binding a package at the DB2 server. The information appears in a plan table at the server, not at the requester.
- If the requester does not support the propagation of the EXPLAIN(YES) option, rebind the package at the requester and specify EXPLAIN(YES). 

### Capturing EXPLAIN information under DB2 QMF

You can use DB2 QMF to display the results of EXPLAIN to the terminal.

**PSPI** You can create your own form to display the output or use the default form for DB2 QMF. **PSPI**

**Parameter markers in place of host variables:**

If you have host variables in a predicate for an original query in a static application and if you are using DB2 QMF or SPUFI to execute EXPLAIN for the query, you should consider using parameter markers where you use host variables in the original query.

**PSPI** If you use a constant value instead, you might see different access paths for your static and dynamic queries. For instance, compare the queries in the following table:

Table 143. Three example queries for the use of parameter markers

Original Static SQL	DB2 QMF Query Using Parameter Marker	DB2 QMF Query Using Literal
DECLARE C1 CURSOR FOR SELECT * FROM T1	EXPLAIN PLAN SET QUERYNO=1 FOR SELECT * FROM T1 WHERE C1 > ?	EXPLAIN PLAN SET QUERYNO=1 FOR SELECT * FROM T1 WHERE C1 > 10

Using the constant '10' would likely produce a different filter factor and might produce a different access path from the original static SQL statement. (A filter factor is the proportion of rows that remain after a predicate has "filtered out" the rows that do not satisfy the predicate. The parameter marker behaves just like a host variable, in that the predicate is assigned a default filter factor. **PSPI**

**When to use a constant:**

| If you know that a static plan or package was bound with REOPT(ALWAYS) or  
| REOPT(AUTO) and you have some idea of what is returned in the host variable,  
| including the constant in the DB2 QMF EXPLAIN results can be more accurate.

| **PSPI** REOPT(ALWAYS) means that DB2 replaces the value of the host variable  
| with the true value at run time and then determine the access path. REOPT(AUTO)  
| means that DB2 might replace the value of the host variable, depending on how it  
| changed since the last execution.. . **PSPI**

**Static and dynamic queries:**

Even when using parameter markers, you could see different access paths for static and dynamic queries.

**PSPI** DB2 assumes that the value that replaces a parameter marker has the same length and precision as the column it is compared to. That assumption determines whether the predicate is stage 1 indexable or stage 2, which is always non-indexable.

If the column definition and the host variable definition are both strings, the predicate becomes stage 1 but not indexable when any of the following conditions are true:

- The column definition is CHAR or VARCHAR, and the host variable definition is GRAPHIC or VARGRAPHIC.
- The column definition is GRAPHIC or VARGRAPHIC, the host variable definition is CHAR or VARCHAR, and the length of the column definition is less than the length of the host variable definition.
- Both the column definition and the host variable definition are CHAR or VARCHAR, the length of the column definition is less than the length of the host variable definition, and the comparison operator is any operator other than "=".
- Both the column definition and the host variable definition are GRAPHIC or VARGRAPHIC, the length of the column definition is less than the length of the host variable definition, and the comparison operator is any operator other than "=".

The predicate becomes stage 2 when any of the following conditions are true:

- The column definition is DECIMAL(p,s), where p>15, and the host variable definition is REAL or FLOAT.
- The column definition is CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC, and the host variable definition is DATE, TIME, or TIMESTAMP.



## Working with EXPLAIN table data

Although DB2 automatically populates EXPLAIN tables, you might need to modify the data in your EXPLAIN tables.

### Populating and maintaining a plan table

When you populate the plan table through EXPLAIN, any INSERT triggers on the table are not activated. If you insert rows yourself, then those triggers are activated.

#### Deleting EXPLAIN table rows:


DB2 adds rows to EXPLAIN tables as you choose; it does not automatically delete rows.



To clear the table of obsolete rows:

Issue one of the following SQL statements, just as you would for deleting rows from any table:

- DELETE
- TRUNCATE

You can also use DROP TABLE to drop a plan table completely. 

#### Reordering EXPLAIN table rows:

Several processes can insert rows into the same plan table. To understand access paths, you must retrieve the rows for a particular query in an appropriate order.

#### Retrieving EXPLAIN table rows for a plan:

You can find the EXPLAIN table rows for every explainable statements of a particular plan, in their logical order.

**PSPI** The rows for a particular plan are identified by the value of the APPLNAME column.

To retrieve all the rows for all the explainable statements in a plan in their logical order:

Issue the following SQL statement

```
SELECT * FROM userID.PLAN_TABLE
WHERE APPLNAME = 'APPL1'
ORDER BY TIMESTAMP, QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
```

The result of the ORDER BY clause shows whether any of the following conditions exist:

- Multiple QBLOCKNO values within a QUERYNO
- Multiple PLANNO values within a QBLOCKNO
- Multiple MIXOPSEQ values within a PLANNO

**PSPI**

All rows with the same non-zero value for QBLOCKNO and the same value for QUERYNO relate to a step within the query. QBLOCKNO values are not necessarily executed in the order shown in PLAN\_TABLE. But within a QBLOCKNO, the PLANNO column gives the sub-steps in the order they execute.

For each sub-step, the TNAME column identifies the table accessed. Sorts can be shown as part of a table access or as a separate step.

For entries that contain QUERYNO=0, use the timestamp, which is guaranteed to be unique, to distinguish individual statements.

### Retrieving EXPLAIN table rows for a package:

You can retrieve the rows for every explainable statement in a package, in their logical order.

**PSPI** The rows for a particular package are identified by the values of PROGNAME, COLLID, and VERSION. Those columns correspond to the following four-part naming convention for packages:

*location.collection.packageID.version*

COLLID gives the COLLECTION name, and PROGNAME gives the PACKAGE\_ID.

To find the rows for all the explainable statements in a package in their logical order:

Issue the following SQL statement:

```
SELECT * FROM userID.PLAN_TABLE
WHERE PROGNAME = 'PACK1' AND COLLID = 'COLL1' AND VERSION = 'PROD1'
ORDER BY QUERYNO, QBLOCKNO, PLANNO, MIXOPSEQ;
```

**PSPI**

### Populating and maintaining a statement table:

You populate a statement table at the same time as you populate the corresponding plan table.

**PSPI** Just as with the plan table, DB2 just adds rows to the statement table; it does not automatically delete rows. INSERT triggers are not activated unless you insert rows yourself using and SQL INSERT statement.

To clear the table of obsolete rows, use DELETE or TRUNCATE, just as you would for deleting rows from any table. You can also use DROP TABLE to drop a statement table completely. **PSPI**

### Retrieving rows from a statement table:

You can use rows in the statement table to determine the cost of executing an SQL statement.

**PSPI** To retrieve all rows in a statement table, you can use a query like the following statement, which retrieves all rows about the statement that is represented by query number 13:

```
SELECT * FROM JOE.DSN_STATEMNT_TABLE
WHERE QUERYNO = 13;
```

The QUERYNO, APPLNAME, PROGNAME, COLLID, and EXPLAIN\_TIME columns contain the same values as corresponding columns of PLAN\_TABLE for a given plan. You can use these columns to join the plan table and statement table:

```
SELECT A.*, PROCMS, COST_CATEGORY
FROM JOE.PLAN_TABLE A, JOE.DSN_STATEMNT_TABLE B
WHERE A.APPLNAME = 'APPL1' AND
A.APPLNAME = B.APPLNAME AND
A.QUERYNO = B.QUERYNO AND A.PROGNAME = B.PROGNAME AND
A.COLLID = B.COLLID AND
A.BIND_TIME = B.EXPLAIN_TIME
ORDER BY A.QUERYNO, A.QBLOCKNO, A.PLANNO, A.MIXOPSEQ;
```

**PSPI**


---

## Monitoring and optimizing queries with profile tables

If you are an experienced system programmer or database administrator, you might choose to do some optimization work manually by creating and manipulating profile tables.

**PSPI** The easiest way to monitor and tune your queries is to use the DB2 for z/OS Optimization Service Center or DB2 Optimization Expert for z/OS. Both of these products are IBM workstation tools that analyze queries and workloads and provide recommendations to improve performance.

**Restriction:** Previously bound static SQL statements must be rebound in Version 9.1 new-function mode before you can use monitor profiles to monitor them. This restriction does not apply to dynamic SQL statements.

These topics explain how to create and use these profile tables to monitor and optimize your queries. 

## Profiles

A *profile* is a set of criteria that identifies a particular query or set of queries.

 You can to create a profile to perform the following tasks:

Profiles are stored in the table SYSIBM.DSN\_PROFILE\_TABLE. For more information about this table, see “SYSIBM.DSN\_PROFILE\_TABLE” on page 579.

**Example profiles:** The following table shows sample data in SYSIBM.DSN\_PROFILE\_TABLE. Each of the five rows defines a different profile.

| Table 144. Sample data in SYSIBM.DSN\_PROFILE\_TABLE

AUTHID	PLANNAME	COLLID	PKGNAME	IPADDR	PROFILEID
null	PLANA	COLL1	PKG1	null	12
Tom	null	null	null	9.30.36.181	13

These rows define the following profiles:

- The first row defines a profile that includes all statements in plan PLANA, in collection COLL1, and in package PKG1.
- The second row defines a profile that includes all statements that are run by authorization ID Tom from the IP address 9.30.36.181.

For instructions on how to create a profile, see “Creating a profile” on page 588.


After you have created a profile, you can monitor the statements that are defined by the profile and define subsystem parameter values for those statements. Use the profile attributes table, SYSIBM.DSN\_PROFILE\_ATTRIBUTES, to specify these actions. For example, for the sample data in Table 144, you might have a profile attribute table that looks similar to the one in Table 145.

| Table 145. Sample data in SYSIBM.DSN\_PROFILE\_ATTRIBUTES

PROFILEID	KEYWORDS	ATTRIBUTE1	ATTRIBUTE2	ATTRIBUTE3	ATTRIBUTE_TIMESTAMP	REMARKS
12	MONITOR CPUTIME		3	2.5	2005-06-23...	
12	MONITOR SPIKE	PERCENT	3	200.0	2005-06-23...	
12	MAX EXC PUSHOUT	TOTAL	150		2005-06-23...	
13	MONITOR RLF		3		2005-06-23...	

For more information about this table and what the values mean, see “SYSIBM.DSN\_PROFILE\_ATTRIBUTES” on page 580.

For information on the rows that you need to insert to monitor statements, see “Monitoring statements by using profile tables” on page 592.

For information on the rows that you need to insert to set subsystem parameter values, see “Modifying subsystem parameter values by using profile tables” on page 599. 

## Profile tables

*Profile tables* contain information about how DB2 executes or monitors a group of statements.



You can use profile tables to perform the following tasks:

**Recommendation:** Create and manipulate profile tables only if you are an experienced user.

You can set up the following profile tables on your DB2 subsystem:

### **SYSIBM.DSN\_PROFILE\_TABLE**

Used to specify which SQL statements are monitored or tuned by each monitor profile. Each row defines a particular monitor profile.

### **SYSIBM.DSN\_PROFILE\_HISTORY**

Contains all of the profiles that were in effect at a specific point in time. Profiles are defined in the profile table, SYSIBM.DSN\_PROFILE\_TABLE. Profiles in the profile history table might no longer be in the profile table. Use the profile history table when diagnosing optimization problems. IBM Software Support might request this table if you contact them to help you with a problem.

### **SYSIBM.DSN\_PROFILE\_ATTRIBUTES**

Used to specify the monitoring and tuning functions that particular monitor profile performs.

### **SYSIBM.DSN\_PROFILE\_ATTRIBUTES\_HISTORY**

Contains the attributes that were in effect at a specific point in time. These attributes are defined in the profile attributes table, SYSIBM.DSN\_PROFILE\_ATTRIBUTES. Use the profile attributes history table when diagnosing optimization problems. IBM Software Support might request this table if you contact them to help you with a problem.

### **DSN\_STATEMENT\_RUNTIME\_INFO**

Contains runtime information for statements. When you monitor statements, you can specify that DB2 records information in this table.

### **DSN\_OBJECT\_RUNTIME\_INFO**

Contains runtime information for statements. When you monitor statements, you can specify that DB2 records information in this table.

The following EXPLAIN tables can be used to perform profile table functions:

### **PLAN\_TABLE**

Contains information about access paths. Your subsystem or data sharing group can have more than one of these tables. One table can be created with the qualifier SYSIBM. Other tables can be created with user IDs as the qualifiers.

### **DSN\_STATEMNT\_TABLE**

Contains information about the estimated cost of a statement. Your subsystem or data sharing group can have more than one of these tables. One table can be created with the qualifier SYSIBM. Other tables can be created with user IDs as the qualifiers.

## DSN\_FUNCTION\_TABLE

Contains descriptions of functions that are used in specified SQL statements. Your subsystem or data sharing group can have more than one of these tables. One table can be created with the qualifier SYSIBM. Other tables can be created with user IDs as the qualifiers.

### PSPI

#### Related concepts

“Investigating SQL performance with EXPLAIN” on page 421

## Profile tables that are used by optimization tools

Optimization tools, such as IBM Optimization Service Center for DB2 for z/OS and IBM DB2 Optimization Expert for z/OS can create and use profile tables to enable monitoring of query performance.

## SYSIBM.DSN\_VIRTUAL\_INDEXES:

The virtual indexes table, DSN\_VIRTUAL\_INDEXES, enables optimization tools to test the effect of creating and dropping indexes on the performance of particular queries.

### PSPI

The following table describes the columns in DSN\_VIRTUAL\_INDEXES.

**Recommendation:** Do not manually insert data into or delete data from this table, it is intended to be used only by optimization tools.

Table 146. SYSIBM.DSN\_VIRTUAL\_INDEXES description

Column name	Data type	Description
TBCREATOR	VARCHAR(128)	The schema or authorization ID of the owner of the table on which the index is being created or dropped.
TBNAME	VARCHAR(128)	The name of the table on which the index is being created or dropped.
IXCREATOR	VARCHAR(128)	The schema or authorization ID of the owner of the index.
IXNAME	VARCHAR(128)	The index name.
ENABLE	CHAR(1)	Indicates whether this index should be considered in the scenario that is being tested. This column can have one of the following values:  <b>Y</b> Use this index.  <b>N</b> Do not use this index.  If this column contains 'Y', but the index definition is not valid, the index is ignored.
MODE	CHAR(1)	Indicates whether the index is being created or dropped. This column can have one of the following values:  <b>C</b> This index is to be created.  <b>D</b> This index is to be dropped.

Table 146. SYSIBM.DSN\_VIRTUAL\_INDEXES description (continued)

Column name	Data type	Description
UNIQUERULE	CHAR(1)	Indicates whether the index is unique. This column can have one of the following values:  <b>D</b> The index is not unique. (Duplicates are allowed.) <b>U</b> This index is unique.
COLCOUNT	SMALLINT	The number of columns in the key.
CLUSTERING	CHAR(1)	Indicates whether the index is clustered. This column can have one of the following values:  <b>Y</b> The index is clustered. <b>N</b> The index is not clustered.
NLEAF	INTEGER	The number of active leaf pages in the index. If unknown, the value is -1.
NLEVELS	SMALLINT	The number of levels in the index tree. If unknown, the value is -1.
INDEXTYPE	CHAR(1)	The index type. This column can have one of the following values:  <b>2</b> The index is a nonpartitioned secondary index. <b>D</b> The index is a data-partitioned secondary index.
PGSIZE	SMALLINT	The size, in kilobytes, of the leaf pages in the index. This column can have one of the following values: 4, 8, 16, or 32.
FIRSTKEYCARDF	FLOAT	The number of distinct values of the first key column. If unknown, the value is -1.
FULLKEYCARDF	FLOAT	The number of distinct values of the key. If unknown, the value is -1.
CLUSTERRATIOF	FLOAT	The percentage of rows that are in clustering order. Multiply this column value by 100 to get the percent value. For example, a value of .9125 in this column indicates that 91.25% of the rows are in clustering order. If unknown, the value is -1.
PADDED	CHAR(1)	Indicates whether keys within the index are padded for varying-length column data. This column can have one of the following values:  <b>Y</b> The keys are padded. <b>N</b> The keys are not padded.
COLNO1	SMALLINT	The column number of the first column in the index key.
ORDERING1	CHAR(1)	Indicates the order of the first column in the index key. This column can have one of the following values:  <b>A</b> Ascending <b>D</b> Descending
COLNO $n$	SMALLINT	The column number of the $n$ th column in the index key, where $n$ is a number between 2 and 64, including 2 and 64. If the number of index keys is less than $n$ , this column is null.


Table 146. SYSIBM.DSN\_VIRTUAL\_INDEXES description (continued)

Column name	Data type	Description
ORDERING $n$	CHAR(1)	Indicates the order of the $n$ th column in the index key, where $n$ is a number between 2 and 64, including 2 and 64. This column can have one of the following values:  <b>A</b> Ascending <b>D</b> Descending  If the number of index keys is less than $n$ , this column is null.

The SQL statement to create DSN\_VIRTUAL\_INDEXES is in member DSNTESC of the SDSNSAMP library. 

#### **SYSIBM.DSN\_PROFILE\_TABLE:**

Each row in the profile table, SYSIBM.DSN\_PROFILE\_TABLE, defines a profile. A *profile* is a set of criteria that identifies a particular query or set of queries.


 These profiles are used by optimization tools to perform various monitoring and tuning tasks.

**Recommendation:** Do not manually insert data into or delete data from this table, it is intended to be used only by optimization tools.

The following table describes the columns in SYSIBM.DSN\_PROFILE\_TABLE.


Table 147. SYSIBM.DSN\_PROFILE\_TABLE description

Column name	Data type	Description
AUTHID	VARCHAR(128)	The authorization ID of the user that is running the statement.
PLANNAME	VARCHAR(24)	The name of the plan that contains the statement.
COLLID	VARCHAR(128)	The name of the collection that contains the statement.
PKGNAME	VARCHAR(128)	The name of the package that contains the statement.
IPADDR	VARCHAR(254)	The IP address or domain name of a remote TCP/IP host from which the statement is issued. Use IP address 127.0.0.1 to specify the local host.
PROFILEID	INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY NOT NULL	The unique identifier for the profile that is defined by this row.
PROFILE_TIMESTAMP	TIMESTAMP NOT NULL WITH DEFAULT	The time when the row was inserted or updated.
PROFILE_ENABLED	CHAR(1) NOT NULL WITH DEFAULT 'Y'	Indicates whether the profile is enabled. This column can have one of the following values:  <b>Y</b> The profile is enabled. <b>N</b> The profile is disabled.
GROUP_MEMBER	VARCHAR(24)	The name of the DB2 member that inserted this row. This column is null if the DB2 subsystem is not in a data sharing environment.
REMARKS	VARCHAR(762)	Comments about this profile.

The SQL statement to create SYSIBM.DSN\_PROFILE\_TABLE is in member DSNTIJOS of the SDSNSAMP library. 

### **SYSIBM.DSN\_PROFILE\_HISTORY:**

The profile history table, SYSIBM.DSN\_PROFILE\_HISTORY, contains all of the profiles that were in effect at some point in time.


 Profiles are defined in the profile table, SYSIBM.DSN\_PROFILE\_TABLE. Profiles in the profile history table might no longer exist in the profile table.

This table is used by optimization tools when diagnosing optimization problems.

**Recommendation:** Do not manually insert data into or delete data from this table, it is intended to be used only by optimization tools.

The columns for SYSIBM.DSN\_PROFILE\_HISTORY are the same as the columns in SYSIBM.DSN\_PROFILE\_TABLE, except that the REMARKS column is named STATUS instead. The STATUS column has the following possible values:

- 'REJECTED - DUPLICATED SCOPE SPECIFIED'
- 'REJECTED - INVALID IPADDR SPECIFIED'
- 'REJECTED - INVALID SCOPE SPECIFIED'
- 'REJECTED - NO VALID RECORD FOUND IN ATTRIBUTE TABLE'
- 'REJECTED - INVALID KEYWORD SPECIFIED'
- 'REJECTED - Non-ZPARM keyword can not have a \* planname'
- 'REJECTED - ZPARM keyword must have a \* planname'
- 'REJECTED - The keyword for this scope is not allowed'
- 'REJECTED - INVALID ATTRIBUTE SPECIFIED'
- 'REJECTED - the latest (timestamp) row exist'
- 'REPLACED THE PROFILEID profileid KEYWORDS : keyword'
- 'ACCEPTED'
- 'ACCEPTED - Replaces MAX NORMAL PUSHOUT TOTAL with MAX PUSHOUT'
- 'ACCEPTED - Replaces MAX NORMAL PUSHOUT STMT with MAX PUSHOUT'
- 'ACCEPTED - Replaces MAX EXC PUSHOUT TOTAL with MAX PUSHOUT'
- 'ACCEPTED - Replaces MAX EXC PUSHOUT STMT with MAX PUSHOUT'
- 'ACCEPTED - Replaces MAX NORMAL PUSHOUT STMT with MAX NORMAL PUSHOUT TOTAL'
- 'ACCEPTED - Replaces MAX EXC PUSHOUT STMT with MAX EXC PUSHOUT TOTAL'
- 'ACCEPTED - Replaces the specified CPUTIME value with the maximum allowable value of 2<sup>32</sup> seconds'

The SQL statement to create SYSIBM.DSN\_PROFILE\_HISTORY is in member DSNTIJOS of the SDSNSAMP library. 

### **SYSIBM.DSN\_PROFILE\_ATTRIBUTES:**

The profile attributes table, SYSIBM.DSN\_PROFILE\_ATTRIBUTES, defines the attributes that are associated with a given profile.

**PSPI** This table is used by optimization tools when monitoring and tuning statements.

**Recommendation:** Do not manually insert data into or delete data from this table, it is intended to be used only by optimization tools.

The following tables describes the columns in SYSIBM.DSN\_PROFILE\_ATTRIBUTES.

Table 148. SYSIBM.DSN\_PROFILE\_ATTRIBUTES description

Column name	Data type	Description
PROFILEID	INTEGER NOT NULL FOREIGN KEY REFERENCES SYSIBM.DSN_PROFILE_TABLE ON DELETE CASCADE	The unique identifier for the profile. This value is from the PROFILEID column in SYSIBM.DSN_PROFILE_TABLE.
KEYWORDS	VARCHAR(128) NOT NULL	The function that DB2 performs. The possible function keywords are described in “Function keywords for SYSIBM.DSN_PROFILE_ATTRIBUTES.”
ATTRIBUTE1	VARCHAR(1024)	The string attribute that is associated with the function in the KEYWORDS column, if any.
ATTRIBUTE2	INTEGER	The integer attribute that is associated with the function in the KEYWORDS column, if any.
ATTRIBUTE3	FLOAT	The float attribute that is associated with the function in the KEYWORDS column, if any.
ATTRIBUTE_TIMESTAMP	TIMESTAMP NOT NULL WITH DEFAULT	The time when the row was inserted or updated.
REMARKS	VARCHAR(762)	Comments for this row.

The SQL statements to create SYSIBM.DSN\_PROFILE\_ATTRIBUTES and its index, SYSIBM.DSN\_PROFILE\_ATTRIBUTES\_IX\_ALL, are in member DSNTIJOS of the SDSNSAMP library. **PSPI**

**SYSIBM.DSN\_PROFILE\_ATTRIBUTES\_HISTORY:**

The profile attributes history table, SYSIBM.DSN\_PROFILE\_ATTRIBUTES\_HISTORY, contains the attributes that were in effect at some point in time.

The profile attributes history table, SYSIBM.DSN\_PROFILE\_ATTRIBUTES\_HISTORY, contains the attributes that were in effect at some point in time. These attributes are defined in the profile attributes table, SYSIBM.DSN\_PROFILE\_ATTRIBUTES.

However, the ATTRIBUTE\_TIMESTAMP column contains the timestamp of the associated PROFILEID in the profile history table.

This table is used by optimization tools when diagnosing optimization problems.

The columns for SYSIBM.DSN\_PROFILE\_ATTRIBUTES\_HISTORY are the same as the columns in SYSIBM.DSN\_PROFILE\_ATTRIBUTES. Except for the REMARKS column, which is named STATUS instead. The STATUS column has the following possible values:

- 'REJECTED - DUPLICATED SCOPE SPECIFIED'
- 'REJECTED - INVALID IPADDR SPECIFIED'
- 'REJECTED - INVALID SCOPE SPECIFIED'
- 'REJECTED - NO VALID RECORD FOUND IN ATTRIBUTE TABLE'
- 'REJECTED - INVALID KEYWORD SPECIFIED'
- 'REJECTED - Non-ZPARM keyword can not have a \* planname'
- 'REJECTED - ZPARM keyword must have a \* planname'
- 'REJECTED - The keyword for this scope is not allowed'
- 'REJECTED - INVALID ATTRIBUTE SPECIFIED'
- 'REJECTED - the latest (timestamp) row exist'
- 'REPLACED THE PROFILEID profileid KEYWORDS : keyword'
- 'ACCEPTED'
- 'ACCEPTED - Replaces MAX NORMAL PUSHOUT TOTAL with MAX PUSHOUT'
- 'ACCEPTED - Replaces MAX NORMAL PUSHOUT STMT with MAX PUSHOUT'
- 'ACCEPTED - Replaces MAX EXC PUSHOUT TOTAL with MAX PUSHOUT'
- 'ACCEPTED - Replaces MAX EXC PUSHOUT STMT with MAX PUSHOUT'
- 'ACCEPTED - Replaces MAX NORMAL PUSHOUT STMT with MAX NORMAL PUSHOUT TOTAL'
- 'ACCEPTED - Replaces MAX EXC PUSHOUT STMT with MAX EXC PUSHOUT TOTAL'
- 'ACCEPTED - Replaces the specified CPUTIME value with the maximum allowable value of 2<sup>32</sup> seconds'

The SQL statement to create SYSIBM.DSN\_PROFILE\_ATTRIBUTES\_HISTORY is in member DSNTIJOS of the SDSNSAMP library.

#### **SYSIBM.DSN\_STATEMENT\_RUNTIME\_INFO** or *user ID.DSN\_STATEMENT\_RUNTIME\_INFO*:

The statement runtime information table, SYSIBM.DSN\_STATEMENT\_RUNTIME\_INFO, contains runtime information for statements.



Optimization tools use this table when monitoring statements.

Your subsystem or data sharing group can have more than one of these tables. One table can be created with the qualifier SYSIBM. Other tables can be created with user IDs as the qualifiers. These tables are used by optimization tools.

**Restriction:** Turn off IFCID 318 trace while the profile facility is active. IFCID 318 trace activity interferes with the collection and reporting of statistics in DSN\_STATEMENT\_RUNTIME\_INFO.

**Recommendation:** Do not manually insert data into or delete data from this table, it is intended to be used only by optimization tools.

The following table describes the columns in DSN\_STATEMENT\_RUNTIME\_INFO.

*Table 149. SYSIBM.DSN\_STATEMENT\_RUNTIME\_INFO description*

Column name	Data type	Description
STMT_RUNTIME_INFO_ID	INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY NOT NULL	The ID for the row.
PROFILEID	INTEGER NOT NULL WITH DEFAULT	Contains the value of the PROFILEID column from DSN_PROFILE_TABLE, and associate this row with the row in the profile table.
STMT_ID	INTEGER NOT NULL WITH DEFAULT	The identifier for the SQL statement; this the unique ID assigned by EDM for this statement.
STMT_TOKEN	VARCHAR(240) NOT NULL WITH DEFAULT	The statement token. This value is a number identification string.
APPLNAME	VARCHAR(24) NOT NULL WITH DEFAULT	The name of the application plan that contains the statement.
PROGNAME	VARCHAR(128) NOT NULL WITH DEFAULT	The name of the package or DBRM that issued the initial PREPARE statement for the SQL statement.
COLLID	VARCHAR(128) NOT NULL WITH DEFAULT	The collection ID for the package. Blank if not applicable.
VERSION	VARCHAR(122) NOT NULL WITH DEFAULT	The version identifier of the package that contains the statement.
SECTNO	INTEGER NOT NULL WITH DEFAULT	The section number of the statement. This value applies only to static SQL. This number is set to -1 for a dynamic statement.
CACHED_TS	TIMESTAMP NOT NULL WITH DEFAULT	The timestamp when the statement was stored in the dynamic statement cache EDM pool.
PRIMAUTH	VARCHAR(128) NOT NULL WITH DEFAULT	For dynamic SQL: The primary authorization ID that issued the initial PREPARE statement.  For static SQL: The value of the OWNER bind option.
CURSQLID	VARCHAR(128) NOT NULL WITH DEFAULT	The value of the CURRENT SQLID special register when the initial PREPARE statement was run. This value is applicable only to dynamic SQL. For static SQL, this value is blank.
SCHEMA	VARCHAR(128) NOT NULL WITH DEFAULT	The value of the CURRENT SCHEMA special register. This value is applicable only to dynamic SQL. For static SQL, this value is blank.
BIND_QUALIFIER	VARCHAR(128) NOT NULL WITH DEFAULT	The QUALIFIER option of the BIND subcommand. For unqualified table names, this value is the object qualifier.
BIND_ISO	CHAR(2) NOT NULL WITH DEFAULT	The value of the ISOLATION bind option that is in effect for the statement. The value can be one of the following values:  <b>UR</b> Uncommitted read  <b>CS</b> Cursor stability  <b>RS</b> Read stability  <b>RR</b> Repeatable read

Table 149. SYSIBM.DSN\_STATEMENT\_RUNTIME\_INFO description (continued)


Column name	Data type	Description
BIND_CDATA	CHAR(1) NOT NULL WITH DEFAULT	The value of the CURRENTDATA bind option that is in effect for the statement. The value can be one of the following values:  Y      CURRENTDATA(YES) N      CURRENTDATA(NO)
BIND_DYNRL	CHAR(1) NOT NULL WITH DEFAULT	The value of the DYNAMICRULES bind option that is in effect for the statement. The value can be one of the following values:  B      DYNAMICRULE(BIND) R      DYNAMICRULES(RUN)
BIND_DEGREE	CHAR(1) NOT NULL WITH DEFAULT	The value of the CURRENT DEGREE special register that is in effect for the statement. The value can be one of the following values:  A      CURRENT DEGREE = ANY 1      CURRENT DEGREE = 1
BIND_SQLRL	CHAR(1) NOT NULL WITH DEFAULT	The value of the CURRENT RULES special register that is in effect for the statement. The value can be one of the following values:  D      CURRENT RULES = DB2 S      CURRENT RULES = SQL
BIND_CHOLD	CHAR(1) NOT NULL WITH DEFAULT	The value of the WITH HOLD attribute of the PREPARE statement for the SQL statement. The value can be one of the following values:  Y      The initial PREPARE statement was specified with the WITH HOLD option. N      The initial PREPARE was specified with the WITHOUT HOLD option.
STAT_TS	TIMESTAMP NOT NULL WITH DEFAULT	The time at which statistics were gathered. This value is applicable only to dynamic SQL. For static SQL, this value is the current timestamp.
STAT_EXEC	INTEGER NOT NULL WITH DEFAULT	The number of times that the statement has been run. For a statement with a cursor, this value is the number of times that an OPEN statement has been issued.
STAT_GPAG	INTEGER NOT NULL WITH DEFAULT	The number of getpage operations that DB2 performed for the statement.
STAT_SYNR	INTEGER NOT NULL WITH DEFAULT	The number of synchronous buffer reads that DB2 performed for the statement.
STAT_WRIT	INTEGER NOT NULL WITH DEFAULT	The number of buffer write operations that DB2 performed for the statement.
STAT_EROW	INTEGER NOT NULL WITH DEFAULT	The number of rows that DB2 examined for the statement.
STAT_PROW	INTEGER NOT NULL WITH DEFAULT	The number of rows that DB2 processed for the statement.
STAT_SORT	INTEGER NOT NULL WITH DEFAULT	The number of sorts that DB2 performed for the statement.

Table 149. SYSIBM.DSN\_STATEMENT\_RUNTIME\_INFO description (continued)

Column name	Data type	Description
STAT_INDX	INTEGER NOT NULL WITH DEFAULT	The number of index scans that DB2 performed for the statement.
STAT_RSCN	INTEGER NOT NULL WITH DEFAULT	The number of table space scans that DB2 performed for the statement.
STAT_PGRP	INTEGER NOT NULL WITH DEFAULT	The number of parallel groups that DB2 created for the statement.
STAT_ELAP	FLOAT NOT NULL WITH DEFAULT	The accumulated elapsed time of the statement.
STAT_CPU	FLOAT NOT NULL WITH DEFAULT	The accumulated CPU time of the statement.
STAT_CPU_STMT	FLOAT NOT NULL WITH DEFAULT	The CPU time that DB2 uses only for the current execution of this statement.
STAT_SUS_SYNIO	FLOAT NOT NULL WITH DEFAULT	The accumulated wait time for synchronous I/O operations.
STAT_SUS_LOCK	FLOAT NOT NULL WITH DEFAULT	The accumulated wait time for lock and latch requests.
STAT_SUS_SWIT	FLOAT NOT NULL WITH DEFAULT	The accumulated wait time for synchronous execution unit switch.
STAT_SUS_GLCK	FLOAT NOT NULL WITH DEFAULT	The accumulated wait time for global locks.
STAT_SUS_OTHM	FLOAT NOT NULL WITH DEFAULT	The accumulated wait time for read activity from another thread.
STAT_SUS_OTHW	FLOAT NOT NULL WITH DEFAULT	The accumulated wait time for write activity from another thread.
STAT_RIDLMT	INTEGER NOT NULL WITH DEFAULT	The number of times that DB2 did not use a RID list, because the number of RIDs exceeded DB2 limits.
STAT_RIDSTOR	INTEGER NOT NULL WITH DEFAULT	The number of times that DB2 did not use a RID list, because not enough storage was available to hold the list of RIDs.
PUSHOUT_TS	TIMESTAMP NOT NULL WITH DEFAULT	The time at which the report is pushed out.
REASON	SMALLINT NOT NULL WITH DEFAULT	The reason or cause why this row is generated. Possible values are:  1      A stop profile command was issued 2      The cache entry was purged from the statement cache 3      An exception occurred 4      An EXPLAIN statement was executed 5      An IFCID 318 trace was started


Table 149. SYSIBM.DSN\_STATEMENT\_RUNTIME\_INFO description (continued)

Column name	Data type	Description										
ROW_TYPE	SMALLINT NOT NULL WITH DEFAULT	<p>The type of row. This column can have one of the following values:</p> <table><tr><td>-1</td><td>Error</td></tr><tr><td>0</td><td>Normal monitored row</td></tr><tr><td>1</td><td>Summary row</td></tr><tr><td>2</td><td>Exception row</td></tr><tr><td>3</td><td>Warning</td></tr></table>	-1	Error	0	Normal monitored row	1	Summary row	2	Exception row	3	Warning
-1	Error											
0	Normal monitored row											
1	Summary row											
2	Exception row											
3	Warning											
EXCEPTION_TYPE	INTEGER NOT NULL WITH DEFAULT	<p>Indicates what exceptions are generated. The value is a bitmap combination of the following values:</p> <table><tr><td>'0000'B</td><td>No exception occurred</td></tr><tr><td>'0001'B</td><td>RLF exception</td></tr><tr><td>'0100'B</td><td>CPUTIME exception</td></tr><tr><td>'1000'B</td><td>Spike exception</td></tr></table>	'0000'B	No exception occurred	'0001'B	RLF exception	'0100'B	CPUTIME exception	'1000'B	Spike exception		
'0000'B	No exception occurred											
'0001'B	RLF exception											
'0100'B	CPUTIME exception											
'1000'B	Spike exception											
EXCEPTION_COUNT	INTEGER NOT NULL WITH DEFAULT	The number of exceptions that occurred for the statement during the monitoring period. This value is applicable only to the summary rows. For exception rows, the value is 0.										
STMT_TEXT	CLOB(2M) NOT NULL	The text of the statement. For static statements, this column contains the first 4K bytes of the statement text.										
STMT_TRUNCATED	CHAR(1) NOT NULL	The value can be either 'Y' or 'N'. The value 'Y' indicates that the statement text is truncated to 4K long.										
STMT_ROWID	ROWID NOT NULL GENERATED ALWAYS	The row ID of the statement.										
GROUP_MEMBER	VARCHAR(24) NOT NULL WITH DEFAULT	The name of the DB2 member that inserted this row. If the DB2 subsystem is not in a data-sharing environment, this value is blank.										

The SQL statements to create SYSIBM.DSN\_STATEMENT\_RUNTIME\_INFO and the associated auxiliary table to store the CLOB data for the STMT\_TEXT column are in member DSNTIJOS of the SDSNSAMP library. 

#### **SYSIBM.DSN\_OBJECT\_RUNTIME\_INFO** or *user ID.DSN\_OBJECT\_RUNTIME\_INFO*:

The object runtime information table, SYSIBM.DSN\_OBJECT\_RUNTIME\_INFO, contains runtime information about tables and indexes that DB2 accesses when it processes queries.

 Optimization tools use this table when monitoring statements.

Your subsystem or data sharing group can have more than one of these tables. One table can be created with the qualifier SYSIBM. Other tables can be created with user IDs as the qualifiers. These tables are used by optimization tools.

**Recommendation:** Do not manually insert data into or delete data from this table, it is intended to be used only by optimization tools.


The following table describes the columns in  
SYSIBM.DSN\_OBJECT\_RUNTIME\_INFO.

*Table 150. SYSIBM.DSN\_OBJECT\_RUNTIME\_INFO description*

Column name	Data type	Description
STMT_RUNTIME_ INFO_ID	INTEGER NOT NULL FOREIGN KEY REFERENCES SYSIBM.DSN_ STATEMENT_ RUNTIME_INFO ON DELETE CASCADE	The foreign key for joining to the table SYSIBM.DSN_STATEMENT_RUNTIME_INFO.
QBLOCKNO	SMALLINT NOT NULL WITH DEFAULT	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.
PLANNO	SMALLINT NOT NULL WITH DEFAULT	The number of step in which the query indicated in QBLOCKNO was processed. This column indicates the order in which the steps were executed.
MIXOPSEQ	SMALLINT NOT NULL WITH DEFAULT	The sequence number of a step in a multiple index operation.  <b>1,2,...n</b> For the steps of multiple index procedure (ACCESSTYPE is MX, MI, or MU)  <b>0</b> For any other rows (ACCESSTYPE is I, I1, M, N, R, or blank.)
PARENT_ QBLOCKNO	SMALLINT NOT NULL WITH DEFAULT	A number that indicates the QBLOCKNO of the parent query block.
PARENT_PLANNO	SMALLINT NOT NULL WITH DEFAULT	The plan number of the parent query.
DMSROW_EST	FLOAT NOT NULL WITH DEFAULT	The estimate of rows scanned by DB2 after all indexed predicates are applied.
DMROWS_EST	FLOAT NOT NULL WITH DEFAULT	The estimate of rows returned by DB2 after all stage 1 predicates are applied.
RDSROW_EST	FLOAT NOT NULL WITH DEFAULT	The estimate of rows that are qualified by DB2 after all stage 2 predicates are applied.
COMPCD_EST	FLOAT NOT NULL WITH DEFAULT	The estimate on composite cardinality.
PROCESSED_ ROWS	FLOAT NOT NULL WITH DEFAULT	The total number of rows that DB2 processes.
LOOKAT_ROWS	FLOAT NOT NULL WITH DEFAULT	The total number of rows that DB2 reads. For simple table spaces, this value is different than the value in the PROCESSED_ROWS column.
DMROWS	FLOAT NOT NULL WITH DEFAULT	The total number of rows that are qualified by DB2 after all stage 1 predicates are applied
RDSROWS	FLOAT NOT NULL WITH DEFAULT	The total number of rows that are qualified by DB2 after all stage 2 predicates are applied.
INDEX_FF_EST	FLOAT NOT NULL WITH DEFAULT	The filter factor estimate for an index object.
INDEX_FFADJ_EST	FLOAT NOT NULL WITH DEFAULT	The adjusted filter factor for an index object.

Table 150. SYSIBM.DSN\_OBJECT\_RUNTIME\_INFO description (continued)

Column name	Data type	Description
IX_PROCESSED_ROWS	FLOAT NOT NULL WITH DEFAULT	The number of rows processed for an index object.
IX_LOOKAT_ROWS	FLOAT NOT NULL WITH DEFAULT	The number of rows looked at for an index object.
IX_DMROWS	FLOAT NOT NULL WITH DEFAULT	The number of rows, for an index object, that are qualified by DB2 after all stage 1 predicates are applied.
IX_RDSROWS	FLOAT NOT NULL WITH DEFAULT	The number of rows, for an index object, that are qualified by DB2 after all stage 2 predicates are applied.
PUSHOUT_TS	TIMESTAMP NOT NULL WITH DEFAULT	The timestamp at which the report is pushed out.
GROUP_MEMBER	VARCHAR(24) NOT NULL WITH DEFAULT	The name of the DB2 data sharing member that inserted this row. If the DB2 subsystem is not in a data-sharing environment, or is the migrating member of a data sharing system, this value is blank.

The SQL statement to create SYSIBM.DSN\_OBJECT\_RUNTIME\_INFO is in member DSNTIJOS of the SDSNSAMP library. 

## Creating a profile

A profile is a set of criteria that identifies a particular query or queries.


 To create a profile, insert a row into SYSIBM.DSN\_PROFILE\_TABLE with one of the following sets of values:

Table 151. Values to insert into SYSIBM.DSN\_PROFILE\_TABLE

The statements that you want the profile to include	Values to insert into SYSIBM.DSN_PROFILE_TABLE
All statements that are run by a specific authorization ID from a specified IP address	Use the values in Table 152.
All statements in a specific plan, collection, or package	Use the values in Table 153 on page 589.

Table 152. Values to insert in SYSIBM.DSN\_PROFILE\_TABLE if you want the profile to include all statements that are run by a specific authorization ID

Column name	Value
AUTHID	The authorization ID that is running the statements
PLANNAME	null
COLLID	null
PKGNAME	null
IPADDR	The IP address that is used for establishing the connection
PROFILEID	Do not specify a value. DB2 generates this value.

Table 152. Values to insert in SYSIBM.DSN\_PROFILE\_TABLE if you want the profile to include all statements that are run by a specific authorization ID (continued)

Column name	Value
PROFILE_TIMESTAMP	CURRENT_TIMESTAMP
PROFILE_ENABLED	Y
GROUP_MEMBER	For a data-sharing environment, specify the member from which you are issuing the INSERT statement. Otherwise, leave this column blank.
REMARKS	Optionally, specify any comments that you want to save in this row, such as a description of the profile. Otherwise, leave this column blank.

Table 153. Values to insert in SYSIBM.DSN\_PROFILE\_TABLE if you want the profile to include all statements in a specific plan, collection, or package

Column name	Value
AUTHID	null
PLANNAME	The name of the plan that contains the statements.
COLLID	The name that uniquely identifies the collection or a null value.
PKGNAME <sup>1</sup>	The name of the package for the statements or a null value.
IPADDR	null
PROFILEID	Do not specify a value. DB2 generates this value.
PROFILE_TIMESTAMP	CURRENT_TIMESTAMP
PROFILE_ENABLED	Y
GROUP_MEMBER	For a data-sharing environment, specify the member from which you are issuing the INSERT statement. Otherwise, leave this column blank.
REMARKS	Optionally, specify any comments that you want to save in this row, such as a description of the profile. Otherwise, leave this column blank.

**Note:**


1. If you specify a value for either PKGNAME or COLLID, you must specify values for both.

For more information about the SYSIBM.DSN\_PROFILE\_TABLE, see

“SYSIBM.DSN\_PROFILE\_TABLE” on page 579 

## Function keywords for SYSIBM.DSN\_PROFILE\_ATTRIBUTES

Function keywords specify the function that you want DB2 to perform.

 Specify these keywords in the profile attributes table, SYSIBM.DSN\_PROFILE\_ATTRIBUTES, in the columns KEYWORDS, ATTRIBUTE1, ATTRIBUTE2, and ATTRIBUTE3. For more information about this table, see “SYSIBM.DSN\_PROFILE\_ATTRIBUTES” on page 580.

You can specify functions for the following tasks:

- Monitoring queries. Those functions are described in Table 154 on page 590

- Modifying subsystem parameter values. Those functions are described in Table 156 on page 592

The following table lists the functions that you can use to specify that DB2 monitors queries.

*Table 154. Functions for monitoring queries*

Keyword	Attribute1	Attribute2	Attribute3	Description
MONITOR	NULL	An integer that indicates the location where DB2 records monitoring information.	NULL	Specifies that DB2 monitors statements that are included in the specified profile. DB2 records the monitoring information in the appropriate tables when the statement exits the cache.
MONITOR RLF	NULL	An integer that indicates the location where DB2 records monitoring information.	NULL	Specifies that DB2 monitors the execution of statements that are included in the specified profile. DB2 records the monitoring information in the appropriate tables only when an RLF exception occurs.
MONITOR CPUTIME	NULL	An integer that indicates the location where DB2 records monitoring information.	A float value that specifies the given CPUTIME threshold	Specifies that DB2 monitors the execution of statements that match the criteria expressed by the values in the first 5 columns of the profile table. Collected information is pushed into the appropriate tables only when the actual CPU time exceeds the specified absolute CPUTIME value. The maximum allowable value for CPUTIME threshold is 2 <sup>32</sup> seconds.
MONITOR SPIKE	PERCENT	An integer that indicates the location where DB2 records monitoring information.	A float value that specifies the given CPU time threshold	Specifies that DB2 monitors the execution of statements that are included in the specified profile. DB2 records the monitoring information in the appropriate tables only when the actual CPU time differs from the average CPU time by more than the specified percentage. For DB2 to calculate the average CPU time, the query needs to be executed at least 10 times.

## The granularity of reports sent by monitor profiles

The following table summarizes the granularity of reported information, and when reports are sent to profile tables.

The Attribute2 column specifies the granularity of recorded statistics. Low-order bits indicate which types of information to collect. High-order bits indicate the granularity of statistics collection. Minimal statistics collection records data for the STAT\_EXC and STAT\_CPU columns in the DSN\_STATEMENT\_RUNTIME\_INFO table. The following table lists the possible values for the Attribute2 column and their meanings.

Table 155. The Attribute2 column values for monitoring functions

Attribute2 value	Description
First byte (bits 0 to 7, right to left order)	<p>Group of information requested: statement runtime, object runtime, or EXPLAIN information. Bits 4 to 7 are reserved and should be set to 0.</p> <p><b>'0001'B</b> Statement runtime information is requested. T1 receives the report</p> <p><b>'0010'B</b> Object runtime information is requested. T2 receives the report</p> <p><b>'0100'B</b> EXPLAIN information is requested. T3 or T4 receives the report, depending on the value of bits 14 to 16</p>
Remaining bytes (bits 8 to 31, right to left order)	<p>Granularity of information for each group. Bits 8 to 10 are for statement runtime, 11 to 13 are for object runtime, and bits 14 to 16 are for EXPLAIN information. Bits 17 to 31 are reserved and should be set to 0.</p> <p><b>'000'B</b> Minimum statistics and reports for statement runtime information</p> <p><b>'111'B</b> Maximum statistics and reports for statement runtime information</p> <p><b>'000xxx'B</b> Minimum statistics and reports for object runtime information</p> <p><b>'111xxx'B</b> Maximum statistics and reports for object runtime information</p> <p><b>'000xxxxxx'B</b> Minimum statistics and reports for EXPLAIN information.</p> <p><b>'111xxxxxx'B</b> Maximum statistics and reports for EXPLAIN information.</p>

#### Notes:

- Tables that receive reports from monitor profiles are identified above as follows:

**T1** DSN\_STATEMENT\_RUNTIME\_INFO

**T2** DSN\_OBJECT\_RUNTIME\_INFO

**T3** EXPLAIN tables, including PLAN\_TABLE, DSN\_STATEMENT\_TABLE, and DSN\_FUNCTION\_TABLE.

**T4** Other explain tables that are used by optimization tools.

- Your subsystem or data sharing group can have more than one of each these tables. One table can be created with the qualifier SYSIBM. Other tables can be created with user IDs as the qualifiers. DB2 records information in the appropriate table according to the following logic:

- When a monitored statement exits the cache or the specified exception occurs for a monitored statement, DB2 records the specified information in tables that are qualified by SYSIBM.
- If you issue an EXPLAIN statement for a monitored statement while monitoring is active, DB2 records the information in the tables that are qualified by *userID*.

The table below lists the functions that you can use to specify that DB2 modifies a subsystem parameter value.

Table 156. Functions for modifying subsystem parameters

Keyword	Attribute1	Attribute2	Attribute3	Description	Name of changed subsystem parameter
NPAGES THRESHOLD	NULL	An integer 0 or greater	NULL	Specifies that DB2 is to set the pages threshold to the value that is specified for the Attribute2 column. This value indicates when DB2 should use index access. You can specify one of the following values:  -1 DB2 is to use index access for as many tables as possible.  0 DB2 is to select the access path based on the cost. This value is the default.  1 to <i>n</i> DB2 should use index access on tables for which the total number of pages (NPAGES) is less than <i>n</i> . Ensure that statistics are up to date before you specify a value of 1 or greater.	NPGTHRSH
STAR JOIN	DISABLE or ENABLE	NULL	NULL	Specifies whether DB2 is to use star join processing. You can specify one of the following values for Attribute1:  <b>DISABLE</b> DB2 is not to use star join processing.  <b>ENABLE</b> DB2 is to use star join processing when possible.	STARJOIN
MIN STAR JOIN TABLES	NULL	An integer between 3 and 225	NULL	Specifies that DB2 is to set the minimum number of tables for star join processing to the value that is specified for the Attribute2 column. This value indicates that DB2 is to use star joins when the number of tables in the statement is equal to or greater than the specified integer.	SJTABLES



## Monitoring statements by using profile tables

Monitoring statements helps you to view query performance and isolate any problem queries.



If you have already identified a specific SQL statement as a problem query, you should monitor only that statement.

Before you can monitor any statements, you must create all of the following tables:

- SYSIBM.DSN\_PROFILE\_TABLE
- SYSIBM.DSN\_PROFILE\_HISTORY
- SYSIBM.DSN\_PROFILE\_ATTRIBUTES
- SYSIBM.DSN\_PROFILE\_ATTRIBUTES\_HISTORY

- SYSIBM.DSN\_STATEMENT\_RUNTIME\_INFO

Depending on the information that you want DB2 to record (as described in Table 155 on page 591 ), you might also need to create some of the following tables:

- SYSIBM.DSN\_OBJECT\_RUNTIME\_INFO
- SYSIBM.PLAN\_TABLE
- SYSIBM.DSN\_STATEMENT\_TABLE
- SYSIBM.DSN\_FUNCTION\_TABLE
- Other EXPLAIN tables that are used by optimization tools.

For information about these tables and how to create them, see “Profile tables” on page 576.

To monitor a group of statements:

1. Specify the statements that you want to monitor by performing one of the following actions:
  - Create a profile. For instructions on how to create a profile, see “Creating a profile” on page 588.
  - Find an existing profile that includes the statement or statements that you want to monitor by querying SYSIBM.DSN\_PROFILE\_TABLE. Ensure that, in the row for that profile, the value for the PROFILE\_ENABLED column is Y. If the value is N, change it to Y. For more information about this table, see “SYSIBM.DSN\_PROFILE\_TABLE” on page 579. For more information about profiles, see “Profiles” on page 575.
2. Define the type of monitoring that you want to perform by inserting a row into SYSIBM.DSN\_PROFILE\_ATTRIBUTES with the following values:

#### PROFILE ID column

Specify the profile that defines the statements that you want to monitor. Use a value from the PROFILEID column in SYSIBM.DSN\_PROFILE\_TABLE.

#### KEYWORDS

Specify one of the following monitoring keywords:

- MONITOR
- MONITOR RLF
- MONITOR CPU TIME
- MONITOR SPIKE

For a complete description of each of these keywords, see Table 154 on page 590.

#### ATTRIBUTE1, ATTRIBUTE2, ATTRIBUTE3

Specify the appropriate attribute values depending on the keyword that you specify in the KEYWORDS column. For a description of the valid attributes, see Table 154 on page 590.

You do not need to specify values for any other columns.

**Example:** Suppose that you insert the three rows in the following table into SYSIBM.DSN\_PROFILE\_ATTRIBUTES:

Table 157. Sample data in SYSIBM.DSN\_PROFILE\_ATTRIBUTES

PROFILEID	KEYWORDS	ATTRIBUTE1	ATTRIBUTE2	ATTRIBUTE3	ATTRIBUTE TIMESTAMP	REMARKS
10	MONITOR		1			
11	MONITOR SPIKE	PERCENT	3	300.00	2005-06-23...	

Table 157. Sample data in SYSIBM.DSN\_PROFILE\_ATTRIBUTES (continued)

PROFILEID	KEYWORDS	ATTRIBUTE1	ATTRIBUTE2	ATTRIBUTE3	ATTRIBUTE TIMESTAMP	REMARKS
12	MONITOR RLF		3		2005-06-23...	

These rows specify that you want DB2 to monitor SQL statements as follows:

- The first row specifies that DB2 monitors the statements that satisfy the scope that is defined by profile 10. DB2 records the monitoring information for these statements in SYSIBM.DSN\_STATEMENT\_RUNTIME\_INFO.
- The second row indicates that DB2 monitors the statements that satisfy the scope that is defined by profile 11. When the CPU time differs from the average CPU time by more than 300%, DB2 records monitoring information in SYSIBM.DSN\_STATEMENT\_RUNTIME\_INFO and SYSIBM.DSN\_OBJECT\_RUNTIME\_INFO.
- The third row indicates that DB2 monitors the statements that satisfy the scope that is defined by profile 12. When an RLF exception occurs, DB2 records monitoring information in SYSIBM.DSN\_STATEMENT\_RUNTIME\_INFO and SYSIBM.DSN\_OBJECT\_RUNTIME\_INFO.

For more information about SYSIBM.DSN\_PROFILE\_ATTRIBUTES, see “SYSIBM.DSN\_PROFILE\_ATTRIBUTES” on page 580.

For more information about how DB2 resolves any conflicts in the profile tables, see “How DB2 resolves conflicting rows in the profile attributes table” on page 597.

3. Load or reload the profile tables into memory by issuing the following command:

```
START PROFILE
```

Any rows with a Y in the PROFILE\_ENABLED column in SYSIBM.DSN\_PROFILE\_TABLE are now in effect. DB2 monitors any statements that meet the specified criteria.

DB2 records any monitoring information. The value that you specified for Attribute2 of the monitoring keyword indicates the tables where the monitoring information is recorded. For more information about those keywords and their attributes, see Table 154 on page 590.

If you specify the MONITOR keyword, DB2 inserts information in the specified tables every time that the statement leaves the cache. If you specify one of the other monitoring keywords, DB2 inserts information in the specified tables only when an exception occurs. The total number of rows that DB2 inserts is limited by the value of the MAX PUSHOUT subsystem parameter.

4. **Optional:** If any statement runs for a long time, take a snapshot of the current status. For instructions, see “Obtaining snapshot information for monitored queries” on page 598.
5. View the monitoring information by querying the tables that you specified, which might include any of the following tables:
  - SYSIBM.DSN\_STATEMENT\_RUNTIME\_INFO
  - SYSIBM.DSN\_OBJECT\_RUNTIME\_INFO
  - SYSIBM.PLAN\_TABLE
  - SYSIBM.DSN\_STATEMNT\_TABLE
  - SYSIBM.DSN\_FUNCTION\_TABLE
6. When you finish monitoring these statements, stop the monitoring process by performing one of the following tasks:

Option	Description
To disable the monitoring function for a specific profile	Delete that row from DSN_PROFILE_TABLE, or change the PROFILE_ENABLED column value to N. Then, reload the profile table by issuing the following command: START PROFILE
To disable all monitoring and subsystem parameters that have been specified in the profile tables	Issue the following command: STOP PROFILE

**Recommendation:** To limit unnecessary overhead that might affect performance, stop the monitoring process if you no longer need to monitor the statement.



#### Related concepts

“EXPLAIN tables that are used by optimization tools” on page 503

## Limiting the number of statement reports from a monitor profile

You can use keywords to limit the number of statements that a monitor profile captures and the number of times that the monitor profile captures information about a single statement to prevent information overflow.

When any of these limits is reached, DB2 writes a summary record to the DSN\_STATEMENT\_RUNTIME\_INFO table to indicate this event. Note that the reports generated by EXPLAIN MONITORED STMTS do not count toward these limits, and no summary records are counted.

To limit the number of statement reports issued from a monitor profile:

Insert the appropriate values into the function table from the following choices:

Table 158. Functions for modifying subsystem parameters

Keyword	Attribute1	Attribute2	Attribute3	Description
MAX PUSHOUT	NULL	10000	NULL	<p>The maximum number of total reports allowed for all (normal and exception) statement monitoring during the lifetime of a monitor profile. When this value is reached, the monitor profile does not issue any more statement reports and ends its monitoring activities.</p> <p>The default value is set to 10000, but you can modify the value by adding an entry in the profile table without any restrictive filters and an associated with an entry in the profile attributes table that specifies the new maximum.</p> <p>The MAX PUSHOUT keyword is only relevant when associated with a global scope entry in the profile table and implicitly associated with the monitor function.</p> <p>MAX PUSHOUT is the only keyword that can be created with a global scope in the profile table.</p>
MAX NORMAL PUSHOUT	STMT	1 (unchangeable)	NULL	<p>The maximum number of statement reports allowed for each profile entry for basic or normal monitoring, during the lifetime of a single statement in the cache. The default value is 1 (unchangeable), which means one statement report when the statement is purged from cache. The keyword 'MAX NORMAL PUSHOUT' is only relevant if associated with the MONITOR function.</p>
MAX NORMAL PUSHOUT	TOTAL	5000	NULL	<p>The maximum number of total statement reports allowed for each profile entry for basic monitor during the lifetime of the monitor function.</p> <p>The default value is 5000. You can specify a different value by specifying the new value in the profile attributes table that is associated with the profile entry.</p>
MAX EXC PUSHOUT	STMT	An integer	NULL	<p>The maximum number of statement reports allowed for each profile entry for exception monitor profiles during the lifetime of each single statement in the cache.</p> <p>In the absence of this entry, the system will assume default value of 1, which means only one report per statement, when the first exception is detected. The keyword 'MAX EXC PUSHOUT' is only relevant if associated with the MONITOR exception function.</p>

Table 158. Functions for modifying subsystem parameters (continued)

Keyword	Attribute1	Attribute2	Attribute3	Description
MAX EXC PUSHOUT	TOTAL	200	NULL	The maximum number of total statement reports allowed for each profile entry for exception monitor profiles during the lifetime of the monitor function.  The default value is 200. You can specify a different value by specifying a profile attribute entry associated with the appropriate scope.

## How DB2 resolves conflicting rows in the profile attributes table

Rows in SYSIBM.DSN\_PROFILE\_ATTRIBUTES can specify that DB2 is to take more than one action for certain queries.



DB2 uses the following rules to process those rows and resolve any conflicts:

- When a row negates another row, DB2 uses the row with the latest timestamp.

**Example:** Suppose that you insert the following two rows into SYSIBM.DSN\_PROFILE\_ATTRIBUTES:

Table 159. Sample conflicting rows in SYSIBM.DSN\_PROFILE\_ATTRIBUTES

PROFILEID	KEYWORDS	ATTRIBUTE1	ATTRIBUTE2	ATTRIBUTE3	ATTRIBUTE TIMESTAMP	REMARKS
10	STAR JOIN	DISABLE			2005-06-22	
10	STAR JOIN	ENABLE			2005-06-23	

Both rows in this table specify how DB2 is to process the statements that satisfy the scope that is defined by profile 10. The first row tells DB2 to not use star join processing. The second row tells DB2 to use star join processing if possible. In this case, the row that has the ENABLE keyword has a later timestamp and is used by DB2

- When a row compliments another row, DB2 uses both rows.

**Example:** Suppose that you insert the following two rows into SYSIBM.DSN\_PROFILE\_ATTRIBUTES:

Table 160. Sample complementary rows in SYSIBM.DSN\_PROFILE\_ATTRIBUTES

PROFILEID	KEYWORDS	ATTRIBUTE1	ATTRIBUTE2	ATTRIBUTE3	ATTRIBUTE TIMESTAMP	REMARKS
11	MONITOR		1		2005-06-22..	
11	MONITOR SPIKE	25	1	300.0	2005-06-23...	

Both rows in this table give directions to DB2 for how to monitor the statements that satisfy the scope that is defined by profile 11. Because these rows do not conflict, but complement one another, DB2 performs both specified monitoring functions. These functions specify that DB2 monitors the statements and record the monitoring information for each statement when the statement leaves the cache and when the CPU time increases dramatically.

- When statements satisfy the scope of more than one profile, and the actions that are specified for those profiles conflict, the action for the most specific profile

takes precedence. DB2 treats a profile that filters statements based on AUTHID as more specific than a profile that filters statements based on PLANNAME, COLLID, and PKGNAME.

**Example:** Suppose that you have the following profiles defined in SYSIBM.DSN\_PROFILE\_TABLE.

Table 161. Sample profiles with overlapping scopes in SYSIBM.DSN\_PROFILE\_TABLE


AUTHID	PLAN NAME	COLLID	PKGNAME	IPADDR	PROFILE ID
null	P1	C1	null	null	12
null	P1	C1	PK1	null	13

The first row defines a profile that includes all statements that are in plan P1 and in collection C1. The second row defines a profile that includes all statements that are in plan P1, in collection C1, and in package PK1. Notice that all of the statements that satisfy the scope of profile 13 also satisfy the scope of profile 12.

Suppose that you also have the following two rows in SYSIBM.DSN\_PROFILE\_ATTRIBUTES:


Table 162. Sample rows in SYSIBM.DSN\_PROFILE\_ATTRIBUTES for profiles with overlapping scopes

PROFILEID	KEYWORDS	ATTRIBUTE1	ATTRIBUTE2	ATTRIBUTE3	ATTRIBUTE TIMESTAMP	REMARKS
12	STAR JOIN	DISABLE			2005-06-30..	
13	STAR JOIN	ENABLE			2005-06-30...	

The first row tells DB2 to not use star join processing for the statements in profile 12. The second row tells DB2 to use star join processing if possible for the statements in profile 13. Because all of the statements in profile 13 also satisfy the scope of profile 12, this table tells DB2 to perform two conflicting actions for the statements in profile 13. In this case, the more specific profile, profile 13, takes precedence. DB2 does not use star join processing for statements that are in plan P1 and collection C1, except for those statements that are also in package PK1. 

## Obtaining snapshot information for monitored queries

When you monitor a statement, DB2 does not report the query information until the statement leaves the cache. If you want information sooner, you can force DB2 to report some of the query information by taking a snapshot of the current status.


 This feature is especially useful if the query has been running for a long time.

Before you can obtain snapshot information for monitored queries, you must create the DSN\_STATEMENT\_RUNTIME\_INFO table under your user schema. If the query is being monitored, you must also create the SYSIBM.DSN\_STATEMENT\_RUNTIME\_INFO table. For information about this table and how to create it, see “SYSIBM.DSN\_STATEMENT\_RUNTIME\_INFO or user ID.DSN\_STATEMENT\_RUNTIME\_INFO” on page 582.

To obtain snapshot information for monitored queries, issue an EXPLAIN statement and specify the appropriate scope. **Examples:** The following table shows several example EXPLAIN statements that limit the scope of the EXPLAIN function to a group of SQL statements.


Table 163. Example EXPLAIN statements that limit the scope of the EXPLAIN function to a group of SQL statements

EXPLAIN statement	Meaning
EXPLAIN MONITORED STMTS SCOPE AUTHID 'TOMAS' IPADDR '9.30.36.181'	DB2 records information for monitored statements that are running under the authorization ID TOMAS from the IP address 9.30.36.181.
EXPLAIN MONITORED STMTS SCOPE PLAN 'p1'	DB2 records information for monitored statements that are running in plan p1.
EXPLAIN MONITORED STMTS SCOPE PLAN 'p2' COLLECTION 'c1' PACKAGE 'pak3'	DB2 records information for monitored statements that are running in plan p2, collection c1, and package pak3.

When you issue any of these EXPLAIN statements, DB2 records the information in the tables that you specified when you set up monitoring for that query. For a list of the possible tables to which you can have DB2 report information, see Table 155 on page 591. For more information about setting up monitoring, see “Monitoring statements by using profile tables” on page 592. 

## Subsystem parameters for optimization


You can set certain subsystem parameter values to help optimize how DB2 processes SQL queries.

 For a list of these subsystem parameters, see Table 156 on page 592.

You set the subsystem parameter values when you install DB2. You can subsequently change these values in one of the following three ways:


- Using the SET SYSPARM command. For more information about how to use this command, see *DB2 Command Reference*.
- Using the profile tables. For instructions, see “Modifying subsystem parameter values by using profile tables.”

The precedence and scope of these methods for setting subsystem parameter values are as follows:

1. DB2 first applies any subsystem parameter values that are set by the profile tables. These parameters apply to all statements that are included in the specified profile, and the parameters are in effect only for the time in which that profile is enabled.
2. Then DB2 applies any subsystem parameter values that are set by the installation panels or the SET SYSPARM command. These parameters apply to the entire subsystem until the values are changed. 

## Modifying subsystem parameter values by using profile tables

You can use the profile tables to modify certain subsystem parameter values.

 Before you can modify a subsystem parameter value by using the profile tables, you must create the following tables:

- SYSIBM.DSN\_PROFILE\_TABLE

- SYSIBM.DSN\_PROFILE\_HISTORY
- SYSIBM.DSN\_PROFILE\_ATTRIBUTES
- SYSIBM.DSN\_PROFILE\_ATTRIBUTES\_HISTORY

For information about these tables and how to create them, see “Profile tables” on page 576.

To modify a subsystem parameter:

1. Specify the statements that you want to assign new subsystem parameters to by performing one of the following actions:
  - Create a profile. For instructions on how to create a profile, see “Creating a profile” on page 588.
  - Find an existing profile that includes any statement that you want to assign new subsystem parameters to by querying SYSIBM.DSN\_PROFILE\_TABLE. Ensure that, in the row for that profile, the value for the PROFILE\_ENABLED column is Y. If the value is N, change it to Y. For more information about this table, see “SYSIBM.DSN\_PROFILE\_TABLE” on page 579. For more information about profiles, see “Profiles” on page 575.
2. Specify the subsystem parameter that you want to modify by inserting a row into SYSIBM.DSN\_PROFILE\_ATTRIBUTES with the following column values:

#### PROFILE ID column

Specify the profile that defines the scope of statements to which you want the subsystem parameter to apply. Use a value from the PROFILEID column in SYSIBM.DSN\_PROFILE\_TABLE.

#### KEYWORDS

Specify a valid subsystem parameter keyword as described in Table 156 on page 592.

#### ATTRIBUTE1, ATTRIBUTE2, ATTRIBUTE 3

Specify the appropriate attribute values depending on the keyword that you specify in the KEYWORDS column. For a description of the valid attributes, see Table 156 on page 592.

For more information about SYSIBM.DSN\_PROFILE\_ATTRIBUTES, see “SYSIBM.DSN\_PROFILE\_ATTRIBUTES” on page 580.

**Example:** Suppose that you insert the row in the following table into SYSIBM.DSN\_PROFILE\_ATTRIBUTES:

Table 164. Sample data in SYSIBM.DSN\_PROFILE\_ATTRIBUTES.

PROFILEID	KEYWORDS	ATTRIBUTE1	ATTRIBUTE2	ATTRIBUTE3	ATTRIBUTE TIMESTAMP	REMARKS
17	STAR JOIN	DISABLE			2005-06-23...	

This row specifies that DB2 is to disable star join processing for all statements that are included in profile 17.

3. Load or reload the profile tables into memory by issuing the following command:

```
START PROFILE
```

Any rows with a Y in the PROFILE\_ENABLED column in SYSIBM.DSN\_PROFILE\_TABLE and SYSIBM.DSN\_PROFILE\_ATTRIBUTES are now in effect. DB2 applies the values for the specified parameters to the statements in the specified profile.

4. When you no longer want DB2 to use the parameter values that you specified in the profile tables, disable the values by performing one of the following actions:

Option	Description
To disable the parameter values for a specific profile	Delete that row from DSN_PROFILE_TABLE, or change the PROFILE_ENABLED column value to N. Then, reload the profile table by issuing the following command: START PROFILE
To disable all monitoring and subsystem parameters that are specified in the profile tables	Issue the following command: STOP PROFILE

PSPI

## Checking for invalid plans and packages

You can check whether application plans and packages remain valid and operative.

The following statements identify plans and packages that meet the following criteria:

- Might redo validity checks at run time; if an invalid object or missing authority is found, DB2 issues a warning and checks again for the object or authorization at run time
- Use repeatable read
- Are invalid (must be rebound before use), for example, deleting an index or revoking authority can render a plan or package invalid
- Are inoperative (require an explicit BIND or REBIND before use). A plan or package can be marked inoperative after an unsuccessful REBIND

To check for invalid plans and packages:

Issue the following SQL statements:

```
SELECT NAME, VALIDATE, ISOLATION, VALID, OPERATIVE
FROM SYSIBM.SYSPLAN
WHERE VALIDATE = 'R' OR ISOLATION = 'R'
OR VALID = 'N' OR OPERATIVE = 'N';
SELECT COLLID, NAME, VERSION, VALIDATE, ISOLATION, VALID, OPERATIVE
FROM SYSIBM.SYSPACKAGE
WHERE VALIDATE = 'R' OR ISOLATION = 'R'
OR VALID = 'N' OR OPERATIVE = 'N';
```

## Monitoring parallel operations

The number of parallel operations or tasks used to access data is initially determined at bind time, and later adjusted when the query is executed.

**Bind time:** At bind time, DB2 collects partition statistics from the catalog, estimates the processor cycles for the costs of processing the partitions, and determines the optimal number of parallel tasks to achieve minimum elapsed time.

When a planned degree exceeds the number of online CPs, the query might not be completely processor-bound. Instead it might be approaching the number of partitions because it is I/O-bound. In general, the more I/O-bound a query is, the closer the degree of parallelism is to the number of partitions.

In general, the more processor-bound a query is, the closer the degree of parallelism is to the number of online CPs, and it can even exceed the number of CPs by one.

**Example:** Suppose that you have a processor-intensive query on a 10-partition table, and that this query is running on a 6-way CPC. The degree of parallelism can be up to 7 in this case.

To help DB2 determine the optimal degree of parallelism, use the utility RUNSTATS to keep your statistics current.

PLAN\_TABLE shows the planned degree of parallelism in the columns ACCESS\_DEGREE and JOIN\_DEGREE.

**Execution time:** For each parallel group, parallelism (either CP or I/O) can execute at a reduced degree or degrade to sequential operations for the following reasons:

- Amount of virtual buffer pool space available
- Host variable values
- Availability of the hardware sort assist facility
- Ambiguous cursors
- A change in the number or configuration of online processors
- The join technique that DB2 uses (I/O parallelism not supported when DB2 uses the star join technique)

At execution time, a plan using Sysplex query parallelism can use CP parallelism. All parallelism modes can degenerate to a sequential plan. No other changes are possible.

## Using DISPLAY BUFFERPOOL

You can use the output from DISPLAY BUFFERPOOL DETAIL report to see how well the buffer pool is able to satisfy parallel operations.

**GUIP**

```
DSNB440I = PARALLEL ACTIVITY -  
          PARALLEL REQUEST =      282  DEGRADED PARALLEL=      5
```

The PARALLEL REQUEST field in this example shows that DB2 was negotiating buffer pool resource for 282 parallel groups. Of those 282 groups, only 5 were degraded because of a lack of buffer pool resource. A large number in the DEGRADED PARALLEL field could indicate that your subsystem does not have enough buffers that can be used for parallel processing. **GUIP**

## Using DISPLAY THREAD

The DISPLAY THREAD command displays parallel tasks. Whereas previously you would only see information about the originating task, now you can see information about the parallel tasks associated with that originating task.

**GUIP**

Whereas previously you would only see information about the originating task, now you can see information about the parallel tasks associated with that originating task. The status field contains PT for parallel tasks. All parallel tasks are displayed immediately after their corresponding originating thread. **GUIP**

## Using DB2 trace

The statistics trace indicates when parallel groups do not run to the planned degree or run sequentially. These are possible indicators that some queries that are not achieving the best possible response times.

Use the accounting trace to ensure that your parallel queries are meeting their response time goals. If you discover a potential problem with a parallel query, use the performance trace to do further analysis.

### Accounting trace

By default, DB2 rolls task accounting into an accounting record for the originating task. OMEGAMON also summarizes all accounting records generated for a parallel query and presents them as one logical accounting record.

OMEGAMON presents the times for the originating tasks separately from the accumulated times for all the parallel tasks.

As shown in Figure 88, CP CPU TIME-AGENT is the time for the originating tasks, while CP CPU TIME-PAR.TASKS ( **A** ) is the accumulated processing time for the parallel tasks.

TIMES/EVENTS	APPL (CLASS 1)	DB2 (CLASS 2)	CLASS 3 SUSP.	ELAPSED TIME
-----	-----	-----	-----	-----
ELAPSED TIME	32.578741	32.312218	LOCK/LATCH	25.461371
NONNESTED	28.820003	30.225885	SYNCHRON. I/O	0.142382
STORED PROC	3.758738	2.086333	DATABASE I/O	0.116320
UDF	0.000000	0.000000	LOG WRTE I/O	0.026062
TRIGGER	0.000000	0.000000	OTHER READ I/O	3:00.404769
			OTHER WRTE I/O	0.000000
CPU CP TIME	1:29.695300	1:29.644026	SER.TASK SWTCH	0.000000
AGENT	0.225153	0.178128	UPDATE COMMIT	0.000000
NONNESTED	0.132351	0.088834	OPEN/CLOSE	0.000000
STORED PRC	0.092802	0.089294	SYSLGRNG REC	0.000000
UDF	0.000000	0.000000	EXT/DEL/DEF	0.000000
TRIGGER	0.000000	0.000000	OTHER SERVICE	0.000000
PAR.TASKS	<b>A</b> 1:29.470147	1:29.465898	ARC.LOG(QUIES)	0.000000
...				
	QUERY PARALLEL.	TOTAL		
	-----	-----		
	MAXIMUM MEMBERS	1		
	MAXIMUM DEGREE	10		
	GROUPS EXECUTED	1		
	RAN AS PLANNED <b>B</b>	1		
	RAN REDUCED <b>C</b>	0		
	ONE DB2 COOR=N	0		
	ONE DB2 ISOLAT	0		
	ONE DB2 DCL TTABLE	0		
	SEQ - CURSOR <b>D</b>	0		
	SEQ - NO ESA <b>E</b>	0		
	SEQ - NO BUF <b>F</b>	0		
	SEQ - ENCL.SER.	0		
	MEMB SKIPPED(%)	0		
	DISABLED BY RLF <b>G</b>	NO		
	REFORM PARAL-CONFIG <b>H</b>	0		
	REFORM PARAL-NO BUF	0		

Figure 88. Partial accounting trace, query parallelism

As the report shows, the values for CPU TIME and I/O WAIT TIME are larger than the elapsed time. The processor and suspension time can be greater than the elapsed time because these two times are accumulated from multiple parallel tasks. The elapsed time would be less than the processor and suspension time if these two times are accumulated sequentially.

If you have baseline accounting data for the same thread run without parallelism, the elapsed times and processor times should not be significantly larger when that query is run in parallel. If it is significantly larger, or if response time is poor, you need to examine the accounting data for the individual tasks. Use the OMEGAMON Record Trace for the IFCID 0003 records of the thread you want to examine. Use the performance trace if you need more information to determine the cause of the response time problem.

### **Performance trace**

The performance trace can give you information about tasks within a group.

To determine the actual number of parallel tasks use:

Refer to field QW0221AD in IFCID 0221, as mapped by macro DSNDQW03. The 0221 record also gives you information about the key ranges used to partition the data.

IFCID 0222 contains the elapsed time information for each parallel task and each parallel group in each SQL query. OMEGAMON presents this information in its SQL Activity trace.

If your queries are running sequentially or at a reduced degree because of a lack of buffer pool resources, the QW0221C field of IFCID 0221 indicates which buffer pool is constrained.

---

## **Monitoring DB2 in a distributed environment**

DB2 provides several ways to monitor DB2 data and events in a distributed environment.

You can use the DISPLAY command and the trace facility to obtain information. DB2 can also return server-elapsed time to certain types of client applications.

### **The DISPLAY command**

The DB2 DISPLAY command gives you information about the status of threads, databases, tracing, allied subsystems, and applications.

#### **GUIP**

Several forms of the DISPLAY command are particularly helpful for monitoring DB2 including:

- DISPLAY THREAD
- DISPLAY LOCATION
- DISPLAY DDF DETAIL
- DISPLAY DATABASE
- DISPLAY TRACE

## Tracing distributed events

A number of IFCIDs, including IFCID 0001 (statistics) and IFCID 0003 (accounting), record distributed data and events.

If your applications update data at other sites, turn on the statistics class 4 trace and always keep it active. This statistics trace covers error situations surrounding in doubt threads; it provides a history of events that might impact data availability and data consistency.

DB2 accounting records are created separately at the requester and each server. Events are recorded in the accounting record at the location where they occur. When a thread becomes active, the accounting fields are reset. Later, when the thread becomes inactive or is terminated, the accounting record is created.

Figure 89 on page 606 shows the relationship of the accounting class 1 and 2 times and the requester and server accounting records. Figure 90 on page 607 and Figure 91 on page 608 show the server and requester distributed data facility blocks from the OMEGAMON accounting long trace.

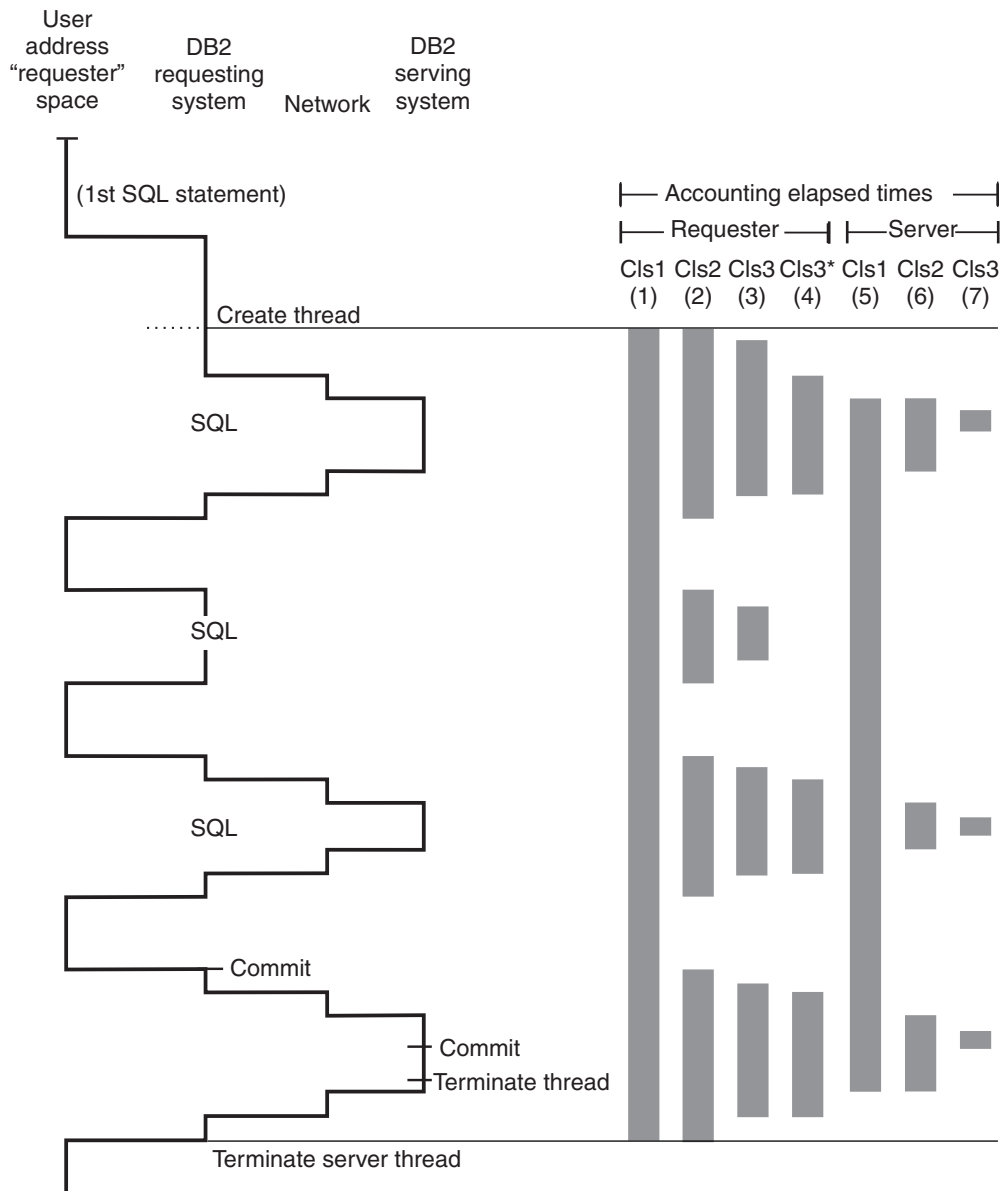


Figure 89. Elapsed times in a DDF environment as reported by OMEGAMON. These times are valid for access that uses either DRDA or private protocol (except as noted).

This figure is a simplified picture of the processes that go on in the serving system. It does not show block fetch statements and is only applicable to a single row retrieval.

The various elapsed times referred to in the header are:

- (1) - Requester Cls1  
This time is reported in the ELAPSED TIME field under the APPL (CLASS 1) column near the top of the OMEGAMON accounting long trace for the requesting DB2 subsystem. It represents the elapsed time from the creation of the allied distributed thread until the termination of the allied distributed thread.
- (2) - Requester Cls2  
This time is reported in the ELAPSED TIME field under the DB2 (CLASS 2) column near the top of the OMEGAMON accounting long trace for the

requesting DB2 subsystem. It represents the elapsed time from when the application passed the SQL statements to the local DB2 system until return. This is considered "In DB2" time.

- (3) - Requester Cls3

This time is reported in the TOTAL CLASS 3 field under the CLASS 3 SUSP column near the top of the OMEGAMON accounting long trace for the requesting DB2 system. It represents the amount of time the requesting DB2 system spent suspended waiting for locks or I/O.

- (4) - Requester Cls3\* (Requester wait time for activities not in DB2)

This time is reported in the SERVICE TASK SWITCH, OTHER SERVICE field of the OMEGAMON accounting report for the requesting DB2 subsystem. It is typically time spent waiting for the network and server to process the request.

- (5) - Server Cls1

This time is reported in the ELAPSED TIME field under the APPL (CLASS 1) column near the top of the OMEGAMON accounting long trace for the serving DB2 subsystem. It represents the elapsed time from the creation of the database access thread until the termination of the database access thread.

- (6) - Server Cls2

This time is reported in the ELAPSED TIME field under the DB2 (CLASS 2) column near the top of the OMEGAMON accounting long trace of the serving DB2 subsystem. It represents the elapsed time to process the SQL statements and the commit at the server.

- (7) - Server Cls3

This time is reported in the TOTAL CLASS 3 field under the CLASS 3 SUSP column near the top of the OMEGAMON accounting long trace for the serving DB2 subsystem. It represents the amount of time the serving DB2 system spent suspended waiting for locks or I/O.

The Class 2 processing time (the TCB time) at the requester does not include processing time at the server. To determine the total Class 2 processing time, add the Class 2 time at the requester to the Class 2 time at the server.

Likewise, add the getpage counts, prefetch counts, locking counts, and I/O counts of the requester to the equivalent counts at the server. For private protocol, SQL activity is counted at both the requester and server. For DRDA, SQL activity is counted only at the server.

---- DISTRIBUTED ACTIVITY -----				
SERVER	: BOEBDB2SERV	SUCCESSFULLY ALLOC.CONV:	<b>C</b> N/A	MSG.IN BUFFER <b>E</b> : 0
PRODUCT ID	: DB2	CONVERSATION TERMINATED:	N/A	
PRODUCT VERSION	: V8 R1 M0	MAX OPEN CONVERSATIONS :	N/A	PREPARE SENT : 1
METHOD	: DRDA PROTOCOL	CONT->LIM.BL.FTCH SWCH :	<b>D</b> N/A	LASTAGN.SENT : 0
REQUESTER ELAP.TIME	: 0.685629			MESSAGES SENT : 3
SERVER ELAPSED TIME	: N/A	COMMIT(2) RESP.RECV. :	1	MESSAGES RECEIVED: 2
SERVER CPU TIME	: N/A	BKOUT(2) R.R. :	0	BYTES SENT : 9416
DBAT WAITING TIME	: 0.026118	TRANSACT.SENT :	1	BYTES RECEIVED : 1497
COMMIT (2) SENT	: 1	COMMIT(1)SENT :	0	BLOCKS RECEIVED : 0
BACKOUT(2) SENT	: 0	ROLLB(1)SENT :	0	STMT BOUND AT SER: <b>F</b> N/A
CONVERSATIONS INITIATED:	<b>A</b> 1	SQL SENT :	0	
CONVERSATIONS QUEUED :	<b>B</b> 0	ROWS RECEIVED :	1	FORGET RECEIVED : 0

Figure 90. DDF block of a requester thread from a OMEGAMON accounting long trace

```

----- DISTRIBUTED ACTIVITY -----
REQUESTER      : BOEBDB2REQU      ROLLBK(1) RECEIVED :    0    PREPARE RECEIVED   :    1
PRODUCT ID     : DB2              SQL RECEIVED       :    0    LAST AGENT RECV.   :    1
PRODUCT VERSION : V8 R1 M0        COMMIT(2) RESP.SENT:    1    THREADS INDOUBT   :    0
METHOD         : DRDA PROTOCOL    BACKOUT(2)RESP.SENT:    0    MESSAGES.IN BUFFER :    0
COMMIT(2) RECEIVED :              1 BACKOUT(2)PERFORMED:    0    ROWS SENT          :    0
BACKOUT(2) RECEIVED:              0 MESSAGES SENT      :    3    BLOCKS SENT        :    0
COMMIT(2) PERFORMED:              1 MESSAGES RECEIVED  :    5    CONVERSAT.INITIATED:    1
TRANSACTIONS RECV. :              1 BYTES SENT         :   643   FORGET SENT         :    0
COMMIT(1) RECEIVED :              0 BYTES RECEIVED     :  3507

```

Figure 91. DDF block of a server thread from a OMEGAMON accounting long trace

The accounting distributed fields for each serving or requesting location are collected from the viewpoint of this thread communicating with the other location identified. For example, SQL *sent* from the requester is SQL *received* at the server. Do not add together the distributed fields from the requester and the server.

Several fields in the distributed section merit specific attention. The number of conversations is reported in several fields:

- The number of conversation allocations is reported as CONVERSATIONS INITIATED ( **A** ).
- The number of conversation requests queued during allocation is reported as CONVERSATIONS QUEUED ( **B** ).
- The number of successful conversation allocations is reported as SUCCESSFULLY ALLOC.CONV ( **C** ).
- The number of times a switch was made from continuous block fetch to limited block fetch is reported as CONT->LIM.BL.FTCH ( **D** ). This is only applicable to access that uses DB2 private protocol.

You can use the difference between initiated allocations and successful allocations to identify a session resource constraint problem. If the number of conversations queued is high, or if the number of times a switch was made from continuous to limited block fetch is high, you might want to tune VTAM to increase the number of conversations. VTAM and network parameter definitions are important factors in the performance of DB2 distributed processing. For more information, see *VTAM for MVS/ESA Network Implementation Guide*.

Bytes sent, bytes received, messages sent, and messages received are recorded at both the requester and the server. They provide information on the volume of data transmitted. However, because of the way distributed SQL is processed for private protocol, more bytes might be reported as sent than are reported as received.

To determine the percentage of the rows transmitted by block fetch, compare the total number of rows sent to the number of rows sent in a block fetch buffer, which is reported as MSG.IN BUFFER ( **E** ). The number of rows sent is reported at the server, and the number of rows received is reported at the requester. Block fetch can significantly affect the number of rows sent across the network.

The number of SQL statements bound for remote access is the number of statements dynamically bound at the server for private protocol. This field is maintained at the requester and is reported as STMT BOUND AT SER ( **F** ).

Because of the manner in which distributed SQL is processed, the number of rows that are reported might differ slightly from the number of rows that are received.

However, a significantly lower number of rows received might indicate that the application did not fetch the entire answer set. This is especially true for access that uses private protocol.

## Reporting server-elapsed time

Client applications that access DB2 data using DRDA access can request that DB2 return the server-elapsed time.

Server-elapsed time allows remote clients to determine the actual amount of time it takes for DB2 to parse a remote request, process any SQL statements required to satisfy the request, and generate the reply. Because server-elapsed time does not include any of the network time used to receive the request or send the reply, client applications can use the information to quickly isolate poor response times to the network or to the DB2 server without you having to perform traces on the server. Only application clients using DB2 Connect Version 7 or later can request the server-elapsed time. When the System Monitor statement switch has been turned on, DB2 returns server-elapsed time information as a new element through the regular Snapshot Monitoring APIs.

---

## Monitoring distributed processing with RMF

If you use RMF to monitor DDF work, understand how DDF is using the enclave SRBs.

The information that is reported using RMF or an equivalent product in the SMF 72 records are the portions of the client's request that are covered by individual enclaves. The way DDF uses enclaves relates directly to whether the DDF thread can become inactive.

## Duration of an enclave

If a thread is always active, the duration of the thread is the duration of the enclave. Otherwise, certain conditions control the duration of an enclave.

If the thread can be pooled, the following conditions determine the duration of an enclave:

- If the associated package is bound with `KEEPDYNAMIC(NO)`, or there are no open held cursors, or there are active declared temporary tables, the duration of the enclave is the period during which the thread is active.
- If the associated package is bound with `KEEPDYNAMIC(YES)`, and no held cursors or active declared temporary tables exist, and only `KEEPDYNAMIC(YES)` keeps the thread from being pooled, the duration of the enclave is the period from the beginning to the end of the transaction.

While a thread is pooled, such as during think time, it is not using an enclave. Therefore, SMF 72 record does not report inactive periods.

ACTIVE MODE threads are treated as a single enclave from the time it is created until the time it is terminated. This means that the entire life of the database access thread is reported in the SMF 72 record, regardless of whether SQL work is actually being processed. Figure 92 on page 610 contrasts the two types of threads.

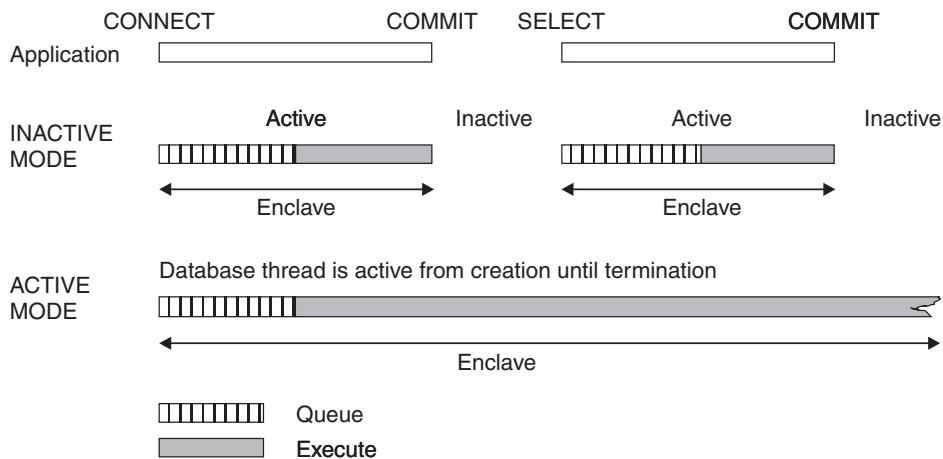


Figure 92. Contrasting ACTIVE MODE threads and POOLED MODE threads

**Queue time:** Note that the information that is reported back to RMF includes queue time. This particular queue time includes waiting for a new or existing thread to become available.

## RMF records for enclaves

You can pull out total enclave usage from the record, but you must use DB2 accounting traces to see resource consumption for a particular enclave.

The two most frequently used SMF records are types 30 and 72. The type 30 record contains resource consumption at the address space level.

Each enclave contributes its data to one type 72 record for the service class and to zero or one (0 or 1) type 72 records for the report class. You can use WLM classification rules to separate different enclaves into different service or report classes. Separating the enclaves in this way enables you to understand the DDF work better.

---

## Part 5. Analyzing performance data

The first step to investigating a performance problem is to look at the overall system to determine whether the problem might be outside of DB2. After you determine that the problem is inside DB2, you can begin detailed analysis of performance data that you captured during monitoring.

When analyzing performance data, keep in mind that almost all symptoms of poor performance are magnified when contention occurs.

For example, if a slowdown in disk operations occurs:

- Transactions can pile up, waiting for data set activity.
- Transactions can wait for I/O and locks.
- Paging can be delayed.

In addition, more transactions in the system means greater processor overhead, greater real-storage demand, greater I/O demand, and greater locking demand. This increased overhead and demand can have a large impact.

In such situations, the system shows heavy use of all of its resources. However, the system is actually experiencing typical system stress, with a constraint that is yet to be found.



---

## Chapter 33. Looking at the entire system

You should start by looking at the overall system before you decide that you have a problem in DB2

| . In general, look in some detail to see why application processes are progressing  
| slowly, or why a given resource is being heavily used. The best tools for that kind  
| of investigation include the resource measurement facility (RMF) of z/OS and IBM  
| Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS. If the application  
| processes are running on distributed platforms, you must look at the performance  
| on those platforms and the network that connects them with z/OS.



---

## Chapter 34. Beginning to look at DB2

Before you analyze DB2 for performance problems, look at the overall system. If your initial analysis suggests that the performance problem is within DB2, the problem might be poor response time, an unexpected and unexplained high use of resources, or locking conflicts.

Check factors such as total processor usage, disk activity, and paging.

First, turn on classes 1, 2, and 3 of the accounting trace to get a picture of task activity and determine whether the performance problem is within DB2. Although accounting trace class 2 has a high overhead cost for fetch-intensive applications that do not use multi-row fetch, it is generally recommended that you keep it on in all cases. For other applications, accounting trace class 2 has an overhead of less than 5%. Accounting trace class 3 can help you determine which resources DB2 is waiting on.

Next, focus on particular activities, such as specific application processes or a specific time interval. You might see certain problems:

### **Slow response time**

You could look at detailed traces of one slow task, a problem for which there could be several reasons. For instance, the users could be trying to do too much work with certain applications, work that clearly takes time, and the system simply cannot do all the work that they want done.

### **Real storage constraints**

Applications progress more slowly than expected because of paging interrupts. The constraints show up as delays between successive requests recorded in the DB2 trace.

### **Contention for a particular function**

For example, processes might have to wait on a particular data set, or a certain application might cause many application processes to put the same item in their queues. Use the DB2 performance trace to distinguish most of these cases.

### **Poor performance from specific individual queries or query workloads**

Particular queries, and query workloads, might perform poorly because of how the queries are written, because important statistics have not been collected, or because useful indexes have not been created. All of these problems can cause DB2 to make select suboptimal access paths. Use EXPLAIN and IBM Optimization Service Center for DB2 for z/OS, to analyze your queries and to get performance tuning recommendations.

For information about packages or DBRMs, run accounting trace classes 7 and 8. To determine which packages are consuming excessive resources, compare accounting trace classes 7 and 8 to the elapsed time for the whole plan on accounting classes 1 and 2.

A number greater than 1 in the QXMAXDEG field of the accounting trace indicates that parallelism was used. Interpreting such records requires special considerations.

The easiest way to read and interpret the trace data is through the reports produced by OMEGAMON. If you do not have OMEGAMON or an equivalent program, but you can also read reports from DB2 traces.

You can also use tools for performance measurement, and analyze statistics in the DB2 catalog and directory.

**Related concepts**

Chapter 20, "Using tools to monitor performance," on page 419

"Monitoring parallel operations" on page 601

Chapter 36, "Interpreting DB2 trace output," on page 621

"Maintaining statistics in the catalog" on page 153

---

## Chapter 35. A general approach to problem analysis in DB2

The following is a suggested sequence for investigating a response-time problem:

1. If the problem is inside DB2, determine which plan has the longest response time. If the plan can potentially allocate many different packages or DBRMs, determine which packages or DBRMs have the longest response time. Or, if you have a record of past history, determine which transactions show the largest increases. Compare class 2 CPU time, class 3 time, and not accounted time. If your performance monitoring tool does not specify times other than class 2 and class 3, then you can determine the not accounted for time with the following formula:

Not accounted time = Class 2 elapsed time - Class 2 CPU time - Total class 3 time

2. If the class 2 CPU time is high, investigate by doing the following:
    - Check to see whether unnecessary trace options are enabled. Excessive performance tracing can be the reason for a large increase in class 2 CPU time.
    - Check the SQL statement count, the getpage count, and the buffer update count on the OMEGAMON accounting report. If the profile of the SQL statements has changed significantly, review the application. If the getpage counts or the buffer update counts change significantly, check for changes to the access path and significant increases in the number of rows in the tables that are accessed.
    - Use the statistics report to check buffer pool activity, including the buffer pool thresholds. If buffer pool activity has increased, be sure that your buffer pools are properly tuned. For more information on buffer pools, see “Tuning database buffer pools” on page 56.
    - Use IBM Optimization Service Center for DB2 for z/OSDB2 EXPLAIN to check the efficiency of the access paths for your application. Based on the EXPLAIN results:
      - Use package-level accounting reports to determine which package or DBRM has a long elapsed time. In addition, use the class 7 CPU time for packages to determine which package or DBRM has the largest CPU time or the greatest increase in CPU time.
      - Use the OMEGAMON SQL activity report to analyze specific SQL statements. You can also use OMEGAMON to analyze specific SQL statements, including the currently running SQL statement.
      - If you have a history of the performance of the affected application, compare current EXPLAIN output to previous access paths and costs.
      - Check that RUNSTATS statistics are current.
      - Check that databases have been reorganized using the REORG utility.
      - Check which indexes are used and how many columns are accessed. Has your application used an alternative access path because an index was dropped?
      - Examine joins and subqueries for efficiency.
- To analyze access path problems, use IBM Optimization Service Center for DB2 for z/OS and DB2 Explain.
- Check the counts in the locking section of the OMEGAMON accounting report. For a more detailed analysis, use the deadlock or timeout traces from statistics trace class 3 and the lock suspension report or trace.

3. If class 3 time is high, check the individual types of suspensions in the “Class 3 Suspensions” section of the OMEGAMON accounting report.

Option	Description
If LOCK/LATCH ( <b>A</b> ), DRAIN LOCK ( <b>I</b> ), or CLAIM RELEASE ( <b>J</b> ) time is high	See Chapter 15, “Programming for concurrency,” on page 319.
SYNCHRON. I/O ( <b>B</b> ) time is high	See Synchronous I/O suspension time.
OTHER READ I/O ( <b>D</b> ) time is high	Check prefetch I/O operations, disk contention and the tuning of your buffer pools.
If OTHER WRITE I/O ( <b>E</b> ) time is high	Check the I/O path, disk contention, and the tuning of your buffer pools.
If SER.TASK SWTCH ( <b>F</b> ) is high	<p>Check open and close activity, as well as commit activity. A high value could also be caused by:</p> <ul style="list-style-type: none"> <li>• SYSLGRNG recording service</li> <li>• Data set extend/delete/define service</li> </ul> <p>Consider also, the possibility that DB2 is waiting for Hierarchical Storage Manager (HSM) to recall data sets that had been migrated to tape. The amount of time that DB2 waits during the recall is specified on the RECALL DELAY parameter on installation panel DSNTIPO.</p>

If accounting class 8 trace was active, each of these suspension times is available on a per-package or per-DBRM basis in the package block of the OMEGAMON accounting report.

4. If NOT ACCOUNT. ( **L** ) time is high, check for paging activity, processor wait time, return wait time for requests to be returned from VTAM or TCP/IP, wait time for completion of parallel tasks, and the use of online performance monitors. Turn off or reduce the intensity of online monitoring to eliminate or significantly reduce a high NOT ACCOUNT time that is caused by some monitors. A high NOT ACCOUNT time is acceptable if it is caused by wait time for completion of parallel tasks.
  - Use RMF reports to analyze paging, CPU utilization, and other workload activities.
  - Check the SER.TASK SWTCH field in the “Class 3 Suspensions” section of the OMEGAMON accounting reports.

The figure below shows which reports you might use, depending on the nature of the problem, and the order in which to look at them.

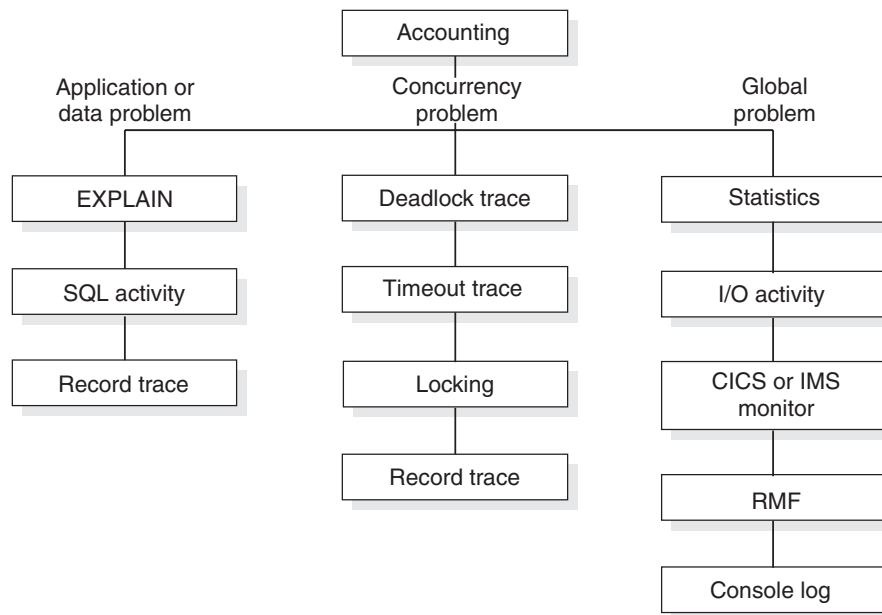


Figure 93. OMEGAMON reports used for problem analysis

If you suspect that the problem is in DB2, it is often possible to discover its general nature from the accounting reports. You can then analyze the problem in detail based on one of the branches shown in the figure above:

- Follow the first branch, **Application or data problem**, when you suspect that the problem is in the application itself or in the related data. Also use this path for a further breakdown of the response time when no reason can be identified.
- The second branch, **Concurrency problem**, shows the reports required to investigate a lock contention problem..
- Follow the third branch for a **Global problem**, such as an excessive average elapsed time per I/O. A wide variety of transactions could suffer similar problems.

Before starting the analysis in any of the branches, start the DB2 trace to support the corresponding reports. When starting the DB2 trace:

- Refer to *OMEGAMON Report Reference* for the types and classes needed for each report.
- To make the trace data available as soon as an experiment has been carried out, and to avoid flooding the SMF data sets with trace data, use a GTF data set as the destination for DB2 performance trace data.

Alternatively, use the Collect Report Data function in OMEGAMON to collect performance data. You specify only the report set, not the DB2 trace types or classes you need for a specific report. Collect Report Data lets you collect data in a TSO data set that is readily available for further processing. No SMF or GTF handling is required.

- To limit the amount of trace data collected, you can restrict the trace to particular plans or users in the reports for SQL activity or locking. However, you cannot so restrict the records for performance class 4, which traces asynchronous I/O for specific page sets. You might want to consider turning on selective traces and be aware of the added costs incurred by tracing.

If the problem is not in DB2, check the appropriate reports from a CICS or IMS reporting tool.

When CICS or IMS reports identify a commit, the time stamp can help you locate the corresponding OMEGAMON accounting trace report.

You can match DB2 accounting records with CICS accounting records. If you specify ACCOUNTREC(UOW) or ACCOUNTREC(TASK) on the DB2ENTRY RDO definition, the CICS LU 6.2 token is included in the DB2 trace records, in field QWHCTOKEN of the correlation header. To help match CICS and DB2 accounting records, specify ACCOUNTREC(UOW) or ACCOUNTREC(TASK) in the DB2ENTRY definition. That writes a DB2 accounting record after every transaction. As an alternative, you can produce OMEGAMON accounting reports that summarize accounting records by CICS transaction ID. Use the OMEGAMON function Correlation Translation to select the subfield containing the CICS transaction ID for reporting.

You can synchronize the statistics recording interval with the RMF reporting interval, using the STATIME and SYNCVAL subsystem parameters. STATIME specifies the length of the statistics interval, and SYNCVAL specifies that the recording interval is synchronized with some part of the hour.

**Example:** If the RMF reporting interval is 15 minutes, you can set the STATIME to 15 minutes and SYNCVAL to 0 to synchronize with RMF at the beginning of the hour. These values cause DB2 statistics to be recorded at 15, 30, 45, and 60 minutes past the hour, matching the RMF report interval. Alternatively, because the RMF reporting interval is typically in multiples of 5 minutes, you could specify 0 for STATIME and 5 for SYNCVAL to synchronize the reporting times. These values produce synchronized statistics reports with more granularity and better synchronization.

Synchronizing the statistics recording interval across data sharing members and with the RMF reporting interval is helpful because having the DB2 statistics, RMF, and CF data for identical time periods makes the problem analysis more accurate.

**Related concepts**

“Investigating SQL performance with EXPLAIN” on page 421

“Scenario for analyzing concurrency” on page 485

**Related tasks**

“Tuning database buffer pools” on page 56

Chapter 15, “Programming for concurrency,” on page 319

**Related reference**

“The accounting report (long format)” on page 638

---

## Chapter 36. Interpreting DB2 trace output

When you activate a DB2 trace, it produces trace records based on the parameters you specified for the START TRACE command.

**PSPI** Each record identifies one or more significant DB2 events. You can use OMEGAMON to format, print, and interpret DB2 trace output. If you do not have OMEGAMON, or you want to do your own analysis of the trace output, you can use this information and the trace field descriptions that are shipped with DB2. By examining a DB2 trace record, you can determine the type of trace that produced the record (statistics, accounting, audit, performance, monitor, or global) and the event the record reports.

Note that when the trace output indicates a particular release level, 'xx' varies according to the actual release of DB2. **PSPI**

---

### The sections of the trace output

Trace records can be written to SMF or GTF.

**PSPI** In both cases, the record contains up to four basic sections:

- An SMF or GTF writer header section
- A self-defining section
- A product section
- Zero or more data sections

The following figure shows the format of DB2 trace records.

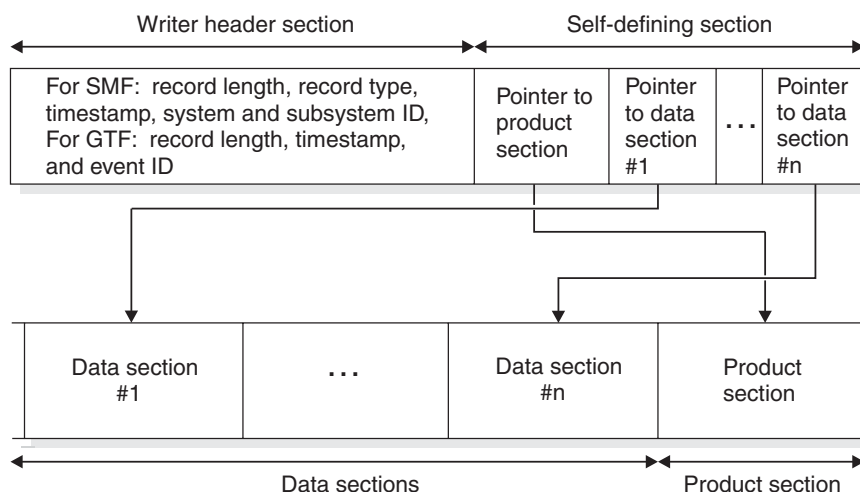



Figure 94. General format of trace records written by DB2

The writer header section begins at the first byte of the record and continues for a fixed length. (The GTF writer header is longer than the SMF writer header.)


The self-defining section follows the writer header section (both GTF and SMF). The first self-defining section always points to a special data section called the

product section. Among other things, the product section contains an instrumentation facility component identifier (IFCID). Descriptions of the records in the data section differ for each IFCID. For a list of records, by IFCID, for each class of a trace, see the description of the START TRACE command in *DB2 Command Reference*.

The product section also contains field QWHSNSDA, which indicates how many self-defining data sections the record contains. You can use this field to keep from trying to access data sections that do not exist. In trying to interpret the trace records, remember that the various keywords you specified when you started the trace determine whether any data is collected. If no data has been collected, field QWHSNSDA shows a data length of zero. 

## SMF writer header section

In SMF, writer headers for statistics records are mapped by macro DSNDQWST, for accounting records by DSNDQWAS, and for performance, audit, and monitor records by DSNDQWSP.

 When these macros are assembled, they include the other macros necessary to map the remainder of the trace records sent to SMF.

The SMF writer header section begins at the first byte of the record. After establishing addressability, you can examine the header fields. The fields are described in Table 165.

*Table 165. Contents of SMF writer header section*

Hex Offset	DSNDQWST	DSNDQWAS	DSNDQWSP	Description
0	SM100LEN	SM101LEN	SM102LEN	Total length of SMF record
2	SM100SGD	SM101SGD	SM102SGD	Segment descriptor
4	SM100FLG	SM101FLG	SM102FLG	System indicator
5	SM100RTY	SM101RTY	SM102RTY	SMF record type: <ul style="list-style-type: none"> <li>Statistics=100(dec)</li> <li>Accounting=101(dec)</li> <li>Monitor=102(dec)</li> <li>Audit=102(dec)</li> <li>Performance=102(dec)</li> </ul>
6	SM100TME	SM101TME	SM102TME	SMF record timestamp, time portion
A	SM100DTE	SM101DTE	SM102DTE	SMF record timestamp, date portion
E	SM100SID	SM101SID	SM102SID	System ID
12	SM100SSI	SM101SSI	SM102SSI	Subsystem ID
16	SM100STF	SM101STF	SM102STF	Reserved
17	SM100RI	SM101RI	SM102RI	Reserved
18	SM100BUF	SM101BUF	SM102BUF	Reserved
1C	SM100END	SM101END	SM102END	End of SMF header

Figure 95 is a sample of the first record of the DB2 performance trace output sent to SMF.

```

000000  A 01240000 B 0E660030 C 9EEC0093 D 018FF3F0 E F9F0E2E2 F D6D70000 G 00000000 H 0000008C
000020  I 00980001 J 0000002C K 005D0001 L 00550053 M 40E2E3C1 N D9E340E3 O D9C1C3C5 P 4040E2E3
000040  Q C1E3405D R C3D3C1E2 S E2404D5C T 405D09D4 U C9C4404D V 5C405D07 W D3C1D540 X 405C405D
000060  Y C1E4E3C8 Z C9C4404D AA 5C405DC9 AB C6C3C9C4 AC 404D5C40 AD 5DC2E4C6 AE E2C9E9C5 AF 404D5C40
000080  AG 5D000000 AH 01000101 AI 01000000 AJ 004C0110 AK 000402xx AL 00B3AB78 AM E2E2D6D7 AN A6E9BACB
0000A0  AO F6485E02 AP 00000003 AQ 00000021 AR 00000001 AS E2C1D5E3 AT C16DE3C5 AU D9C5E2C1 AV 6DD3C1C2
0000C0  AW C4C2F2D5 AX C5E34040 AY D3E4D5C4 AZ F0404040 BA A6E9BACB BB F4570001 BC 004C0200 BD E2E8E2D6
0000E0  BE D7D94040 BF F0F2F34B BG C7C3E2C3 BH D5F6F0F2 BI E2E2D6D7 BJ 40404040 BK 40404040 BL 40404040
000100  BM E2E8E2D6 BN D7D94040 BO 00000000 BP 00000000 BQ 00000000 BR 00000000 BS 00000000
000120  BT 00000000

```

Figure 95. DB2 trace output sent to SMF (printed with DFSERA10 print program of IMS)

Key to Figure 95	Description
<b>A</b> 0124	Record length (field SM102LEN); beginning of SMF writer header section
<b>B</b> 66	Record type (field SM102RTY)
<b>C</b> 0030 9EEC	Time (field SM102TME)
<b>D</b> 0093 018F	Date (field SM102DTE)
<b>E</b> F3F0 F9F0	System ID (field SM102SID)
<b>F</b> E2E2 D6D7	Subsystem ID (field SM102SSI)
<b>G</b>	End of SMF writer header section
<b>H</b> 0000008C	Offset to product section; beginning of self-defining section
<b>I</b> 0098	Length of product section
<b>J</b> 0001	Number of times the product section is repeated
<b>K</b> 0000002C	Offset to first (in this case, only) data section
<b>L</b> 005D	Length of data section
<b>M</b> 0001	Number of times the data section is repeated
<b>N</b> 00550053	Beginning of data section
<b>O</b>	Beginning of product section
<b>P</b> 0004	IFCID (field QWHSIID)
<b>Q</b> 02	Number of self-defining sections in the record (field QWHSNSDA)
<b>R</b> xx	Release indicator number (field QWHSRN); this varies according to the actual level of DB2 you are using.
<b>S</b> E2C1D5E3...	Local location name (16 bytes)
<b>T</b>	End of first record



## GTF writer header section

The length and content of the writer header section differs between SMF and GTF records, but the other sections of the records are the same for SMF and GTF.

**PSPI** The GTF writer header section begins at the first byte of the record. After establishing addressability, you can examine the fields of the header. The writer headers for trace records sent to GTF are always mapped by macro DSNDQWGT. The fields are described in Table 166 on page 624.

Table 166. Contents of GTF writer header section

Offset	Macro DSNDQWGT field	Description
0	QWGTLEN	Length of Record
2		Reserved
4	QWGTAID	Application identifier
5	QWGTFID	Format ID
6	QWGTTIME	Timestamp; you must specify TIME=YES when you start GTF.
14	QWGTEID	Event ID: X'EFB9'
16	QWGTAISC	ASCB address
20	QWGTJOB	Job name
28	QWGTHDRE	Extension to header
28	QWGTDLEN	Length of data section
30	QWGTDSCC	Segment control code
		0=Complete 2=Last 1=First 3=Middle
31	QWGTDZZ2	Reserved
32	QWGTSSID	Subsystem ID
36	QWGTWSEQ	Sequence number
40	QWGTEND	End of GTF header

Figure 96 on page 625 contains trace output sent to GTF.

DFSERA10 - PRINT PROGRAM

```

000000 001A0000 0001FFFF 94B6A6E9 BD6636FA 5C021000 00010000 0000
000000 011C0000 FF00A6E9 C33E28F7 DD03EFB9 00F91400 E2E2D6D7 D4E2E3D9 01000100
000020 E2E2D6D7 00000001 000000A0 00980001 00000038 00680001 0060005E 4DE2E3C1
000040 D9E340E3 D9C1C3C5 404DE2E3 C1E3405D C3D3C1E2 E2404D5C 405DD9D4 C9C4404D
000060 5C405DC4 C5E2E340 4DC7E3C6 405DD7D3 C1D5404D 5C405DC1 E4E3C8C9 C4404D5C
000080 405DC9C6 C3C9C440 4D5C405D C2E4C6E2 C9E9C540 4D5C405D FFFFFFFF 00040101
0000A0 004C0110 000402xx 00B3ADB8 E2E2D6D7 A6E9C33E 28EF4403 00000006 00000001
0000C0 00000001 E2C1D5E3 C16DE3C5 D9C5E2C1 6DD3C1C2 C4C2F2D5 C5E34040 D3E4D5C4
0000E0 F0404040 A6E9C33E 271F0001 004C0200 E2E8E2D6 D7D94040 F0F2F34B C7C3E2C3
000100 D5F6F0F2 E2E2D6D7 40404040 40404040 40404040 E2E8E2D6 D7D94040
000000 00440000 FF00A6E9 C33E2901 1303EFB9 00F91400 E2E2D6D7 D4E2E3D9 00280200
000020 E2E2D6D7 00000001 00000000 00000000 00000000 00000000 00000000 00000000
000040 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000000 011C0000 FF00A6E9 C33E2948 E203EFB9 00F91400 E2E2D6D7 D4E2E3D9 01000100
000020 E2E2D6D7 00000002 000000D8 004C0001 00000090 001C0004 00000100 001C000E
000040 00000288 0018000E 00000590 00400001 000005D0 00740001 00000480 00440001
000060 000003D8 00800001 00000458 00280001 00000644 00480001 000004E4 00AC0001
000080 0000068C 004C0001 000004C4 00200001 D4E2E3D9 00000001 762236F2 00000000
0000A0 59F48900 001E001E 00F91400 C4C2D4F1 00000001 1A789573 00000000 95826100
0000C0 001F001F 00F90E00 C4C9E2E3 00000000 3413C60E 00000000 1C4D0A00 00220022
0000E0 00F90480 C9D9D3D4 00000000 0629E2BC 00000000 145CE000 001D001D 00F91600
000100 E2D4C640 00000046 00000046 00000000 00000000 00000000 00000000 00000000
000000 011C0000 FF00A6E9 C33E294B 1603EFB9 00F91400 E2E2D6D7 D4E2E3D9 01000300
000020 E2E2D6D7 00000002 D9C5E240 00000000 00000000 00000000 00000000 00000000
000040 00000000 C7E3C640 00000001 00000001 00000000 00000000 00000000 00000000
000060 E2D9E540 00000000 00000000 00000000 00000000 00000000 00000000 E2D9F140

```

```

000080 00000156 000000D2 00000036 00000036 00000000 00000004 E2D9F240 00000000
0000A0 00000000 00000000 00000000 00000000 00000000 D6D7F140 00000000 00000000
0000C0 00000000 00000000 00000000 00000000 00000000 D6D7F240 00000000 00000000
0000E0 00000000 00000000 00000000 00000000 D6D7F340 00000000 00000000 00000000
000100 00000000 00000000 D6D7F440 00000000 00000000 00000000 00000000

                                Y
000000 011C0000 FF00A6E9 C33E294D 3C03EFB9 00F91400 E2E2D6D7 D4E2E3D9 01000300
000020 E2E2D6D7 00000002 00000000 00000000 D6D7F540 00000000 00000000 00000000
000040 00000000 00000000 00000000 00000000 D6D7F640 00000000 00000000 00000000
000060 00000000 00000000 D6D7F740 00000000 00000000 00000000 00000000 00000000
000080 00000000 D6D7F840 00000000 00000000 00000000 00000000 00000000 00000000
0000A0 00010000 0000000E 0000000D 00000000 00000000 00000000 00000000 00020000 0000000D
0000C0 0000000D 00000000 00000000 00000000 00030000 00000003 00000003 00000000
0000E0 00000000 00000000 00040000 00000006 00000006 00000000 00000000 00000000
000100 00050000 00000005 00000005 00000000 00000000 00000000 006A0000

                                Y
000000 011C0000 FF00A6E9 C33E294F 6103EFB9 00F91400 E2E2D6D7 D4E2E3D9 01000300
000020 E2E2D6D7 00000002 00000005 00000005 00000000 00000000 00000000 008C0000
000040 00000000 00000000 00000000 00000000 00000000 008D0000 00000000

:
                                Z
000000 00780000 FF00A6E9 C33E2957 D103EFB9 00F91400 E2E2D6D7 D4E2E3D9 005C0200

                                AA
000020 E2E2D6D7 00000002 00000000 004C011A 00010D31 02523038 E2E2D6D7 A6E9C33E
000040 29469A03 0000000E 00000002 00000001 E2C1D5E3 C16DE3C5 D9C5E2C1 6DD3C1C2
000060 40404040 40404040 40404040 40404040 A6E9B6B4 9A2B0001

```

Figure 96. DB2 trace output sent to GTF (spanned records printed with DFSERA10 print program of IMS)

Key to Figure 96	Description
<b>A</b> 011C	Record length (field QWGTLEN); beginning of GTF writer header section
<b>B</b> A6E9 C33E28F7 DD03	Timestamp (field QWGTTIME)
<b>C</b> EFB9	Event ID (field QWGTEID)
<b>D</b> E2E2D6D7 D4E2E3D9	Job name (field QWGTJOBN)
<b>E</b> 0100	Length of data section
<b>F</b> 01	Segment control code (01 = first segment of the first record)
<b>G</b> E2E2D6D7	Subsystem ID (field QWGTSSID)
<b>H</b>	End of GTF writer header section
<b>I</b> 000000A0	Offset to product section; beginning of self-defining section
<b>J</b> 0098	Length of product section
<b>K</b> 0001	Number of times the product section is repeated
<b>L</b> 00000038	Offset to first (in this case, only) data section
<b>M</b> 0068	Length of data section
<b>N</b> 0001	Number of times the data section is repeated
<b>O</b> 0060005E	Beginning of data section
<b>P</b> 004C0110...	Beginning of product section
<b>Q</b> 0004	IFCID (field QWHSIID)
<b>R</b> 02	Number of self-defining sections in the record (field QWHSNSDA)
<b>S</b> xx	Release indicator number (field QWHSRN); this varies according to the actual release level of DB2 you are using.

Key to Figure 96 on page 625	
	Description
<b>T</b> E2C1D5E3...	Local location name (16 bytes)
<b>U</b> 02	Last segment of the first record
<b>V</b>	End of first record
<b>W</b>	Beginning of GTF header for new record
<b>X</b> 01	First segment of a spanned record (QWGTDS01 = QWGTDS01)
<b>Y</b> 03	Middle segment of a spanned record (QWGTDS03 = QWGTDS03)
<b>Z</b> 02	Last segment of a spanned record (QWGTDS02 = QWGTDS02)
<b>AA</b> 004C	Beginning of product section

GTF records are blocked to 256 bytes. Because some of the trace records exceed the GTF limit of 256 bytes, they have been blocked by DB2. Use the following logic to process GTF records:

1. Is the GTF event ID of the record equal to the DB2 ID (that is, does QWGTID = X'x'FB9')?
  - If it is *not* equal, get another record.
  - If it is equal, continue processing.
2. Is the record spanned?
  - If it is spanned (that is, QWGTDS01 = QWGTDS00), test to determine whether it is the first, middle, or last segment of the spanned record.
    - a. If it is the *first* segment (that is, QWGTDS01 = QWGTDS01), save the entire record including the sequence number (QWGTWSEQ) and the subsystem ID (QWGTSSID).
    - b. If it is a *middle* segment (that is, QWGTDS03 = QWGTDS03), find the first segment matching the sequence number (QWGTSEQ) and on the subsystem ID (QWGTSSID). Then move the data portion immediately after the GTF header to the end of the previous segment.
    - c. If it is the *last* segment (that is, QWGTDS02 = QWGTDS02), find the first segment matching the sequence number (QWGTSEQ) and on the subsystem ID (QWGTSSID). Then move the data portion immediately after the GTF header to the end of the previous record.
  - Now process the completed record.
  - If it is **not** spanned, process the record.

The following figure shows the same output after it has been processed by a user-written routine, which follows the logic that was outlined previously.

000000	01380000	FF00A6E9	DCA7E275	1204EFB9	00F91400	E2E2D6D7	D4E2E3D9	011C0000
000020	E2E2D6D7	00000019	000000A0	00980001	00000038	00680001	0060005E	4DE2E3C1
000040	D9E340E3	D9C1C3C5	404DE2E3	C1E3405D	C3D3C1E2	E2404D5C	405DD9D4	C9C4404D
000060	5C405DC4	C5E2E340	4DC7E3C6	405DD7D3	C1D5404D	5C405DC1	E4E3C8C9	C4404D5C
000080	405DC9C6	C3C9C440	4D5C405D	C2E4C6E2	C9E9C540	4D5C405D	00000001	00040101
0000A0	004C0110	000402xx	00B3ADB8	E2E2D6D7	0093018F	11223310	0000000C	00000019
0000C0	00000001	E2C1D5E3	C16DE3C5	D9C5E2C1	6DD3C1C2	C4C2F2D5	C5E34040	D3E4D5C4
0000E0	F0404040	A6E9DCA7	DF960001	004C0200	E2E8E2D6	D7D94040	F0F2F34B	C7C3E2C3
000100	D5F6F0F2	E2E2D6D7	40404040	40404040	40404040	E2E8E2D6	D7D94040	00000000
000120	00000000	00000000	00000000	00000000	00000000	00000000		
	<b>A</b>			<b>B</b>				
000000	07240000	FF00A6E9	DCA8060C	2803EFB9	00F91400	E2E2D6D7	D4E2E3D9	07080000
		<b>C</b>	<b>D</b>		<b>E</b>			
000020	E2E2D6D7	0000001A	000006D8	004C0001	00000090	001C0004	00000100	001C000E
000040	00000288	0018000E	00000590	00400001	000005D0	00740001	00000480	00440001
000060	000003D8	00800001	00000458	00280001	00000644	00480001	000004E4	00AC0001
			<b>F</b>					
000080	0000068C	004C0001	000004C4	00200001	D4E2E3D9	00000003	27BCFDBC	00000000
0000A0	AB000300	001E001E	00F91400	C4C2D4F1	00000001	1DE8AEE2	00000000	DB0CB200
0000C0	001F001F	00F90E00	C4C9E2E3	00000000	4928674B	00000000	217F6000	00220022
0000E0	00F90480	C9D9D3D4	00000000	07165F79	00000000	3C2EF500	001D001D	00F91600
000100	E2D4C640	0000004D	0000004D	00000000	00000000	00000000	00000000	D9C5E240
000120	00000000	00000000	00000000	00000000	00000000	00000000	C7E3C640	00000019
000140	00000019	00000000	00000000	00000000	00000000	E2D9E540	00000000	00000000
000160	00000000	00000000	00000000	00000000	E2D9F140	00000156	000000D2	00000036
000180	00000036	00000000	00000004	E2D9F240	00000092	00000001	00000091	00000091
0001A0	00000000	0000000C	D6D7F140	00000002	00000001	00000001	00000000	00010000
0001C0	20000004	D6D7F240	00000000	00000000	00000000	00000000	00000000	00000000
0001E0	D6D7F340	00000000	00000000	00000000	00000000	00000000	00000000	D6D7F440
000200	00000000	00000000	00000000	00000000	00000000	00000000	D6D7F540	00000000
000220	00000000	00000000	00000000	00000000	00000000	D6D7F640	00000000	00000000
000240	00000000	00000000	00000000	00000000	D6D7F740	00000000	00000000	00000000
000260	00000000	00000000	00000000	D6D7F840	00000000	00000000	00000000	00000000
000280	00000000	00000000	00010000	00000042	00000011	00000030	00000000	00000000
0002A0	00020000	00000041	00000011	00000030	00000000	00000000	00030000	00000003
0002C0	00000003	00000000	00000000	00000000	00040000	0000000C	0000000C	00000000
0002E0	00000000	00000000	00050000	0000000B	0000000A	00000001	00000000	00000000
000300	006A0000	0000000C	0000000B	00000001	00000000	00000000	008C0000	00000000
000320	00000000	00000000	00000000	00000000	008D0000	00000000	00000000	00000000
000340	00000000	00000000	008E0000	00000000	00000000	00000000	00000000	00000000
000360	008F0000	00000000	00000000	00000000	00000000	00000000	00900000	00000000

Figure 97. DB2 trace output sent to GTF (assembled with a user-written routine and printed with DFSERA10 print program of IMS)

```

000380 00000000 00000000 00000000 00000000 00910000 00000000 00000000 00000000
0003A0 00000000 00000000 00920000 00000000 00000000 00000000 00000000 00000000
0003C0 00CA0000 00000041 00000011 00000030 00000000 00000000 00000000 00000000
0003E0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000400 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000420 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000440 00000000 00000000 00000000 00000004 00000000 00000000 000005D4 00000130
000460 0000000D 0000000A 00000029 00000009 000000C3 00000000 00000000 00000000
000480 00000001 0000000C 00000000 04A29740 00000000 00000000 00000001 00000000
0004A0 00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0004C0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0004E0 00000000 E2C1D56D D1D6E2C5 40404040 40404040 00000000 00000002 00000003
000500 00000000 0000004A8 000005C7 00000000 00000001 00000003 00000003 00000000
000520 00000001 00000000 00000001 00000000 00000000 00000000 00000000 00000000
000540 00000002 00000001 00000000 00000000 00000000 00000000 00000000 00000000
000560 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000580 00000000 00000000 00000002 00000000 00000003 00000000 00000003 00000006
0005A0 00000000 00000000 00000000 00000000 00000005 00000003 00000000 00000000
0005C0 00000000 00000003 00000000 00000000 00000000 00000000 00000000 00000000
0005E0 00000000 00000000 0000000C 00000001 00000000 00000007 00000000 00000000
000600 00000000 00000000 00000000 00000001 00000000 00000000 00000000 00000000
000620 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000640 00000000 003C0048 D8E2E2E3 00000035 00000006 00000002 0000009E 0000002B
000660 00000078 00000042 00000048 000000EE 0000001B 0000007B 0000004B 00000000
000680 00000000 00000000 00000000 0093004C D8D1E2E3 00000000 000000FC 0000000E
0006A0 00000000 00000000 0000009D 00000000 00000000 00000016 0000000F 00000018

0006C0 00000000 00000000 00000000 00000000 00000000 004C011A 00010Dxx
0006E0 02523038 E2E2D6D7 0093018F 11223324 00000042 0000001A 00000001 E2C1D5E3
000700 C16DE3C5 D9C5E2C1 6DD3C1C2 40404040 40404040 40404040 40404040 A6E9B6B4
000720 9A2B0001 H

```

Figure 98. DB2 trace output sent to GTF (assembled with a user-written routine and printed with DFSERA10 print program of IMS) continued

Key to Figure 97 on page 627	Description
<b>A</b> 0724	Length of assembled record; beginning of GTF writer header section of second record (field QWGTLEN)
<b>B</b> EFB9	GTF event ID (field QWGTEID)
<b>C</b>	End of GTF writer header section of second record
<b>D</b> 000006D8	Offset to product section
<b>E</b> 00000090	Offset to first data section
<b>F</b> 000004C4	Offset to last data section
<b>G</b> 004C011A	Beginning of product section
<b>H</b>	End of second record

PSPI

## Self-defining section

The self-defining section, which follows the writer header, contains pointers that enable you to find the product and data sections, which contain the actual trace data.

PSPI

Each “pointer” is a descriptor that contains fields, which are:

- A fullword that contains the offset from the beginning of the record to the data section.
- A halfword that contains the length of each item in the data section. If this value is zero, the length of the items are varied.

- A halfword that contains the number of times that the data section is repeated. If the field contains “0”, the data section is not in the record. If it contains a number greater than 1, multiple data items are stored contiguously within that data section. To find the second data item, add the length of the first data item to the address of the first data item (and so forth).

Pointers occur in a fixed order, and their meanings are determined by the IFCID of the record. Different sets of pointers can occur, and each set is described by a separate DSECT. Therefore, to examine the pointers, you must first establish addressability by using the DSECT that provides the appropriate description of the self-defining section. To do this, perform the following steps:

1. Compute the address of the self-defining section.

The self-defining section begins at label “SM100END” for statistics records, “SM101END” for accounting records, and “SM102END” for performance and audit records. It does not matter which mapping DSECT you use because the length of the SMF writer header is always the same.


For GTF, use QWGTEND.

2. Determine the IFCID of the record.

Use the first field in the self-defining section; it contains the offset from the beginning of the record to the product section. The product section contains the IFCID.


The product section is mapped by DSNDQWHS; the IFCID is mapped by QWHSIID.

For statistics records that have IFCID 0001, establish addressability using label “QWS0”; for statistics records having IFCID 0002, establish addressability using label “QWS1”. For accounting records, establish addressability using label “QWA0”. For performance and audit records, establish addressability using label “QWT0”.

After establishing addressability using the appropriate DSECT, use the pointers in the self-defining section to locate the record’s data sections. 

### Reading the self-defining section for same-length data items

If the length of each item in a data section is the same, the self-defining section indicates the length in a halfword that follows the fullword offset. If this value is “0”, the length of the data items varies.

 For more information about variable-length data items, see “Reading the self-defining section for variable-length data items” on page 630.

The relationship between the contents of the self-defining section “pointers” and the items in a data section for same-length data items is shown in The figure below.

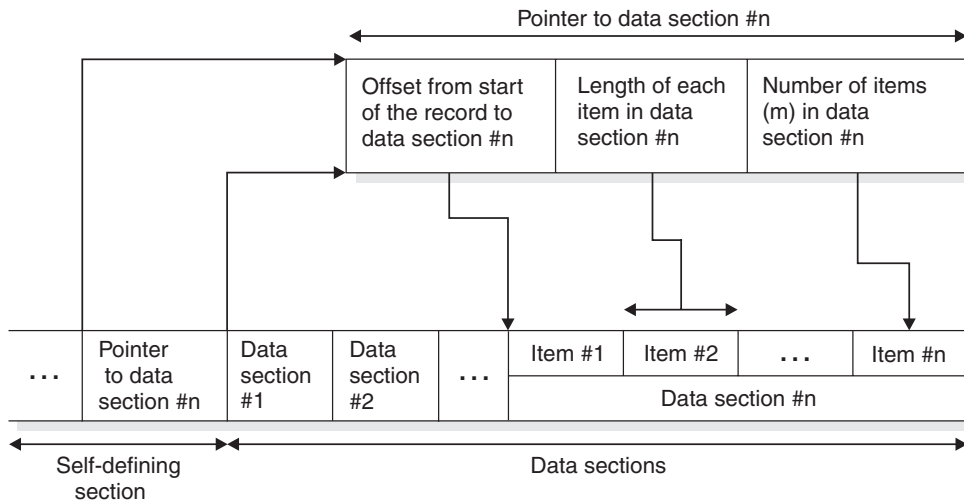


Figure 99. Relationship between self-defining section and data sections for same-length data items

#### PSPI

### Reading the self-defining section for variable-length data items

If the length of each item in a data section varies, the self-defining section indicates the value "0" in the halfword that follows the fullword offset. The length of each variable-length data item is indicated by two bytes that precede each data item.

#### PSPI

The relationship between the contents of the self-defining section "pointers" and the items in a data section for variable-length data items is shown in the following figure.

#### PSPI

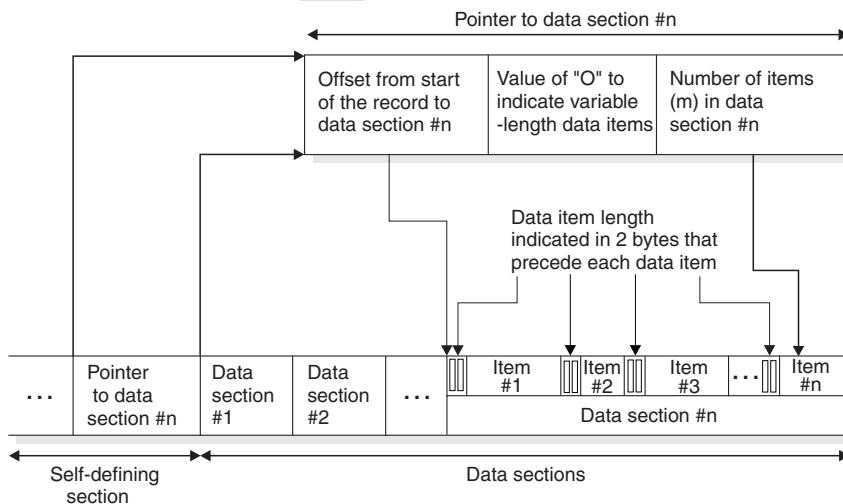


Figure 100. Relationship between self-defining section and data sections for variable-length data items

## Product section

The product section for all record types contains the standard header. The other headers (correlation, CPU, distributed, and data sharing data) might also be present.



The following table shows the contents of the product section standard header.

Table 167. Contents of product section standard header

Hex Offset	Macro DSNDQWHS field	Description
0	QWHSLEN	Length of standard header
2	QWHSTYP	Header type
3	QWHSRMID	RMID
4	QWHSIID	IFCID
6	QWHSRELN	Release number section
6	QWHSNSDA	Number of self-defining sections
7	QWHSRN	DB2release identifier
8	QWHSACE	ACE address
C	QWHSSSID	Subsystem ID
10	QWHSSTCK	Timestamp—STORE CLOCK value assigned by DB2
18	QWHSISEQ	IFCID sequence number
1C	QWHSWSEQ	Destination sequence number
20	QWHSMTN	Active trace number mask
24	QWHSLOCN	Local location Name
34	QWHS LWID	Logical unit of work ID
34	QWHSNID	Network ID
3C	QWHS LUNM	LU name
44	QWHS LUUV	Uniqueness value
4A	QWHS LUCC	Commit count
4C	QWHSFLAG	Flags
4D	QWHSLOCN_Off	If QWHSLOCN is truncated, this is the offset from the beginning of QWHS to QWHSLOCN_LEN. If the value is zero, refer to QWHSLOCN.
4F	QWHS SUBV	The sub-version for the base release.
51	QWHSSEND	End of product section standard header

The following table shows the contents of the product section correlation header.

Table 168. Contents of product section correlation header

Hex Offset	Macro DSNDQWHC field	Description
0	QWHCLEN	Length of correlation header
2	QWHCTYP	Header type
3		Reserved
4	QWHCAID	Authorization ID
C	QWHCCV	Correlation ID
18	QWHCCN	Connection name
20	QWHCPLAN	Plan name

Table 168. Contents of product section correlation header (continued)

Hex Offset	Macro DSNDQWHC field	Description
28	QWHCOPID	Original operator ID
30	QWHCATYP	The type of system that is connecting
34	QWHCTOKN	Trace accounting token field
4A		Reserved
4C	QWHCEUID	User ID of at the workstation for the end user
5C	QWHCEUTX	Transaction name for the end user
7C	QWHCEUWN	Workstation name for the end user
7E	QWHCAID_Off	If QWHCAID is truncated, this is the offset from the beginning of QWHC to QWHCAID_LEN. If the value is zero, refer to QWHCAID.
80	QWHCOPID_Off	If QWHCOPID is truncated, this is the offset from the beginning of QWHC to QWHCAID_LEN. If the value is zero, refer to QWHCOPID.
82	QWHCEUID_Off	If QWHCEUID is truncated, this is the offset from the beginning of QWHC to QWHCEUID_LEN. If the value is zero, refer to QWHCEUID.
84	QWHCTCXT_Off	If QWHCTCXT is truncated, this is the offset from the beginning of QWHC to QWHCTCXT_LEN. If the value is zero, refer to QWHCTCXT.
86	QWHCROLE_Off	If QWHCROLE is truncated, this is the offset from the beginning of QWHC to QWHCROLE_LEN. If the value is zero, refer to QWHCROLE.
8E	QWHCEND	End of product section correlation header

The following table shows the contents of the CPU header.

Table 169. Contents of CPU header

Hex Offset	Macro DSNDQWHU field	Description
0	QWHULEN	Length of CPU header
2	QWHUTYP	Header type
3		Reserved
4	QWHUCPU	CPU time of the currently dispatched execution unit (TCB or SRB). This time includes CPU consumed on an IBM zIIP.
C	QWHUCNT	Count field reserved
E	QWHUEND	End of header

The following table shows the contents of the distributed data header.

*Table 170. Contents of distributed data header*

Hex Offset	Macro DSNDQWHD field	Description
0	QWHDLEN	Length of the distributed header
2	QWHD TYP	Header type
3		Reserved
4	QWHDRQNM	Requester location name
14	QWHD TSTP	Timestamp for DBAT trace record
1C	QWHDSVNM	EXCSAT SRVNAM parameter
2C	QWHDPRID	ACCRDB PRDID parameter
34	QWHDRGNM_Off	If QWHDRQNM is truncated, this is the offset from the beginning of QWHD to QWHDRQNM_LEN. If zero, refer to QWHDRQNM.
36	QWHDSVNM_Off	If QWHDSVNM is truncated, this is the offset from the beginning of QWHD to QWHDSVNM_LEN. If zero, refer to QWHDSVNM
30	QWHDEND	End of distributed header

The following table shows the contents of the trace header.

*Table 171. Contents of trace header*

Hex Offset	Macro DSNDQWHT field	Description
0	QWHTLEN	Length of the trace header
2	QWHTTYP	Header type
3		Reserved
4	QWHTTID	Event ID
6	QWHTTAG	ID specified on DSNWTRC macro
7	QWHTFUNC	Resource manager function code. Default is 0.
8	QWHTEB	Execution block address
C	QWHTPASI	Prior address space ID - EPAR
E	QWHTR14A	Register 14 address space ID
10	QWHTR14	Contents of register 14
14	QWHTR15	Contents of register 15
18	QWHTR0	Contents of register 0
1C	QWHTR1	Contents of register 1
20	QWHTEXU	Address of MVS execution unit
24	QWHTDIM	Number of data items
26	QWHTHASI	Home address space ID
28	QWHTDATA	Address of the data
2C	QWHTFLAG	Flags in the trace list
2E	QWHTDATL	Length of the data list

Table 171. Contents of trace header (continued)

Hex Offset	Macro DSNDQWHT field	Description
30	QWHTEND	End of header

The following table shows the contents of the data sharing header.

Table 172. Contents of data sharing header

Hex Offset	Macro DSNDQWHA field	Description
0	QWHALEN	Length of the data sharing header
2	QWHATYP	Header type
3		Reserved
4	QWHAMEMN	DB2 member name
C	QWHADSGN	DB2 data sharing group name
14	QWHAEND	End of header

Figure 101 on page 635 is a sample accounting trace for a distributed transaction sent to SMF.

```

000000 07000000 1E650059 A8B50104 019FF3F0 F9F0E5F8 F1C10000 00000000 0000061A
000020 00E60001 00000084 01F00001 000003CE 01CC0001 0000059A 00400002 00000376
000040 00580001 00000000 00000000 00000274 01020001 00000000 00000000 00000000
000060 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000080 00000000 BAA5E4D7 8373558E BAA5E4E2 B6DB4F46 00000000 0349EA39 00000000
0000A0 05578AFD 00000000 00000000 00000000 00000000 00000000 40404040 40404040
0000C0 00000000 00000000 00000001 00000002 00000000 006349C0 00000000 005735A0
0000E0 00000000 00000000 00000000 00000000 00000000 00000000 00000006 00000000
000100 00000000 00000000 00000000 00000000 00000000 005536E0 00000000 00000000
000120 00000000 00000000 00000000 00000006 00000000 00000000 00000000 00000000
000140 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000160 00000000 00030004 00000000 00000000 00000000 00000000 00000000 00000000
000180 00000000 03BDA0A1 00000000 0131DD20 0000000C 00000000 0C90ADB5 00000006
0001A0 00000000 00000000 00001816 00000000 00000000 00000000 00000000 00000000
0001C0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0001E0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000200 00432BFA 00000000 00010F80 00000009 7C68DEB2 00000000 033B2BAB 00000000
000220 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000240 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000260 00000000 00000000 00000000 00000000 00000000 5FC4E2D5 F0F8F0F1 F5E2E3D3
000280 C5C3F140 40404040 40404040 40E4E2C9 C2D4E2E8 40E2E8C5 C3F1C4C2 F2C2C1E3
0002A0 C3C84040 40C2C1E3 C3C84040 40E6D5C5 E2E3D5D7 40404040 40E2E8E2 C1C4D440
0002C0 40D7D3D5 C1D7D7D3 F8E4E2C5 D97EE2E8 E2C1C4D4 00000000 00000000 00000000
0002E0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000300 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000320 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000340 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000360 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000380 00000000 00000000 00000000 00010000 00000000 00000000 00000000 00000000
0003A0 00000000 00000000 00000000 00170000 00100000 00000000 00040000 00000000
0003C0 00060000 00000000 00000000 00002095 01CCD8E7 E2E30000 00020000 00000000
0003E0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000400 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

```

```

000420 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000440 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
                                S
000460 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000480 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0004A0 00000000 00000000 00000000 00000000 00000000 00000000 00030000 00000000
0004C0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0004E0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000500 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000520 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
                                T
000540 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000560 00000000 00000000 00030000 00000000 00000000 00000000 00000000 00000000
                                U
000580 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0005A0 00080000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
                                V
0005C0 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00640000
0005E0 000A0000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
000600 00000000 00000000 00000000 00000000 00000000 00000000 00000052 011A0003
000620 0D8106E0 D7B8E5F8 F1C1BAA5 E4E2B6E5 85A60000 00180000 00230000 0004E2E3
000640 D3C5C3F1 40404040 40404040 4040E4E2 C9C2D4E2 E840E2E8 C5C3F1C4 C2F2BAA5
                                W
000660 E4D78307 00010000 00000000 00940200 E2E8E2C1 C4D44040 E6D5C5E2 E3D5D740
000680 40404040 C2C1E3C3 C8404040 D7D3D5C1 D7D7D3F8 E2E8E2C1 C4D44040 00000001
0006A0 00000000 00000000 00000000 00000000 00000000 00000000 40404040 40404040
0006C0 40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040

```

Figure 101. DB2 distributed data trace output sent to SMF (printed with IMS DFSERA10 print program). This example has one accounting record (IFCID 0003) from the server site (SILICON\_VALLEY\_LAB). DSNDQWA0 maps the self-defining section for IFCID 0003.

Key to Figure 101	Description
<b>A</b> 0000061A	Offset to product section; beginning of self-defining section
<b>B</b> 00E6	Length of product section
<b>C</b> 0001	Number of times product section is repeated
<b>D</b> 00000084	Offset to accounting section
<b>E</b> 01F0	Length of accounting section
<b>F</b> 0001	Number of times accounting section is repeated
<b>G</b> 000003CE	Offset to SQL accounting section
<b>H</b> 0000059A	Offset to buffer manager accounting section
<b>I</b> 00000376	Offset to locking accounting section
<b>J</b> 00000000	Offset to distributed section
<b>K</b> 00000274	Offset to MVS/DDF accounting section
<b>L</b> 00000000	Offset to IFI accounting section
<b>M</b> 00000000	Offset to package/DBRM accounting section
<b>N</b> 00000000...	Beginning of accounting section (DSNDQWAC)
<b>O</b> 00000000...	Beginning of distributed section (DSNDQLAC)
<b>P</b> 00000000...	Beginning of MVS/DDF accounting section (DSNDQMDA)
<b>Q</b> 00000000...	Beginning of package/DBRM accounting section (DSNDQPAC)
<b>R</b> 00000000...	Beginning of locking accounting section (DSNDQTXA)
<b>S</b> 00000000...	Beginning of SQL accounting section (DSNDQXST)
<b>T</b> 00000000...	Beginning of buffer manager accounting section (DSNDQBAC)

Key to Figure 101 on page 635	Description
<b>U</b> 00000000...	Beginning of product section (DSNDQWHS); beginning of standard header
<b>V</b> 00640000...	Beginning of correlation header (DSNDQWHC)
<b>W</b> 00000094...	Beginning of distributed header (DSNDQWHD)

PSPI

## Trace field descriptions

You can find descriptions of trace records in *prefix.SDSNIVPD(DSNWMSG)*.

PSPI If you intend to write a program to read DB2 trace records, use the assembler mapping macros in *prefix.SDSNMACS*. The macros have member names like 'DSNDQW

You can use the TSO or ISPF browse function to look at the field descriptions in the trace record mapping macros online, even when DB2 is down. If you prefer to look at the descriptions in printed form, you can use ISPF to print a listing of the data set. PSPI

---

## Chapter 37. Reading accounting reports from OMEGAMON

You can obtain OMEGAMON reports of accounting data in long or short format and in various levels of detail.

The examples of OMEGAMON reports in this information are based on the default formats, which might have been modified for your installation. Furthermore, the OMEGAMON reports have been reformatted or modified for this publication. Refer to *OMEGAMON Report Reference* for an exact description of each report. You can also time results for nested activities such as triggers, stored procedures, and user-defined functions.

### Related concepts

Chapter 10, “Accounting for nested activities,” on page 113

---

### The accounting report (short format)

The OMEGAMON accounting report, short format, allows you to monitor application distribution, resources used by each major group of applications, and the average DB2 elapsed time for each major group.

The report summarizes application-related performance data and orders the data by selected DB2 identifiers.

Monitoring application distribution helps you to identify the most frequently used transactions or queries, and is intended to cover the 20% of the transactions or queries that represent about 80% of the total work load. The TOP list function of OMEGAMON lets you identify the report entries that represent the largest user of a given resource.

To get an overall picture of the system work load, you can use the OMEGAMON GROUP command to group several DB2 plans together.

You can use the accounting report, short format, to:

- Monitor the effect of each application or group on the total work load
- Monitor, in each application or group:
  - DB2 response time (elapsed time)
  - Resources used (processor, I/Os)
  - Lock suspensions
  - Application changes (SQL used)
  - Usage of packages and DBRMs
  - Processor, I/O wait, and lock wait time for each package

An accounting report in the short format can list results in order by package. Thus you can summarize package or DBRM activity independently of the plan under which the package or DBRM executed.

Only class 1 of the accounting trace is needed for a report of information by plan. Classes 2 and 3 are recommended for additional information. Classes 7 and 8 are needed to give information by package or DBRM.

## The accounting report (long format)

Use the OMEGAMON accounting report, long format, when an application seems to have a problem, and you need very detailed analysis.

Use the OMEGAMON accounting report, short format, to monitor your applications. For a partial example of the long format report, see the figure below.

AVERAGE	APPL (CL.1)	DB2 (CL.2)	IFI (CL.5)	CLASS 3 SUSPENSIONS	AVERAGE TIME	AV.EVENT	HIGHLIGHTS
ELAPSED TIME	0.015167	0.004024	N/P	LOCK/LATCH(DB2+IRLM) <b>A</b>	0.000746	0.18	#OCCURRENCES : 151203
NONNESTED	0.015167	0.004024	N/A	SYNCHRON. I/O <b>B</b>	0.001382	2.48	#ALLIEDS : 0
STORED PROC	0.000000	0.000000	N/A	DATABASE I/O	0.000850	2.02	#ALLIEDS DISTRIB: 0
UDF	0.000000	0.000000	N/A	LOG WRITE I/O	0.000532	0.46	#DBATS : 151203
TRIGGER	0.000000	0.000000	N/A	OTHER READ I/O <b>D</b>	0.000000	0.00	#DBATS DISTRIB. : 0
				OTHER WRTE I/O <b>E</b>	0.000016	0.01	#NO PROGRAM DATA: 7
CP CPU TIME <b>C</b>	0.002643	0.001721	N/P	SER.TASK SWITCH <b>F</b>	0.000021	0.00	#NORMAL TERMINAT: 151203
AGENT	0.002643	0.001721	N/A	UPDATE COMMIT	0.000000	0.00	#ABNORMAL TERMIN: 0
NONNESTED	0.002643	0.001721	N/P	OPEN/CLOSE	0.000000	0.00	#CP/X PARALLEL. : 0
STORED PRC	0.000000	0.000000	N/A	SYSLGRNG REC	0.000000	0.00	#IO PARALLELISM : 0
UDF	0.000000	0.000000	N/A	EXT/DEL/DEF	0.000021	0.00	#INCREMENT. BIND: 0
TRIGGER	0.000000	0.000000	N/A	OTHER SERVICE	0.000000	0.00	#COMMITTS : 151218
PAR.TASKS	0.000000	0.000000	N/A	ARC.LOG(QUIES) <b>G</b>	0.000000	0.00	#ROLLBACKS : 0
				LOG READ <b>H</b>	0.000000	0.00	#SVPT REQUESTS : 0
IIPCP CPU <b>L</b>	0.000000	N/A	N/A	DRAIN LOCK <b>I</b>	0.000000	0.00	#SVPT RELEASE : 0
				CLAIM RELEASE <b>J</b>	0.000000	0.00	#SVPT ROLLBACK : 0
IIP CPU TIME <b>M</b>	0.000000	0.000000	N/A	PAGE LATCH <b>K</b>	0.000000	0.00	MAX SQL CASC LVL: 0
STORED PROC	0.000000	0.000000	N/A	NOTIFY MSGS	0.000000	0.00	UPDATE/COMMIT : 6.82
				GLOBAL CONTENTION	0.000000	0.00	SYNCH I/O AVG. : 0.000557
SUSPEND TIME	0.000000	0.002164	N/A	COMMIT PH1 WRITE I/O	0.000000	0.00	
AGENT	N/A	0.002164	N/A	ASYNCH CF REQUESTS	0.000000	0.00	
PAR.TASKS	N/A	0.000000	N/A	TCP/IP LOB	0.000000	0.00	
STORED PROC	0.000000	N/A	N/A	TOTAL CLASS 3	0.002164	2.67	
UDF	0.000000	N/A	N/A				
NOT ACCOUNT. <b>N</b>	N/A	0.000138	N/A				
DB2 ENT/EXIT	N/A	71.84	N/A				
EN/EX-STPROC	N/A	0.00	N/A				
EN/EX-UDF	N/A	0.00	N/A				
DCAPT.DESCR.	N/A	N/A	N/P				
LOG EXTRACT.	N/A	N/A	N/P				

SQL DML	AVERAGE	TOTAL	SQL DCL	TOTAL	SQL DDL	CREATE	DROP	ALTER	LOCKING	AVERAGE	TOTAL
SELECT	0.00	0	LOCK TABLE	0	TABLE	0	0	0	TIMEOUTS	0.00	0
INSERT	2.88	436038	GRANT	0	CRT TTABLE	0	N/A	N/A	DEADLOCKS	0.00	0
UPDATE	3.73	564533	REVOKE	0	DCL TTABLE	0	N/A	N/A	ESCAL.(SHARED)	0.00	0
MERGE	0.00	0	SET CURR.SQLID	0	AUX TABLE	0	N/A	N/A	ESCAL.(EXCLUS)	0.00	0
DELETE	0.21	31260	SET HOST VAR.	0	INDEX	0	0	0	MAX PG/ROW LOCKS HELD	4.96	49
			SET CUR.DEGREE	0	TABLESPACE	0	0	0	LOCK REQUEST	17.82	2694651
DESCRIBE	0.00	0	SET RULES	0	DATABASE	0	0	0	UNLOCK REQUEST	1.77	268170
DESC.TBL	0.00	0	SET CURR.PATH	0	STOGROUP	0	0	0	QUERY REQUEST	0.00	0
PREPARE	14.46	2186391	SET CURR.PREC.	0	SYNONYM	0	0	N/A	CHANGE REQUEST	2.87	433788
OPEN	8.06	1219330	CONNECT TYPE 1	0	VIEW	0	0	N/A	OTHER REQUEST	0.00	0
FETCH	8.06	1219330	CONNECT TYPE 2	0	ALIAS	0	0	N/A	TOTAL SUSPENSIONS	0.06	8700
CLOSE	2.89	437058	SET CONNECTION	0	PACKAGE	N/A	0	N/A	LOCK SUSPENSIONS	0.05	7770
			RELEASE	0	PROCEDURE	0	0	0	IRLM LATCH SUSPENS.	0.01	930
DML-ALL	40.30	6093940	CALL	0	FUNCTION	0	0	0	OTHER SUSPENS.	0.00	0
			ASSOC LOCATORS	0	TRIGGER	0	0	N/A			
			ALLOC CURSOR	0	DIST TYPE	0	0	N/A			
			HOLD LOCATOR	0	SEQUENCE	0	0	0			
			FREE LOCATOR	0							
			DCL-ALL	0	TOTAL	0	0	0			
					RENAME TBL	0					

Figure 102. Partial accounting report (long format)

In analyzing a detailed accounting report, consider the following components of response time. (Fields of the report that are referred to are labeled in the figure above.)

### Class 1 elapsed time

Compare this with the CICS, IMS, WebSphere, or distributed application transit times:

- In CICS, you can use CICS Performance Analyzer to find the attach and detach times; use this time as the transit time.
- In IMS, use the PROGRAM EXECUTION time reported in IMS Performance Analyzer.

Differences between the CICS, IMS, WebSphere, or distributed application times and the DB2 accounting times arise mainly because the DB2 times do not include:

- Time before the first SQL statement, which for a distributed application includes the inbound network delivery time
- DB2 create thread
- DB2 terminate thread
- For a distributed application, the time to deliver the response to a commit if database access thread pooling is used

Differences can also arise from thread reuse in CICS, IMS, WebSphere, or distributed application processing or through multiple commits in CICS. If the class 1 elapsed time is significantly less than the CICS or IMS time, check the report from Tivoli Decision Support for z/OS, IMS Performance Analyzer, or an equivalent reporting tool to find out why. Elapsed time can occur:

- In DB2, during sign-on, create, or terminate thread
- Outside DB2, during CICS or IMS processing

### Not-in-DB2 time

This time is the calculated difference between the class 1 and the class 2 elapsed time. This value is the amount of time spent outside of DB2, but within the DB2 accounting interval. A lengthy time can be caused by thread reuse, which can increase class 1 elapsed time, or a problem in the application program, CICS, IMS, or the overall system.

For a distributed application, not-in-DB2 time is calculated with this formula:

$$\text{Not\_in\_DB2 time} = A - (B + C + (D - E))$$

Where the variables have the following values:

- |   |   |
|---|---|
| A | Class 1 elapsed time  |
| B | Class 2 non-nested elapsed time   |
| C | Class 1 non-nested elapsed time of any stored procedures, user-defined functions, or triggers |
| D | Class 1 non-nested CPU time   |
| E | Class 2 non-nested CPU time   |

The calculated not-in-DB2 time might be zero. Furthermore, this time calculation is only an estimate. A primary factor that is not included in the equation is the amount of time that requests wait for CPU resources while executing within the DDF address space. To determine how long requests wait for CPU resources, look at the NOT ACCOUNT field. The NOT ACCOUNT field shows the time that requests wait for CPU resources while a distributed task is inside DB2.

### Lock/latch suspension time

This shows contention for DB2 and IRLM resources. If contention is high, check the locking summary section of the report, and then proceed with the locking

reports. For more information, see “Scenario for analyzing concurrency” on page 485.

In the OMEGAMON accounting report, see the field LOCK/LATCH(DB2+IRLM) ( **A** ).

## Synchronous I/O suspension time

This is the total application wait time for synchronous I/Os. It is the total of database I/O and log write I/O. In the OMEGAMON accounting report, check the number reported for SYNCHRON. I/O ( **B** ).

If the number of synchronous read or write I/Os is higher than expected, check for:

- A change in the access path to data. First, check getpage count. If it has significantly increased, then a change in access path must have occurred. However, if the getpage count remained about the same, but the number of I/Os increased significantly, the problem is not an access path change. If you have data from accounting trace class 8, the number of synchronous and asynchronous read I/Os is available for individual packages. Determine which package or packages have unacceptable counts for synchronous and asynchronous read I/Os. Activate the necessary performance trace classes for the OMEGAMON SQL activity reports to identify the SQL statement or cursor that is causing the problem. If you suspect that your application has an access path problem, consider using IBM Optimization Service Center for DB2 for z/OS and DB2 EXPLAIN.
- A lower than expected buffer pool hit ratio. You can improve the ratio by tuning the buffer pool. Look at the number of synchronous reads in the buffer pool that are associated with the plan, and look at the related buffer pool hit ratio. If the buffer pool size and the buffer pool hit ratio for random reads is small, consider increasing the buffer pool size. By increasing the buffer pool size, you might reduce the amount of synchronous database I/O and reduce the synchronous I/O suspension time.
- A change to the catalog statistics, or statistics that no longer match the data.
- Changes in the application. Check the “SQL ACTIVITY” section and compare with previous data. Some inserts might have changed the amount of data. Also, check the names of the packages or DBRMs being executed to determine if the pattern of programs being executed has changed.
- Pages might be out of order so that sequential detection is not used, or data might have been moved to other pages. Run the REORG utility in these situations.
- A system-wide problem in the database buffer pool. You can also use OMEGAMON DB2 Buffer Pool Analyzer (DB2 BPA) or OMEGAMON DB2 Performance Expert, which contains the function of DB2 BPA, to manage and optimize buffer pool activity.
- A RID pool failure.
- A system-wide problem in the EDM pool.

If I/O time is greater than expected, and not caused by more read I/Os, check for:

- Synchronous write I/Os. (You can also use OMEGAMON DB2 Buffer Pool Analyzer (DB2 BPA) or OMEGAMON DB2 Performance Expert, which contains the function of DB2 BPA, to manage and optimize buffer pool activity)

- I/O contention. In general, each synchronous read I/O typically takes from 5 to 20 milliseconds, depending on the disk device. This estimate assumes that no prefetch or deferred write I/Os exist on the same device as the synchronous I/Os.
- An increase in the number of users, or an increase in the amount of data. Also, drastic changes to the distribution of the data can cause the problem.

## Processor resource consumption

The problem might be caused by DB2 or IRLM traces, a change in access paths, or an increase in the number of users. In the OMEGAMON accounting report, DB2 processor resource consumption is indicated in the field for class 2 CPU TIME ( **C** ). If a System z™ Integrated Information Processor (zIIP) is used, any CPU that is redirected to the zIIP processor is not included in the CP CPU TIME field.

## Processing redirected to zIIP

Certain eligible processing tasks might be eligible to be redirected to a System z Integrated Information Processor (zIIP) including:

- DRDA TCP/IP workload
- Parallel query child tasks
- Build index processing for LOAD, REBUILD, and REORG utilities

Any CPU time consumed on the zIIP is shown in the IIP CPU TIME ( **M** ) field, and is not included in the CP CPU TIME ( **C** ). IIPCP CPU TIME ( **L** ) indicates that zIIP-eligible processing that was performed under a normal CP and the CPU time is included in the CP CPU TIME.

When zIIP processor is used, a high value for IIPCP CPU TIME indicates that zIIP eligible processing was redirected back to normal CP processing because the zIIP processor was busy. In this situation, you might consider configuring additional zIIP processors.

When zIIP processor is not used, which is indicated by zero value for IIP CPU TIME, a non-zero value for IIPCP CPU TIME indicates the projected estimate of CPU time that could be redirected to a zIIP processor. The projected benefit of redirecting processing to a zIIP is shown under the following conditions:

- A zIIP processor is configured in the LPAR, but is offline
- zIIP processor is not configured, but the PROJECTCPU parameter is set to YES in the SYS1.PARMLIB(IEAOPTxx) member in z/OS.

## Other read suspensions

The accumulated wait time for read I/O done under a thread other than this one. It includes time for:

- Sequential prefetch
- List prefetch
- Dynamic prefetch
- Synchronous read I/O performed by a thread other than the one being reported

Generally, an asynchronous read I/O for sequential prefetch or dynamic prefetch takes about 0.035 to 0.4 milliseconds per page. List prefetch takes about the same time on the low end but can, on the high end, take 5 to 10 milliseconds per page.

In the OMEGAMON accounting report, other read suspensions are reported in the field OTHER READ I/O ( **D** ).

### Other write suspensions

The accumulated wait time for write I/O done under a thread other than this one. It includes time for:

- Asynchronous write I/O
- Synchronous write I/O performed by a thread other than the one being reported

As a guideline, an asynchronous write I/O takes 0.1 to 2 milliseconds per page.

In the OMEGAMON accounting report, other write suspensions are reported in the field OTHER WRTE I/O ( **E** ).

### Service task suspensions

The accumulated wait time from switching synchronous execution units, by which DB2 switches from one execution unit to another. The most common contributors to service task suspensions are:

- Wait for phase 2 commit processing for updates, inserts, and deletes (UPDATE COMMIT). You can reduce this wait time by allocating the DB2 primary log on a faster disk. You can also help to reduce the wait time by reducing the number of commits per unit of work.
- Wait for OPEN/CLOSE service task (including HSM recall). You can minimize this wait time by using two strategies. If DSMAX is frequently reached, increase DSMAX. If DSMAX is not frequently reached, change CLOSE YES to CLOSE NO on data sets that are used by critical applications.
- Wait for SYSLGRNG recording service task.
- Wait for data set extend/delete/define service task (EXT/DEL/DEF). You can minimize this wait time by defining larger primary and secondary disk space allocation for the table space.
- Wait for other service tasks (OTHER SERVICE). On DB2 requester systems, the amount of time waiting for requests to be returned from either VTAM or TCP/IP, including time spent on the network and time spent handling the request in the target or server systems (OTHER SERVICE).

In the OMEGAMON accounting report, the total of this information is reported in the field SER.TASK SWTCH ( **F** ). The field is the total of the five fields that follow it. If several types of suspensions overlap, the sum of their wait times can exceed the total clock time that DB2 spends waiting. Therefore, when service task suspensions overlap other types, the wait time for the other types of suspensions is not counted.

### Archive log mode (QUIESCE)

The accumulated time the thread was suspended while processing ARCHIVE LOG MODE(QUIESCE). In the OMEGAMON accounting report, this information is reported in the field ARCH.LOG (QUIES) ( **G** ).

### Archive log read suspension

This is the accumulated wait time the thread was suspended while waiting for a read from an archive log on tape. In the OMEGAMON accounting report, this information is reported in the field ARCHIVE LOG READ ( **H** ).

## Drain lock suspension

The accumulated wait time the thread was suspended while waiting for a drain lock. If this value is high, check the installation options that relate to wait times, and consider running the OMEGAMON locking reports for additional detail. In the OMEGAMON accounting report, this information is reported in the field DRAIN LOCK ( **I** ).

## Claim release suspension

The accumulated wait time the drainer was suspended while waiting for all claim holders to release the object. If this value is high, check the installation options for wait times, and consider running the OMEGAMON locking reports for additional details.

In the OMEGAMON accounting report, this information is reported in the field CLAIM RELEASE ( **J** ).

## Page latch suspension

This field shows the accumulated wait time because of page latch contention. As an example, when the RUNSTATS and COPY utilities are run with the SHRLEVEL(CHANGE) option, they use a page latch to serialize the collection of statistics or the copying of a page. The page latch is a short duration “lock”. If this value is high, the OMEGAMON locking reports can provide additional data to help you determine which object is the source of the contention.

In a data sharing environment, high page latch contention could occur in a multithreaded application that runs on multiple members and requires many inserts. The OMEGAMON lock suspension report shows this suspension for page latch contention in the “other” category. If the suspension is on the index leaf page, use one of the following strategies:

- Make the inserts random
- Drop the index
- Perform the inserts from a single member

If the page latch suspension is on a space map page, use the member cluster option for the table space.

In the OMEGAMON accounting report, this information is reported in the field PAGE LATCH ( **K** ).

## Not-accounted-for DB2 time

Usually the DB2 Class 2 Not Accounted time is very small or negligible. It represents time that DB2 is unable to account for. The DB2 accounting class 2 elapsed time that is not recorded as class 2 CPU time or class 3 suspensions. If you see significant DB2 Class 2 Not Accounted time, it could be because one of the following reasons:

- Too much detailed online tracing, or problems with some vendor performance monitors. This situation is usually the primary cause of high not-accounted-for time on systems that are not CPU-constrained.
- In a non-data sharing environment, running in a very high CPU utilization environment and waiting for CPU cycles, especially with lower dispatching

priority in Work Load Manager goal mode. A non-dedicated LPAR can be interrupted by another LPAR and not be dispatched on a processor for some time.

- In a non-data sharing environment, running in a high MVS paging environment and waiting for storage allocation; DB2 gets swapped out and has to wait for a processor.
- Waiting for return from requests to be returned from VTAM or TCP/IP
- Reading the instrumentation facility interface (IFI) log.
- In very I/O intensive environments, the Media Manager might be running out of request blocks.
- Waiting for package accounting.
- A PREPARE statement which is not found in the dynamic statement cache.
- Data set open contention related to PCLOSET being too small.
- Gathering RMF interval data set statistics.
- DB2 internal suspend and resume looping when several threads are waiting for the same resource. This case is very rare.
- Time spent waiting for parallel tasks to complete (when query parallelism is used for the query)

In the OMEGAMON accounting report, this information is reported in the field NOT ACCOUNT ( **N** ).

#### **Related concepts**

“Investigating SQL performance with EXPLAIN” on page 421

“EDM storage” on page 70

#### **Related tasks**

“Using OMEGAMON to monitor buffer pool statistics” on page 481

“Increasing RID pool size” on page 75

“Monitoring I/O activity of data sets” on page 479

“Setting installation options for wait times” on page 348

---

## Chapter 38. Interpreting records returned by IFI

This information describes the format of the records returned by IFI as a result of READA, READS, and COMMAND requests.

---

### Trace data record format

Trace records that are returned from READA and READS requests contain the following sections:

**PSPI**

- A writer header that reports the length of the entire record, whether the record was in the first, middle, or last section of data, and other specific information for the writer.
  - The writer header for IFI is mapped by DSNDQWIW or the header QW0306OF for IFCID 306 requests. See the mapping macros in *prefix.SDSNMACS* for the formats.
  - In SMF, writer headers for statistics records are mapped by macro DSNDQWST, for accounting records by DSNDQWAS, and for performance, audit, and monitor records by DSNDQWSP. When these macros are assembled, they include the other macros necessary to map the remainder of the trace records sent to SMF. The length and content of the writer header section differs between SMF and GTF records, but the other sections of the records are the same for SMF and GTF.
  - The GTF writer header section begins at the first byte of the record. After establishing addressability, you can examine the fields of the header. The writer headers for trace records sent to GTF are always mapped by macro DSNDQWGT. The fields are described in Table 166 on page 624.
- A self-defining section
- A product section containing specific DB2 information based on the active trace. The product section for all record types contains the standard header, DSNDQWHS. The other headers (correlation, CPU, distributed, and data sharing data) might also be present.
  - DSNDQWHC**  
Product section correlation header
  - DSNDQWHD**  
Distributed data header
  - DSNDQWHT**  
Trace header
  - DSNDQWHA**  
Data sharing header
  - DSNWMSGs.**  
Descriptive text for all IFCID records
- Data areas containing the actual recorded data are mapped by multiple mapping macros described in *prefix.SDSNMACS*.

The following figure shows the return area after a READS request successfully executed.

DFSERA10 - PRINT PROGRAM

Figure 103. Example of IFI return area after READS request (IFCID 106). This output was assembled by a user-written routine and printed with the DFSERA10 print program of IMS.

Figure label	Description
<b>A</b> 05A8	Length of record. The next two bytes are reserved.
<b>B</b> 00000510	Offset to product section standard header.
<b>C</b> 00000054	Offset to first data section.
<b>D</b> 80000018	Beginning of first data section.
<b>E</b> 004C011A	Beginning of product section standard header.
<b>F</b> 006A	IFCID (decimal 106).

## Command record format

The record that is returned from a command request can contain none or many message text segments.

**PSPI** Each segment is a varying-length message (LLZZ, where LL is the 2-byte length and ZZ is a 2-byte reserved area) followed by message text. The IFCA's IFCABM field contains the total number of bytes moved.

The following figure shows the return area after a `START TRACE` command successfully executes.


DFSERA10 - PRINT PROGRAM

•  
•  
•

	A	B	C	D				
000000	007E0000	0000007A	003C0001	C4E2D5E6	F1F3F0C9	406F40D4	D6D540E3	D9C1C3C5
000020	40E2E3C1	D9E3C5C4	6B40C1E2	E2C9C7D5	C5C440E3	D9C1C3C5	40D5E4D4	C2C5D940
		E	F					
000040	F0F24015	003A0001	C4E2D5F9	F0F2F2C9	406F40C4	E2D5E6E5	C3D4F140	7D60E2E3
000060	C1D9E340	E3D9C1C3	C57D40D5	D6D9D4C1	D340C3D6	D4D7D3C5	E3C9D6D5	4015

Figure 104. Example of IFI return area after a START TRACE command. This output was assembled with a user-written routine and printed with DFSERA10 program of IMS

Figure label	Description
<b>A</b> 007E0000	Field entered by print program
<b>B</b> 0000007A	Length of return area
<b>C</b> 003C	Length of record (003C). The next two bytes are reserved.
<b>D</b> C4E2D5E6	Beginning of first message
<b>E</b> 003A	Length of record. The next two bytes are reserved.
<b>F</b> C4E2D5F9	Beginning of second message

The IFCABM field in the IFCA would indicate that X'00000076' ( **C** + **E** ) bytes have been moved to the return area. 



---

## Chapter 39. Interpreting data access

You can use the values captured in EXPLAIN tables, and instances of PLAN\_TABLE in particular, to analyze how DB2 accesses data when it processes a particular SQL statement.

---

### EXPLAIN tables

*EXPLAIN tables* contain information about statements and functions that run on DB2. Certain optimization tools create and maintain EXPLAIN tables to hold performance data about SQL statements that run on your DB2 subsystem.

### PLAN\_TABLE columns

The plan table, PLAN\_TABLE, contains information about access paths that is collected from the results of EXPLAIN statements.

**PSPI** Your subsystem or data sharing group can contain more than one of these tables, including a table with the qualifier SYSIBM, a table with the qualifier DB2OSCA, and additional tables that are qualified by user IDs.

The following table shows the descriptions of the columns in PLAN\_TABLE.

*Table 173. Descriptions of columns in PLAN\_TABLE*

Column name	Description
QUERYNO	A number that identifies the statement that is being explained. For a row that is produced by an EXPLAIN statement, specify the number in the QUERYNO clause. For a row that is produced by non-EXPLAIN statements, specify the number by using the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE, MERGE, and DELETE statement syntax. Otherwise, DB2 assigns a number that is based on the line number of the SQL statement in the source program.  When the values of QUERYNO are based on the statement number in the source program, values that exceed 32767 are reported as 0. However, in a very long program, the value is not guaranteed to be unique. If QUERYNO is not unique, the value of TIMESTAMP is unique.
QBLOCKNO	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.
APPLNAME	The name of the application plan for the row. Applies only to embedded EXPLAIN statements that are executed from a plan or to statements that are explained when binding a plan. A blank indicates that the column is not applicable.
PROGNAME	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. A blank indicates that the column is not applicable.
PLANNO	The number of the step in which the query that is indicated in QBLOCKNO was processed. This column indicates the order in which the steps were executed.

Table 173. Descriptions of columns in PLAN\_TABLE (continued)

Column name	Description
METHOD	<p>A number that indicates the join method that is used for the step:</p> <p><b>0</b> The table in this step is the first table that is accessed, a continuation of a previous table that was accessed, or a table that is not used.</p> <p><b>1</b> A nested loop join is used. For each row of the current composite table, matching rows of a new table are found and joined.</p> <p><b>2</b> A merge scan join is used. The current composite table and the new table are scanned in the order of the join columns, and matching rows are joined.</p> <p><b>3</b> Sorts are needed by ORDER BY, GROUP BY, SELECT DISTINCT, UNION, INTERSECT, EXCEPT, a quantified predicate, or an IN predicate. This step does not access a new table.</p> <p><b>4</b> A hybrid join was used. The current composite table is scanned in the order of the join-column rows of the new table. The new table is accessed using list prefetch.</p>
CREATOR	The creator of the new table that is accessed in this step, blank if METHOD is 3.
TNAME	<p>The name of a table, materialized query table, created or declared temporary table, materialized view, or materialized table expression. The value is blank if METHOD is 3. The column can also contain the name of a table in the form DSNWFQB(<i>qblockno</i>).</p> <p>DSNWFQB(<i>qblockno</i>) is used to represent the intermediate result of a UNION ALL, INTERSECT ALL, EXCEPT ALL, or an outer join that is materialized. If a view is merged, the name of the view does not appear. DSN_DIM_TBLX(<i>qblockno</i>) is used to the represent the work file of a star join dimension table.</p>
TABNO	Values are for IBM use only.

Table 173. Descriptions of columns in PLAN\_TABLE (continued)

Column name	Description
ACCESSTYPE	<p>The method of accessing the new table:</p> <p><b>DI</b> By an intersection of multiple DOCID lists to return the final DOCID list</p> <p><b>DU</b> By a union of multiple DOCID lists to return the final DOCID list</p> <p><b>DX</b> By an XML index scan on the index that is named in ACCESSNAME to return a DOCID list</p> <p><b>E</b> By direct row access using a row change timestamp column.</p> <p><b>I</b> By an index (identified in ACCESSCREATOR and ACCESSNAME)</p> <p><b>I1</b> By a one-fetch index scan</p> <p><b>M</b> By a multiple index scan (followed by MX, MI, or MU)</p> <p><b>MI</b> By an intersection of multiple indexes</p> <p><b>MU</b> By a union of multiple indexes</p> <p><b>MX</b> By an index scan on the index named in ACCESSNAME. When the access method MX follows the access method DX, DI, or DU, the table is accessed by the DOCID index by using the DOCID list that is returned by DX, DI, or DU.</p> <p><b>N</b> By an index scan when the matching predicate contains the IN keyword</p> <p><b>P</b> By a dynamic pair-wise index scan</p> <p><b>R</b> By a table space scan</p> <p><b>RW</b> By a work file scan of the result of a materialized user-defined table function</p> <p><b>V</b> By buffers for an INSERT statement within a SELECT</p> <p><b>blank</b> Not applicable to the current row</p>
MATCHCOLS	For ACCESSTYPE I, I1, N, MX, or DX, the number of index keys that are used in an index scan; otherwise, 0.
ACCESSCREATOR	For ACCESSTYPE I, I1, N, MX, or DX, the creator of the index; otherwise, blank.
ACCESSNAME	For ACCESSTYPE I, I1, N, MX, or DX, the name of the index; for ACCESSTYPE P, DSNPJW( <i>mioxpseqno</i> ) is the starting pair-wise join leg in MIXOPSEQNO; otherwise, blank.
INDEXONLY	<p>Indication of whether access to an index alone is enough to perform the step, or Indication of whether data too must be accessed.</p> <p><b>Y</b> Yes</p> <p><b>N</b> No</p>
SORTN_UNIQ	<p>Indication of whether the new table is sorted to remove duplicate rows.</p> <p><b>Y</b> Yes</p> <p><b>N</b> No</p>
SORTN_JOIN	<p>Indication of whether the new table is sorted for join method 2 or 4.</p> <p><b>Y</b> Yes</p> <p><b>N</b> No</p>
SORTN_ORDERBY	<p>Indication of whether the new table is sorted for ORDER BY.</p> <p><b>Y</b> Yes</p> <p><b>N</b> No</p>

Table 173. Descriptions of columns in PLAN\_TABLE (continued)

Column name	Description
SORTN_GROUPBY	Indication of whether the new table is sorted for GROUP BY.
	Y Yes
	N No
SORTC_UNIQ	Indication of whether the composite table is sorted to remove duplicate rows.
	Y Yes
	N No
SORTC_JOIN	Indication of whether the composite table is sorted for join method 1, 2 or 4.
	Y Yes
	N No
SORTC_ORDERBY	Indication of whether the composite table is sorted for an ORDER BY clause or a quantified predicate.
	Y Yes
	N No
SORTC_GROUPBY	Indication of whether the composite table is sorted for a GROUP BY clause.
	Y Yes
	N No
TSLOCKMOD	An indication of the mode of lock that is acquired on either the new table, or its table space or table space partitions. If the isolation can be determined at bind time, the values are:
	IS Intent share lock
	IX Intent exclusive lock
	S Share lock
	U Update lock
	X Exclusive lock
	SIX Share with intent exclusive lock
	N UR isolation; no lock
	If the isolation level cannot be determined at bind time, the lock mode is determined by the isolation level at run time is shown by the following values.
	NS For UR isolation, no lock; for CS, RS, or RR, an S lock.
	NIS For UR isolation, no lock; for CS, RS, or RR, an IS lock.
	NSS For UR isolation, no lock; for CS or RS, an IS lock; for RR, an S lock.
	SS For UR, CS, or RS isolation, an IS lock; for RR, an S lock.
	The data in this column is right justified. For example, IX appears as a blank, followed by I, followed by X. If the column contains a blank, then no lock is acquired.
	If the access method in the ACCESTYPE column is DX, DI, or DU, no latches are acquired on the XML index page and no lock is acquired on the new base table data page or row, nor on the XML table and the corresponding table spaces. The value of TSLOCKMODE is a blank in this case.
TIMESTAMP	Usually, the time at which the row is processed, to the last .01 second. If necessary, DB2 adds .01 second to the value to ensure that rows for two successive queries have different values.

Table 173. Descriptions of columns in PLAN\_TABLE (continued)

Column name	Description
REMARKS	A field into which you can insert any character string of 762 or fewer characters.
PREFETCH	Indication of whether data pages are to be read in advance by prefetch:  <b>D</b> Optimizer expects dynamic prefetch <b>S</b> Pure sequential prefetch <b>L</b> Prefetch through a page list <b>blank</b> Unknown or no prefetch
COLUMN_FN_EVAL	When an SQL aggregate function is evaluated:  <b>R</b> While the data is being read from the table or index <b>S</b> While performing a sort to satisfy a GROUP BY clause <b>blank</b> After data retrieval and after any sorts
MIXOPSEQ	The sequence number of a step in a multiple index operation.  <b>1, 2, ... n</b> For the steps of the multiple index procedure (ACCESSTYPE is MX, MI, MU, DX, DI, or DU.)  <b>0</b> For any other rows
VERSION	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable.
COLLID	The collection ID for the package. Applies only to an embedded EXPLAIN statement that is executed from a package or to a statement that is explained when binding a package. A blank indicates that the column is not applicable. The value DSNODYNAMICSQLCACHE indicates that the row is for a cached statement.
ACCESS_DEGREE	The number of parallel tasks or operations that are activated by a query. This value is determined at bind time; the actual number of parallel operations that are used at execution time could be different. This column contains 0 if a host variable is used. This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.
ACCESS_PGROUP_ID	The identifier of the parallel group for accessing the new table. A parallel group is a set of consecutive operations, executed in parallel, that have the same number of parallel tasks. This value is determined at bind time; it could change at execution time. This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.
JOIN_DEGREE	The number of parallel operations or tasks that are used in joining the composite table with the new table. This value is determined at bind time and can be 0 if a host variable is used. The actual number of parallel operations or tasks used at execution time could be different. This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.

Table 173. Descriptions of columns in PLAN\_TABLE (continued)

Column name	Description
JOIN_PGROUP_ID	The identifier of the parallel group for joining the composite table with the new table. This value is determined at bind time; it could change at execution time. This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.
SORTC_PGROUP_ID	The parallel group identifier for the parallel sort of the composite table. This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.
SORTN_PGROUP_ID	The parallel group identifier for the parallel sort of the new table. This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.
PARALLELISM_MODE	The kind of parallelism, if any, that is used at bind time: <b>I</b> Query I/O parallelism <b>C</b> Query CP parallelism <b>X</b> Sysplex query parallelism This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.
MERGE_JOIN_COLS	The number of columns that are joined during a merge scan join (Method=2). This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.
CORRELATION_NAME	The correlation name of a table or view that is specified in the statement. If no correlation name exists, then the column is null. This column contains the null value if the plan or package was bound using a plan table with fewer than 43 columns. Otherwise, it can contain null if the method that it refers to does not apply.
PAGE_RANGE	Indication of whether the table qualifies for page range screening, so that plans scan only the partitions that are needed. <b>Y</b> Yes <b>blank</b> No
JOIN_TYPE	The type of join: <b>F</b> FULL OUTER JOIN <b>L</b> LEFT OUTER JOIN <b>S</b> STAR JOIN <b>blank</b> INNER JOIN or no join RIGHT OUTER JOIN converts to a LEFT OUTER JOIN when you use it, so that JOIN_TYPE contains L.
GROUP_MEMBER	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

Table 173. Descriptions of columns in *PLAN\_TABLE* (continued)

Column name	Description
IBM_ SERVICE_ DATA	This column contains values that are for IBM use only.
WHEN_OPTIMIZE	When the access path was determined: <div> <div><b>blank</b></div> <div><b>B</b></div> <div><b>R</b></div> </div> At bind time, using a default filter factor for any host variables, parameter markers, or special registers. At bind time, using a default filter factor for any host variables, parameter markers, or special registers; however, the statement is re-optimized at run time using input variable values for input host variables, parameter markers, or special registers. The bind option REOPT(ALWAYS), REOPT(AUTO), or REOPT(ONCE) must be specified for reoptimization to occur. At run time, using input variables for any host variables, parameter markers, or special registers. The bind option REOPT(ALWAYS), REOPT(AUTO), or REOPT(ONCE) must be specified for this to occur.

Table 173. Descriptions of columns in *PLAN\_TABLE* (continued)

Column name	Description
QBLOCK_TYPE	For each query block, an indication of the type of SQL operation that is performed. For the outermost query, this column identifies the statement type. Possible values include:  <b>SELECT</b> SELECT  <b>INSERT</b> INSERT  <b>UPDATE</b> UPDATE  <b>MERGE</b> MERGE  <b>DELETE</b> DELETE  <b>SELUPD</b> SELECT with FOR UPDATE OF  <b>DELCUR</b> DELETE WHERE CURRENT OF CURSOR  <b>UPDCUR</b> UPDATE WHERE CURRENT OF CURSOR  <b>CORSUB</b> Correlated subselect or fullselect  <b>TRUNCA</b> TRUNCATE  <b>NCOSUB</b> Noncorrelated subselect or fullselect  <b>TABLEX</b> Table expression  <b>TRIGGR</b> WHEN clause on CREATE TRIGGER  <b>UNION</b> UNION  <b>UNIONA</b> UNION ALL  <b>INTERS</b> INTERSECT  <b>INTERA</b> INTERSECT ALL  <b>EXCEPT</b> EXCEPT  <b>EXCEPTA</b> EXCEPT ALL

Table 173. Descriptions of columns in PLAN\_TABLE (continued)

Column name	Description																		
BIND_TIME	For non-cached static SQL statements, the time at which the plan or package for this statement or query block was bound. For cached static and dynamic statements, the time at which the statement entered the cache. For non-cached static, cached static, and cached dynamic SQL statements, this is a full-precision timestamp value. For non-cached dynamic SQL statements, this is the value contained in the TIMESTAMP column of PLAN_TABLE appended by 4 zeros.																		
OPTHINT	A string that you use to identify this row as an optimization hint for DB2. DB2 uses this row as input when choosing an access path.																		
HINT_USED	If DB2 used one of your optimization hints, it puts the identifier for that hint (the value in OPTHINT) in this column.																		
PRIMARY_ACCESTYPE	Indicates Indication of whether direct row access is attempted first: <table> <tr> <td><b>D</b></td><td>DB2 tries to use direct row access with a rowid column. If DB2 cannot use direct row access with a rowid column at run time, it uses the access path that is described in the ACCESTYPE column of PLAN_TABLE.</td></tr> <tr> <td><b>T</b></td><td>The base table or result file is materialized into a work file, and the work file is accessed via sparse index access. If a base table is involved, then ACCESTYPE indicates how the base table is accessed.</td></tr> <tr> <td><b>blank</b></td><td>DB2 does not try to use direct row access by using a rowid column or sparse index access for a work file. The value of the ACCESTYPE column of PLAN_TABLE provides information on the method of accessing the table.</td></tr> </table>	<b>D</b>	DB2 tries to use direct row access with a rowid column. If DB2 cannot use direct row access with a rowid column at run time, it uses the access path that is described in the ACCESTYPE column of PLAN_TABLE.	<b>T</b>	The base table or result file is materialized into a work file, and the work file is accessed via sparse index access. If a base table is involved, then ACCESTYPE indicates how the base table is accessed.	<b>blank</b>	DB2 does not try to use direct row access by using a rowid column or sparse index access for a work file. The value of the ACCESTYPE column of PLAN_TABLE provides information on the method of accessing the table.												
<b>D</b>	DB2 tries to use direct row access with a rowid column. If DB2 cannot use direct row access with a rowid column at run time, it uses the access path that is described in the ACCESTYPE column of PLAN_TABLE.																		
<b>T</b>	The base table or result file is materialized into a work file, and the work file is accessed via sparse index access. If a base table is involved, then ACCESTYPE indicates how the base table is accessed.																		
<b>blank</b>	DB2 does not try to use direct row access by using a rowid column or sparse index access for a work file. The value of the ACCESTYPE column of PLAN_TABLE provides information on the method of accessing the table.																		
PARENT_QBLOCKNO	A number that indicates the QBLOCKNO of the parent query block.																		
TABLE_TYPE	The type of new table: <table> <tr> <td><b>B</b></td><td>Buffers for SELECT from INSERT, SELECT from UPDATE, SELECT from MERGE, or SELECT from DELETE statement.</td></tr> <tr> <td><b>C</b></td><td>Common table expression</td></tr> <tr> <td><b>F</b></td><td>Table function</td></tr> <tr> <td><b>M</b></td><td>Materialized query table</td></tr> <tr> <td><b>Q</b></td><td>Temporary intermediate result table (not materialized). For the name of a view or nested table expression, a value of Q indicates that the materialization was virtual and not actual. Materialization can be virtual when the view or nested table expression definition contains a UNION ALL that is not distributed.</td></tr> <tr> <td><b>R</b></td><td>Recursive common table expression</td></tr> <tr> <td><b>S</b></td><td>Subquery (correlated or non-correlated)</td></tr> <tr> <td><b>T</b></td><td>Table</td></tr> <tr> <td><b>W</b></td><td>Work file</td></tr> </table> <p>The value of the column is null if the query uses GROUP BY, ORDER BY, or DISTINCT, which requires an implicit sort.</p>	<b>B</b>	Buffers for SELECT from INSERT, SELECT from UPDATE, SELECT from MERGE, or SELECT from DELETE statement.	<b>C</b>	Common table expression	<b>F</b>	Table function	<b>M</b>	Materialized query table	<b>Q</b>	Temporary intermediate result table (not materialized). For the name of a view or nested table expression, a value of Q indicates that the materialization was virtual and not actual. Materialization can be virtual when the view or nested table expression definition contains a UNION ALL that is not distributed.	<b>R</b>	Recursive common table expression	<b>S</b>	Subquery (correlated or non-correlated)	<b>T</b>	Table	<b>W</b>	Work file
<b>B</b>	Buffers for SELECT from INSERT, SELECT from UPDATE, SELECT from MERGE, or SELECT from DELETE statement.																		
<b>C</b>	Common table expression																		
<b>F</b>	Table function																		
<b>M</b>	Materialized query table																		
<b>Q</b>	Temporary intermediate result table (not materialized). For the name of a view or nested table expression, a value of Q indicates that the materialization was virtual and not actual. Materialization can be virtual when the view or nested table expression definition contains a UNION ALL that is not distributed.																		
<b>R</b>	Recursive common table expression																		
<b>S</b>	Subquery (correlated or non-correlated)																		
<b>T</b>	Table																		
<b>W</b>	Work file																		

Table 173. Descriptions of columns in PLAN\_TABLE (continued)

Column name	Description
TABLE_ENCODE	The encoding scheme of the table. The possible values are: <b>A</b> ASCII <b>E</b> EBCDIC <b>U</b> Unicode <b>M</b> The table contains multiple CCSID sets
TABLE_SCCSID	The SBCS CCSID value of the table. If column TABLE_ENCODE is M, the value is 0.
TABLE_MCCSID	The mixed CCSID value of the table. If column TABLE_ENCODE is M, the value is 0
TABLE_DCCSID	The DBCS CCSID value of the table. If column TABLE_ENCODE is M, the value is 0.
ROUTINE_ID	The values in this column are for IBM use only.
CTEREF	If the referenced table is a common table expression, the value is the top-level query block number.
STMTTOKEN	User-specified statement token.
PARENT_PLANNO	Corresponds to the plan number in the parent query block where a correlated subquery is invoked. Or, for non-correlated subqueries, corresponds to the plan number in the parent query block that represents the work file for the subquery.

#### PSPI

##### Related tasks

“Populating and maintaining a plan table” on page 572

## DSN\_FUNCTION\_TABLE columns

The function table, DSN\_FUNCTION\_TABLE, contains descriptions of functions that are used in specified SQL statements.

#### PSPI

Your subsystem or data sharing group can contain more than one of these tables, including a table with the qualifier SYSIBM, a table with the qualifier DB2OSCA, and additional tables that are qualified by user IDs.

**Recommendation:** Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

The following table describes the columns of DSN\_FUNCTION\_TABLE.

Table 174. Descriptions of columns in DSN\_FUNCTION\_TABLE

Column name	Description
QUERYNO	A number intended to identify the statement being explained. If QUERYNO is not unique, the value of EXPLAIN_TIME is unique.
QBLOCKNO	A number that identifies each query block within a query.
APPLNAME	The name of the application plan for the row, or blank.

Table 174. Descriptions of columns in DSN\_FUNCTION\_TABLE (continued)

Column name	Description
PROGNAME	The name of the program or package containing the statement being explained, or blank.
COLLID	The collection ID for the package, or DSN_DYNAMICSQLCACHE. When the row in the table is for a cached statement, the value of the column is DSN_DYNAMICSQLCACHE.
GROUP_MEMBER	The member name of the DB2 that executed EXPLAIN, or blank. For more information about the GROUP_MEMBER column, see
EXPLAIN_TIME	The time at which the statement is processed. This time is the same as the BIND_TIME column in PLAN_TABLE.
SCHEMA_NAME	The schema name of the function invoked in the explained statement.
FUNCTION_NAME	The name of the function invoked in the explained statement.
SPEC_FUNC_ID	The specific name of the function invoked in the explained statement.
FUNCTION_TYPE	The type of function invoked in the explained statement. Possible values are: CU Column function SU Scalar function TU Table function
VIEW_CREATOR	If the function specified in the FUNCTION_NAME column is referenced in a view definition, the creator of the view. Otherwise, blank.
VIEW_NAME	If the function specified in the FUNCTION_NAME column is referenced in a view definition, the name of the view. Otherwise, blank.
PATH	The value of the SQL path that was used to resolve the schema name of the function.
FUNCTION_TEXT	The text of the function reference (the function name and parameters). If the function reference is over 100 bytes, this column contains the first 100 bytes. For functions specified in infix notation, FUNCTION_TEXT contains only the function name. For example, for a function named /, which overloads the SQL divide operator, if the function reference is A/B, FUNCTION_TEXT contains only /.



## DSN\_STATEMNT\_TABLE columns

The statement table, DSN\_STATEMNT\_TABLE, contains information about the estimated cost of specified SQL statements.



Your subsystem or data sharing group can contain more than one of these tables, including a table with the qualifier SYSIBM, a table with the qualifier DB2OSCA, and additional tables that are qualified by user IDs.

**Recommendation:** Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

The following table describes the content of each column in STATEMNT\_TABLE.

Table 175. Descriptions of columns in DSN\_STATEMNT\_TABLE

Column name	Description
QUERYNO	A number that identifies the statement being explained. See the description of the QUERYNO column in PLAN_TABLE for more information. If QUERYNO is not unique, the value of EXPLAIN_TIME is unique.
APPLNAME	The name of the application plan for the row, or blank. See the description of the APPLNAME column in PLAN_TABLE for more information.
PROGNAME	The name of the program or package containing the statement being explained, or blank. See the description of the PROGNAME column in PLAN_TABLE for more information.
COLLID	The collection ID for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. Blank if not applicable. The value DSNDYNAMICSQLCACHE indicates that the row is for a cached statement.
GROUP_MEMBER	The member name of the DB2 that executed EXPLAIN, or blank. See the description of the GROUP_MEMBER column in PLAN_TABLE for more information.
EXPLAIN_TIME	The time at which the statement is processed. This time is the same as the BIND_TIME column in PLAN_TABLE.
STMT_TYPE	<p>The type of statement being explained. Possible values are:</p> <p><b>SELECT</b> SELECT</p> <p><b>INSERT</b> INSERT</p> <p><b>UPDATE</b> UPDATE</p> <p><b>MERGE</b> MERGE</p> <p><b>DELETE</b> DELETE</p> <p><b>TRUNCATE</b> TRUNCATE</p> <p><b>SELUPD</b> SELECT with FOR UPDATE OF</p> <p><b>DELCUR</b> DELETE WHERE CURRENT OF CURSOR</p> <p><b>UPDCUR</b> UPDATE WHERE CURRENT OF CURSOR</p>
COST_CATEGORY	<p>Indicates if DB2 was forced to use default values when making its estimates. Possible values:</p> <p><b>A</b> Indicates that DB2 had enough information to make a cost estimate without using default values.</p> <p><b>B</b> Indicates that some condition exists for which DB2 was forced to use default values. See the values in REASON to determine why DB2 was unable to put this estimate in cost category A.</p>
PROCMS	The estimated processor cost, in milliseconds, for the SQL statement. The estimate is rounded up to the next integer value. The maximum value for this cost is 2147483647 milliseconds, which is equivalent to approximately 24.8 days. If the estimated value exceeds this maximum, the maximum value is reported.

Table 175. Descriptions of columns in DSN\_STATEMNT\_TABLE (continued)

Column name	Description
PROCSU	The estimated processor cost, in service units, for the SQL statement. The estimate is rounded up to the next integer value. The maximum value for this cost is 2147483647 service units. If the estimated value exceeds this maximum, the maximum value is reported.
REASON	<p>A string that indicates the reasons for putting an estimate into cost category B.</p> <p><b>HAVING CLAUSE</b> A subselect in the SQL statement contains a HAVING clause.</p> <p><b>HOST VARIABLES</b> The statement uses host variables, parameter markers, or special registers.</p> <p><b>REFERENTIAL CONSTRAINTS</b> Referential constraints of the type CASCADE or SET NULL exist on the target table of a DELETE statement.</p> <p><b>TABLE CARDINALITY</b> The cardinality statistics are missing for one or more of the tables that are used in the statement.</p> <p><b>TRIGGERS</b> Triggers are defined on the target table of an insert, update, or delete operation.</p> <p><b>UDF</b> The statement uses user-defined functions.</p> <p><b>MATERIALIZATION</b> Statistics are missing because the statement uses materialized views or nested table expresses.</p>
STMT_ENCODE	<p>Encoding scheme of the statement. If the statement represents a single CCSID set, the possible values are:</p> <p><b>A</b> ASCII <b>E</b> EBCDIC <b>U</b> Unicode</p> <p>If the statement has multiple CCSID sets, the value is M.</p>
TOTAL_COST	The overall estimated cost of the statement. This cost should be used only for reference purposes.

#### PSPI

#### Related tasks

“Creating a statement table” on page 562

## DSN\_STATEMENT\_CACHE\_TABLE columns

The statement cache table, DSN\_STATEMENT\_CACHE\_TABLE, contains information about the SQL statements in the statement cache, information captured as the results of an EXPLAIN STATEMENT CACHE ALL statement.

#### PSPI

Your subsystem or data sharing group can contain more than one of these tables, including a table with the qualifier SYSIBM, a table with the qualifier DB2OSCA, and additional tables that are qualified by user IDs.

**Recommendation:** Do not manually insert data into or delete data from EXPLAIN tables. The data is intended to be manipulated only by the DB2 EXPLAIN function and various optimization tools.

The following table shows the descriptions of the columns in DSN\_STATEMENT\_CACHE\_TABLE.

Table 176. Descriptions of columns in DSN\_STATEMENT\_CACHE\_TABLE

Column name	Description
STMT_ID	The statement ID; this value is the EDM unique token for the statement.
STMT_TOKEN	The statement token; you provide this value as an identification string.
COLLID	The collection ID; because DSN_STATEMENT_CACHE_TABLE is used to explain cached statements, the value of this column is DSNDYNAMICSQLCACHE.
PROGRAM_NAME	The name of the package or DBRM that performed the initial PREPARE for the statement.
INV_DROPALT	This column indicates if the statement has been invalidated by a DROP or ALTER statement.
INV_REVOKE	This column indicates if the statement has been invalidated by a REVOKE statement.
INV_LRU	This column indicates if the statement has been removed from the cache by LRU.
INV_RUNSTATS	This column indicates if the statement has been invalidated by RUNSTATS.
CACHED_TS	The timestamp when the statement was stored in the dynamic statement cache.
USERS	The number of current users of the statement. This number indicates the users that have prepared or run the statement during their current unit of work.
COPIES	The number of copies of the statement that are owned by all threads in the system.
LINES	The precompiler line number from the initial PREPARE of the statement.
PRIMAUTH	The primary authorization ID that did the initial PREPARE of the statement.
CURSQLID	The CURRENT SQLID that did the initial PREPARE of the statement.
1 BIND_QUALIFIER	The BIND qualifier. For unqualified table names, this is the object qualifier.
BIND_ISO	The value of the ISOLATION BIND option that is in effect for this statement. The value will be one of the following values: 'UR' Uncommitted read 'CS' Cursor stability 'RS' Read stability 'RR' Repeatable read
BIND_CDATA	The value of the CURRENTDATA BIND option that is in effect for this statement. The value will be one of the following values: 'Y' CURRENTDATA(YES) 'N' CURRENTDATA(NO)
BIND_DYNRL	The value of the DYNAMICRULES BIND option that is in effect for this statement. The value will be one of the following values: 'B' DYNAMICRULE(BIND) 'R' DYNAMICRULES(RUN)
BIND_DEGRE	The value of the CURRENT DEGREE special register that is in effect for this statement. The value will be one of the following values: 'A' CURRENT DEGREE = ANY '1' CURRENT DEGREE = 1
BIND_SQLRL	The value of the CURRENT RULES special register that is in effect for this statement. The value will be one of the following values: 'D' CURRENT RULES = DB2 'S' CURRENT RULES = SQL

Table 176. Descriptions of columns in DSN\_STATEMENT\_CACHE\_TABLE (continued)

Column name	Description
BIND_CHOLD	The value of the WITH HOLD attribute of the PREPARE for this statement. The value will be one of the following values: 'Y' Initial PREPARE specified WITH HOLD 'N' Initial PREPARE specified WITHOUT HOLD
STAT_TS	Timestamp of the statistics. This is the timestamp when IFCID 318 is started.
STAT_EXEC	The number of times this statement has been run. For a statement with a cursor, this is the number of OPENS.
STAT_GPAG	The number of getpage operations that are performed for the statement.
STAT_SYNR	The number of synchronous buffer reads that are performed for the statement.
STAT_WRIT	The number of buffer write operations that are performed for the statement.
STAT_EROW	The number of rows that are examined for the statement.
STAT_PROW	The number of rows that are processed for the statement.
STAT_SORT	The number of sorts that are performed for the statement.
STAT_INDX	The number of index scans that are performed for the statement.
STAT_RSCN	The number of table space scans that are performed for the statement.
STAT_PGRP	The number of parallel groups that are created for the statement.
STAT_ELAP	The accumulated elapsed time that is used for the statement.
STAT_CPU	The accumulated CPU time that is used for the statement.
STAT_SUS_SYNIO	The accumulated wait time for synchronous I/O operations for the statement.
STAT_SUS_LOCK	The accumulated wait time for lock and latch requests for the statement.
STAT_SUS_SWIT	The accumulated wait time for synchronous execution unit switch for the statement.
STAT_SUS_GLCK	The accumulated wait time for global locks for this statement.
STAT_SUS_OTHM	The accumulated wait time for read activity that is done by another thread.
STAT_SUS_OTHW	The accumulated wait time for write activity done by another thread.
STAT_RIDLMT	The number of times a RID list was not used because the number of RIDs would have exceeded DB2 limits.
STAT_RIDSTOR	The number of times a RID list was not used because there is not enough storage available to hold the list of RIDs.
EXPLAIN_TS	The timestamp for when the statement cache table is populated.
SCHEMA	The value of the CURRENT SCHEMA special register.
STMT_TEXT	The statement that is being explained.
STMT_ROWID	The ROWID of the statement.
BIND_RA_TOT	The total number of REBIND commands that have been issued for the dynamic statement because of the REOPT(AUTO) option
BIND_RO_TYPE	The current specification of the REOPT option for the statement: N REOPT(NONE) or its equivalent 1 REOPT(ONCE) or its equivalent A REOPT(AUTO) or its equivalent 0 The current plan is deemed optimal and there is no need for REOPT(AUTO)

---

## Single-table access

Certain values in a plan table indicate statements that access only a single table. You might use the plan table to identify better access paths for DB2 to choose from for such SQL statements.

### Table space scan access(**ACCESSTYPE='R'** and **PREFETCH='S'**)

In some cases, table space scan is used in combination with sparse index. A sparse index is created from the initial table space scan, and subsequent access to the table from the same statement uses the sparse index instead of repeatedly scanning the table.

#### **PSPI**

Table space scan is most often used for one of the following reasons:

- Access is through a created temporary table. (Index access is not possible for created temporary tables.)
- A matching index scan is not possible because an index is not available, or no predicates match the index columns.
- A high percentage of the rows in the table is returned. In this case, an index is not really useful because most rows need to be read anyway.
- The indexes that have matching predicates have low cluster ratios and are therefore efficient only for small amounts of data.

### Example

Assume that table T has no index on C1. The following is an example that uses a table space scan:

```
SELECT * FROM T WHERE C1 = VALUE;
```

In this case, at least every row in T must be examined to determine whether the value of C1 matches the given value.

### Table space scans of nonsegmented table spaces

DB2 reads and examines every page in the table space, regardless of which table the page belongs to. It might also read pages that have been left as free space and space not yet reclaimed after deleting data.

### Table space scans of segmented table spaces

If the table space is segmented, DB2 first determines which segments need to be read. It then reads only the segments in the table space that contain rows of T. If the prefetch quantity, which is determined by the size of your buffer pool, is greater than the SEGSIZE and if the segments for T are not contiguous, DB2 might read unnecessary pages. Use a SEGSIZE value that is as large as possible, consistent with the size of the data. A large SEGSIZE value is best to maintain clustering of data rows. For very small tables, specify a SEGSIZE value that is equal to the number of pages required for the table.

*Recommendation for SEGSIZE value:* The following table summarizes the recommendations for SEGSIZE, depending on how large the table is.

Table 177. Recommendations for SEGSIZE


Number of pages	SEGSIZE recommendation
≤ 28	4 to 28
> 28 < 128 pages	32
≥ 128 pages	64

## Table space scans of partitioned table spaces

Partitioned table spaces are nonsegmented. A table space scan on a partitioned table space is more efficient than on a nonpartitioned table space. DB2 takes advantage of the partitions by a limited partition scan, as described under “Scans limited to certain partitions (PAGE\_RANGE=’Y’)” on page 682.

## Table space scans and sequential prefetch

Regardless of the type of table space, DB2 plans to use sequential prefetch for a table space scan. For a segmented table space, DB2 might not actually use sequential prefetch at execution time if it can determine that fewer than four data pages need to be accessed. For guidance on monitoring sequential prefetch, see “Sequential prefetch (PREFETCH=’S’)” on page 686.

If you do not want to use sequential prefetch for a particular query, consider adding to it the clause OPTIMIZE FOR 1 ROW. 

## Aggregate function access (COLUMN\_FN\_EVAL)

When DB2 evaluates aggregate functions are depends evaluated is based on the access path chosen for the SQL statement.



- If the ACESSTYPE column is I1, then a MAX or MIN function can be evaluated by one access of the index named in ACCESSNAME.
- For other values of ACESSTYPE, the COLUMN\_FN\_EVAL column tells when DB2 is evaluating the aggregate functions.

### Value Functions are evaluated ...

- S** While performing a sort to satisfy a GROUP BY clause
- R** While the data is being read from the table or index
- blank** After data retrieval and after any sorts

Generally, values of R and S are considered better for performance than a blank. *Use variance and standard deviation with care:* The VARIANCE and STDDEV functions are always evaluated late (that is, COLUMN\_FN\_EVAL is blank). This causes other functions in the same query block to be evaluated late as well. For example, in the following query, the sum function is evaluated later than it would be if the variance function was not present:

```
SELECT SUM(C1), VARIANCE(C1) FROM T1;
```



## Index access (ACCESSTYPE is 'I', 'I1', 'N', 'MX', or 'DX')

If the ACCESSTYPE column in the plan table has a value of 'I', 'I1', 'N', 'MX', or 'DX', DB2 uses an index to access the table that is named in column TNAME.

**PSPI** The columns ACCESSCREATOR and ACCESSNAME identify the index. For a description of methods of using indexes, see “Index access paths” on page 668.

If a nested loop join is used in processing the query, you might see ACCESSTYPE=R, but the value of the PRIMARY\_ACCESTYPE column is T. This indicates that sparse index access is used. **PSPI**

### Overview of index access

Indexes can provide efficient access to data. In fact, that is the only purpose of non-unique indexes. Unique indexes have the additional purpose of ensuring that key values are unique.

#### Costs of indexes:

Before you begin creating indexes, you should carefully consider their costs.

**PSPI** Indexes have the following costs:

- Indexes require storage space. Padded indexes require more space than nonpadded indexes for long index keys. For short index keys, nonpadded indexes can take more space.
- Each index requires an index space and a data set, or as many data sets as the number of data partitions if the index is partitioned, and operating system restrictions exist on the number of open data sets.
- Indexes must be changed to reflect every insert or delete operation on the base table. If an update operation updates a column that is in the index, then the index must also be changed. The time required by these operations increases accordingly.
- Indexes can be built automatically when loading data, but this takes time. They must be recovered or rebuilt if the underlying table space is recovered, which might also be time-consuming.

#### Recommendation:

In reviewing the access paths described in “Index access paths” on page 668, consider indexes as part of your database design. For a query with a performance problem, ask yourself the following questions:

- Would adding a column to an index allow the query to use index-only access?
- Do you need a new index?
- Is your choice of clustering index correct?

**PSPI**

#### Indexes to avoid sorts:

Beside providing selective access to data, indexes can also order data, and sometime eliminate the need to sort the data.

**PSPI** Some sorts can be avoided if index keys are in the order needed by ORDER BY, GROUP BY, a join operation, or DISTINCT in an aggregate function. In other cases, such as when list prefetch is used, the index does not provide useful ordering, and the selected data might have to be sorted.

When it is absolutely necessary to prevent a sort, consider creating an index on the column or columns necessary to provide that ordering. Consider also using the clause OPTIMIZE FOR 1 ROW to discourage DB2 from choosing a sort for the access path.

Consider the following query:

```
SELECT C1,C2,C3 FROM T
  WHERE C1 > 1
  ORDER BY C1 OPTIMIZE FOR 1 ROW;
```

An ascending index on C1 or an index on (C1,C2,C3) could eliminate a sort.

### Backward index scan

In some cases, DB2 can use a backward index scan on a descending index to avoid a sort on ascending data. Similarly, an ascending index can be used to avoid a sort on descending data. For DB2 to use a backward index scan, the following conditions must be true:

- The index includes the columns in the ORDER BY clause in the same order that they appear in the ORDER BY clause.
- Each column in the sequence must have the opposite sequence (ASC or DESC) of the ORDER BY clause.

### Example: backward index scan

Suppose that an index exists on the ACCT\_STAT table. The index is defined by the following columns: ACCT\_NUM, STATUS\_DATE, STATUS\_TIME. All of the columns in the index are in ascending order. Now, consider the following SELECT statements:

```
SELECT STATUS_DATE, STATUS
  FROM ACCT_STAT
  WHERE ACCT_NUM = :HV
  ORDER BY STATUS_DATE DESC, STATUS_TIME DESC;

SELECT STATUS_DATE, STATUS
  FROM ACCT_STAT
  WHERE ACCT_NUM = :HV
  ORDER BY STATUS_DATE ASC, STATUS_TIME ASC;
```

By using a backward index scan, DB2 can use the same index for both statements.

### Randomized index key columns

You might also be able to avoid a sort by using the RANDOM option to create an index with a randomized key column, as long as the randomized key column is not included within an ORDER BY clause.

### Example: randomized index key columns

You can avoid sorts in query that uses GROUP BY processing by using an index with a randomized key. Consider the following statements:

```
CREATE INDEX I1
ON T1(C1, C2 RANDOM, C3);
SELECT C1, C2, SUM(C3)
FROM T1
WHERE C1 = 17
GROUP BY C2;
```

The query can use index I1 because all equal values of the original column C2 are stored contiguously on the index, and have identical random values stored. Although, the order of the query's output would appear to be arbitrary (as opposed to the output if an ASC or DESC index was used), the correctness of the results is not effected. Only the order in which the result tuples are represented to the application is effected by the randomization. If you wish to see the results in order, you must enforce the order with an ORDER BY statement, which requires a sort.

### When sorts are more efficient

Not all sorts are inefficient. For example, if the index that provides ordering is not an efficient one and many rows qualify, it is possible that using another access path to retrieve and then sort the data could be more efficient than the inefficient, ordering index.

Indexes that are created to avoid sorts can sometimes be non-selective. If these indexes require data access and if the cluster ratio is poor, these indexes are unlikely to be chosen. Accessing many rows by using a poorly clustered index is often less efficient than accessing rows by using a table space scan and sort. Both

table space scan and sort benefit from sequential access. 

#### Related concepts

"Minimizing overhead for retrieving few rows: OPTIMIZE FOR *n* ROWS" on page 301

### Index access paths

DB2 uses the following index access paths.

 **PSPI**

**Note:** DB2 might also use sparse index access (ACCESSTYPE=R and

PRIMARY\_ACCESSTYPE=T) when processing a nested loop join. 

### Matching index scan (MATCHCOLS>0):

In a *matching index scan*, predicates are specified on either the leading or all of the index key columns. These predicates provide *filtering*; only specific index pages and data pages need to be accessed. If the degree of filtering is high, the matching index scan is efficient.

 **PSPI**

In the general case, the rules for determining the number of matching columns are simple, but with a few exceptions.

- Look at the index columns from leading to trailing. For each index column, search for an indexable boolean term predicate on that column. (See "Properties

of predicates” on page 247 for a definition of boolean term.) If such a predicate is found, then it can be used as a matching predicate.

Column MATCHCOLS in a plan table shows how many of the index columns are matched by predicates.

- If no matching predicate is found for a column, the search for matching predicates stops.
- If a matching predicate is a range predicate, then there can be no more matching columns. For example, in the matching index scan example that follows, the range predicate `C2>1` prevents the search for additional matching columns.
- For star joins, a missing key predicate does not cause termination of matching columns that are to be used on the fact table index.

The exceptional cases are:

- For MX, or DX accesses and index access with list prefetch, IN-list predicates cannot be used as matching predicates.
- Join predicates cannot qualify as matching predicates when doing a merge join (METHOD=2). For example, `T1.C1=T2.C1` cannot be a matching predicate when doing a merge join, although any local predicates, such as `C1='5'` can be used. Join predicates can be used as matching predicates on the inner table of a nested loop join or hybrid join.

- The XML index, containing the composite key values, maps the XML values to the DOCID and NODEID pairs. The XML values (the first key value) in the composite keys can be specified in the XPath expressions. By matching the XPath expression in XMLEXISTS to the XPath expression in a particular value index, the index key entries which contain the matched key values can be identified. The DOCID and NODEID pairs of those identified index key entries can be used to locate the corresponding base table rows efficiently. DB2 uses DOCIDs only. The DOCIDs from those identified key entries form the DOCID list with the duplicate DOCID values removed. The DOCID list is then converted to the RID list of the base table rows via the DOCID index on the base table.

DB2 analyzes and matches the XPath expression in an XMLEXISTS predicate to the XPath expression in an XML index. If they match, the XPath expression in the XMLEXISTS predicate can be used as a matching predicate. The following describes some concepts used in the matching process. DB2 might apply various restrictions on the top of the matching concepts. All restrictions must be satisfied in order to be considered as a matching predicate. The restrictions mentioned in the following descriptions are not exhaustive, only some important ones are mentioned.

#### **Truly exact match**

An exact match, meaning that both XPath expressions are identical. This method is used only for the XML index with the SQL data type VARCHAR. For example: XPath expression in XMLEXISTS: `/a/b/c`, and XPath expression in the value index: `/a/b/c`.

#### **Exact match but the ending part of the XPath expression in XMLEXISTS is in a predicate**

Used only when the XPath predicate is a general comparison with operator `=`, `<`, `<=`, `>`, or `>=`. The data type of the operands in the predicate must match to the index data type. For example, XPath expression in XMLEXISTS: `/a[b/@c > 123]`, and XPath expression in the value index: `/a/b/@c`.

### Partial exact match with residual steps

Used to evaluate the XPath expression which has more steps than the first two methods. These extra steps in the XPath expression of XMLEXISTS are called 'residual steps'. For example: XPath expression in XMLEXISTS: /a/b[c > "xyz"]//d[e=8], and XPath expression in the value index: /a/b/c.

### Partial match for index filtering

The methods above have segments in the XPath expressions of XMLEXISTS that match "well" with the XPath expression of an index. This method handles the cases where the XPath expression in XMLEXISTS does not match so well with the XPath expression of an index. For example: XPath expression in XMLEXISTS: /a[b/c = 5]/d, and XPath expression in the value index: //c.

### Partial exact match with ANDing and ORing on DOCID lists

The XPath expression might be decomposed into multiple XPath segments which are ANDed or ORed together to produce a super set of the final result. The methods above apply to each of the decomposed XPath segments to determine whether the value index can be used to evaluate the XPath segment. For example: XPath expression in XMLEXISTS: /a/b[c = "xyz" and d > "abc"], and XPath expressions in the value indexes: /a/b/c, and /a/b/d.


### Partial match for filtering combined with ANDing and ORing on DOCID lists

Partial match for filtering can be combined with partial exact match with ANDing and ORing on DOCID lists. For example: XPath expression in XMLEXISTS: /a/b[@c = 5 or d > "a" ]/e, and XPath expressions in the value indexes: //@c, and /a/b/d.

## Matching index scan example


Assume an index was created on T(C1,C2,C3,C4):

```
SELECT * FROM T
  WHERE C1=1 AND C2>1
        AND C3=1;
```

Two matching columns occur in this example. The first one comes from the predicate C1=1, and the second one comes from C2>1. The range predicate on C2 prevents C3 from becoming a matching column. 

*The number of index columns used for matching (MATCHCOLS=n):*


If MATCHCOLS is 0, the access method is called a *nonmatching index scan* and all the index keys and their RIDs are read.. If MATCHCOLS is greater than 0, the access method is called a *matching index scan* and the query uses predicates that match the index columns.

 In general, the matching predicates on the leading index columns are equal or IN predicates. The predicate that matches the final index column can be an equal, IN, NOT NULL, or range predicate (<, <=, >, >=, LIKE, or BETWEEN).

The following example illustrates matching predicates:


```
SELECT * FROM EMP
  WHERE JOBCODE = '5' AND SALARY > 60000 AND LOCATION = 'CA';

INDEX XEMP5 on (JOBCODE, LOCATION, SALARY, AGE);
```

The index XEMP5 is the chosen access path for this query, with MATCHCOLS = 3. Two equal predicates are on the first two columns and a range predicate is on the third column. Though the index has four columns in the index, only three of them can be considered matching columns. 


### Index screening:

In *index screening*, predicates are specified on index key columns but are not part of the matching columns.

 Those predicates improve the index access by reducing the number of rows that qualify while searching the index. For example, with an index on T(C1,C2,C3,C4) in the following SQL statement, C3>0 and C4=2 are index screening predicates.

```
SELECT * FROM T
WHERE C1 = 1
      AND C3 > 0 AND C4 = 2
      AND C5 = 8;
```

The predicates can be applied on the index, but they are not matching predicates. C5=8 is not an index screening predicate, and it must be evaluated when data is retrieved. The value of MATCHCOLS in the plan table is 1.

EXPLAIN does not directly tell when an index is screened; however, if MATCHCOLS is less than the number of index key columns, it indicates that index screening is possible. 

### Nonmatching index scan (ACCESSTYPE='I' and MATCHCOLS=0):

In a *nonmatching index scan* no matching columns are in the index. Consequently, all of the index keys must be examined.



Because a nonmatching index usually provides no filtering, only a few cases provide an efficient access path. The following situations are examples:

#### When index screening predicates exist

In that case, not all of the data pages are accessed.

#### When the clause OPTIMIZE FOR *n* ROWS is used

That clause can sometimes favor a nonmatching index, especially if the index gives the ordering of the ORDER BY clause or GROUP BY clause.

#### When more than one table exists in a nonsegmented table space

In that case, a table space scan reads irrelevant rows. By accessing the rows through the nonmatching index, fewer rows are read.



### IN-list index scan (ACCESSTYPE=N):

An *IN-list index scan* is a special case of the matching index scan, in which a single indexable IN predicate is used as a matching equal predicate.

**PSPI** You can regard the IN-list index scan as a series of matching index scans with the values in the IN predicate being used for each matching index scan. The following example has an index on (C1,C2,C3,C4) and might use an IN-list index scan:

```
SELECT * FROM T
  WHERE C1=1 AND C2 IN (1,2,3)
        AND C3>0 AND C4<100;
```

The plan table shows MATCHCOLS = 3 and ACCESTYPE = N. The IN-list scan is performed as the following three matching index scans:

(C1=1,C2=1,C3>0), (C1=1,C2=2,C3>0), (C1=1,C2=3,C3>0)

Multiple indexable IN predicates can be used if indexes exist on those the necessary columns. In the following example, if an index exists on C1,C2 both IN-list predicates are used for matching index access:

```
SELECT * FROM T1 WHERE T1.C1 IN (1,2,3) AND T1.C2 IN ('A','B','C')
```

Parallelism is supported for queries that involve IN-list index access. These queries used to run sequentially in previous releases of DB2, although parallelism could have been used when the IN-list access was for the inner table of a parallel group. Now, in environments in which parallelism is enabled, you can see a reduction in elapsed time for queries that involve IN-list index access for the outer table of a

parallel group. **PSPI**

**Multiple index access (ACCESTYPE='M', 'MX', 'MI', 'MU', 'DX', 'DI', or 'DU'):**

*Multiple index access* uses more than one index to access a table.

**PSPI** Multiple index access is a good access path when:

- No single index provides efficient access.
- A combination of index accesses provides efficient access.

RID lists are constructed for each of the indexes involved. The unions or intersections of the RID lists produce a final list of qualified RIDs that is used to retrieve the result rows, using list prefetch. You can consider multiple index access as an extension to list prefetch with more complex RID retrieval operations in its first phase. The complex operators are union and intersection.

DB2 chooses multiple index access for the following query:

```
SELECT * FROM EMP
  WHERE (AGE = 34) OR
        (AGE = 40 AND JOB = 'MANAGER');
```

For this query:

- EMP is a table with columns EMPNO, EMPNAME, DEPT, JOB, AGE, and SAL.
- EMPX1 is an index on EMP with key column AGE.
- EMPX2 is an index on EMP with key column JOB.

The plan table contains a sequence of rows describing the access. For this query, ACCESTYPE uses the following values:

**Value    Meaning**

**M**        Start of multiple index access processing

**MX**      Indexes are to be scanned for later union or intersection

- MI** An intersection (AND) is performed
- MU** A union (OR) is performed

The following steps relate to the previous query and the values shown for the plan table in the following table:

1. Index EMPX1, with matching predicate AGE= 34, provides a set of candidates for the result of the query. The value of MIXOPSEQ is 1.
2. Index EMPX1, with matching predicate AGE = 40, also provides a set of candidates for the result of the query. The value of MIXOPSEQ is 2.
3. The first intersection (AND) is done, and the value of MIXOPSEQ is 3. This MI removes the two previous candidate lists (produced by MIXOPSEQs 2 and 3) by intersecting them to form an intermediate candidate list, IR1, which is not shown in PLAN\_TABLE.
4. Index EMPX2, with matching predicate JOB='MANAGER', also provides a set of candidates for the result of the query. The value of MIXOPSEQ is 4.
5. The last step, where the value MIXOPSEQ is 5, is a union (OR) of the two remaining candidate lists, which are IR1 and the candidate list produced by MIXOPSEQ 1. This final union gives the result for the query.

*Table 178. Plan table output for a query that uses multiple indexes.* Depending on the filter factors of the predicates, the access steps can appear in a different order.

PLAN-NO	TNAME	ACCESS-TYPE	MATCH-COLS	ACCESS-NAME	PREFETCH	MIXOP-SEQ
1	EMP	M	0		L	0
1	EMP	MX	1	EMPX1		1
1	EMP	MX	1	EMPX1		2
1	EMP	MI	0			3
1	EMP	MX	1	EMPX2		4
1	EMP	MU	0			5

In this example, the steps in the multiple index access follow the physical sequence of the predicates in the query. This is not always the case. The multiple index steps are arranged in an order that uses RID pool storage most efficiently and for the least amount of time.

#### Execution order for multiple index access

A set of rows in the plan table contain information about the multiple index access. The rows are numbered in column MIXOPSEQ in the order of execution of steps in the multiple index access. (If you retrieve the rows in order by MIXOPSEQ, the result is similar to postfix arithmetic notation.)

Both of the following examples have these indexes: IX1 on T(C1) and IX2 on T(C2).

#### Example: index access order

Suppose that you issue the following SELECT statement:

```
SELECT * FROM T
WHERE C1 = 1 AND C2 = 1;
```

DB2 processes the query by performing the following steps:

1. DB2 retrieves all the qualifying record identifiers (RIDs) where C1=1, by using index IX1.
2. DB2 retrieves all the qualifying RIDs where C2=1, by using index IX2. The intersection of these lists is the final set of RIDs.
3. DB2 accesses the data pages that are needed to retrieve the qualified rows by using the final RID list.

The plan table for this example is shown in the following table.

*Table 179. PLAN\_TABLE output for example with intersection (AND) operator*

TNAME	ACCESS- TYPE	MATCH- COLS	ACCESS- NAME	INDEX- ONLY	PREFETCH	MIXOP- SEQ
T	M	0		N	L	0
T	MX	1	IX1	Y		1
T	MX	1	IX2	Y		2
T	MI	0		N		3

#### **Example: multiple access to the same index**

Suppose that you issue the following SELECT statement:

```
SELECT * FROM T
  WHERE C1 BETWEEN 100 AND 199 OR
        C1 BETWEEN 500 AND 599;
```

In this case, the same index can be used more than once in a multiple index access because more than one predicate could be matching. DB2 processes the query by performing the following steps:

1. DB2 retrieves all RIDs where C1 is between 100 and 199, using index IX1.
2. DB2 retrieves all RIDs where C1 is between 500 and 599, again using IX1. The union of those lists is the final set of RIDs.
3. DB2 retrieves the qualified rows by using the final RID list.

The plan table for this example is shown in the following table.

*Table 180. PLAN\_TABLE output for example with union (OR) operator*

TNAME	ACCESS- TYPE	MATCH- COLS	ACCESS- NAME	INDEX- ONLY	PREFETCH	MIXOP- SEQ
T	M	0		N	L	0
T	MX	1	IX1	Y		1
T	MX	1	IX1	Y		2
T	MU	0		N		3

#### **Example: multiple index access for XML data**

Suppose that you issue the following SELECT statement that uses indexes to evaluate the XMLEXISTS predicates.

```

SELECT * FROM T
WHERE (C1 = 1 OR C2 = 1) AND
      XMLEXISTS('/a/b[c = 1]' PASSING XML_COL1) AND
      XNLEXISTS('/a/b[(e = 2 or /f[g] = 3) and /h/i[j] = 4]'
                PASSING XML_COL2);

```

The following statement shows the indexes defined on T:

```

IX1: C1
IX2: C2
VIX1: /a/b/c
VIX2: /a/b/e
VIX3: /a/b/f/g
VIX4: /a/b/h/i/j
DOCID index on T: DIX1

```

The XPath expression in the second XMLEXISTS predicate is decomposed into multiple XPath segments which are combined by AND and OR operations. DB2 matches each XPath segment to an XML index. The matching information between the predicates and indexes are as follows:

*Table 181. Matching between predicates and indexes for the XMLEXISTS example*

Predicate N IX2 N VIX1 Y VIX2 Y VIX3 Y VIX4 Y	Matching Index	XML index
C1 = 1	IX1	N
C2 = 1	IX2	N
XMLEXISTS 1: /a/b[c =1]	VIX1	Y
XMLEXISTS 2: /a/b[e = 2]	VIX2	Y
XMLEXISTS 2: /a/b/f[g = 3]	VIX3	Y
XMLEXISTS2: /a/b/h/i[j = 4]	VIX4	Y

DB2 uses the above indexes to access the table T and processes the query by performing the following steps:

1. DB2 retrieves all the qualifying record identifiers (RIDs) where C1=1, by using index IX1.
2. DB2 retrieves all the qualifying RIDs where C2 = 1, by using index IX2.
3. The union of the RID lists from step 1 and 2 is the final set of qualifying RIDs where C1 = 1 OR C2 = 1.
4. DB2 retrieves all the qualifying DOCIDs where /a/b[c =1], by using index VIX1.
5. DB2 retrieves all the qualifying RIDs of the DOCID list from step 4, by using index DIX1.
6. The intersection of the RID lists from step 3 and 5 is the final set of qualifying RIDs where (C1 = 1 OR C2 = 1) AND XPath expression /a/b[c =1].
7. DB2 retrieves all the qualifying DOCIDs where /a/b[e = 2], by using index VIX2.
8. DB2 retrieves all the qualifying DOCIDs where /a/b/f[g = 3], by using index VIX3.
9. The union of the DOCID lists from step 7 and 8 is the final set of qualifying DOCIDs where XPath segment /a/b[e = 2] OR XPath segment /a/b/f[g = 3].
10. DB2 retrieves all the qualifying DOCIDs where /a/b/h/i[j = 4], by using index VIX4.

11. The intersection of the DOCID lists from step 9 and 10 is the final set of qualifying DOCIDs where (XPath segment /a/b[e = 2] OR XPath segment /a/b/f[g = 3]) AND XPath segment / a/b/h/i[j = 4].
12. DB2 retrieves all the qualifying RIDs of the DOCID list from step 11, by using index DIX1.
13. The intersection of the RID lists from step 6 and 12 is the final set of qualifying RIDs where (C1 = 1 OR C2 = 1) AND XPath expression /a/b[c =1] AND ((XPath segment /a/b[e = 2] OR XPath segment /a/b/f[g = 3]) AND XPath segment /a/b/h/i[j = 4]). DB2 accesses the data pages that are needed to retrieve the qualified rows by using the final RID list.

The plan table for this example is shown in the following table:

Table 182. The *PLAN\_TABLE* output for the *XMLEXISTS* predicate example

QUERY NO	Q BLOCK NO	PLAN NO	T NAME	TAB NO	METH OD	ACCT TYPE	MATCH COLS	ACC NAME	INDEX-ONLY	MIX OP SEQ
1	1	1	T	1	0	M	0		N	0
1	1	1	T	1	0	MX	1	IX1	Y	1
1	1	1	T	1	0	MX	1	IX2	Y	2
1	1	1	T	1	0	MU	0		N	3
1	1	1	T	1	0	DX	1	VIX1	Y	4
1	1	1	T	1	0	MX	0	DIX1	Y	5
1	1	1	T	1	0	MI	0		N	6
1	1	1	T	1	0	DX	1	VIX2	Y	7
1	1	1	T	1	0	DX	1	VIX3	Y	8
1	1	1	T	1	0	DU	0		N	9
1	1	1	T	1	0	DX	1	VIX4	Y	10
1	1	1	T	1	0	DI	0		N	11
1	1	1	T	1	0	MX	1	DIX1	Y	12
1	1	1	T	1	0	MI	0		N	13

#### PSPI

#### One-fetch access (ACCESSTYPE='I1'):

*One-fetch index access* requires retrieving only one row. It is the best possible access path and is chosen whenever it is available.

#### PSPI

One-fetch index access applies only to statements with MIN or MAX aggregate functions: the order of the index allows a single row to give the result of the function.

One-fetch index access is a possible access path when:

- The query includes only one table.
- The query includes only one aggregate function (either MIN or MAX).
- Either no predicate or all predicates are matching predicates for the index.

- The query includes no GROUP BY clause.
- Aggregate functions are on:
  - The first index column if no predicates exist
  - The last matching column of the index if the last matching predicate is a range type
  - The next index column (after the last matching column) if all matching predicates are equal type

### Example queries that use one-fetch index scan

Assuming that an index exists on T(C1,C2,C3), each of the following queries use one-fetch index scan:

```
SELECT MIN(C1) FROM T;
SELECT MIN(C1) FROM T WHERE C1>5;
SELECT MIN(C1) FROM T WHERE C1>5 AND C1<10;
SELECT MIN(C2) FROM T WHERE C1=5;
SELECT MAX(C1) FROM T;
SELECT MAX(C2) FROM T WHERE C1=5 AND C2<10;
SELECT MAX(C2) FROM T WHERE C1=5 AND C2>5 AND C2<10;
SELECT MAX(C2) FROM T WHERE C1=5 AND C2 BETWEEN 5 AND 10;
```

#### PSPI

### Index-only access (INDEXONLY='Y'):

With index-only access, the access path does not require any data pages because the access information is available in the index.

#### PSPI

For a SELECT operation, when all the columns needed for the query can be found in the index and DB2 does not access the table the method is called *index-only access*. Conversely, when an SQL statement requests a column that is not in the index, updates any column in the table, or deletes a row, DB2 has to access the associated data pages. Because the index is almost always smaller than the table itself, an index-only access path usually processes the data efficiently.


With an index on T(C1,C2), the following queries can use index-only access:

```
SELECT C1, C2 FROM T WHERE C1 > 0;
SELECT C1, C2 FROM T;
SELECT COUNT(*) FROM T WHERE C1 = 1;
```

Index-only access to data is not possible for any step that uses list prefetch. Index-only access is not possible for padded indexes when varying-length data is returned or a VARCHAR column has a LIKE predicate, unless the VARCHAR FROM INDEX field of installation panel DSNTIP4 is set to YES and plan or packages have been rebound to pick up the change. Index-only access is always possible for non-padded indexes.


If access is by more than one index, INDEXONLY is Y for a step with access type MX, or DX, because the data pages are not actually accessed until all the steps for intersection (MI or DI) or union (MU or DU) take place.

When an SQL application uses index-only access for a ROWID column, the application claims the table space or table space partition. As a result, contention might occur between the SQL application and a utility that drains the table space or partition. Index-only access to a table for a ROWID column is not possible if the

associated table space or partition is in an incompatible restrictive state. For example, an SQL application can make a read claim on the table space only if the restrictive state allows readers. 

### Equal unique index (**MATCHCOLS=number of index columns**):

An index that is fully matched and unique, and in which all matching predicates are equal-predicates, is called an *equal unique index* case.

 This case guarantees that only one row is retrieved. If one-fetch index access is unavailable, this is considered the most efficient access over all other indexes that are not equal unique. (The uniqueness of an index is determined by whether or not it was defined as unique.)

Sometimes DB2 can determine that an index that is not fully matching is actually an equal unique index case. Assume the following case:

Unique Index1: (C1, C2)  
Unique Index2: (C2, C1, C3)


```
SELECT C3 FROM T
WHERE C1 = 1 AND C2 = 5;
```

Index1 is a fully matching equal unique index. However, Index2 is also an equal unique index even though it is not fully matching. Index2 is the better choice because, in addition to being equal and unique, it also provides index-only access.



### UPDATE using an index:

If no index key columns are updated, you can use an index while performing an UPDATE operation.

 To use a matching index scan to update an index in which its key columns are being updated, the following conditions must be met:

- Each updated key column must have a corresponding predicate of the form "index\_key\_column = constant" or "index\_key\_column IS NULL".
- If a view is involved, WITH CHECK OPTION must not be specified.

For updates that do not involve dynamic scrollable cursors, DB2 can use list prefetch, multiple index access, or IN-list access. With list prefetch or multiple index access, any index or indexes can be used in an UPDATE operation. Of course, to be chosen, those access paths must provide efficient access to the data.

A positioned update that uses a dynamic scrollable cursor cannot use an access path with list prefetch, or multiple index access. This means that indexes that do not meet the preceding criteria cannot be used to locate the rows to be updated.



### Direct row access (**PRIMARY\_ACCESTYPE='D'**)

If an application selects a row from a table that contains a ROWID column, the row ID value implicitly contains the location of the row. If you use that row ID value in the search condition of subsequent SELECT, DELETE, or UPDATE operations, DB2 might be able to use *direct row access* navigate directly to the row.

**PSPI** Direct row access is very fast because DB2 does not need to use the index or a table space scan to find the row. Direct row access can be used on any table that has a ROWID column.

To use direct row access, you first select the values of a row into host variables. The value that is selected from the ROWID column contains the location of that row. Later, when you perform queries that access that row, you include the row ID value in the search condition. If DB2 determines that it can use direct row access, it uses the row ID value to navigate directly to the row.

If an application selects RID built-in function from a table, the result contains the location of the row. If you use the RID built-in function in the search condition of subsequent SELECT, DELETE, or UPDATE statements, DB2 might be able to choose direct row access to navigate directly to the row. For a query to qualify for direct row access, the search condition must be a Boolean term stage 1 predicate that fits one of these descriptions: If DB2 cannot locate the row through direct row access, it does not switch to another access method and just returns no row found. The EXPLAIN output in the PLAN\_TABLE is changed if DB2 chooses the direct row access when the above conditions are satisfied, with the RID built-in function used as a search condition. **PSPI**

### Predicates that qualify for direct row access

For a query to qualify for direct row access, the search condition must be a Boolean term stage 1 predicate, and must meet certain conditions.

**PSPI** SQL queries that meet the following criteria qualify for direct row access.

- A simple Boolean term predicate of the form `COL=noncolumn expression`, where COL has the ROWID data type and *noncolumn expression* contains a row ID
- A simple Boolean term predicate of the form `COL IN list`, where COL has the ROWID data type and the values in *list* are row IDs, and an index is defined on COL
- A compound Boolean term that combines several simple predicates using the AND operator, and one of the simple predicates fits description 1 or 2
- A simple Boolean term predicate of the form `RID (table designator) = noncolumn expression`, where *noncolumn expression* contains a result of the RID built-in function.
- A compound Boolean term that combines several simple predicates using the AND operator, and one of the simple predicates fits description 4.

However, just because a query qualifies for direct row access does not mean that access path is always chosen. If DB2 determines that another access path is better, direct row access is not chosen.

### Examples

In the following predicate example, ID is a ROWID column in table T1. A unique index exists on that ID column. The host variables are of the ROWID type.

```
WHERE ID IN (:hv_rowid1,:hv_rowid2,:hv_rowid3)
```

The following predicate also qualifies for direct row access:

```
WHERE ID = ROWID(X'F0DFD230E3C0D80D81C201AA0A280100000000000203')
```

## Searching for propagated rows

If rows are propagated from one table to another, do not expect to use the same row ID value from the source table to search for the same row in the target table, or vice versa. This does not work when direct row access is the access path chosen.

Assume that the host variable in the following statement contains a row ID from SOURCE:

```
SELECT * FROM TARGET
WHERE ID = :hv_rowid
```

Because the row ID location is not the same as in the source table, DB2 probably cannot find that row. Search on another column to retrieve the row you want.

**PSPI**

## Reverting to ACESSTYPE

Although DB2 might plan to use direct row access, circumstances can cause DB2 to not use direct row access at run time.

**PSPI**

DB2 remembers the location of the row as of the time it is accessed. However, that row can change locations (such as after a REORG) between the first and second time it is accessed, which means that DB2 cannot use direct row access to find the row on the second access attempt. Instead of using direct row access, DB2 uses the access path that is shown in the ACESSTYPE column of PLAN\_TABLE.

If the predicate you are using to do direct row access is not indexable and if DB2 is unable to use direct row access, then DB2 uses a table space scan to find the row. This can have a profound impact on the performance of applications that rely on direct row access. Write your applications to handle the possibility that direct row access might not be used. Some options are to:

- Ensure that your application does not try to remember ROWID columns across reorganizations of the table space.

When your application commits, it releases its claim on the table space; it is possible that a REORG can run and move the row, which disables direct row access. Plan your commit processing accordingly; use the returned row ID value before committing, or re-select the row ID value after a commit is issued.

If you are storing ROWID columns from another table, update those values after the table with the ROWID column is reorganized.

- Create an index on the ROWID column, so that DB2 can use the index if direct row access is disabled.
- Supplement the ROWID column predicate with another predicate that enables DB2 to use an existing index on the table. For example, after reading a row, an application might perform the following update:

```
EXEC SQL UPDATE EMP
SET SALARY = :hv_salary + 1200
WHERE EMP_ROWID = :hv_emp_rowid
AND EMPNO = :hv_empno;
```

If an index exists on EMPNO, DB2 can use index access if direct access fails. The additional predicate ensures DB2 does not revert to a table space scan.

**PSPI**

## Access methods that prevent direct row access

Certain access methods prevent DB2 from using direct row access.

### Parallelism

**PSPI** Direct row access and parallelism are mutually exclusive. If a query qualifies for both direct row access and parallelism, direct row access is used. If direct row access fails, DB2 does not revert to parallelism; instead it reverts to the backup access type (as designated by column `ACCESSTYPE` in the `PLAN_TABLE`). This might result in a table space scan. To avoid a table space scan in case direct row access fails, add an indexed column to the predicate.

### RID list processing

Direct row access and RID list processing are mutually exclusive. If a query qualifies for both direct row access and RID list processing, direct row access is used. If direct row access fails, DB2 does not revert to RID list processing; instead it reverts to the backup access type. **PSPI**

### Example: Coding with row IDs for direct row access

You can obtain the row ID value for a row, and then to use that value to find the row efficiently when you want to modify it.

**PSPI** The following figure is a portion of a C program that shows you how to obtain the row ID value for a row and use that value to find the row efficiently.

```

/*****
/* Declare host variables */
*****/
EXEC SQL BEGIN DECLARE SECTION;
    SQL TYPE IS BLOB_LOCATOR hv_picture;
    SQL TYPE IS CLOB_LOCATOR hv_resume;
    SQL TYPE IS ROWID hv_emp_rowid;
    short hv_dept, hv_id;
    char hv_name[30];
    decimal hv_salary[5,2];
EXEC SQL END DECLARE SECTION;

/*****
/* Retrieve the picture and resume from the PIC_RES table */
*****/
strcpy(hv_name, "Jones");
EXEC SQL SELECT PR.PICTURE, PR.RESUME INTO :hv_picture, :hv_resume
FROM PIC_RES PR
WHERE PR.Name = :hv_name;

/*****
/* Insert a row into the EMPDATA table that contains the
/* picture and resume you obtained from the PIC_RES table */
*****/
EXEC SQL INSERT INTO EMPDATA
VALUES (DEFAULT,9999,'Jones', 35000.00, 99,
:hv_picture, :hv_resume);

/*****
/* Now retrieve some information about that row,
/* including the ROWID value.
*****/
hv_dept = 99;
EXEC SQL SELECT E.SALARY, E.EMP_ROWID
```

```

        INTO :hv_salary, :hv_emp_rowid
        FROM EMPDATA E
        WHERE E.DEPTNUM = :hv_dept AND E.NAME = :hv_name;

/*****
/* Update columns SALARY, PICTURE, and RESUME. Use the
/* ROWID value you obtained in the previous statement
/* to access the row you want to update.
/* smiley_face and update_resume are
/* user-defined functions that are not shown here.
*****/
EXEC SQL UPDATE EMPDATA
        SET SALARY = :hv_salary + 1200,
        PICTURE = smiley_face(:hv_picture),
        RESUME = update_resume(:hv_resume)
        WHERE EMP_ROWID = :hv_emp_rowid;

/*****
/* Use the ROWID value to obtain the employee ID from the
/* same record.
*****/
EXEC SQL SELECT E.ID INTO :hv_id
        FROM EMPDATA E
        WHERE E.EMP_ROWID = :hv_emp_rowid;

/*****
/* Use the ROWID value to delete the employee record
/* from the table.
*****/
EXEC SQL DELETE FROM EMPDATA
        WHERE EMP_ROWID = :hv_emp_rowid;

```

Figure 105. Example of using a row ID value for direct row access



## Scans limited to certain partitions (PAGE\_RANGE='Y')

DB2 can limit a scan of data in a partitioned table space to one or more partitions. The method is called a *limited partition scan*.



Subject to certain exceptions, a predicate on any column of the partitioning key might be used for limited partition scan if that predicate can eliminate partitions from the scan.

A limited partition scan can be combined with other access methods. For example, consider the following query:


```

SELECT .. FROM T
        WHERE (C1 BETWEEN '2002' AND '3280'
        OR C1 BETWEEN '6000' AND '8000')
        AND C2 = '6';

```

Assume that table T has a partitioned index on column C1 and that values of C1 between 2002 and 3280 all appear in partitions 3 and 4 and the values between 6000 and 8000 appear in partitions 8 and 9. Assume also that T has another index on column C2. DB2 could choose any of these access methods:

- A matching index scan on column C1. The scan reads index values and data only from partitions 3, 4, 8, and 9. (PAGE\_RANGE='N')

- A matching index scan on column C2. (DB2 might choose that if few rows have C2=6.) The matching index scan reads all RIDs for C2=6 from the index on C2 and corresponding data pages from partitions 3, 4, 8, and 9. (PAGE\_RANGE='Y')
- A table space scan on T. DB2 avoids reading data pages from any partitions except 3, 4, 8 and 9. (PAGE\_RANGE='Y'). 

## Parallel processing access (PARALLELISM\_MODE='I', 'C', or 'X')


DB2 can use parallel processing for read-only queries.

 PSPI

If mode is:


DB2 plans to use:

- I Parallel I/O operations
- C Parallel CP operations
- X Sysplex query parallelism

Non-null values in columns ACCESS\_DEGREE and JOIN\_DEGREE indicate to what degree DB2 plans to use parallel operations. At execution time, however, DB2 might not actually use parallelism, or it might use fewer operations in parallel than were originally planned. 

## Complex trigger WHEN clause access (QBLOCKTYPE='TRIGGR')

The plan table does not report simple trigger WHEN clauses, such as WHEN (N.C1 < 5). However, the plan table does report complex trigger WHEN clauses, which are clauses that involve other base tables and transition tables.

 The QBLOCK\_TYPE column of the top level query block shows TRIGGR to indicate a complex trigger WHEN clause.

Consider the following trigger:

```
CREATE TRIGGER REORDER
  AFTER UPDATE OF ON_HAND, MAX_STOCKED ON PARTS
  REFERENCING NEW TABLE AS NT OLD AS O
  FOR EACH STATEMENT MODE DB2SQL
  WHEN (O.ON_HAND < (SELECT MAX(ON_HAND) FROM NT))
  BEGIN ATOMIC
    INSERT INTO ORDER_LOG VALUES (O.PARTNO, O.ON_HAND);
  END
```

The following table shows the corresponding plan table for the WHEN clause.

Table 183. Plan table for the WHEN clause

QBLOCKNO	PLANNO	TABLE	ACCESSTYPE	QBLOCK_TYPE	PARENT_QBLOCKNO
1	1			TRIGGR	0
2	1	NT	R	NCOSUB	1

 PSPI

## Prefetch access

*Prefetch* is a mechanism for reading a set of pages, usually 32, into the buffer pool with only one asynchronous I/O operation.

**PSPI** Prefetch can allow substantial savings in both processor cycles and I/O costs. To achieve those savings, monitor the use of prefetch. Additionally, you can choose not to use prefetch. **PSPI**

### Number for pages read by prefetch

The number of pages read by prefetch depends on the type of prefetch, the buffer pool size (VPSIZE), in some cases, the sequential steal threshold (VPSEQT), and the number of buffers. The following table shows the number pages read by prefetch for each asynchronous I/O for each buffer pool size (4 KB, 8 KB, 16 KB, and 32 KB).

Table 184. The number of pages read by prefetch, by buffer pool size

Buffer pool size	Number of buffers	Pages read by prefetch (for each asynchronous I/O)		
		Sequential and LOB list	Dynamic and non-LOB list	Utility sequential
4 KB	VPSIZE < 224	8	8	16
	225 < VPSIZE < 1,000	16	16	32
	1000 <= VPSIZE < 40,000 or VPSIZE*VPSEQT < 40000	32	32	64
	40,000 <= VPSIZE*VPSEQT < 80,000	64	32	64
	80,000 <= VPSIZE*VPSEQT	64	32	128
8 KB	VPSIZE < 48	4	4	8
	48 < VPSIZE < 400	8	8	16
	400 <= VPSIZE < 20,000 or VPSIZE*VPSEQT < 20000	16	16	32
	20,000 <= VPSIZE*VPSEQT < 40,000	32	16	32
	40,000 <= VPSIZE*VPSEQT	32	16	64
16 KB	VPSIZE < 24	2	2	4
	24 < VPSIZE < 200	4	4	8
	200 <= VPSIZE < 10,000 or VPSIZE*VPSEQT < 10000	8	8	16
	10,000 <= VPSIZE*VPSEQT < 20,000	16	8	16
	20,000 <= VPSIZE*VPSEQT	16	8	32

Table 184. The number of pages read by prefetch, by buffer pool size (continued)

Buffer pool size	Number of buffers	Pages read by prefetch (for each asynchronous I/O)		
		Sequential and LOB list	Dynamic and non-LOB list	Utility sequential
32 KB	VPSIZE < 12	1	1	2
	12 < VPSIZE < 100	2	2	4
	100 <= VPSIZE < 5,000 or VPSIZE*VPSEQT < 5,000	4	4	8
	5,000 <= VPSIZE*VPSEQT < 10,000	8	4	8
	10,000 <= VPSIZE*VPSEQT	8	4	17

#### PSPI

### The expected type of prefetch (PREFETCH = 'D', 'S', 'L', or blank)

*Prefetch* is a method of determining in advance that a set of data pages is about to be used and then reading the entire set into a buffer with a single asynchronous I/O operation. You use EXPLAIN results to determine what kind of prefetch DB2 used to process an SQL statement.

#### PSPI

If the value of PREFETCH is:

- D** *Dynamic prefetch.* DB2 expects that data on the pages to be accessed is sufficiently nonsequential to invoke dynamic prefetch. DB2 uses dynamic prefetch in most situations. For a more complete description, see “Dynamic prefetch (PREFETCH='D').”
- S** *Sequential prefetch.* The data pages that are read in advance are sequential. A table space scan always uses sequential prefetch. An index scan might not use it. For a more complete description, see “Sequential prefetch (PREFETCH='S')” on page 686.
- L** *List prefetch.* One or more indexes are used to select the RIDs for a list of data pages to be read in advance; the pages need not be sequential. Usually, the RIDs are sorted. The exception is the case of a hybrid join (described under “Hybrid join (METHOD=4)” on page 700) when the value of column SORTN\_JOIN is N. For a more complete description, see “List prefetch (PREFETCH='L')” on page 686.
- Blank** No prefetch is expected. However, depending on the pattern of the page access, data can be prefetched at execution time through a process called *sequential detection*. For a description of that process, see “Sequential detection at execution time” on page 687.

#### PSPI

### Dynamic prefetch (PREFETCH='D'):

With *dynamic prefetch*, DB2 can automatically adjust between multi-page prefetch and single page synchronous read, as needed for optimal I/O performance.

**PSPI** Dynamic prefetch can reduce paging and improve performance over sequential prefetch, especially for access to data that might be arranged sequentially for some sets of pages but scattered randomly on other pages. When DB2 expects to use dynamic prefetch, DB2 sets PREFETCH='D'. At runtime, dynamic prefetch might or might not actually be used. However, DB2 expects dynamic prefetch and optimizes for that behavior.

#### When dynamic prefetch is used

Because dynamic prefetch uses sequential detection, it is more adaptable to dynamically changing access patterns than sequential prefetch. DB2 uses dynamic prefetch in almost all situations, the main exception being for table space scans.

**PSPI**

#### Sequential prefetch (PREFETCH='S'):

*Sequential prefetch* reads a sequential set of pages. The maximum number of pages read by a request issued from your application program is determined by the size of the buffer pool used.

**PSPI**

For certain utilities (LOAD, REORG, RECOVER), the prefetch quantity can be twice as much.

#### When sequential prefetch is used

Sequential prefetch is used only for table space scans.

**PSPI**

#### List prefetch (PREFETCH='L'):

*List prefetch* reads a set of data pages determined by a list of RIDs taken from an index.

**PSPI**

The data pages need not be contiguous. The maximum number of pages that can be retrieved in a single list prefetch is 32 (64 for utilities).

List prefetch can be used in conjunction with either single or multiple index access.

List prefetch uses the following steps:

1. RID retrieval: A list of RIDs for needed data pages is found by matching index scans of one or more indexes.
2. RID sort: The list of RIDs is sorted in ascending order by page number.
3. Data retrieval: The needed data pages are prefetched in order using the sorted RID list.

List prefetch does not preserve the data ordering given by the index. Because the RIDs are sorted in page number order before accessing the data, the data is not retrieved in order by any column. If the data must be ordered for an ORDER BY clause or any other reason, it requires an additional sort.

In a hybrid join, if the index is highly clustered, the page numbers might not be sorted before accessing the data.

List prefetch can be used with most matching predicates for an index scan. IN-list predicates are the exception; they cannot be the matching predicates when list prefetch is used.

### When list prefetch is used

List prefetch is used:

- Usually with a single index that has a cluster ratio lower than 80%
- Sometimes on indexes with a high cluster ratio, if the estimated amount of data to be accessed is too small to make sequential prefetch efficient, but large enough to require more than one regular read
- Always to access data by multiple index access
- Always to access data from the inner table during a hybrid join
- Usually for updatable cursors when the index contains columns that might be updated.

#### PSPI

*Bind time and execution time thresholds for list prefetch:*

DB2 does not consider list prefetch if the estimated number of RIDs to be processed would take more than 50% of the RID pool when the query is executed.

#### PSPI

You can change the size of the RID pool in the field RID POOL SIZE on installation panel DSNTIPC. The maximum size of a RID pool is 10 000 MB. The maximum size of a single RID list is approximately 26 million RIDs.

During execution, DB2 ends list prefetching if more than 25% of the rows in the table (with a minimum of 6524) must be accessed. Record IFCID 0125 in the performance trace, mapped by macro DSNDQW01, indicates whether list prefetch ended.

When list prefetch ends, the query continues processing by a method that depends on the current access path.

- For access through a single index or through the union of RID lists from two indexes, processing continues by a table space scan.
- For index access before forming an intersection of RID lists, processing continues with the next step of multiple index access. If no step remains and no RID list has been accumulated, processing continues by a table space scan.

When DB2 forms an intersection of RID lists, if any list has 32 or fewer RIDs, intersection stops and the list of 32 or fewer RIDs is used to access the data.

#### PSPI

### Sequential detection at execution time

Even if DB2 does not choose prefetch at bind time, it can sometimes use prefetch at execution time nevertheless. The method is called *sequential detection*.

### When sequential detection is used

DB2 can use sequential detection for both index leaf pages and data pages. It is most commonly used on the inner table of a nested loop join, if the data is accessed sequentially. If a table is accessed repeatedly using the same statement

(for example, DELETE in a do-while loop), the data or index leaf pages of the table can be accessed sequentially. This is common in a batch processing environment. Sequential detection can then be used if access is through:

- SELECT or FETCH statements
- UPDATE and DELETE statements
- INSERT statements when existing data pages are accessed sequentially

DB2 can use sequential detection if it did not choose sequential prefetch at bind time because of an inaccurate estimate of the number of pages to be accessed.

Sequential detection is not used for an SQL statement that is subject to referential constraints.

## How sequential detection works

The pattern of data access on a page is tracked when the application scans DB2 data through an index. Tracking is done to detect situations where the access pattern that develops is sequential or nearly sequential.

The most recent eight pages are tracked. A page is considered page-sequential if it is within  $P/2$  advancing pages of the current page, where  $P$  is the prefetch quantity.  $P$  is usually 32.

If a page is page-sequential, DB2 determines further if data access is sequential or nearly sequential. Data access is declared sequential if more than 4 out of the last eight pages are page-sequential; this is also true for index-only access. The tracking is continuous, allowing access to slip into and out of data access sequential.

When data access is first declared sequential, which is called *initial data access sequential*, three page ranges are calculated as follows:

- Let  $A$  be the page being requested. RUN1 is defined as the page range of length  $P/2$  pages starting at  $A$ .
- Let  $B$  be page  $A + P/2$ . RUN2 is defined as the page range of length  $P/2$  pages starting at  $B$ .
- Let  $C$  be page  $B + P/2$ . RUN3 is defined as the page range of length  $P$  pages starting at  $C$ .

## Sequential detection example

For example, assume that page  $A$  is 10. The following figure illustrates the page ranges that DB2 calculates.

	A	B	C
	RUN1	RUN2	RUN3
Page #	10	26	42
P=32 pages	16	16	32

Figure 106. Initial page ranges to determine when to use prefetch


For initial data access sequential, prefetch is requested starting at page  $A$  for  $P$  pages (RUN1 and RUN2). The prefetch quantity is always  $P$  pages.

For subsequent page requests where the page is 1) page sequential and 2) data access sequential is still in effect, prefetch is requested as follows:

- If the desired page is in RUN1, no prefetch is triggered because it was already triggered when data access sequential was first declared.
- If the desired page is in RUN2, prefetch for RUN3 is triggered and RUN2 becomes RUN1, RUN3 becomes RUN2, and RUN3 becomes the page range starting at C+P for a length of P pages.

If a data access pattern develops such that data access sequential is no longer in effect and, thereafter, a new pattern develops that is sequential, then initial data access sequential is declared again and handled accordingly.


Because, at bind time, the number of pages to be accessed can only be estimated, sequential detection acts as a safety net and is employed when the data is being accessed sequentially.

In extreme situations, when certain buffer pool thresholds are reached, sequential prefetch can be disabled. 

#### **Determining whether sequential detection was used:**

A plan table does not indicate sequential detection, which is not determined until run time.

 To determine whether sequential detection was used:


Refer to the IFCID 0003 record in the accounting trace or the IFCID 0006 record in the performance trace. 

## **Sort access**

DB2 can use two general types of sorts that DB2 can use when accessing data. One is a sort of data rows; the other is a sort of row identifiers (RIDs) in a RID list. You can use DB2 EXPLAIN to determine whether a SQL statement requires sort operations.

### **Sorts of data**

After you run EXPLAIN, DB2 sorts are indicated in PLAN\_TABLE. The sorts can be either sorts of the composite table or the new table.

 If a single row of PLAN\_TABLE has a 'Y' in more than one of the sort composite columns, then one sort accomplishes two things. (DB2 does not perform two sorts when two 'Y's are in the same row.) For instance, if both SORTC\_ORDERBY and SORTC\_UNIQ are 'Y' in one row of PLAN\_TABLE, then a single sort puts the rows in order and removes any duplicate rows as well.

The only reason DB2 sorts the new table is for join processing, which is indicated by SORTN\_JOIN.

### **Sorts for GROUP BY and ORDER BY**

These sorts are indicated by SORTC\_ORDERBY, and SORTC\_GROUPBY in PLAN\_TABLE.

If the statement includes both a GROUP BY clause and an ORDER BY clause, and if every item in the ORDER-BY list is in the GROUP-BY list, then only one sort is performed, which is marked as SORTC\_ORDERBY.

The performance of the sort by the GROUP BY clause is improved when the query accesses a single table and when the GROUP BY column has no index.

### Sorts to remove duplicates

This type of sort is used to process a query with SELECT DISTINCT, with a set function such as COUNT(DISTINCT COL1), or to remove duplicates in UNION processing. It is indicated by SORTC\_UNIQ in PLAN\_TABLE.

### Sorts used in join processing

For hybrid join (METHOD 4) and nested loop join (METHOD 1), the composite table can be sorted to make the join more efficient. For merge join (METHOD 2), both the composite table and new table need to be sorted unless an index is used for accessing these tables that gives the correct order already. The sorts needed for join processing are indicated by SORTN\_JOIN and SORTC\_JOIN in the PLAN\_TABLE.

When SORTN\_JOIN is set, each of the following join method behaves differently:

#### Nested loop join

SORTN\_JOIN is only valid in support of star join. When SORTN\_JOIN = Y for nested loop join, the qualified rows from the dimension or snowflake are sorted into the fact table join column sequence. A sparse index might also be created as a result of the sort to support efficient feedback loop skipping processing which is part of the star-join execution.

#### Sort merge join

The new table is accessed and sorted into join column sequence.

#### Hybrid join

When SORTN\_JOIN is on, the intermediate table is sorted into inner table rid sequence to support efficient list prefetch access to the inner table data.

PSPI

### Sorts for subquery processing

When a noncorrelated IN or NOT IN subquery is present in the query, the results of the subquery are sorted and put into a work file for later reference by the parent query.

The results of the subquery are sorted because this allows the parent query to be more efficient when processing the IN or NOT IN predicate. Duplicates are not needed in the work file, and are removed. Noncorrelated subqueries used with =ANY or =ALL, or NOT=ANY or NOT=ALL also need the same type of sort as IN or NOT IN subqueries. When a sort for a noncorrelated subquery is performed, you see both SORTC\_ORDERBY and SORTC\_UNIQUE in PLAN\_TABLE. This is because DB2 removes the duplicates and performs the sort.

SORTN\_GROUPBY, SORTN\_ORDERBY, and SORTN\_UNIQ are not currently used by DB2. PSPI

## Sorts of RIDs

To perform list prefetch, DB2 sorts RIDs into ascending page number order. This sort is very fast and is done totally in memory.

**PSPI** A RID sort is usually not indicated in the PLAN\_TABLE, but a RID sort normally is performed whenever list prefetch is used. The only exception to this rule is when a hybrid join is performed and a single, highly clustered index is used on the inner table. In this case SORTN\_JOIN is 'N', indicating that the RID list for the inner table was not sorted. **PSPI**

## The effect of sorts on OPEN CURSOR

The type of sort processing required by the cursor affects the amount of time it can take for DB2 to process the OPEN CURSOR statement.

**PSPI** This information outlines the effect of sorts and parallelism on OPEN CURSOR.

### *Without parallelism:*

- If no sorts are required, then OPEN CURSOR does not access any data. It is at the first fetch that data is returned.
- If a sort is required, then the OPEN CURSOR causes the materialized result table to be produced. Control returns to the application after the result table is materialized. If a cursor that requires a sort is closed and reopened, the sort is performed again.
- If a RID sort is performed, but no data sort, then it is not until the first row is fetched that the RID list is built from the index and the first data record is returned. Subsequent fetches access the RID pool to access the next data record.

### *With parallelism:*

- At OPEN CURSOR, parallelism is asynchronously started, regardless of whether a sort is required. Control returns to the application immediately after the parallelism work is started.
- If a RID sort is performed, but no data sort, then parallelism is not started until the first fetch. This works the same way as with no parallelism. **PSPI**

---

## Access to more than one table

A single SQL statement might access data from more than one table.

## Join operations

A *join* operation retrieves rows from more than one table and combines them. The operation specifies at least two tables, but the two tables need not be distinct.

## Composite tables

**PSPI**

A *composite table* represents the result of accessing one or more tables in a query. If a query contains a single table, only one composite table exists. If one or more joins are involved, an *outer composite table* consists of the intermediate result rows from the previous join step. This intermediate result might, or might not, be materialized into a work file.

The *new table* (or *inner table*) in a join operation is the table that is newly accessed in the step.

A join operation can involve more than two tables. In these cases, the operation is carried out in a series of steps. For non-star joins, each step joins only two tables.

Composite table example

The following figure shows a two-step join operation.

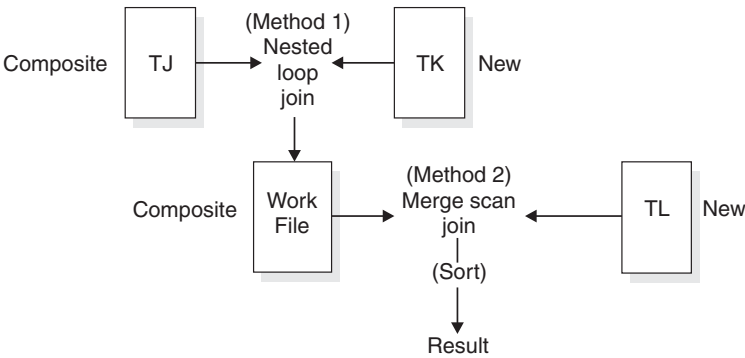


Figure 107. Two-step join operation

DB2 performs the following steps to complete the join operation:

- 1. Accesses the first table (METHOD=0), named TJ (TNAME), which becomes the composite table in step 2.
- 2. Joins the new table TK to TJ, forming a new composite table.
- 3. Sorts the new table TL (SORTN\_JOIN=Y) and the composite table (SORTC\_JOIN=Y), and then joins the two sorted tables.
- 4. Sorts the final composite table (TNAME is blank) into the desired order (SORTC\_ORDERBY=Y).

The following tables show a subset of columns in a plan table for this join operation:

Table 185. Subset of columns for a two-step join operation

METHOD	TNAME	ACCESS- TYPE	MATCH- COLS	ACCESS- NAME	INDEX- ONLY	TSLOCK- MODE
0	TJ	I	1	TJX1	N	IS
1	TK	I	1	TKX1	N	IS
2	TL	I	0	TLX1	Y	S
3			0		N	

Table 186. Subset of columns for a two-step join operation

SORTN UNIQ	SORTN JOIN	SORTN ORDERBY	SORTN GROUPBY	SORTC UNIQ	SORTC JOIN	SORTC ORDERBY	SORTC GROUPBY
N	N	N	N	N	N	N	N
N	N	N	N	N	N	N	N

Table 186. Subset of columns for a two-step join operation (continued)

SORTN UNIQ	SORTN JOIN	SORTN ORDERBY	SORTN GROUPBY	SORTC UNIQ	SORTC JOIN	SORTC ORDERBY	SORTC GROUPBY
N	Y	N	N	N	Y	N	N
N	N	N	N	N	N	Y	N

## Join conditions

A join operation typically matches a row of one table with a row of another on the basis of a *join condition*. For example, the condition might specify that the value in column A of one table equals the value of column X in the other table (WHERE T1.A = T2.X).

Two kinds of joins differ in what they do with rows in one table that do not match on the join condition with any row in the other table:

- An *inner join* discards rows of either table that do not match any row of the other table.
- An *outer join* keeps unmatched rows of one or the other table, or of both. A row in the composite table that results from an unmatched row is filled out with null values. As the following table shows, outer joins are distinguished by which unmatched rows they keep.

Table 187. Join types and kept unmatched rows

Outer join type	Included unmatched rows
Left outer join	The composite (outer) table
Right outer join	The new (inner) table
Full outer join	Both tables

## Join condition example

Suppose that you issue the following statement to explain an outer join:

```
EXPLAIN PLAN SET QUERYNO = 10 FOR
SELECT PROJECT, COALESCE(PROJECTS.PROD#, PRODNUM) AS PRODNUM,
       PRODUCT, PART, UNITS
FROM PROJECTS LEFT JOIN
  (SELECT PART,
   COALESCE(PARTS.PROD#, PRODUCTS.PROD#) AS PRODNUM,
   PRODUCTS.PRODUCT
  FROM PARTS FULL OUTER JOIN PRODUCTS
   ON PARTS.PROD# = PRODUCTS.PROD#) AS TEMP
ON PROJECTS.PROD# = PRODNUM
```

The following table shows a subset of the plan table for the outer join.

Table 188. Plan table output for an example with outer joins


QUERYNO	QBLOCKNO	PLANNO	TNAME	JOIN_TYPE
10	1	1	PROJECTS	
10	1	2	TEMP	L
10	2	1	PRODUCTS	
10	2	2	PARTS	F

Column JOIN\_TYPE identifies the type of outer join with one of these values:

- F for FULL OUTER JOIN
- L for LEFT OUTER JOIN
- Blank for INNER JOIN or no join


At execution, DB2 converts every right outer join to a left outer join; thus JOIN\_TYPE never identifies a right outer join specifically.

## Materialization with outer join

Sometimes DB2 has to materialize a result table when an outer join is used in conjunction with other joins, views, or nested table expressions. You can tell when this happens by looking at the TABLE\_TYPE and TNAME columns of the plan table. When materialization occurs, TABLE\_TYPE contains a W, and TNAME shows the name of the materialized table as DSNWFQB(xx), where xx is the number of the query block (QBLOCKNO) that produced the work file. 

## Cartesian join with small tables first

A *Cartesian join* is a form of nested loop join in which no join predicates exist between the two tables.


 DB2 usually avoids a Cartesian join, but sometimes it is the most efficient method, as in the following example. The query uses three tables: T1 has 2 rows, T2 has 3 rows, and T3 has 10 million rows.

```
SELECT * FROM T1, T2, T3
  WHERE T1.C1 = T3.C1 AND
        T2.C2 = T3.C2 AND
        T3.C3 = 5;
```

Join predicates are between T1 and T3 and between T2 and T3. No predicate joins T1 and T2.

Assume that 5 million rows of T3 have the value C3=5. Processing time is large if T3 is the outer table of the join and tables T1 and T2 are accessed for each of 5 million rows.

However if all rows from T1 and T2 are joined, without a join predicate, the 5 million rows are accessed only six times, once for each row in the Cartesian join of T1 and T2. It is difficult to say which access path is the most efficient. DB2 evaluates the different options and could decide to access the tables in the

sequence T1, T2, T3. 


## Nested loop join (METHOD=1)

In *nested loop join* DB2 scans the composite (outer) table. For each row in that table that qualifies (by satisfying the predicates on that table), DB2 searches for matching rows of the new (inner) table.



DB2 concatenates any matching rows that it finds with the current row of the composite table. If no rows match the current row, then:

- For an inner join, DB2 discards the current row.
- For an outer join, DB2 concatenates a row of null values.

Stage 1 and stage 2 predicates eliminate unqualified rows during the join. (For an explanation of those types of predicate, see “Stage 1 and stage 2 predicates” on page 249.) DB2 can scan either table using any of the available access methods, including table space scan. 

## Performance considerations for nested loop join

A *nested loop join* repetitively scans the inner table of the join.

That is, DB2 scans the outer table once, and scans the inner table as many times as the number of qualifying rows in the outer table. Therefore, the nested loop join is usually the most efficient join method when the values of the join column passed to the inner table are in sequence and the index on the join column of the inner table is clustered, or the number of rows retrieved in the inner table through the index is small.

### When nested loop join is used

DB2 often uses a nested loop join in the following situations.

- The outer table is small.
- Predicates with small filter factors reduce the number of qualifying rows in the outer table.
- Either an efficient, highly clustered index exists on the join columns of the inner table, or DB2 can dynamically create a sparse index on the inner table, and use that index for subsequent access.
- The number of data pages accessed in the inner table is small.
- No join columns exist. Hybrid and sort merge joins require join columns; nested loop joins do not.

### Example: left outer join

The following figure illustrates a nested loop for a left outer join. The outer join preserves the unmatched row in OUTERT with values A=10 and B=6. The same join method for an inner join differs only in discarding that row. The following figure illustrates a nested loop join.

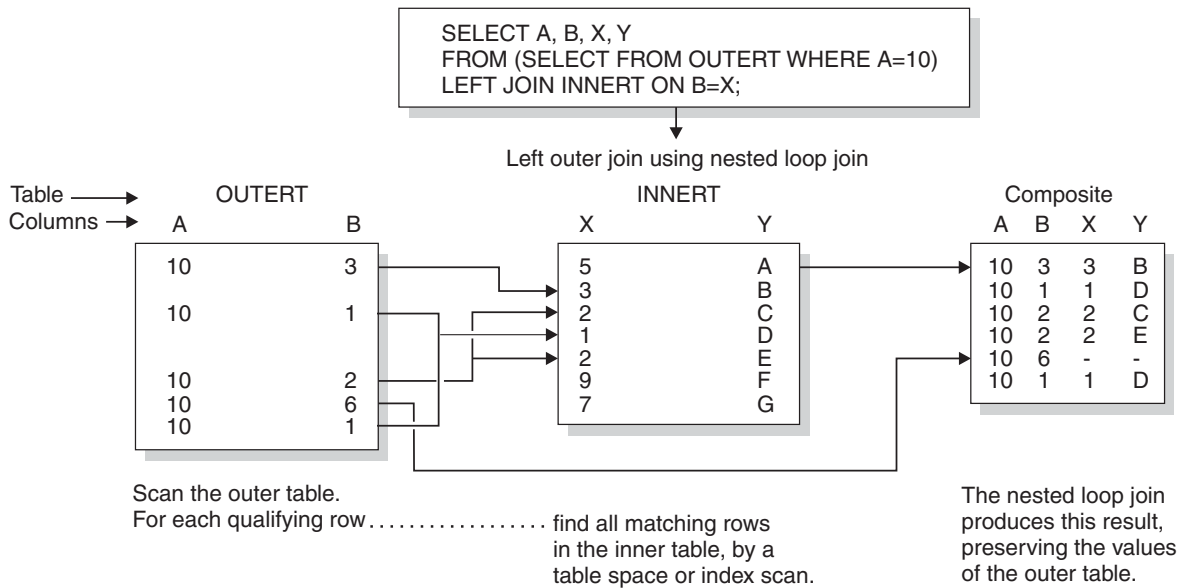


Figure 108. Nested loop join for a left outer join

### Example: one-row table priority

For a case like the following example, with a unique index on T1.C2, DB2 detects that T1 has only one row that satisfies the search condition. DB2 makes T1 the first table in a nested loop join.

```
SELECT * FROM T1, T2
WHERE T1.C1 = T2.C1 AND
      T1.C2 = 5;
```

### Example: Cartesian join with small tables first

A *Cartesian join* is a form of nested loop join in which no join predicates exist between the two tables. DB2 usually avoids a Cartesian join, but sometimes it is the most efficient method, as in the following example. The query uses three tables: T1 has 2 rows, T2 has 3 rows, and T3 has 10 million rows.

```
SELECT * FROM T1, T2, T3
WHERE T1.C1 = T3.C1 AND
      T2.C2 = T3.C2 AND
      T3.C3 = 5;
```

Join predicates are between T1 and T3 and between T2 and T3. No predicate joins T1 and T2.

Assume that 5 million rows of T3 have the value C3=5. Processing time is large if T3 is the outer table of the join and tables T1 and T2 are accessed for each of 5 million rows.

However if all rows from T1 and T2 are joined, without a join predicate, the 5 million rows are accessed only six times, once for each row in the Cartesian join of T1 and T2. It is difficult to say which access path is the most efficient. DB2 evaluates the different options and could decide to access the tables in the

sequence T1, T2, T3. PSPI

## Sorts on the composite table

**PSPI** DB2 might sort the composite table under the following conditions:

- The join columns in the composite table and the new table are not in the same sequence.
- The join column of the composite table has no index.
- The index is poorly clustered.

Nested loop join with a sorted composite table has the following performance advantages:

- Uses sequential detection efficiently to prefetch data pages of the new table, reducing the number of synchronous I/O operations and the elapsed time.
- Avoids repetitive full probes of the inner table index by using the index look-aside. **PSPI**

## Nested loop join with sparse index access

A value of PRIMARY\_ACCESTYPE = T indicates that DB2 dynamically creates a sparse index on the inner table and uses the sparse index to search the work file that is built on the inner table.

Nested loop join with sparse index has the following performance advantages:

- Access to the inner table is more efficient when the inner has no efficient index on the join columns.
- A sort of the composite table are avoided when composite table is relatively large. **PSPI**

## When a MERGE statement is used (QBLOCK\_TYPE = 'MERGE')

You can determine whether a MERGE statement was used and how it was processed by analyzing the QBLOCK\_TYPE and PARENT\_QBLOCKNO columns.

**PSPI** If the QBLOCK\_TYPE column contains MERGE, a MERGE statement was used. In most cases, a MERGE is processed in multiple query blocks with QBLOCK\_TYPES MERGE, UPDATE, and INSERT.

## Example

Consider the following MERGE statement:

```
MERGE INTO ARCHIVE AR
  USING VALUES (:hv_activity, :hv_description) FOR
    :hv_nrows ROWS as AC (ACTIVITY, DESCRIPTION)
  ON (AR.ACTIVITY = AC.ACTIVITY)
  WHEN MATCHED THEN UPDATE SET DESCRIPTION = AC.DESCRPTION
  WHEN NOT MATCHED THEN INSERT (ACTIVITY, DESCRIPTION)
    VALUES (AC.ACTIVITY, AC.DESCRPTION)
  NOT ATOMIC CONTINUE ON SQL EXCEPTION
```

The following table shows the corresponding plan table for the MERGE statement.

Table 189. Plan table for MERGE statement

QBLOCK NO	PARENT_QBLOCKNO	QBLOCK_TYPE	PLANNO	TNAME	TABLE_TYPE	JOIN_TYPE	METHOD
1	0	MERGE	1	ACTIVITIES	B		
1	0	MERGE	2	ARCHIVE	T	L	1
2	1	UPDATE	1	ARCHIVE	T		
3	1	INSERT	1	ARCHIVE	T		

PSPI

## Merge scan join (METHOD=2)

*Merge scan join* is also known as *merge join* or *sort merge join*. For this method, there must be one or more predicates of the form `TABLE1.COL1=TABLE2.COL2`, where the two columns have the same data type and length attribute.

PSPI

The following figure illustrates a merge scan join.

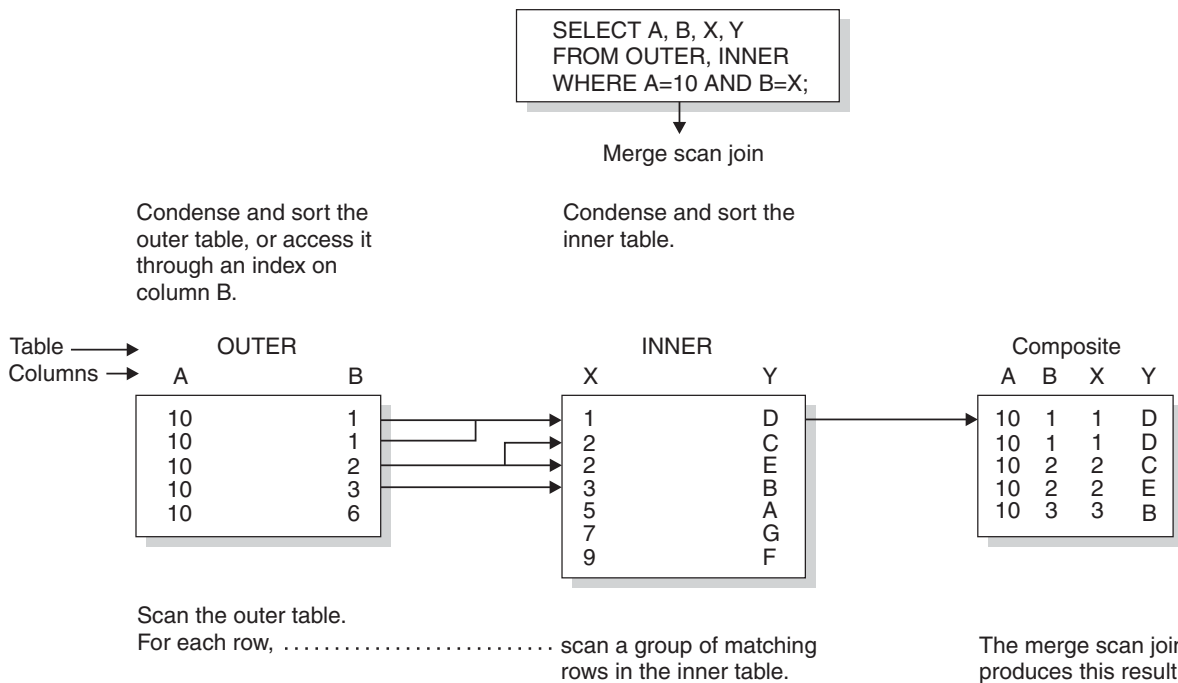


Figure 109. Merge scan join

DB2 scans both tables in the order of the join columns. If no efficient indexes on the join columns provide the order, DB2 might sort the outer table, the inner table, or both. The inner table is put into a work file; the outer table is put into a work file only if it must be sorted. When a row of the outer table matches a row of the inner table, DB2 returns the combined rows.

DB2 then reads another row of the inner table that might match the same row of the outer table and continues reading rows of the inner table as long as a match is found. When a match is no longer found, DB2 reads another row of the outer table.

- If that row has the same value in the join column, DB2 reads again the matching group of records from the inner table. Thus, a group of duplicate records in the inner table is scanned one time for each matching record in the outer table.
- If the outer row has a new value in the join column, DB2 searches ahead in the inner table. It can find any of the following rows:
  - Unmatched rows in the inner table, with lower values in the join column.
  - A new matching inner row. DB2 then starts the process again.
  - An inner row with a higher value of the join column. Now the row of the outer table is unmatched. DB2 searches ahead in the outer table, and can find any of the following rows:
    - Unmatched rows in the outer table.
    - A new matching outer row. DB2 then starts the process again.
    - An outer row with a higher value of the join column. Now the row of the inner table is unmatched, and DB2 resumes searching the inner table.

If DB2 finds an unmatched row:

For an inner join, DB2 discards the row.

For a left outer join, DB2 discards the row if it comes from the inner table and keeps it if it comes from the outer table.

For a full outer join, DB2 keeps the row.

When DB2 keeps an unmatched row from a table, it concatenates a set of null values as if that matched from the other table. A merge scan join must be used for a full outer join.

## Performance considerations for merge scan join

A full outer join by this method uses all predicates in the ON clause to match the two tables and reads every row at the time of the join. Inner and left outer joins use only stage 1 predicates in the ON clause to match the tables. If your tables match on more than one column, it is generally more efficient to put all the predicates for the matches in the ON clause, rather than to leave some of them in the WHERE clause.

For an inner join, DB2 can derive extra predicates for the inner table at bind time and apply them to the sorted outer table to be used at run time. The predicates can reduce the size of the work file needed for the inner table.

If DB2 has used an efficient index on the join columns, to retrieve the rows of the inner table, those rows are already in sequence. DB2 puts the data directly into the work file without sorting the inner table, which reduces the elapsed time.

You cannot use RANDOM order index columns as part of a sort merge join. If a join is between one table with that has an ASC index on the join column and a second table that has a RANDOM index, the indexes are in completely different orders, and cannot be merged.

## When merge scan join is used

- The qualifying rows of the inner and outer table are large, and the join predicate does not provide much filtering; that is, in a many-to-many join.
- The tables are large and have no indexes with matching columns.
- Few columns are selected on inner tables. This is the case when a DB2 sort is used. The fewer the columns to be sorted, the more efficient the sort is.

## Hybrid join (METHOD=4)

The hybrid join method applies only to an inner join, and requires an index on the join column of the inner table.

The following figure illustrates a hybrid join.

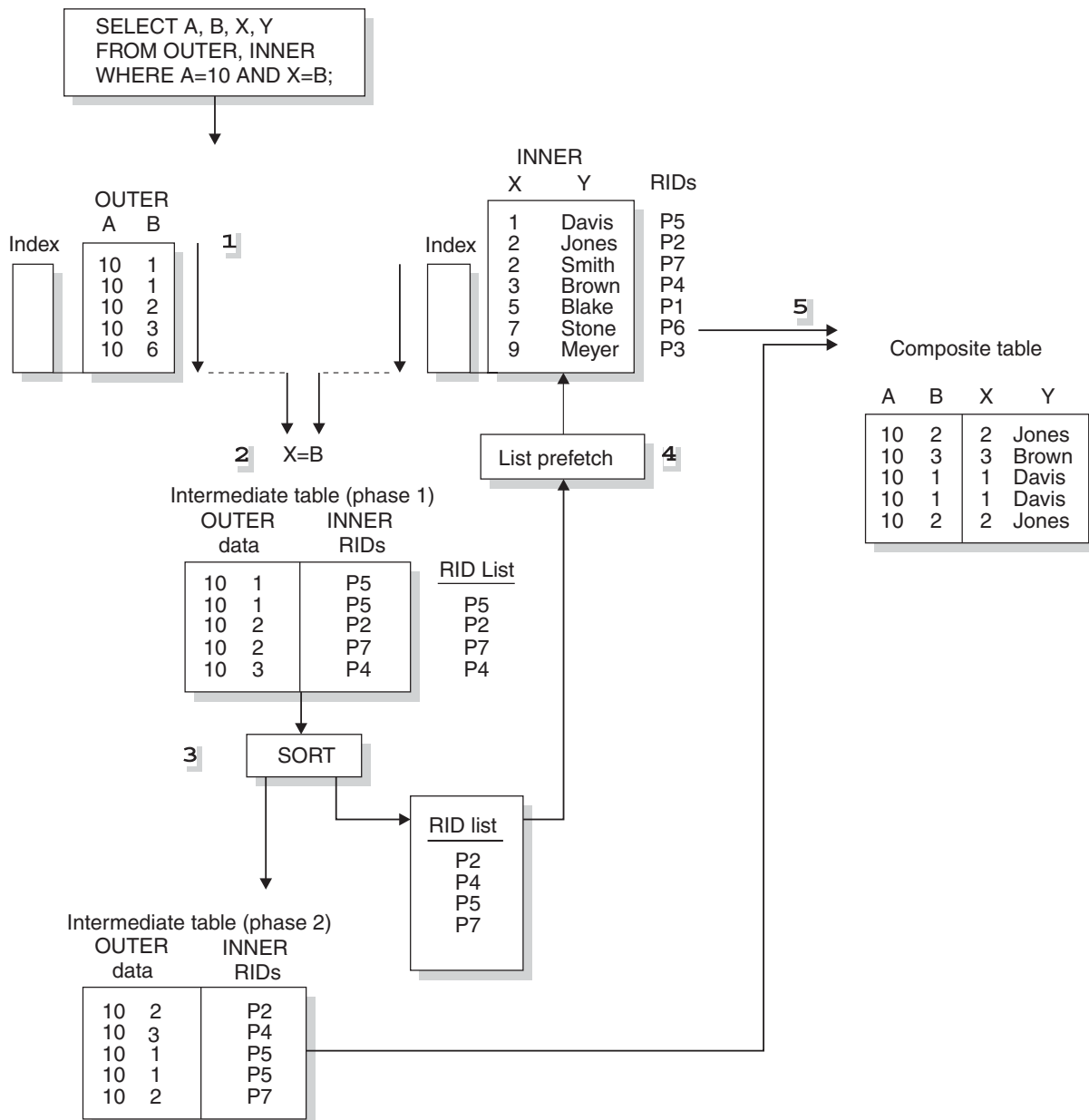


Figure 110. Hybrid join (SORTN\_JOIN='Y')

The method requires obtaining RIDs in the order needed to use list prefetch. The steps are shown in Figure 110. In that example, both the outer table (OUTER) and the inner table (INNER) have indexes on the join columns.

DB2 performs the following steps:

- 1** Scans the outer table (OUTER).
- 2** Joins the outer table with RIDs from the index on the inner table. The result is the phase 1 intermediate table. The index of the inner table is scanned for every row of the outer table.
- 3** Sorts the data in the outer table and the RIDs, creating a sorted RID list and the phase 2 intermediate table. The sort is indicated by a value of Y in column SORTN\_JOIN of the plan table. If the index on the inner table is a well-clustered index, DB2 can skip this sort; the value in SORTN\_JOIN is then N.
- 4** Retrieves the data from the inner table, using list prefetch.
- 5** Concatenates the data from the inner table and the phase 2 intermediate table to create the final composite table.

## Possible EXPLAIN results for hybrid join

The following table shows possible EXPLAIN results from a hybrid join and an explanation of each column value.

Table 190. Explanation of EXPLAIN results for a hybrid join

Column value	Explanation
METHOD='4'	A hybrid join was used.
SORTC_JOIN='Y'	The composite table was sorted.
SORTN_JOIN='Y'	The intermediate table was sorted in the order of inner table RIDs. A non-clustered index accessed the inner table RIDs.
SORTN_JOIN='N'	The intermediate table RIDs were not sorted. A clustered index retrieved the inner table RIDs, and the RIDs were already well ordered.
PREFETCH='L'	Pages were read using list prefetch.


## Performance considerations for hybrid join

Hybrid join uses list prefetch more efficiently than nested loop join, especially if indexes exist on the join predicate with low cluster ratios. It also processes duplicates more efficiently because the inner table is scanned only once for each set of duplicate values in the join column of the outer table.

If the index on the inner table is highly clustered, it is unnecessary to sort the intermediate table (SORTN\_JOIN=N). The intermediate table is placed in a table in memory rather than in a work file.

## When hybrid join is used

Hybrid join is often used under the following situations.

- A non-clustered index or indexes are used on the join columns of the inner table.
- The outer table has duplicate qualifying rows. 

## Star schema access

DB2 can use special join methods, such as star join and pair-wise join, to efficiently join tables that form a star schema.

**PSPI**

A *star schema* is a logical database design that is included in decision support applications. A star schema is composed of a *fact table* and a number of *dimension tables* that are connected to it. A dimension table contains several values that are given an ID, which is used in the fact table instead of all the values.

You can think of the fact table, which is much larger than the dimension tables, as being in the center surrounded by dimension tables; the result resembles a star formation. The following figure illustrates the star formation created by a star schema.

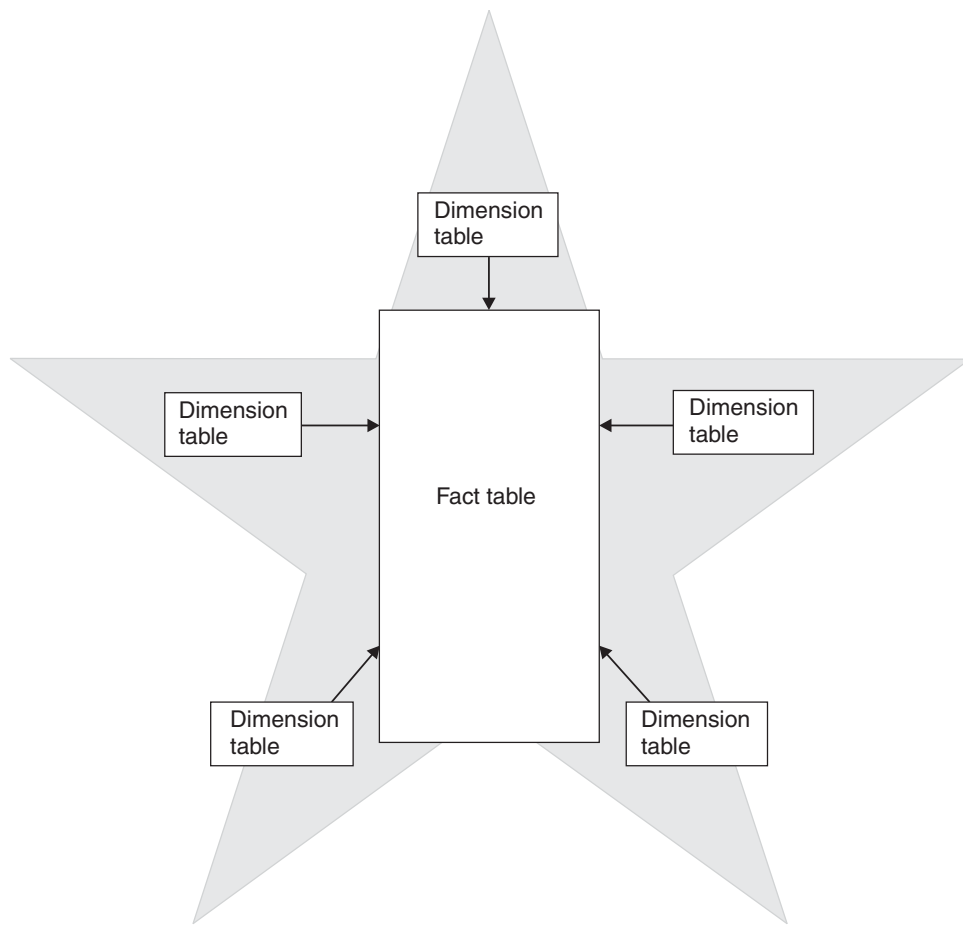


Figure 111. Star schema with a fact table and dimension tables

Unlike the steps in the other join methods, such as nested loop join, merge scan join, and hybrid join, in which only two tables are joined in each step, a single step in the star schema method can involve three or more tables. If the required indexes exist, DB2 might choose special methods such as star join and pair-wise join to process queries on a star schema more efficiently.

### Example star schema

In this typical example, the star schema is composed of a fact table, SALES, and a number of dimension tables connected to it for time, products, and geographic locations.

The TIME table has an column for each month, quarter, and year. The PRODUCT table has columns for each product item, its class, and its inventory. The

LOCATION table has columns of geographic data, including city and country.

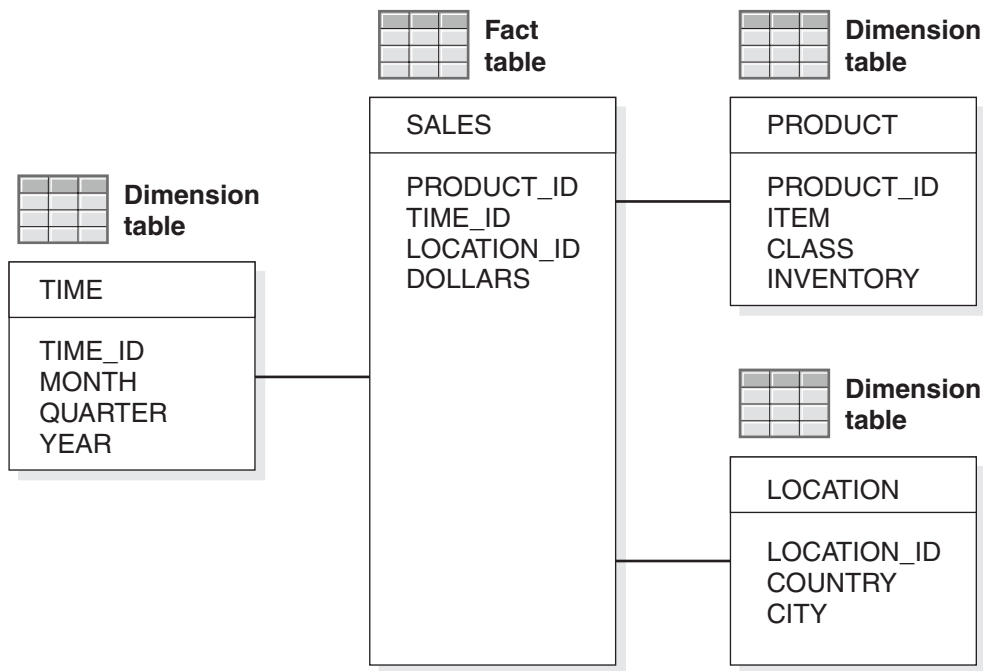


Figure 112. Example star schema with 3 dimension tables

In this scenario, the SALES table contains only three columns with IDs from the dimension tables, TIME, PRODUCT, and LOCATION, instead of three columns for time data, three columns for product data, and two columns for location data. Thus, the size of the fact table is greatly reduced. In addition, when you need to change an item, you need only make a single change in the dimension table, instead of making many changes in the fact table.

You can create even more complex star schemas by normalizing a dimension table into several tables. The normalized dimension table is called a *snowflake*. Only one of the tables in the snowflake joins directly with the fact table.

## Example star join query with three dimension tables

**PSPI** Suppose that you have a store in San Jose and want information about sales of audio equipment from that store in 2005. For this example, you want to join the following tables:

- A fact table for SALES (S)
- A dimension table for TIME (T) with columns for an ID, month, quarter, and year
- A dimension table for geographic LOCATION (L) with columns for an ID, city, region, and country
- A dimension table for PRODUCT (P) with columns for an ID, product item, class, and inventory

You could write the following query to join the tables:

```

SELECT *
FROM SALES S, TIME T, PRODUCT P, LOCATION L
WHERE S.TIME = T.ID      AND

```

```

S.PRODUCT = P.ID    AND
S.LOCATION = L.ID   AND
T.YEAR = 2005       AND
P.CLASS = 'SAN JOSE';

```

You would use the following index:

```
CREATE INDEX XSALES_TPL ON SALES (TIME, PRODUCT, LOCATION);
```

Your EXPLAIN output looks like the following table.

Table 191. Plan table output for a star join example with TIME, PRODUCT, and LOCATION

QUERYNO	QBLOCKNO	METHOD	TNAME	JOIN TYPE	SORTN JOIN	ACCESS TYPE
1	1	0	TIME	S	Y	R
1	1	1	PRODUCT	S	Y	R
1	1	1	LOCATION	S	Y	R
1	1	1	SALES	S		I

All snowflakes are processed before the central part of the star join, as individual query blocks, and are materialized into work files. A work file exists for each snowflake. The EXPLAIN output identifies these work files by naming them DSN\_DIM\_TBLX(*nn*), where *nn* indicates the corresponding QBLOCKNO for the snowflake.

This next example shows the plan for a star join that contains two snowflakes. Suppose that two new tables MANUFACTURER (M) and COUNTRY (C) are added to the tables in the previous example to break dimension tables PRODUCT (P) and LOCATION (L) into snowflakes:

- The PRODUCT table has a new column MID that represents the manufacturer.
- Table MANUFACTURER (M) has columns for MID and name to contain manufacturer information.
- The LOCATION table has a new column CID that represents the country.
- Table COUNTRY (C) has columns for CID and name to contain country information.

You could write the following query to join all the tables:

```

SELECT *
  FROM SALES S, TIME T, PRODUCT P, MANUFACTURER M,
       LOCATION L, COUNTRY C
 WHERE S.TIME = T.ID    AND
       S.PRODUCT = P.ID  AND
       P.MID = M.MID     AND
       S.LOCATION = L.ID AND
       L.CID = C.CID     AND
       T.YEAR = 2005     AND
       M.NAME = 'some_company';

```

The joined table pairs (PRODUCT, MANUFACTURER) and (LOCATION, COUNTRY) are snowflakes. The EXPLAIN output of this query looks like the following table.

Table 192. Plan table output for a star join example with snowflakes

QUERYNO	QBLOCKNO	METHOD	TNAME	JOIN TYPE	SORTN JOIN	ACCESS TYPE
1	1	0	TIME	S	Y	R
1	1	1	DSN_DIM_TBLX(02)	S	Y	R
1	1	1	SALES	S		I
1	1	1	DSN_DIM_TBLX(03)		Y	T
1	2	0	PRODUCT			R
1	2	1	MANUFACTURER			I
1	3	0	LOCATION			R
1	3	4	COUNTRY			I

**Note:** This query consists of three query blocks:

- QBLOCKNO=1: The main star join block
- QBLOCKNO=2: A snowflake (PRODUCT, MANUFACTURER) that is materialized into work file DSN\_DIM\_TBLX(02)
- QBLOCKNO=3: A snowflake (LOCATION, COUNTRY) that is materialized into work file DSN\_DIM\_TBLX(03)

The joins in the snowflakes are processed first, and each snowflake is materialized into a work file. Therefore, when the main star join block (QBLOCKNO=1) is processed, it contains four tables: SALES (the fact table), TIME (a base dimension table), and the two snowflake work files.

In this example, in the main star join block, the star join method is used for the first three tables (as indicated by S in the JOIN TYPE column of the plan table) and the remaining work file is joined by the nested loop join with sparse index access on the work file (as indicated by T in the ACCESTYPE column for

DSN\_DIM\_TBLX(3)). 

## When DB2 uses star schema access

To access the data in a star schema, you often write SELECT statements that include join operations between the fact table and the dimension tables, but no join operations between dimension tables. DB2 uses star schema processing as the join type for the query if the following conditions are true:

- The number of tables in the star schema query block, including the fact table, dimensions tables, and snowflake tables, is greater than or equal to the value of the SJTABLES subsystem parameter.
- The value of subsystem parameter STARJOIN is 1, or the cardinality of the fact table to the largest dimension table meets the requirements specified by the value of the subsystem parameter. The values of STARJOIN and cardinality requirements are:

**-1** Star schema processing is disabled. This is the default.

**1** Star schema processing is enabled. The one table with the largest cardinality is the fact table. However, if more than one table has this cardinality, star join is not enabled.

**0** Star schema processing is enabled if the cardinality of the fact table is at least 25 times the cardinality of the largest dimension that is a base table that is joined to the fact table.

- $n$  Star schema processing is enabled if the cardinality of the fact table is at least  $n$  times the cardinality of the largest dimension that is a base table that is joined to the fact table, where  $2 \leq n \leq 32768$ .

You can set the subsystem parameter STARJOIN by using the STAR JOIN QUERIES field on the DSNTIP8 installation panel.

In addition to the settings of SJTABLES and STARJOIN, the following conditions must also be met:

- The query references at least two dimensions.
- All join predicates are between the fact table and the dimension tables, or within tables of the same snowflake. If a snowflake is connected to the fact table, only one table in the snowflake (the central dimension table) can be joined to the fact table.
- All join predicates between the fact table and dimension tables are equi-join predicates.
- All join predicates between the fact table and dimension tables are Boolean term predicates..
- None of the predicates consist of a local predicate on a dimension table and a local predicate on a different table that are connected with an OR logical operator.
- No correlated subqueries cross dimensions.
- No single fact table column is joined to columns of different dimension tables in join predicates. For example, fact table column F1 cannot be joined to column D1 of dimension table T1 and also joined to column D2 of dimension table T2.
- After DB2 simplifies join operations, no outer join operations exist..
- The data type and length of both sides of a join predicate are the same.

Star join, which can reduce bind time significantly, does not provide optimal performance in all cases. Performance of star join depends on a number of factors such as the available indexes on the fact table, the cluster ratio of the indexes, and the selectivity of rows through local and join predicates. Follow these general guidelines for setting the value of SJTABLES:

- If you have queries that reference fewer than 10 tables in a star schema database and you want to make the star join method applicable to all qualified queries, set the value of SJTABLES to the minimum number of tables used in queries that you want to be considered for star join.

For example, suppose that you query a star schema database that has one fact table and three dimension tables. You should set SJTABLES to 4.

- If you want to use star join for relatively large queries that reference a star schema database but are not necessarily suitable for star join, use the default. The star join method is considered for all qualified queries that have 10 or more tables.
- If you have queries that reference a star schema database but, in general, do not want to use star join, consider setting SJTABLES to a higher number, such as 15, if you want to drastically cut the bind time for large queries and avoid a potential bind time SQL return code -101 for large qualified queries.

#### **Related concepts**

“Boolean term predicates” on page 250

“When DB2 simplifies join operations” on page 273

## Star join access (JOIN\_TYPE='S')

In *star join* processing, DB2 joins dimension tables to the fact table according to a multi-column index that is defined on the fact table.

Having a well-defined, multi-column index on the fact table is critical for efficient star join processing. Star join processing consolidates the dimensions to form a Cartesian product for matching to the single index on the fact table. Star join applies only to queries that qualify for star schema processing. This access method requires a single index on the fact table that supports the filtering that is provided by the dimension tables of the star schema. If the optimizer does not consider star join to be a cost effective access path for processing the star schema query, DB2 might choose pair-wise join (JOIN\_TYPE='P'), or more-traditional join methods, such as nested loop, hybrid, or merge-scan.

### Example: star schema with three dimension tables

**PSPI**

Suppose that you have a store in San Jose and want information about sales of audio equipment from that store in 2005. For this example, you want to join the following tables:

- A fact table for SALES (S)
- A dimension table for TIME (T) with columns for an ID, month, quarter, and year
- A dimension table for geographic LOCATION (L) with columns for an ID, city, region, and country
- A dimension table for PRODUCT (P) with columns for an ID, product item, class, and inventory

You could write the following query to join the tables:

```
SELECT *
FROM SALES S, TIME T, PRODUCT P, LOCATION L
WHERE S.TIME = T.ID      AND
      S.PRODUCT = P.ID    AND
      S.LOCATION = L.ID   AND
      T.YEAR = 2005      AND
      P.CLASS = 'AUDIO'  AND
      L.LOCATION = 'SAN JOSE';
```

You would use the following statement to create an index:

```
CREATE INDEX XSALES_TPL ON SALES (TIME, PRODUCT, LOCATION);
```

The following table shows the EXPLAIN output for this example.

Table 193. Plan table output for a star join example with TIME, PRODUCT, and LOCATION

QBLOCKNO	METHOD	TNAME	JOIN TYPE	SORTN JOIN	ACCESS TYPE	PRIMARY ACCESS TYPE
1	0	TIME	S	Y	R	
1	1	PRODUCT	S	Y	R	T
1	1	LOCATION	S	Y	R	T
1	1	SALES	S		I	

In the example above, all dimensions were accessed before the fact table. However, DB2 might choose to access only certain filtering dimensions before accessing the

fact table, and remaining dimensions might be joined later. This is a cost-based decision that is made by optimizer.

**Example: star schema with two snowflakes**

All snowflakes are processed before the central part of a star join, as individual query blocks, and they are materialized into work files. A work file exists for each snowflake. The EXPLAIN output identifies these work files by naming them DSN\_DIM\_TBLX(*nn*), where *nn* indicates the corresponding QBLOCKNO for the snowflake.

Suppose that two new tables MANUFACTURER (M) and COUNTRY (C) are added to the tables in the previous example to break dimension tables PRODUCT (P) and LOCATION (L) into snowflakes:

- The PRODUCT table has a new column MID that represents the manufacturer.
- The MANUFACTURER table has columns for MID and name to contain manufacturer information.
- The LOCATION table has a new column CID that represents the country.
- The COUNTRY table has columns for CID and name to contain country information.

You could write the following query to join all the tables:

```
SELECT *
FROM SALES S, TIME T, PRODUCT P, MANUFACTURER M,
     LOCATION L, COUNTRY C
WHERE S.TIME = T.ID      AND
      S.PRODUCT = P.ID   AND
      P.MID = M.MID      AND
      S.LOCATION = L.ID  AND
      L.CID = C.CID      AND
      T.YEAR = 2005      AND
      M.NAME = 'some_company';
```

The joined table pairs (PRODUCT, MANUFACTURER and LOCATION, COUNTRY) are snowflakes. The following table shows the EXPLAIN output for this example.

Table 194. Plan table output for a star join example with snowflakes

QBLOCKNO	METHOD	TNAME	JOIN TYPE	SORTN JOIN	ACCESS TYPE	PRIMARY ACCESS TYPE
1	0	TIME	S	Y	R	
1	1	DSN_DIM_TBLX(02)	S	Y	R	T
1	1	SALES	S		I	
1	1	DSN_DIM_TBLX(03)		Y	R	T
2	0	PRODUCT			R	
2	1	MANUFACTURER			I	
3	0	LOCATION			R	
3	4	COUNTRY			I	

The sample explain output above shows that this query consists of several query blocks:

#### QBLOCKNO=1

The main star join block


#### QBLOCKNO=2

A snowflake (PRODUCT, MANUFACTURER) that is materialized into work file DSN\_DIM\_TBLX(02)

#### QBLOCKNO=3

A snowflake (LOCATION, COUNTRY) that is materialized into work file DSN\_DIM\_TBLX(03)

The joins in the snowflakes are processed first, and each snowflake is materialized into a work file. Therefore, when the main star join block (QBLOCKNO=1) is processed, it contains four tables: SALES (the fact table), TIME (a base dimension table), and the two snowflake work file tables.

In this example, in the main star join block, the star join method is used for the first three tables (as indicated by S in the JOIN TYPE column of the plan table). The remaining work file is joined by the nested loop join with sparse index access on the work file (as indicated by T in the PRIMARY\_ACCESTYPE column for DSN\_DIM\_TBLX(3)). 

#### Pair-wise join access (JOIN\_TYPE='P')

In *pair-wise* join processing DB2 matches each dimension to fact table independently, according to a single-column index for each dimension table.

RID lists from each join to the fact table index are intersected in pairs. The first pair of RID lists are intersected, the result of that intersection is paired with the next RID list and so on, until all viable dimension filtering is intersected. Pair-wise join applies only to queries that qualify for star schema processing. This access method requires separate indexes on the fact table that support each individual filtering dimension tables of the star schema. Pair-wise join requires a separate fact table index for each filtering dimension.

If the optimizer does not consider pair-wise join to be a cost effective access path for processing the star schema query, DB2 might chose star join (JOIN\_TYPE='S') or more-traditional join methods, such nested loop join, hybrid join, or merge-scan join.

#### Example:



Suppose that you have a store in San Jose and want information about sales of audio equipment from that store in 2005. For this example, you want to join the following tables:

- A fact table for SALES (S)
- A dimension table for TIME (T) with columns for an ID, month, quarter, and year
- A dimension table for geographic LOCATION (L) with columns for an ID, city, region, and country
- A dimension table for PRODUCT (P) with columns for an ID, product item, class, and inventory

You could write the following query to join the tables:

```
SELECT *
FROM SALES S, TIME T, PRODUCT P, LOCATION L
WHERE S.TIME = T.ID      AND
```

```

S.PRODUCT = P.ID   AND
S.LOCATION = L.ID  AND
T.YEAR = 2005      AND
P.CLASS = 'AUDIO'  AND
L.LOCATION = 'SAN JOSE';

```

Instead a creating a single index on the fact table that supports the various combinations of filtering, you can create separate indexes on the fact table for each dimension table. For this example you would use the following three statements to create the indexes:

```

CREATE INDEX XSALES_T ON SALES (TIME);
CREATE INDEX XSALES_P ON SALES (PRODUCT);
CREATE INDEX XSALES_L ON SALES (LOCATION);

```

The following table shows the EXPLAIN output for this example.

*Table 195. Plan table output for a pair-wise join example with TIME, PRODUCT, and LOCATION*

QBLOCKNO	METHOD	TNAME	JOIN TYPE	ACCESS NAME	ACCESS TYPE	MIXOPSEQ
1	0	SALES	P		P	0
1	0	SALES	P	DSNPWJ(06)	MX	1
1	0	SALES	P	DSNPWJ(08)	MX	2
1	0	SALES	P		MI	3
1	0	SALES	P	DSNPWJ(10)	MX	4
1	0	SALES	P		MI	5
1	0	DSN_DIM_TBLX(02)			R	6
1	1	SALES		XSALES_T	I	7
1	0	DSN_DIM_TBLX(03)			R	8
1	1	SALES		XSALES_P	I	9
1	0	DSN_DIM_TBLX(04)			R	10
1	1	SALES		XSALES_L	I	11
1	1	DSN_DIM_TBLX(02)			R	
1	1	DSN_DIM_TBLX(03)			R	
1	1	DSN_DIM_TBLX(04)			R	
2	0	TIME			R	
3	0	PRODUCT			R	
4	0	LOCATION			R	

Dimension tables chosen for pair-wise join before the fact table must be materialized into their own dimension tables similar to snowflakes in star join access method. Query block 2, 3 and 4 build the work files for TIME, PRODUCT and LOCATION dimensions respectively. These dimensions are accessed before the fact table, and are also required to be joined back after the fact table, to ensure data consistency.

The sample EXPLAIN output above shows the following multi-index access steps:  
**MIXOPSEQ = 0**

Access the data pages for the fact table

**MIXOPSEQ = 1**

ACCESSNAME='DSNPWJ(06)' represents the pair-wise RID list built from multi-index access steps 6 and 7

**MIXOPSEQ = 2**

ACCESSNAME='DSNPWJ(08)' represents the pair-wise RID list built from multi-index access steps 8 and 9

**MIXOPSEQ = 3**

Intersection of RID lists from steps 1 and 2

**MIXOPSEQ = 4**

ACCESSNAME='DSNPWJ(10)' represents the pair-wise RID list built from multi-index access steps 10 and 11

**MIXOPSEQ = 5**

Intersection of RID lists from steps 3 and 4

**MIXOPSEQ = 6**

Materialized dimension table, DSN\_DIM\_TBLX(02), built from query block 2

**MIXOPSEQ = 7**

Join from step 6 to the fact table index XSALES\_T to build RID list as input to step 1

**MIXOPSEQ = 8**

Materialized dimension table DSN\_DIM\_TBLX(03) built from query block 3

**MIXOPSEQ = 9**


Join from step 6 to the fact table index XSALES\_P to build RID list as input to step 2

**MIXOPSEQ = 10**

Materialized dimension table DSN\_DIM\_TBLX(04) built from query block 4


**MIXOPSEQ = 11**

Join from step 6 to the fact table index XSALES\_L to build RID list as input to step 4

Based upon cost, DB2 might choose to access one or more dimensions before the fact table because each dimension has a corresponding fact table index. 

## Enabling data caching for star schema queries

You can enable data caching to improve the performance of queries on star schemas.

 When data caching is enabled for star schema queries, DB2 caches data from work files that are used by star schema queries. Data caching provides the following advantages:

### Immediate data availability

During a star join operation, work files might be scanned many times. If the work file data is cached in the dedicated virtual memory pool, that data is immediately available for join operations.

### Reduced buffer pool contention

Because the virtual memory space for data caching is separated from the work file buffer pool, contention with the buffer pool is reduced. Reduced contention improves performance particularly when sort operations are performed concurrently.

The default virtual memory pool size is 20 MB. To set the pool size, use the SJMX, or DXPOOL parameter on the DSNTIP8 installation panel.

To determine the best setting of the MAX DATA CACHING parameter for star schema queries:

1. Determine the value of A. Estimate the average number of work files that a star schema query uses. In typical cases, with highly normalized star schemas, the average number is about three to six work files.
2. Determine the value of B. Estimate the number of work file rows, the maximum length of the key, and the total of the maximum length of the relevant columns. Multiply these three values together to find the size of the data caching space for the work file, or the value of B.
3. Multiply (A)  $\times$  (B) to determine the size of the pool in MB.

### Example

The following example shows how to determine the size of the virtual memory for data caching. Suppose that you issue the following star join query, where SALES is the fact table:

```
SELECT C.COUNTRY, P.PRDNAM, SUM(F.SPRICE)
FROM SALES F, TIME T, PROD P, LOC L, SCOUN C
WHERE F.TID = T.TID AND
      F.PID = P.PID AND
      F.LID = L.LID AND
      L.CID = C.CID AND
      P.PCODE IN (4, 7, 21, 22, 53)
GROUP BY .COUNTRY, P.PRDNAM;
```

The following table shows the EXPLAIN output for this example query.

Table 196. EXPLAIN output for a star schema query

QBLOCKNO	PLANNO	TNAME	METHOD	JOIN_ TYPE	ACCESS TYPE	ACCESS NAME
1	1	TIME	0	S	R	
1	2	PROD	1	S	T	
1	3	SALES	1	S	I	XSALES
1	4	DSN_DIM_TBLX(02)	1		T	
1	5		3			
2	1	LOC	0		R	
2	2	SCOUN	4		I	XSCOUN

For this query, two work files can be cached in memory. These work files, PROD and DSN\_DIM\_TBLX(02), are indicated by ACCESTYPE=T.

1. In this example, the star join query uses two work files, PROD and DSN\_DIM\_TBLX(02). Therefore B = 2.
2. Both PROD and DSN\_DIM\_TBLX(02) are used to determine the value of B.

**Recommendation:** Average the values for a representative sample of work files, and round the value up to determine an estimate for a value of C:

- The number of work file rows depends on the number of rows that match the predicate. For PROD, 87 rows are stored in the work file because 87 rows match the IN-list predicate. No selective predicate is used for DSN\_DIM\_TBLX(02), so the entire result of the join is stored in the work file. The work file for DSN\_DIM\_TBLX(02) holds 2800 rows.

- The maximum length of the key depends on the data type definition of the table's key column. For PID, the key column for PROD, the maximum length is 4. DSN\_DIM\_TBLX(02) is a work file that results from the join of LOC and SCOUN. The key column that is used in the join is LID from the LOC table. The maximum length of LID is 4.
- The maximum data length depends on the maximum length of the key column and the maximum length of the column that is selected as part of the star join. Add to the maximum data length 1 byte for nullable columns, 2 bytes for varying length columns, and 3 bytes for nullable and varying length columns.

For the PROD work file, the maximum data length is the maximum length of PID, which is 4, plus the maximum length of PRDNAME, which is 24. Therefore, the maximum data length for the PROD work file is 28. For the DSN\_DIM\_TBLX(02) work file, the maximum data length is the maximum length of LID, which is 4, plus the maximum length of COUNTRY, which is 36. Therefore, the maximum data length for the DSN\_DIM\_TBLX(02) work file is 40.

Consequently, for PROD,  $B = (87) \times (4 + 28) = 2784$  bytes. For DSN\_DIM\_TBLX(02),  $B = (2800) \times (4 + 40) = 123200$  bytes.

The average of these two estimated values for B is approximately 62 KB. Because the number of rows in each work file can vary depending on the selection criteria in the predicate, the value of B should be rounded up to the nearest multiple of 100 KB. Therefore  $B = 100$  KB.

3. The size of the pool is determined by multiplying  $(B) \times (C)$  or, in this example,  $(2) \times (100 \text{ KB}) = 0.2 \text{ MB}$ .

#### PSPI

## Subquery access

The EXPLAIN output in the PLAN\_TABLE might show the position and order in which some subqueries are executed.

#### PSPI

The subqueries are indicated by a row in the PLAN\_TABLE having  $TNAME = "DSNWFQB(nn)"$ , where 'nn' is the query block number associated with the subquery, and  $TABLETYPE = 'S'$ . In PLAN\_TABLE, the PARENT\_PLANNO column corresponds to the plan number in the parent query block where the correlated subquery is invoked for correlated subqueries. For non-correlated subqueries it corresponds to the plan number in the parent query block that represents the work file for the subquery.

## Non-correlated subqueries

The EXPLAIN output below is for the non-correlated form of the following subquery:

```
SELECT * FROM T1 WHERE T1.C2 IN (SELECT T2.C2 FROM T2, T3 WHERE T2.C1 = T3.C1)
```

Table 197. EXPLAIN output for the non-correlated subquery

QB NO	PLAN NO	METHOD	TNAME	AC TYPE	MC	AC NAME	SC_JN	PAR_QB	PAR_PNO	QB TYPE	TB TYPE
1	1	0	DSNWFQB (02)	R	0		N	0	0	SELECT	W

Table 197. EXPLAIN output for the non-correlated subquery (continued)

QB NO	PLAN NO	METHOD	TNAME	AC TYPE	MC	AC NAME	SC_JN	PAR_QB	PAR_PNO	QB TYPE	TB TYPE
1	2	1	T1	I	1	T1_1X_C2	Y	0	0	SELECT	T
2	1	0	T2	R	0		N	1	1	NCOSUB	T
2	2	1	T3	I	1	T3_X_C1	N	1	1	NCOSUB	T

In the example above, the row corresponding to QBNO=2 and PLANNO=1 has PARENT\_PLANNO (abbreviated as PAR\_PNO) = 1 and PARENT\_QBNO (abbreviated as PAR\_QB) = 1. This means that the row corresponding to QBNO=1 and PLANNO=1 is the parent row. The sequence of execution flows from parent to child, then back to parent after the child rows are exhausted. In the example above that means the sequence of execution is (QBNO, PLANNO): (1,1) , (2,1), (2,2), (1,2).

## Correlated subqueries

The following example shows a correlated subquery and the associated EXPLAIN output:

```
SELECT * FROM T1
WHERE EXISTS (SELECT 1 FROM T2, T3
WHERE T2.C1 = T3.C1 AND T2.C2 = T1.C2)
```

Table 198. EXPLAIN output for the correlated subquery:

QB NO	PLAN NO	METHOD	TNAME	AC TYPE	MC	ACNAME	SC_JN	PAR_QB	PAR_PNO	QBTYPE	TB TYPE
1	10	0	T1	R	0		N	0	0	SELECT	T
2	1	1	T2	I	1	T2_IX_C2	N	1	1	CORSUB	T
2	2	1	T3	I	1	T3_IX_C1	N	1	1	CORSUB	T

## Subqueries transformed to joins

**PSPI** A plan table shows that a subquery is transformed into a join by the value in column QBLOCKNO.

- If the subquery is not transformed into a join, that means it is executed in a separate operation, and its value of QBLOCKNO is greater than the value for the outer query.
- If the subquery is transformed into a join, it and the outer query have the same value of QBLOCKNO. A join is also indicated by a value of 1, 2, or 4 in column METHOD. **PSPI**

## View and nested table expression access

A nested *table expression* is the specification of a subquery in the FROM clause of an SQL SELECT statement. The processing of table expressions is similar that for a view.

**PSPI** You can determine the methods that are used by executing EXPLAIN for the statement that contains the view or nested table expression. In addition, you

can use EXPLAIN to determine when set operators are used and how DB2 might eliminate unnecessary subselect operations to improve the performance of a query.

 PSPI

## Materialization for views and nested table expressions

*Materialization* means that the data rows that are selected by the view or nested table expression are put into a work file that is to be processed like a table. When the column TNAME names a view or nested table expression and column TABLE\_TYPE contains a W, it indicates that the view or nested table expression is materialized.

## Merge processing

In *merge* processing, a statement that references a view or table expression is combined with the fullselect that defined the view or table expression. This combination creates a logically equivalent statement. This equivalent statement is executed against the database.

 PSPI

The merge process is more efficient than materialization.

**Important:** The merge process and the MERGE statement are not related concepts, and should not be confused.

## Example: opportunity for merge processing

Consider the following statements, one of which defines a view, the other of which references the view:

View-defining statement:

```
CREATE VIEW VIEW1 (VC1,VC21,VC32) AS
  SELECT C1,C2,C3 FROM T1
  WHERE C1 > C3;
```

View referencing statement:

```
SELECT VC1,VC21
  FROM VIEW1
  WHERE VC1 IN (A,B,C);
```

The fullselect of the view-defining statement can be merged with the view-referencing statement to yield the following logically equivalent statement:

Merged statement:

```
SELECT C1,C2 FROM T1
  WHERE C1 > C3 AND C1 IN (A,B,C);
```

## More examples

The following statements show another example of when a view and table expression can be merged:

```
SELECT * FROM V1 X
  LEFT JOIN
    (SELECT * FROM T2) Y ON X.C1=Y.C1
  LEFT JOIN T3 Z ON X.C1=Z.C1;
```

Merged statement:

```
SELECT * FROM V1 X
  LEFT JOIN
    T2 ON X.C1 = T2.C1
  LEFT JOIN T3 Z ON X.C1 = Z.C1;
```

 PSPI

## Materialization

Views and table expressions cannot always be merged. In certain cases, DB2 materializes the view or table expression

### PSPI

In the following example, DB2 performs materialization of the view or table expression, which is a two step process.

1. The fullselect that defines the view or table expression is executed against the database, and the results are placed in a temporary copy of a result table.
2. The statement that references the view or table expression is then executed against the temporary copy of the result table to obtain the intended result.

Whether materialization is needed depends upon the attributes of the referencing statement, or logically equivalent referencing statement from a prior merge, and the attributes of the fullselect that defines the view or table expression.

### Example

Look at the following statements:

#### View defining statement

```
CREATE VIEW VIEW1 (VC1,VC2) AS
  SELECT SUM(C1),C2 FROM T1
  GROUP BY C2;
```

#### View referencing statement

```
SELECT MAX(VC1)
FROM VIEW1;
```

Column VC1 occurs as the argument of a aggregate function in the view referencing statement. The values of VC1, as defined by the view-defining fullselect, are the result of applying the aggregate function SUM(C1) to groups after grouping the base table T1 by column C2. No equivalent single SQL SELECT statement can be executed against the base table T1 to achieve the intended result.

You cannot specify that aggregate functions be applied successively.

### PSPI

### When views and nested table expressions are materialized

DB2 uses materialization to satisfy a reference to a view or table expression when aggregate processing is involved (such grouping, aggregate functions, and distinct operations). Such processing is indicated by the defining fullselect, in conjunction with either aggregate processing indicated by the statement references the view or table expression, or by the view or table expression that participates in a join. For views and table expressions that are defined with set operators, DB2 can often distribute aggregate processing, joins, and qualified predicates to avoid materialization.

The following table indicates some cases in which materialization occurs. DB2 can also use materialization in statements that contain multiple outer joins, outer joins that combine with inner joins, or merges that cause a join of greater than 15 tables.

Table 199. Cases when DB2 performs view or table expression materialization. The "X" indicates a case of materialization. Notes follow the table.

SELECT FROM view or table expression uses... <sup>1</sup>	View definition or table expression uses... <sup>2</sup>					UNION ALL(4)
	GROUP BY	DISTINCT	Aggregate function	Aggregate function DISTINCT	UNION	
Joins (3)	X	X	X	X	X	
GROUP BY	X	X	X	X	X	
DISTINCT		X		X	X	
Aggregate function	X	X	X	X	X	X
Aggregate function DISTINCT	X	X	X	X	X	
SELECT subset of view or table expression columns		X			X	

#### Notes to Table 199:

1. If the view is referenced as the target of an insert, update, or delete operation used to satisfy the view reference. Only updatable views can be the target in these statements.

An SQL statement can reference a particular view multiple times where some of the references can be merged and some must be materialized.

2. If a SELECT list contains a host variable in a table expression, then materialization occurs. For example:

```
SELECT C1 FROM
  (SELECT :HV1 AS C1 FROM T1) X;
```

If a view or nested table expression is defined to contain a user-defined function, and if that user-defined function is defined as NOT DETERMINISTIC or EXTERNAL ACTION, then the view or nested table expression is always materialized.

3. Additional details about materialization with outer joins:

- If a WHERE clause exists in a view or table expression, and it does not contain a column, materialization occurs.

```
SELECT X.C1 FROM
  (SELECT C1 FROM T1
   WHERE 1=1) X LEFT JOIN T2 Y
    ON X.C1=Y.C1;
```

- If the outer join is a full outer join and the SELECT list of the view or nested table expression does not contain a standalone column for the column that is used in the outer join ON clause, then materialization occurs.

```
SELECT X.C1 FROM
  (SELECT C1+10 AS C2 FROM T1) X FULL JOIN T2 Y
    ON X.C2=Y.C2;
```

- If the SELECT list of a view or nested table expression contains no column, materialization occurs.

```
SELECT X.C1 FROM
  (SELECT 1+2+:HV1 AS C1 FROM T1) X LEFT JOIN T2 Y
    ON X.C1=Y.C1;
```

- If the SELECT list of a view or nested table expression contains a CASE expression, and the result of the CASE expression is referenced in the outer query block, then materialization occurs.

### Example

```
SELECT X.C1 FROM
      T1 X LEFT JOIN
      (SELECT CASE C2 WHEN 5 THEN 10 ELSE 20 END AS YC1 FROM T2) Y
      ON X.C1 = Y.YC1;
```

4. DB2 cannot avoid materialization for UNION ALL in all cases. Some of the situations in which materialization occurs includes:
  - When the view is the operand in an outer join for which nulls are used for non-matching values, materialization occurs. This situation happens when the view is either operand in a full outer join, the right operand in a left outer join, or the left operand in a right outer join.
  - If the number of tables would exceed 225 after distribution, then distribution does not occur, and the result is materialized.
5. For INTERSECT and EXCEPT set operators, the EXPLAIN information might help to determine if the view is materialized.

PSPI

## Performance of merge versus materialization

The merge process, which is not related to the MERGE statement, performs better than materialization. For materialization, DB2 uses a table space scan to access the materialized temporary result. DB2 materializes a view or table expression only if it cannot merge.

PSPI

Materialization is a two-step process with the first step resulting in the formation of a temporary result. The smaller the temporary result, the more efficient is the second step. To reduce the size of the temporary result, DB2 attempts to evaluate certain predicates from the WHERE clause of the referencing statement at the first step of the process rather than at the second step. Only certain types of predicates qualify. First, the predicate must be a simple Boolean term predicate. Second, it must have one of the forms shown in Table 200.

Table 200. Predicate candidates for first-step evaluation

Predicate	Example
COL op constant	V1.C1 > hv1
COL IS (NOT) NULL	V1.C1 IS NOT NULL
COL (NOT) BETWEEN constant AND constant	V1.C1 BETWEEN 1 AND 10
COL (NOT) LIKE constant (ESCAPE constant)	V1.C2 LIKE 'p\%%%' ESCAPE '^'
COL IN (list)	V1.C2 IN (a,b,c)

**Note:** Where "op" is =, <>, >, <, <=, or >=, and constant is either a host variable, constant, or special register. The constants in the BETWEEN predicate need not be identical.

Implied predicates generated through predicate transitive closure are also considered for first step evaluation. PSPI

## Using EXPLAIN to determine when materialization occurs

Rows that describe the access path for both steps of the materialization process, for each reference to a view or table expression that is materialized, appear in the PLAN\_TABLE.

**PSPI** These rows describe the access path used to formulate the temporary result indicated by the view's defining fullselect, and they describe the access to the temporary result as indicated by the referencing statement. The defining fullselect can also refer to views or table expressions that need to be materialized.

When DB2 chooses materialization, TNAME contains the name of the view or table expression, and TABLE\_TYPE contains a W. A value of Q in TABLE\_TYPE for the name of a view or nested table expression indicates that the materialization was virtual and not actual. (Materialization can be virtual when the view or nested table expression definition contains a UNION ALL that is not distributed.) When DB2 chooses merge, EXPLAIN data for the merged statement appears in PLAN\_TABLE; only the names of the base tables on which the view or table expression is defined appear.

## Example

Consider the following statements, which define a view and reference the view:

View defining statement:

```
CREATE VIEW V1DIS (SALARY, WORKDEPT) as
  (SELECT DISTINCT SALARY, WORKDEPT FROM DSN8810.EMP)
```

View referencing statement:

```
SELECT * FROM DSN8810.DEPT
  WHERE DEPTNO IN (SELECT WORKDEPT FROM V1DIS)
```

The following table shows a subset of columns in a plan table for the query.

*Table 201. Plan table output for an example with view materialization*

QBLOCKNO	PLANNO	QBLOCK_ TYPE	TNAME	TABLE_ TYPE	METHOD
1	1	SELECT	DEPT	T	0
2	1	NOCOSUB	V1DIS	W	0
2	2	NOCOSUB		?	3
3	1	NOCOSUB	EMP	T	0
3	2	NOCOSUB		?	3

Notice how TNAME contains the name of the view and TABLE\_TYPE contains W to indicate that DB2 chooses materialization for the reference to the view because of the use of SELECT DISTINCT in the view definition.

## Example

Consider the following statements, which define a view and reference the view:

View defining statement:

```
CREATE VIEW V1NODIS (SALARY, WORKDEPT) as
  (SELECT SALARY, WORKDEPT FROM DSN8810.EMP)
```

View referencing statement:


```
SELECT * FROM DSN8810.DEPT
  WHERE DEPTNO IN (SELECT WORKDEPT FROM V1NODIS)
```

If the VIEW was defined without DISTINCT, DB2 would choose merge instead of materialization. In the sample output, the name of the view does not appear in the plan table, but the table name on which the view is based does appear.

The following table shows a sample plan table for the query.

Table 202. Plan table output for an example with view merge

QBLOCKNO	PLANNO	QBLOCK_ TYPE	TNAME	TABLE_ TYPE	METHOD
1	1	SELECT	DEPT	T	0
2	1	NOCOSUB	EMP	T	0
2	2	NOCOSUB		?	3

When DB2 avoids materialization in such cases, TABLE\_TYPE contains a Q to indicate that DB2 uses an intermediate result that is not materialized, and TNAME shows the name of this intermediate result as DSNWFQB(xx), where xx is the number of the query block that produced the result. 

### Using EXPLAIN to determine UNION, INTERSECT, and EXCEPT activity and query rewrite

For each reference to a view or table expression that is defined with a UNION ALL statement, DB2 tries to rewrite the query into a logically equivalent statement with improved performance.



- DB2 rewrites the queries in the following manner:
- Distributing qualified predicates, joins, and aggregations across the subselects of UNION ALL. Such distribution helps to avoid materialization. No distribution is performed for UNION, INTERSECT, EXCEPT, INTERSECT ALL, or EXCEPT ALL.
- Eliminating unnecessary subselects of a view or table expression that was created by a UNION ALL operation. For DB2 to eliminate subselects, the referencing query and the view or table definition must have predicates that are based on common columns.

The QBLOCK\_TYPE column in the plan table indicates set operator activity. If a set operation was used, the column contains one of the values shown in the table below.

Table 203. Meanings of the set operator values for the QBLOCK\_TYPE column of PLAN\_TABLE

QBLOCK_TYPE value	Set operator
UNION	UNION
UNIONA	UNION ALL
INTERS	INTERSECT
INTERA	INTERSECT ALL
EXCEPT	EXCEPT
EXCEPTA	EXCEPT ALL

When the value of QBLOCK\_TYPE is set to UNION, INTERSECT, or EXCEPT, the METHOD column on the same row is set to 3 and the SORTC\_UNIQ column is set to 'Y' to indicate that a sort is necessary to remove duplicates. As with other views and table expressions, the plan table also shows when DB2 uses materialization instead of merge.

**Example:** Consider the following statements, which define a view by using the UNION ALL operator, reference that view, and demonstrate how DB2 can rewrite the referencing statement.

The statement that defines the view uses data from three tables of weekly data to create the view:

```
CREATE VIEW V1 (CUSTNO, CHARGES, DATE) as
  SELECT CUSTNO, CHARGES, DATE
    FROM WEEK1
   WHERE DATE BETWEEN '01/01/2006' And '01/07/2006'
 UNION ALL
  SELECT CUSTNO, CHARGES, DATE
    FROM WEEK2
   WHERE DATE BETWEEN '01/08/2006' And '01/14/2006'
 UNION ALL
  SELECT CUSTNO, CHARGES, DATE
    FROM WEEK3
   WHERE DATE BETWEEN '01/15/2006' And '01/21/2006';
```

Another statement references the view to find the average charges for each customer in California during the first and third Friday of January 2006:

```
SELECT V1.CUSTNO, AVG(V1.CHARGES)
  FROM CUST, V1
 WHERE CUST.CUSTNO=V1.CUSTNO
    AND CUST.STATE='CA'
    AND DATE IN ('01/07/2006','01/21/2006')
 GROUP BY V1.CUSTNO;
```


DB2 can rewrite the statement (assuming that CHARGES is defined as NOT NULL):

```
SELECT CUSTNO_U, SUM(SUM_U)/SUM(CNT_U)
  FROM
    ( SELECT WEEK1.CUSTNO, SUM(CHARGES), COUNT(CHARGES)
      FROM CUST, WEEK1
     Where CUST.CUSTNO=WEEK1.CUSTNO AND CUST.STATE='CA'
        AND DATE BETWEEN '01/01/2006' And '01/07/2006'
        AND DATE IN ('01/07/2006','01/21/2006')
      GROUP BY WEEK1.CUSTNO
    UNION ALL
    SELECT WEEK3.CUSTNO, SUM(CHARGES), COUNT(CHARGES)
      FROM CUST,WEEK3
     WHERE CUST.CUSTNO=WEEK3 AND CUST.STATE='CA'
        AND DATE BETWEEN '01/15/2006' And '01/21/2006'
        AND DATE IN ('01/07/2006','01/21/2006')
      GROUP BY WEEK3.CUSTNO
    ) AS X(CUSTNO_U,SUM_U,CNT_U)
 GROUP BY CUSNTO_U;
```

The following table shows a subset of columns in a plan table for the query.

Table 204. Plan table output for an example with a view with UNION ALL operations

QBLOCKNO	PLANNO	TNAME	TABLE_ TYPE	METHOD	QBLOCK_ TYPE	PARENT_ QBLOCK
1	1	DSNWFQB(02)	Q	0		0
1	2		?	3		0
2	1		?	0	UNIONA	1
3	1	CUST	T	0		2
3	2	WEEK1	T	1		2
4	1	CUST	T	0		2
4	2	WEEK3	T	2		2

Notice how DB2 eliminates the second subselect of the view definition from the rewritten query and how the plan table indicates this removal by showing a UNION ALL for only the first and third subselect in the view definition. The Q in the TABLE\_TYPE column indicates that DB2 does not materialize the view. 

## Interpreting query parallelism

You can examine plan table data to determine whether DB2 chooses access paths that take advantage of parallel processing.

 PSPI

To understand the likelihood that DB2 chooses parallelism, examine your PLAN\_TABLE output. This information describes a method for examining PLAN\_TABLE columns for parallelism and provides several examples.

 PSPI

## A method for examining PLAN\_TABLE columns for parallelism

You can use the plan table columns to determine whether DB2 uses parallel processing to process a query.

The steps for interpreting the output for parallelism are as follows:

### 1. Determine the likelihood that DB2 chooses parallelism:

For each query block (QBLOCKNO) in a query (QUERYNO), a non-null value in ACCESS\_DEGREE or JOIN\_DEGREE indicates that some degree of parallelism is planned.

### 2. Identify the parallel groups in the query:

All steps (PLANNO) with the same value for ACCESS\_PGROUP\_ID, JOIN\_PGROUP\_ID, SORTN\_PGROUP\_ID, or SORTC\_PGROUP\_ID indicate that a set of operations are in the same parallel group. Usually, the set of operations involves various types of join methods and sort operations. Parallel group IDs can appear in the same row of PLAN\_TABLE output, or in different rows, depending on the operation being performed. The examples in “PLAN\_TABLE examples showing parallelism” on page 723 help clarify this concept.

### 3. Identify the parallelism mode:

The column PARALLELISM\_MODE tells you the kind of parallelism that is planned: I for query I/O, C for query CP, and X for Sysplex query. Within a query block, you cannot have a mixture of “I” and “C” parallel modes. However, a statement that uses more than one query block, such as a UNION, can have “I” for one query block and “C” for another. You can have a mixture of “C” and “X” modes in a query block, but not in the same parallel group.

If the statement was bound while this DB2 is a member of a data sharing group, the PARALLELISM\_MODE column can contain “X” even if only this one DB2 member is active. This lets DB2 take advantage of extra processing power that might be available at execution time. If other members are not available at execution time, then DB2 runs the query within the single DB2 member.

## PLAN\_TABLE examples showing parallelism

For these examples, the other values would not change whether the PARALLELISM\_MODE is I, C, or X.

- **Example 1: single table access**

Assume that DB2 decides at bind time to initiate three concurrent requests to retrieve data from table T1. Part of PLAN\_TABLE appears as shown in Table 205. If DB2 decides not to use parallel operations for a step, ACCESS\_DEGREE and ACCESS\_PGROUP\_ID contain null values.

Table 205. Part of PLAN\_TABLE for single table access

TNAME	METHOD	ACCESS_DEGREE	ACCESS_PGROUP_ID	JOIN_DEGREE	JOIN_PGROUP_ID	SORTC_PGROUP_ID	SORTN_PGROUP_ID
T1	0	3	1	(null)	(null)	(null)	(null)

- **Example 2: nested loop join**

Consider a query that results in a series of nested loop joins for three tables, T1, T2 and T3. T1 is the outermost table, and T3 is the innermost table. DB2 decides at bind time to initiate three concurrent requests to retrieve data from each of the three tables. Each request accesses part of T1 and all of T2 and T3. For the nested loop join method with sort, all the retrievals are in the same parallel group except for star join with ACCESTYPE=T (sparse index). Part of PLAN\_TABLE appears as shown in Table 206:

Table 206. Part of PLAN\_TABLE for a nested loop join

TNAME	METHOD	ACCESS_DEGREE	ACCESS_PGROUP_ID	JOIN_DEGREE	JOIN_PGROUP_ID	SORTC_PGROUP_ID	SORTN_PGROUP_ID
T1	0	3	1	(null)	(null)	(null)	(null)
T2	1	3	1	3	1	(null)	(null)
T3	1	3	1	3	1	(null)	(null)

- **Example 3: merge scan join**

Consider a query that causes a merge scan join between two tables, T1 and T2. DB2 decides at bind time to initiate three concurrent requests for T1 and six concurrent requests for T2. The scan and sort of T1 occurs in one parallel group. The scan and sort of T2 occurs in another parallel group. Furthermore, the

merging phase can potentially be done in parallel. Here, a third parallel group is used to initiate three concurrent requests on each intermediate sorted table. Part of PLAN\_TABLE appears as shown in Table 207:

Table 207. Part of PLAN\_TABLE for a merge scan join

TNAME	METHOD	ACCESS_ DEGREE	ACCESS_ PGROUP_ ID	JOIN_ DEGREE	JOIN_ PGROUP_ ID	SORTC_ PGROUP_ ID	SORTN_ PGROUP_ ID
T1	0	3	d	(null)	(null)	d	(null)
T2	2	6	2	3	3	d	d

In a multi-table join, DB2 might also execute the sort for a composite that involves more than one table in a parallel task. DB2 uses a cost basis model to determine whether to use parallel sort in all cases. When DB2 decides to use parallel sort, SORTC\_PGROUP\_ID and SORTN\_PGROUP\_ID indicate the parallel group identifier. Consider a query that joins three tables, T1, T2, and T3, and uses a merge scan join between T1 and T2, and then between the composite and T3. If DB2 decides, based on the cost model, that all sorts in this query are to be performed in parallel, part of PLAN\_TABLE appears as shown in Table 208:

Table 208. Part of PLAN\_TABLE for a multi-table, merge scan join

TNAME	METHOD	ACCESS_ DEGREE	ACCESS_ PGROUP_ ID	JOIN_ DEGREE	JOIN_ PGROUP_ ID	SORTC_ PGROUP_ ID	SORTN_ PGROUP_ ID
T1	0	3	1	(null)	(null)	(null)	(null)
T2	2	6	2	6	3	1	2
T3	2	6	4	6	5	3	4

- **Example 4: hybrid join**

Consider a query that results in a hybrid join between two tables, T1 and T2. Furthermore, T1 needs to be sorted; as a result, in PLAN\_TABLE the T2 row has SORTC\_JOIN=Y. DB2 decides at bind time to initiate three concurrent requests for T1 and six concurrent requests for T2. Parallel operations are used for a join through a clustered index of T2.

Because the T2 RID can be retrieved by initiating concurrent requests on the clustered index, the joining phase is a parallel step. The retrieval of the T2 RID and the T2 rows are in the same parallel group. Part of PLAN\_TABLE appears as shown in Table 209:

Table 209. Part of PLAN\_TABLE for a hybrid join

TNAME	METHOD	ACCESS_ DEGREE	ACCESS_ PGROUP_ ID	JOIN_ DEGREE	JOIN_ PGROUP_ ID	SORTC_ PGROUP_ ID	SORTN_ PGROUP_ ID
T1	0	3	1	(null)	(null)	(null)	(null)
T2	4	6	2	6	2	1	(null)

---

## Chapter 40. Estimating the cost of a SQL statement

You can use EXPLAIN to populate a statement table, *owner*. DSN\_STATEMENT\_TABLE, at the same time as your PLAN\_TABLE is being populated.

**PSPI** DB2 provides cost estimates, in service units and in milliseconds, for SELECT, INSERT, UPDATE, and DELETE statements, both static and dynamic. The estimates do not take into account several factors, including cost adjustments that are caused by parallel processing, or the use of triggers or user-defined functions.

Use the information provided in the statement table to:

- Determine if a statement is not going to perform within range of your service-level agreements and to tune accordingly.

DB2 puts its cost estimate into one of two *cost categories*: category A or category B. Estimates that go into cost category A are the ones for which DB2 has adequate information to make an estimate. That estimate is not likely to be 100% accurate, but is likely to be more accurate than any estimate that is in cost category B.

DB2 puts estimates into cost category B when it is forced to use default values for its estimates, such as when no statistics are available, or because host variables are used in a query.

- Give a system programmer a basis for entering service-unit values by which to govern dynamic statements with predictive governing. **PSPI**

---

### Cost categories

DB2 uses cost categories to differentiate estimates for which adequate information is available from those for which it is not.

**PSPI** You probably wouldn't want to spend a lot of time tuning a query based on estimates that are returned in cost category B, because the actual cost could be radically different based on such things as what value is in a host variable, or how many levels of nested triggers and user-defined functions exist.

Similarly, if system administrators use these estimates as input into the resource limit specification table for governing (either predictive or reactive), they probably would want to give much greater latitude for statements in cost category B than for those in cost category A.


Because of the uncertainty involved, category B statements are also good candidates for reactive governing.

#### Cost category A

DB2 puts everything that doesn't fall into category B into category A.

## Cost category B

DB2 puts a statement's estimate into cost category B when any of the following conditions exist:

- The statement has UDFs.
- Triggers are defined for the target table:
  - The statement uses an insert operation, and insert triggers are defined on the target table.
  - The statement uses an update operation, and update triggers are defined on the target table.
  - The statement uses a delete operation, and delete triggers are defined on the target table.
- The target table of a delete statement has referential constraints defined on it as the parent table, and the delete rules are either CASCADE or SET NULL.
- The WHERE clause predicate has one of the following forms:
  - COL op constant, and the constant is a host variable, parameter marker, or special register. The operator can be >, >=, <, <=, LIKE, or NOT LIKE.
  - COL BETWEEN constant AND constant where either constant is a host variable, parameter marker, or special register.
  - LIKE with an escape clause that contains a host variable.
- The cardinality statistics are missing for one or more tables that are used in the statement.
- A subselect in the SQL statement contains a HAVING clause. 

---

## Part 6. Tuning DB2 for z/OS performance

You can sometimes improve DB2 performance by modifying statistics in the catalog, and coding SQL statements in a certain way to improve access path selection.



---

## Chapter 41. Providing cost information, for accessing user-defined table functions, to DB2

User-defined table functions add additional access cost to the execution of an SQL statement. For DB2 to factor in the effect of user-defined table functions in the selection of the best access path for an SQL statement, the total cost of the user-defined table function must be determined.

The total cost of a table function consists of the following three components:

- The initialization cost that results from the first call processing
- The cost that is associated with acquiring a single row
- The final call cost that performs the clean up processing

These costs, though, are not known to DB2 when I/O costs are added to the CPU cost.

To assist DB2 in determining the cost of user-defined table functions, you can use four fields in SYSIBM.SYSROUTINES. Use the following fields to provide cost information:

- IOS\_PER\_INVOC for the estimated number of I/Os per row
- INSTS\_PER\_INVOC for the estimated number of instructions
- INITIAL\_IOS for the estimated number of I/Os performed the first and last time the function is invoked
- INITIAL\_INSTS for the estimated number of instructions for the first and last time the function is invoked

These values, along with the CARDINALITY value of the table being accessed, are used by DB2 to determine the cost. The results of the calculations can influence such things as the join sequence for a multi-table join and the cost estimates generated for and used in predictive governing.

Determine values for the four fields by examining the source code for the table function. Estimate the I/Os by examining the code executed during the FIRST call and FINAL call. Look for the code executed during the OPEN, FETCH, and CLOSE calls. The costs for the OPEN and CLOSE calls can be amortized over the expected number of rows returned. Estimate the I/O cost by providing the number of I/Os that are issued. Include the I/Os for any file access.

Calculate the instruction cost by counting the number of high level instructions executed in the user-defined table function and multiplying it by a factor of 20. For assembler programs, the instruction cost is the number of assembler instructions.



**Example:** The following statement shows how these fields can be updated. The authority to update is the same authority as that required to update any catalog statistics column.

```
UPDATE SYSIBM.SYSROUTINES SET
  IOS_PER_INVOC   = 0.0,
  INSTS_PER_INVOC = 4.5E3,
  INITIAL_IOS     = 2.0
  INITIAL_INSTS   = 1.0E4,
  CARDINALITY     = 5E3
```

```
WHERE  
  SCHEMA = 'SYSADM' AND  
  SPECIFICNAME = 'FUNCTION1' AND  
  ROUTINETYPE = 'F';
```



---

## Chapter 42. Improving index and table space access

Statistics from the DB2 catalog help determine the most economical access path.

The statistics that are described in these topics are used to determine index access cost and are found in the corresponding columns of the SYSIBM.SYSINDEXES catalog table.

The statistics show distribution of data within the allocated space, from which you can judge clustering and the need to reorganize.

Space utilization statistics can also help you make sure that access paths that use the index or table space are as efficient as possible. By reducing gaps between leaf pages in an index, or to ensure that data pages are close together, you can reduce sequential I/Os.

**Recommendation:** To provide the most accurate data, gather statistics routinely to provide data about table spaces and indexes over a period of time.

**Recommendation:** Run RUNSTATS some time **after** reorganizing the data or indexes. By gathering the statistics after you reorganize, you ensure that access paths reflect a more “average” state of the data.

---

### How clustering affects access path selection

In general, CLUSTERRATIOF gives an indication of how closely the order of the index entries on the index leaf pages matches the actual ordering of the rows on the data pages.

The closer CLUSTERRATIOF is to 100%, the more closely the ordering of the index entries matches the actual ordering of the rows on the data pages. The actual formula is quite complex and accounts for indexes with many duplicates; in general, for a given index, the more duplicates, the higher the CLUSTERRATIOF value.

Here are some things to remember about the effect of CLUSTERRATIOF on access paths:

- CLUSTERRATIOF is an important input to the cost estimates that are used to determine whether an index is used for an access path, and, if so, which index to use.
- If the access is INDEXONLY, then this value does not apply.
- The higher the CLUSTERRATIOF value, the lower the cost of referencing data pages during an index scan is.
- For an index that has a CLUSTERRATIOF less than 80%, sequential prefetch is not used to access the data pages.
- A slight reduction in CLUSTERRATIOF for a table with a large number of rows can represent a much more significant number of unclustered rows than for a table with a small number of rows.

**Example:** A CLUSTERRATIOF of 99% for a table with 100 000 000 rows represents 100 000 unclustered rows. Whereas, the CLUSTERRATIOF of 95% for a table with 100 000 rows represents 5000 unclustered rows.

- For indexes that contain either many duplicate key values or key values that are highly clustered in reverse order, cost estimation that is based purely on CLUSTERRATIOF can lead to repetitive index scans. In the worst case, an entire page could be scanned one time for each row in the page. DB2 access path selection can avoid this performance problem by using a cost estimation formula based on the DATAREPEATFACTORF statistic to choose indexes. Whether DB2 will use this statistic depends on the value of the STATCLUS subsystem parameter.

Figure 113 shows an index scan on an index with a high cluster ratio. Compare that with Figure 114 on page 733, which shows an index scan on an index with a low cluster ratio.

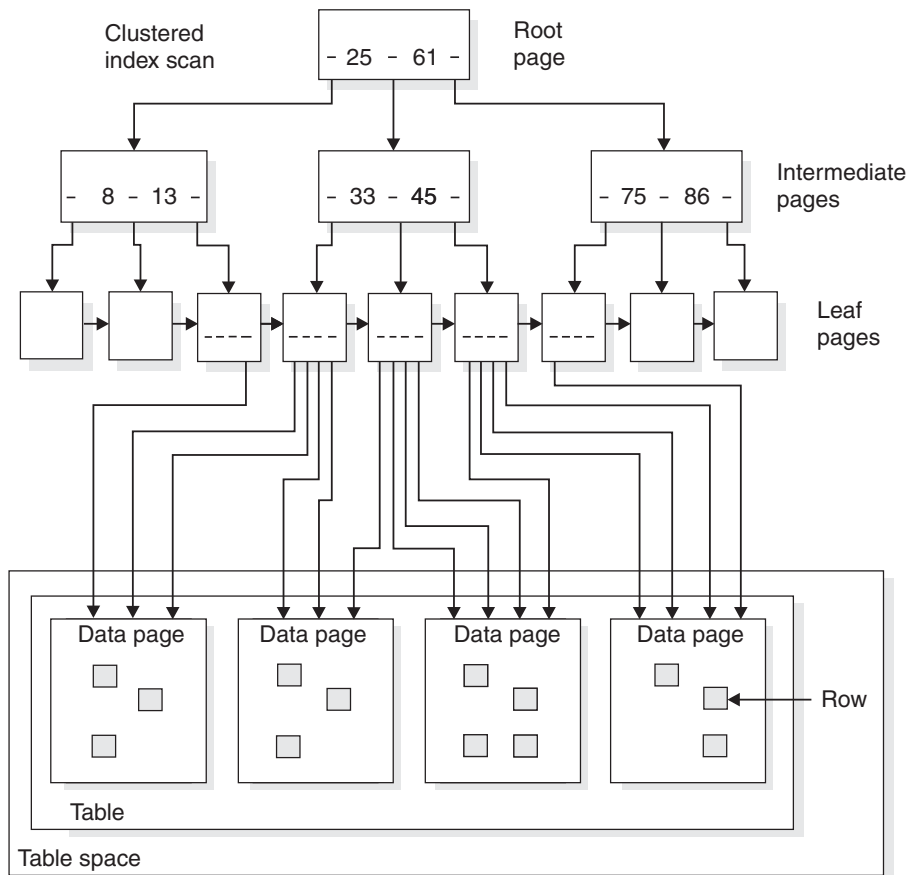


Figure 113. A clustered index scan. This figure assumes that the index is 100% clustered.

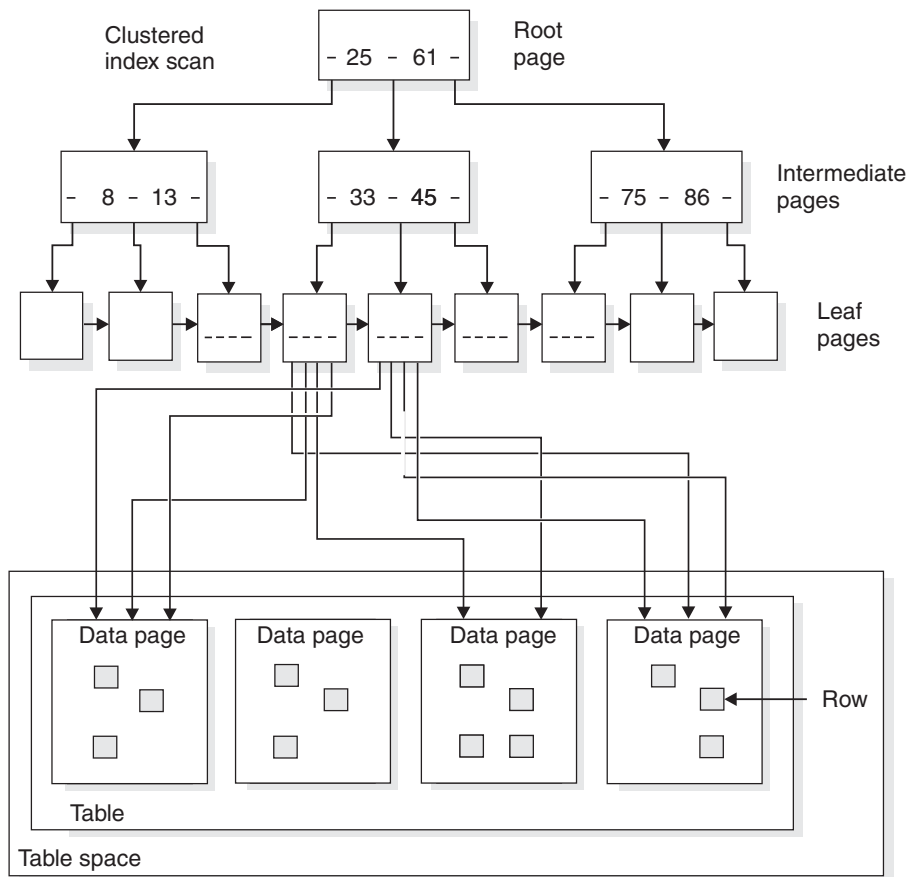


Figure 114. A nonclustered index scan. In some cases, DB2 can access the data pages in order even when a nonclustered index is used.

## When to reorganize indexes and table spaces

Data that is organized well physically can improve the performance of access paths that rely on index or data scans. Well-organized data can also help reduce the amount of disk storage used by the index or table space.

If your main reason for reorganizing is performance, the best way to determine when to reorganize is to watch your statistics for increased I/O, getpages, and processor consumption. When performance degrades to an unacceptable level, analyze the statistics described in the guidelines in this information to help you develop your own rules for when to reorganize in your particular environment. Consider the following general guidelines for when to run the REORG utility:

- **Using useful catalog queries:** Catalog queries you can use to help you determine when to reorganize are included in DSNTESP in SDSNSAMP and can be used as input to SPUFI.
- **Using the REORG utility to determine whether to reorganize:** The REORG utility imbeds the function of catalog queries. If a query returns a certain result (you can use the default or supply your own), REORG either reorganizes or does not reorganize. Optionally, you can have REORG run a report instead of actually doing the reorganization.

The REORG options that imbed these catalog queries are:

- OFFPOSLIMIT and INDREFLIMIT options of REORG TABLESPACE

– LEAFDISTLIMIT of REORG INDEX

The REORG utility does not imbed any function to help you determine when to reorganize LOB table spaces.

- **Reorganizing after table definition changes:** Another time to consider reorganizing data to improve performance is when ALTER TABLE statements have been used to add a column to the table or to change the data types or the lengths of existing columns. Such changes cause the table space to be placed in advisory REORG-pending (AREO\*) status.

In the case of changing the definition of existing column, the table space is placed in AREO\* status because the existing data is not immediately converted to its new definition. Reorganizing the table space causes the rows to be reloaded with the data converted to the new definition. Until the table space is reorganized, the changes must be tracked and applied as the data is accessed, possibly degrading performance. For example, depending on the number of changes, you might see decreased performance for dynamic SQL queries, updates and deletes, other ALTER statements (especially those that are run concurrently). In addition, running multiple REORG and LOAD utilities concurrently might perform slower or create timeouts. Unloading a table might also take longer if the table that has had many changes prior to it being reorganized.

**Related concepts**

Chapter 31, “When to reorganize indexes and table spaces,” on page 501

## Reorganizing Indexes

Fields in the SYSIBM.SYSINDEXSPACESTATS table can help you determine when to reorganize an index.

Consider running REORG INDEX when any of the following conditions are true. These conditions are based on values in the SYSIBM.SYSINDEXSPACESTATS table.

- REORGPSEUDODELETES/TOTALENTRIES > 10% in a data sharing environment, or REORGPSEUDODELETES/TOTALENTRIES > 5% in a data sharing environment.
- REORGLEAFFAR/NACTIVE > 10%
- REORGINSETS/TOTALENTRIES > 25%
- REORGDELETES/TOTALENTRIES > 25%
- REORGAPPENDINSERT/TOTALENTRIES > 20%
- EXTENTS > 254

You should also consider running REORG if one of the following conditions are true:

- The index is in the advisory REORG-pending state (AREO\*) as a result of an ALTER statement.
- An index is in the advisory REBUILD-pending state (ARBDP) as a result an ALTER statement.

### LEAFNEAR and LEAFFAR columns:

The LEAFNEAR and LEAFFAR columns of SYSIBM.SYSINDEXPART measure the disorganization of physical leaf pages by indicating the number of pages that are not in an optimal position.

Leaf pages can have page gaps whenever index pages are deleted or when an insert that cannot fit onto a full page causes an index leaf page that cannot fit onto

a full page. If the key cannot fit on the page, DB2 moves half the index entries onto a new page, which might be far away from the “home” page.

Figure 115 shows the logical and physical view of an index.

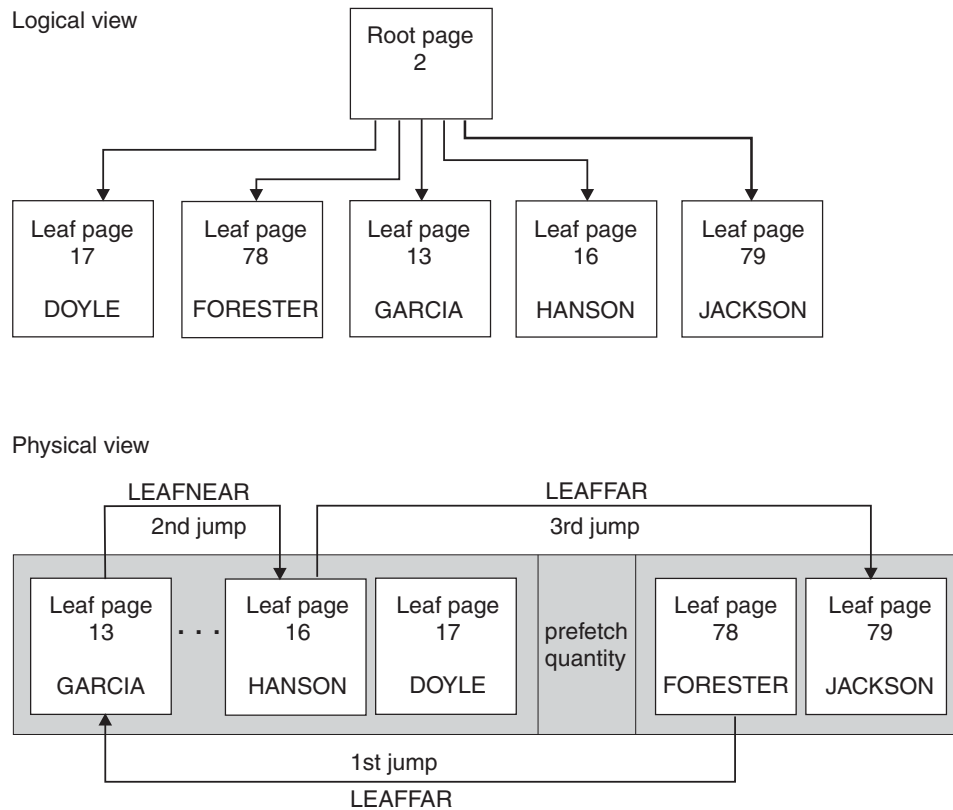


Figure 115. Logical and physical views of an index in which LEAFNEAR=1 and LEAFFAR=2

The logical view at the top of the figure shows that for an index scan four leaf pages need to be scanned to access the data for FORESTER through JACKSON. The physical view at the bottom of the figure shows how the pages are physically accessed. The first page is at physical leaf page 78, and the other leaf pages are at physical locations 79, 13, and 16. A jump forward or backward of more than one page represents non-optimal physical ordering. LEAFNEAR represents the number of jumps within the prefetch quantity, and LEAFFAR represents the number of jumps outside the prefetch quantity. In this example, assuming that the prefetch quantity is 32, two jumps exist outside the prefetch quantity. A jump from page 78 to page 13, and one from page 16 to page 79. Thus, LEAFFAR is 2. Because of the jump within the prefetch quantity from page 13 to page 16, LEAFNEAR is 1.

LEAFNEAR has a smaller impact than LEAFFAR because the LEAFNEAR pages, which are located within the prefetch quantity, are typically read by prefetch without incurring extra I/Os.

The optimal value of the LEAFNEAR and LEAFFAR catalog columns is zero. However, immediately after you run the REORG and gather statistics, LEAFNEAR for a large index might be greater than zero. A non-zero value could be caused by free pages that result from the FREEPAGE option on CREATE INDEX, non-leaf pages, or various system pages; the jumps over these pages are included in LEAFNEAR.

## Reorganizing table spaces

Fields in the SYSIBM.SYSTABLESPACESTATS table can help you determine when to reorganize a table space.

Consider running REORG TABLESPACE when any of the following conditions are true. These conditions are based on values in the SYSIBM.SYSTABLESPACESTATS table.

- $\text{REORGUNCLUSTINS} / \text{TOTALROWS} > 10\%$   
Do not use REORGUNCLUSTINS to determine if you should run REORG if access to the table space is predominantly random access.
- $(\text{REORGNEARINDREF} + \text{REORGFARINDREF}) / \text{TOTALROWS} > 5\%$  in a data sharing environment, or  $(\text{REORGNEARINDREF} + \text{REORGFARINDREF}) / \text{TOTALROWS} > 10\%$  in non-data sharing environment
- $\text{REORGINSERTS} / \text{TOTALROWS} > 25\%$
- $\text{REORGDELETES} / \text{TOTALROWS} > 25\%$
- $\text{EXTENTS} > 254$
- $\text{REORGDISORGLGB} / \text{TOTALROWS} > 50\%$
- $\text{SPACE} > 2 * (\text{DATASIZE} / 1024)$
- $\text{REORGMASDELETE} > 0$

You should also consider running REORG if one of the following conditions are true:

- The table space is in the advisory REORG-pending state (AREO\*) as a result of an ALTER TABLE statement.
- An index on a table in the table space is in the advisory REBUILD-pending state (ARBDP) as result an ALTER TABLE statement.

## Reorganizing LOB table spaces

You can reorganize the data in LOB table spaces to maintain good performance from DB2.

SYSIBM.SYSLOBSTATS contains information about how the data in the table space is physically stored. Consider running REORG on the LOB table space when the percentage in the ORGRATIO column is less than 80.

Additionally, you can use real-time statistics to identify DB2 objects that should be reorganized, have their statistics updated, or be image copied.

## Whether to rebind after gathering statistics

You do not always need to rebind all applications after you gather statistics. A rebind is necessary only if the access path statistics change significantly from the last time you bound the applications and if performance suffers as a result.

When performance degrades to an unacceptable level, analyze the statistics described in the recommendations in this information to help you develop your own guidelines for when to rebind.

Consider the following guidelines regarding when to rebind:

- CLUSTERRATIOF changes to less or more than 80% (a value of 0.80).
- NLEAF changes more than 20% from the previous value.
- NLEVELS changes (only if it was more than a two-level index to begin with).
- NPAGES changes more than 20% from the previous value.

- NACTIVEF changes more than 20% from the previous value.
- The range of HIGH2KEY to LOW2KEY range changes more than 20% from the range previously recorded.
- Cardinality changes more than 20% from previous range.
- Distribution statistics change the majority of the frequent column values.



---

## Chapter 43. Updating the catalog

If you have sufficient privileges, you can change all of statistics values that DB2 uses for access path selection.

To modify statistics values:

Issue an UPDATE statement. If you change values in the catalog and later run RUNSTATS to update those values, your changes are lost.

**Recommendation:** Keep track of the changes you make and of the plans or packages that have an access path change due to changed statistics.

---

### Correlations in the catalog

Relationships exist among certain columns in DB2 catalog tables.

Correlations exist amongst columns in the following catalog tables:

- Columns within table SYSCOLUMNS
- Columns in the tables SYSCOLUMNS and SYSINDEXES
- Columns in the tables SYSCOLUMNS and SYSCOLDIST
- Columns in the tables SYSCOLUMNS, SYSCOLDIST, and SYSINDEXES
- Columns with table space statistics and columns for partition-level statistics, as described in “Statistics for partitioned table spaces” on page 166.

If you plan to update some values, keep in mind the following correlations:

- COLCARDF and FIRSTKEYCARDF. For a column that is the first column of an index, those two values are equal. If the index has only that one column, the two values are also equal to the value of FULLKEYCARDF.
- COLCARDF, LOW2KEY, and HIGH2KEY. If the COLCARDF value is not '-1', DB2 assumes that statistics exist for the column. In particular, it uses the values of LOW2KEY and HIGH2KEY in calculating filter factors. If COLCARDF = 1 or if COLCARDF = 2, DB2 uses HIGH2KEY and LOW2KEY as domain statistics, and generates frequencies on HIGH2KEY and LOW2KEY.
- CARDF in SYSCOLDIST. CARDF is related to COLCARDF in SYSIBM.SYSCOLUMNS and to FIRSTKEYCARDF and FULLKEYCARDF in SYSIBM.SYSINDEXES. CARDF must be the minimum of the following:
  - A value between FIRSTKEYCARDF and FULLKEYCARDF if the index contains the same set of columns
  - A value between MAX(COLCARDF of each column in the column group) and the product of multiplying together the COLCARDF of each column in the column group

**Example:** Assume the following set of statistics:

CARDF = 1000  
NUMCOLUMNS = 3  
COLGROUPCOLNO = 2,3,5

INDEX1 on columns 2,3,5,7,8  
FIRSTKEYCARDF = 100  
FULLKEYCARDF = 10000

CARDF must be between 100  
and 10000

```
column 2 COLCARDF = 100  
column 3 COLCARDF = 50  
column 5 COLCARDF = 10
```

The range between FIRSTKEYCARDF and FULLKEYCARDF is 100 and 10 000. The maximum of the COLCARDF values is 50 000. Thus, the allowable range is between 100 and 10 000.

- CARDF in SYSTABLES. CARDF must be equal to or larger than any of the other cardinalities, SUCH AS COLCARDF, FIRSTKEYCARDF, FULLKEYCARDF, and CARDF in SYSIBM.SYSCOLDIST.
- FREQUENCYF and COLCARDF or CARDF. The number of frequencies collected must be less than or equal to COLCARDF for the column or CARDF for the column group.
- FREQUENCYF. The sum of frequencies collected for a column or column group must be less than or equal to 1.

---

## Recommendation for COLCARDF and FIRSTKEYCARDF

On partitioned indexes, RUNSTATS INDEX calculates the number of distinct column values and saves it in SYSCOLSTATS.COLCARD, by partition.

When the statistics by partition are used to form the aggregate, the aggregate might not be exact because some column values might occur in more than one partition. Without scanning all parts of the index, DB2 cannot detect that overlap. The overlap never skews COLCARD by more than the number of partitions, which is usually not a problem for large values. For small values, update the aggregate COLCARDF value in SYSCOLUMNS because DB2 uses the COLCARDF value when determining access paths.

The exception and the remedy for COLCARD and COLCARDF are also true for the FIRSTKEYCARDF column in SYSIBM.SYSINDEXES and the FIRSTKEYCARDF column in SYSIBM.SYSINDEXSTATS.

---

## Recommendation for HIGH2KEY and LOW2KEY

If you update the COLCARDF value for a column, you must also update HIGH2KEY and LOW2KEY for the column.

HIGH2KEY and LOW2KEY are defined as VARCHAR(2000); thus, an UPDATE statement must provide a character or hexadecimal value. Entering a character value is quite straightforward: SET LOW2KEY = 'ALAS', for instance. But to enter a numeric, date, or time value you must use the hexadecimal value of the DB2 internal format. Be sure to allow for a null indicator in keys that allow nulls.

If you update the COLCARDF value for a column, also update HIGH2KEY and LOW2KEY for the column. HIGH2KEY and LOW2KEY are defined as VARCHAR(2000). To update HIGH2KEY and LOW2KEY:

- Specify a character or hexadecimal value on the UPDATE statement.  
Entering a character value is quite straightforward: SET LOW2KEY = 'ALAS', for example. But to enter a numeric, date, or time value you must use the hexadecimal value of the DB2 internal format. Be sure to allow for a null indicator in keys that allow nulls.
- Ensure that the padding characteristics for the columns are the same as the padding characteristics for COLCARDF.

For example, if the statistics values that are collected for COLCARDF are padded, then the statistics collected for HIGH2KEY or LOW2KEY must also be padded. You can check the STATS\_FORMAT column in SYSCOLUMNS and SYSCOLSTATS to determine whether the statistics gathered for these columns are padded or not.

---

## Statistics for distributions

Statistics for distributions are stored in the SYSCOLDIST, SYSCOLDISTSTATS, SYSKEYTGTDISTSTATS and SYSKEYTGTDIST catalog tables.

| Although you can use RUNSTATS to collect distribution statistics on any column  
| or group of columns, by default, DB2 inserts the 10 most frequent values for the  
| first key column of an index as well as the first and last key values.

You can insert, update, or delete distribution information for any column in these catalog tables. However, to enter a numeric, date, or time value you must use the hexadecimal value of the DB2 internal format. Be sure to allow for a null indicator in keys that allow nulls. See *DB2 Administration Guide*.

---

## Recommendation for using the TIMESTAMP column

Statistics gathered by RUNSTATS include timestamps. Every row updated or inserted during a particular invocation of RUNSTATS contains the same timestamp value.

So that you can always determine when they were last updated:

Update the STATTIME column whenever you update statistics in the catalog.



---

## Chapter 44. Tuning parallel processing

Much of the information in these topics also applies also to Sysplex query parallelism.

A parallel group can run at a parallel degree less than that shown in the PLAN\_TABLE output. The following factors can cause a reduced degree of parallelism:

- Buffer pool availability
- Logical contention.

Consider a nested loop join. The inner table could be in a partitioned or nonpartitioned table space, but DB2 is more likely to use a parallel join operation when the outer table is partitioned.

- Physical contention
- Run-time host variables

A host variable can determine the qualifying partitions of a table for a given query. In such cases, DB2 defers the determination of the planned degree of parallelism until run time, when the host variable value is known.

- Updatable cursor

At run time, DB2 might determine that an ambiguous cursor is updatable.

- A change in the configuration of online processors

If fewer processors are online at run time, DB2 might need to reformulate the parallel degree.

If many parallel groups do not run at the planned degree (see **B** in Figure 88 on page 603), check the following factors:

- Buffer pool availability

Depending on buffer pool availability, DB2 could reduce the degree of parallelism (see **C** in Figure 88 on page 603) or revert to a sequential plan before executing the parallel group (**F** in the figure).

To determine which buffer pool is short on storage, see section QW0221C in IFCID 0221.

**GUI** You can use the ALTER BUFFERPOOL command to increase the buffer pool space available for parallel operations by modifying the following parameters:

- VPSIZE, the size of the virtual buffer pool
- VPSEQT, the sequential steal threshold
- VPPSEQT, the parallel sequential threshold
- VPXPSEQT, the assisting parallel sequential threshold, used only for Sysplex query parallelism.

### **GUI**

If the buffer pool is busy with parallel operations, the sequential prefetch quantity might also be reduced.

The parallel sequential threshold also has an impact on **work file processing** for parallel queries. DB2 assumes that you have all your work files of the same size (4KB or 32KB) in the same buffer pool and makes run time decisions based on a

single buffer pool. A lack of buffer pool resources for the work files can lead to a reduced degree of parallelism or cause the query to run sequentially.

If increasing the parallel thresholds does not help solve the problem of reduced degree, you can increase the total buffer pool size (VPSIZE). Use information from the statistics trace to determine the amount of buffer space you need. Use the following formula:

$$(\text{QBSTJIS} / \text{QBSTPQF}) \times 32 = \text{buffer increase value}$$

QBSTJIS is the total number of requested prefetch I/O streams that were denied because of a storage shortage in the buffer pool. (There is one I/O stream per parallel task.) QBSTPQF is the total number of times that DB2 could not allocate enough buffer pages to allow a parallel group to run to the planned degree.

As an example, assume QBSTJIS is 100 000 and QBSTPQF is 2500:

$$(100000 / 2500) \times 32 = 1280$$

Use ALTER BUFFERPOOL to increase the current VPSIZE by 2560 buffers to alleviate the degree degradation problem. Use the DISPLAY BUFFERPOOL command to see the current VPSIZE.

- Physical contention

As much as possible, put data partitions on separate physical devices to minimize contention. Try not to use a number of partitions greater than the number of internal paths in the controller.

- Run time host variables

A host variable can determine the qualifying partitions of a table for a given query. In such cases, DB2 defers the determination of the planned degree of parallelism until run time, when the host variable value is known.

- Updatable cursor

At run time, DB2 might determine that an ambiguous cursor is updatable. This appears in **D** in the accounting report.

- Proper hardware and software support

If you do not have the hardware sort facility at run time, and a sort merge join is needed, you see a value in **E**.

- A change in the configuration of online processors

If fewer online processors are available at run time than at bind time, DB2 reformulates the parallel degree to take best advantage of the current processing power. This reformulation is indicated by a value in **H** in the accounting report.

**Locking considerations for repeatable read applications:** For CP parallelism, locks are obtained independently by each task. Be aware that this situation can possibly increase the total number of locks taken for applications that:

- Use an isolation level of repeatable read
- Use CP parallelism
- Repeatedly access the table space using a lock mode of IS without issuing COMMITs

**Recommendation:** As is recommended for all repeatable-read applications, issue frequent COMMIT statements to release the lock resources that are held. Repeatable read or read stability isolation cannot be used with Sysplex query parallelism.

---

## Chapter 45. Disabling query parallelism

You can prevent DB2 from using parallel processing.

To disable parallel operations, do any of the following actions:

- For static SQL, rebind to change the option DEGREE(ANY) to DEGREE(1). You can do this by using the DB2I panels, the DSN subcommands, or the DSNH CLIST. The default is DEGREE(1).

- **GUI** For dynamic SQL, execute the following SQL statement:

```
SET CURRENT DEGREE = '1';
```

The default value for CURRENT DEGREE is 1 unless your installation has changed the default for the CURRENT DEGREE special register. **GUI**

- For performance tuning, Set the parallel sequential threshold (VPPSEQT) to 0.
- For performance tuning, add a row to your resource limit facility's specification table (RLST) for your plan, package, or authorization ID with the RLFFUNC value set to "3" to disable I/O parallelism, "4" to disable CP parallelism, or "5" to disable Sysplex query parallelism. To disable all types of parallelism, you need a row for all three types (assuming that Sysplex query parallelism is enabled on your system.) In a system with a very high processor utilization rate (that is, greater than 98 percent), I/O parallelism might be a better choice because of the increase in processor overhead with CP parallelism. In this case, you could disable CP parallelism for your dynamic queries by putting a "4" in the resource limit specification table for the plan or package.

If you have a Sysplex, you might want to use a "5" to disable Sysplex query parallelism, depending on how high processor utilization is in the members of the data sharing group.

To determine if parallelism has been disabled by a value in your resource limit specification table (RLST), look for a non-zero value in field QXRLFDPA in IFCID 0002 or 0003. The QW0022RP field in IFCID 0022 indicates whether this particular statement was disabled.



---

## Part 7. Appendixes



---

## Information resources for DB2 for z/OS and related products

Many information resources are available to help you use DB2 for z/OS and many related products. A large amount of technical information about IBM products is now available online in information centers or on library Web sites.

**Disclaimer:** Any Web addresses that are included here are accurate at the time this information is being published. However, Web addresses sometimes change. If you visit a Web address that is listed here but that is no longer valid, you can try to find the current Web address for the product information that you are looking for at either of the following sites:

- <http://www.ibm.com/support/publications/us/library/index.shtml>, which lists the IBM information centers that are available for various IBM products
- <http://www.elink.ibm.link.ibm.com/public/applications/publications/cgibin/pbi.cgi>, which is the IBM Publications Center, where you can download online PDF books or order printed books for various IBM products

### DB2 for z/OS product information

The primary place to find and use information about DB2 for z/OS is the Information Management Software for z/OS Solutions Information Center (<http://publib.boulder.ibm.com/infocenter/imzic>), which also contains information about IMS, QMF, and many DB2 and IMS Tools products. The majority of the DB2 for z/OS information in this information center is also available in the books that are identified in the following table. You can access these books at the DB2 for z/OS library Web site (<http://www.ibm.com/software/data/db2/zos/library.html>) or at the IBM Publications Center (<http://www.elink.ibm.link.ibm.com/public/applications/publications/cgibin/pbi.cgi>).

Table 210. DB2 Version 9.1 for z/OS book titles

Title	Publication number	Available in information center	Available in PDF	Available in BookManager® format	Available in printed book
<i>DB2 Version 9.1 for z/OS Administration Guide</i>	SC18-9840	X	X	X	X
<i>DB2 Version 9.1 for z/OS Application Programming &amp; SQL Guide</i>	SC18-9841	X	X	X	X
<i>DB2 Version 9.1 for z/OS Application Programming Guide and Reference for Java</i>	SC18-9842	X	X	X	X
<i>DB2 Version 9.1 for z/OS Codes</i>	GC18-9843	X	X	X	X
<i>DB2 Version 9.1 for z/OS Command Reference</i>	SC18-9844	X	X	X	X
<i>DB2 Version 9.1 for z/OS Data Sharing: Planning and Administration</i>	SC18-9845	X	X	X	X
<i>DB2 Version 9.1 for z/OS Diagnosis Guide and Reference</i> <sup>1</sup>	LY37-3218		X	X	X
<i>DB2 Version 9.1 for z/OS Diagnostic Quick Reference</i>	LY37-3219				X

Table 210. DB2 Version 9.1 for z/OS book titles (continued)

Title	Publication number	Available in information center	Available in PDF	Available in BookManager® format	Available in printed book
<i>DB2 Version 9.1 for z/OS Installation Guide</i>	GC18-9846	X	X	X	X
<i>DB2 Version 9.1 for z/OS Introduction to DB2</i>	SC18-9847	X	X	X	X
<i>DB2 Version 9.1 for z/OS Licensed Program Specifications</i>	GC18-9848		X		X
<i>DB2 Version 9.1 for z/OS Messages</i>	GC18-9849	X	X	X	X
<i>DB2 Version 9.1 for z/OS ODBC Guide and Reference</i>	SC18-9850	X	X	X	X
<i>DB2 Version 9.1 for z/OS Performance Monitoring and Tuning Guide</i>	SC18-9851	X	X	X	X
<i>DB2 Version 9.1 for z/OS Optimization Service Center online help</i>	none	X			
<i>DB2 Version 9.1 for z/OS Program Directory</i>	GI10-8737		X		X
<i>DB2 Version 9.1 for z/OS RACF Access Control Module Guide</i>	SC18-9852		X	X	
<i>DB2 Version 9.1 for z/OS Reference for Remote DRDA Requesters and Servers</i>	SC18-9853	X	X	X	
<i>DB2 Version 9.1 for z/OS Reference Summary</i>	SX26-3854		X		X
<i>DB2 Version 9.1 for z/OS SQL Reference</i>	SC18-9854	X	X	X	X
<i>DB2 Version 9.1 for z/OS Utility Guide and Reference</i>	SC18-9855	X	X	X	X
<i>DB2 Version 9.1 for z/OS What's New?</i>	GC18-9856	X	X	X	X
<i>DB2 Version 9.1 for z/OS XML Extender Administration and Programming</i>	SC18-9857	X	X	X	X
<i>DB2 Version 9.1 for z/OS XML Guide</i>	SC18-9858	X	X	X	X

**Notes:**

1. *DB2 Version 9.1 for z/OS Diagnosis Guide and Reference* is available in PDF and BookManager formats on the DB2 Version 9.1 for z/OS Licensed Collection kit, LK3T-7195. You can order this License Collection kit on the IBM Publications Center site (<http://www.elink.ibm.link.ibm.com/public/applications/publications/cgibin/pbi.cgi>). This book is also available in online format in DB2 data set DSN910.SDSNIVPD(DSNDR).

**Information resources for related products**

In the following table, related product names are listed in alphabetic order, and the associated Web addresses of product information centers or library Web pages are indicated.

Table 211. Related product information resource locations

Related product	Information resources
C/C++ for z/OS	Library Web site: <a href="http://www.ibm.com/software/awdtools/czos/library/">http://www.ibm.com/software/awdtools/czos/library/</a>  This product is now called z/OS XL C/C++.
CICS Transaction Server for z/OS	Information center: <a href="http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/index.jsp">http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/index.jsp</a>
COBOL	Information center: <a href="http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp">http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp</a>  This product is now called Enterprise COBOL for z/OS.
DB2 Connect	Information center: <a href="http://publib.boulder.ibm.com/infocenter/db2luw/v9//index.jsp">http://publib.boulder.ibm.com/infocenter/db2luw/v9//index.jsp</a>  This resource is for DB2 Connect 9.
DB2 Database for Linux, UNIX, and Windows	Information center: <a href="http://publib.boulder.ibm.com/infocenter/db2luw/v9//index.jsp">http://publib.boulder.ibm.com/infocenter/db2luw/v9//index.jsp</a>  This resource is for DB2 9 for Linux, UNIX, and Windows.
DB2 Performance Expert for z/OS	Information center: <a href="http://publib.boulder.ibm.com/infocenter/imzic">http://publib.boulder.ibm.com/infocenter/imzic</a>  This product is now called DB2 Tivoli OMEGAMON for XE Performance Expert on z/OS.
DB2 Query Management Facility	Information center: <a href="http://publib.boulder.ibm.com/infocenter/imzic">http://publib.boulder.ibm.com/infocenter/imzic</a>
DB2 Server for VSE & VM VSE	One of the following locations: <ul style="list-style-type: none"> <li>For VSE: <a href="http://www.ibm.com/support/docview.wss?rs=66&amp;uid=swg27003758">http://www.ibm.com/support/docview.wss?rs=66&amp;uid=swg27003758</a></li> <li>For VM: <a href="http://www.ibm.com/support/docview.wss?rs=66&amp;uid=swg27003759">http://www.ibm.com/support/docview.wss?rs=66&amp;uid=swg27003759</a></li> </ul>
DB2 Tools	One of the following locations: <ul style="list-style-type: none"> <li>Information center: <a href="http://publib.boulder.ibm.com/infocenter/imzic">http://publib.boulder.ibm.com/infocenter/imzic</a></li> <li>Library Web site: <a href="http://www.ibm.com/software/data/db2imstools/library.html">http://www.ibm.com/software/data/db2imstools/library.html</a></li> </ul> <p>These resources include information about the following products and others:</p> <ul style="list-style-type: none"> <li>DB2 Administration Tool</li> <li>DB2 Automation Tool</li> <li>DB2 DataPropagator™ (also known as WebSphere Replication Server for z/OS)</li> <li>DB2 Log Analysis Tool</li> <li>DB2 Object Restore Tool</li> <li>DB2 Query Management Facility</li> <li>DB2 SQL Performance Analyzer</li> <li>DB2 Tivoli OMEGAMON for XE Performance Expert on z/OS (includes Buffer Pool Analyzer and Performance Monitor)</li> </ul>
DB2 Universal Database™ for iSeries™	Information center: <a href="http://www.ibm.com/systems/i/infocenter/">http://www.ibm.com/systems/i/infocenter/</a>
Debug Tool for z/OS	Information center: <a href="http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp">http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp</a>
Enterprise COBOL for z/OS	Information center: <a href="http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp">http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp</a>
Enterprise PL/I for z/OS	Information center: <a href="http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp">http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp</a>
IMS	Information center: <a href="http://publib.boulder.ibm.com/infocenter/imzic">http://publib.boulder.ibm.com/infocenter/imzic</a>

Table 211. Related product information resource locations (continued)

Related product	Information resources
IMS Tools	<p>One of the following locations:</p> <ul style="list-style-type: none"> <li>• Information center: <a href="http://publib.boulder.ibm.com/infocenter/imzic">http://publib.boulder.ibm.com/infocenter/imzic</a></li> <li>• Library Web site: <a href="http://www.ibm.com/software/data/db2imstools/library.html">http://www.ibm.com/software/data/db2imstools/library.html</a></li> </ul> <p>These resources have information about the following products and others:</p> <ul style="list-style-type: none"> <li>• IMS Batch Terminal Simulator for z/OS</li> <li>• IMS Connect</li> <li>• IMS HALDB Conversion and Maintenance Aid</li> <li>• IMS High Performance Utility products</li> <li>• IMS DataPropagator</li> <li>• IMS Online Reorganization Facility</li> <li>• IMS Performance Analyzer</li> </ul>
PL/I	<p>Information center: <a href="http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp">http://publib.boulder.ibm.com/infocenter/pdthelp/v1r1/index.jsp</a></p> <p>This product is now called Enterprise PL/I for z/OS.</p>
System z	<a href="http://publib.boulder.ibm.com/infocenter/eserver/v1r2/index.jsp">http://publib.boulder.ibm.com/infocenter/eserver/v1r2/index.jsp</a>
WebSphere Application Server	Information center: <a href="http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp">http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp</a>
WebSphere Message Broker with Rules and Formatter Extension	<p>Information center: <a href="http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp">http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp</a></p> <p>The product is also known as WebSphere MQ Integrator Broker.</p>
WebSphere MQ	<p>Information center: <a href="http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp">http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp</a></p> <p>The resource includes information about MQSeries.</p>
WebSphere Replication Server for z/OS	<p>Either of the following locations:</p> <ul style="list-style-type: none"> <li>• Information center: <a href="http://publib.boulder.ibm.com/infocenter/imzic">http://publib.boulder.ibm.com/infocenter/imzic</a></li> <li>• Library Web site: <a href="http://www.ibm.com/software/data/db2imstools/library.html">http://www.ibm.com/software/data/db2imstools/library.html</a></li> </ul> <p>This product is also known as DB2 DataPropagator.</p>
z/Architecture™	Library Center site: <a href="http://www.ibm.com/servers/eserver/zseries/zos/bkserv/">http://www.ibm.com/servers/eserver/zseries/zos/bkserv/</a>

Table 211. Related product information resource locations (continued)

Related product	Information resources
z/OS	<p>Library Center site: <a href="http://www.ibm.com/servers/eserver/zseries/zos/bkserv/">http://www.ibm.com/servers/eserver/zseries/zos/bkserv/</a></p> <p>This resource includes information about the following z/OS elements and components:</p> <ul style="list-style-type: none"> <li>• Character Data Representation Architecture</li> <li>• Device Support Facilities</li> <li>• DFSORT</li> <li>• Fortran</li> <li>• High Level Assembler</li> <li>• NetView<sup>®</sup></li> <li>• SMP/E for z/OS</li> <li>• SNA</li> <li>• TCP/IP</li> <li>• TotalStorage Enterprise Storage Server<sup>®</sup></li> <li>• VTAM</li> <li>• z/OS C/C++</li> <li>• z/OS Communications Server</li> <li>• z/OS DCE</li> <li>• z/OS DFSMS</li> <li>• z/OS DFSMS Access Method Services</li> <li>• z/OS DFSMSdss<sup>™</sup></li> <li>• z/OS DFSMSHsm</li> <li>• z/OS DFSMSdftp<sup>™</sup></li> <li>• z/OS ICSF</li> <li>• z/OS ISPF</li> <li>• z/OS JES3</li> <li>• z/OS Language Environment</li> <li>• z/OS Managed System Infrastructure</li> <li>• z/OS MVS</li> <li>• z/OS MVS JCL</li> <li>• z/OS Parallel Sysplex</li> <li>• z/OS RMF</li> <li>• z/OS Security Server</li> <li>• z/OS UNIX System Services</li> </ul>
z/OS XL C/C++	<p><a href="http://www.ibm.com/software/awdtools/czos/library/">http://www.ibm.com/software/awdtools/czos/library/</a></p>

The following information resources from IBM are not necessarily specific to a single product:

- The DB2 for z/OS Information Roadmap; available at: <http://www.ibm.com/software/data/db2/zos/roadmap.html>
- DB2 Redbooks<sup>™</sup> and Redbooks about related products; available at: <http://www.ibm.com/redbooks>
- IBM Educational resources:
  - Information about IBM educational offerings is available on the Web at: <http://www.ibm.com/software/sw-training/>

- A collection of glossaries of IBM terms in multiple languages is available on the IBM Terminology Web site at: <http://www.ibm.com/ibm/terminology/index.html>
- National Language Support information; available at the IBM Publications Center at: <http://www.elink.ibm.link.ibm.com/public/applications/publications/cgi-bin/pbi.cgi>
- *SQL Reference for Cross-Platform Development*; available at the following developerWorks® site: <http://www.ibm.com/developerworks/db2/library/techarticle/0206sqlref/0206sqlref.html>

The following information resources are not published by IBM but can be useful to users of DB2 for z/OS and related products:

- Database design topics:
  - *DB2 for z/OS and OS/390® Development for Performance Volume I*, by Gabrielle Wiorkowski, Gabrielle & Associates, ISBN 0-96684-605-2
  - *DB2 for z/OS and OS/390 Development for Performance Volume II*, by Gabrielle Wiorkowski, Gabrielle & Associates, ISBN 0-96684-606-0
  - *Handbook of Relational Database Design*, by C. Fleming and B. Von Halle, Addison Wesley, ISBN 0-20111-434-8
- Distributed Relational Database Architecture™ (DRDA) specifications; <http://www.opengroup.org>
- Domain Name System: *DNS and BIND*, Third Edition, Paul Albitz and Cricket Liu, O'Reilly, ISBN 0-59600-158-4
- Microsoft® Open Database Connectivity (ODBC) information; <http://msdn.microsoft.com/library/>
- Unicode information; <http://www.unicode.org>

---

## How to obtain DB2 information

You can access the official information about the DB2 product in a number of ways.

- “DB2 on the Web”
- “DB2 product information”
- “DB2 education” on page 756
- “How to order the DB2 library” on page 756

### DB2 on the Web

Stay current with the latest information about DB2 by visiting the DB2 home page on the Web:

[www.ibm.com/software/db2zos](http://www.ibm.com/software/db2zos)

On the DB2 home page, you can find links to a wide variety of information resources about DB2. You can read news items that keep you informed about the latest enhancements to the product. Product announcements, press releases, fact sheets, and technical articles help you plan and implement your database management strategy.

### DB2 product information

The official DB2 for z/OS information is available in various formats and delivery methods. IBM provides mid-version updates to the information in the information center and in softcopy updates that are available on the Web and on CD-ROM.

#### Information Management Software for z/OS Solutions Information Center

DB2 product information is viewable in the information center, which is the primary delivery vehicle for information about DB2 for z/OS, IMS, QMF, and related tools. This information center enables you to search across related product information in multiple languages for data management solutions for the z/OS environment and print individual topics or sets of related topics. You can also access, download, and print PDFs of the publications that are associated with the information center topics. Product technical information is provided in a format that offers more options and tools for accessing, integrating, and customizing information resources. The information center is based on Eclipse open source technology.

The Information Management Software for z/OS Solutions Information Center is viewable at the following Web site:

<http://publib.boulder.ibm.com/infocenter/imzic>

#### CD-ROMs and DVD

Books for DB2 are available on a CD-ROM that is included with your product shipment:

- DB2 V9.1 for z/OS Licensed Library Collection, LK3T-7195, in English

The CD-ROM contains the collection of books for DB2 V9.1 for z/OS in PDF and BookManager formats. Periodically, IBM refreshes the books on subsequent editions of this CD-ROM.

The books for DB2 for z/OS are also available on the following CD-ROM and DVD collection kits, which contain online books for many IBM products:

- IBM z/OS Software Products Collection , SK3T-4270, in English
- IBM z/OS Software Products DVD Collection , SK3T-4271, in English

#### **PDF format**

Many of the DB2 books are available in PDF (Portable Document Format) for viewing or printing from CD-ROM or the DB2 home page on the Web or from the information center. Download the PDF books to your intranet for distribution throughout your enterprise.

#### **BookManager format**

You can use online books on CD-ROM to read, search across books, print portions of the text, and make notes in these BookManager books. Using the IBM Softcopy Reader, appropriate IBM Library Readers, or the BookManager Read product, you can view these books in the z/OS, Windows, and VM environments. You can also view and search many of the DB2 BookManager books on the Web.

### **DB2 education**

IBM Education and Training offers a wide variety of classroom courses to help you quickly and efficiently gain DB2 expertise. IBM schedules classes in cities all over the world. You can find class information, by country, at the IBM Learning Services Web site:

[www.ibm.com/services/learning](http://www.ibm.com/services/learning)

IBM also offers classes at your location, at a time that suits your needs. IBM can customize courses to meet your exact requirements. For more information, including the current local schedule, contact your IBM representative.

### **How to order the DB2 library**

To order books, visit the IBM Publication Center on the Web:

[www.elink.ibm.link.ibm.com/public/applications/publications/cgibin/pbi.cgi](http://www.elink.ibm.link.ibm.com/public/applications/publications/cgibin/pbi.cgi)

From the IBM Publication Center, you can go to the Publication Notification System (PNS). PNS users receive electronic notifications of updated publications in their profiles. You have the option of ordering the updates by using the publications direct ordering application or any other IBM publication ordering channel. The PNS application does not send automatic shipments of publications. You will receive updated publications and a bill for them if you respond to the electronic notification.

You can also order DB2 publications and CD-ROMs from your IBM representative or the IBM branch office that serves your locality. If your location is within the United States or Canada, you can place your order by calling one of the toll-free numbers:

- In the U.S., call 1-800-879-2755.
- In Canada, call 1-800-426-4968.

To order additional copies of licensed publications, specify the SOFTWARE option. To order additional publications or CD-ROMs, specify the PUBLICATIONS option.

Be prepared to give your customer number, the product number, and either the feature codes or order numbers that you want.



---

## How to use the DB2 library

Titles of books in the library begin with DB2 Version 9.1 for z/OS. However, references from one book in the library to another are shortened and do not include the product name, version, and release. Instead, they point directly to the section that holds the information.

If you are new to DB2 for z/OS, *Introduction to DB2 for z/OS* provides a comprehensive introduction to DB2 Version 9.1 for z/OS. Topics included in this book explain the basic concepts that are associated with relational database management systems in general, and with DB2 for z/OS in particular.

The most rewarding task associated with a database management system is asking questions of it and getting answers, the task called *end use*. Other tasks are also necessary—defining the parameters of the system, putting the data in place, and so on. The tasks that are associated with DB2 are grouped into the following major categories.

### Installation

If you are involved with DB2 only to install the system, *DB2 Installation Guide* might be all you need.

If you will be using data sharing capabilities you also need *DB2 Data Sharing: Planning and Administration*, which describes installation considerations for data sharing.

### End use

End users issue SQL statements to retrieve data. They can also insert, update, or delete data, with SQL statements. They might need an introduction to SQL, detailed instructions for using SPUFI, and an alphabetized reference to the types of SQL statements. This information is found in *DB2 Application Programming and SQL Guide*, and *DB2 SQL Reference*.

End users can also issue SQL statements through the DB2 Query Management Facility (QMF) or some other program, and the library for that licensed program might provide all the instruction or reference material they need. For a list of the titles in the DB2 QMF library, see the bibliography at the end of this book.

### Application programming

Some users access DB2 without knowing it, using programs that contain SQL statements. DB2 application programmers write those programs. Because they write SQL statements, they need the same resources that end users do.

Application programmers also need instructions for many other topics:

- How to transfer data between DB2 and a host program—written in Java™, C, or COBOL, for example
- How to prepare to compile a program that embeds SQL statements
- How to process data from two systems simultaneously, for example, DB2 and IMS or DB2 and CICS

- How to write distributed applications across operating systems
- How to write applications that use Open Database Connectivity (ODBC) to access DB2 servers
- How to write applications that use JDBC and SQLJ with the Java programming language to access DB2 servers
- How to write applications to store XML data on DB2 servers and retrieve XML data from DB2 servers.

The material needed for writing a host program containing SQL is in *DB2 Application Programming and SQL Guide*.

The material needed for writing applications that use JDBC and SQLJ to access DB2 servers is in *DB2 Application Programming Guide and Reference for Java*. The material needed for writing applications that use DB2 CLI or ODBC to access DB2 servers is in *DB2 ODBC Guide and Reference*. The material needed for working with XML data in DB2 is in *DB2 XML Guide*. For handling errors, see *DB2 Messages* and *DB2 Codes*.

If you will be working in a distributed environment, you will need *DB2 Reference for Remote DRDA Requesters and Servers*.

Information about writing applications across operating systems can be found in *IBM DB2 SQL Reference for Cross-Platform Development*.

## System and database administration

*Administration* covers almost everything else. *DB2 Administration Guide* divides some of those tasks among the following sections:

- DB2 concepts: Introduces DB2 structures, the DB2 environment, and high availability.
- Designing a database: Discusses the decisions that must be made when designing a database and tells how to implement the design by creating and altering DB2 objects, loading data, and adjusting to changes.
- Security and auditing: Describes ways of controlling access to the DB2 system and to data within DB2, to audit aspects of DB2 usage, and to answer other security and auditing concerns.
- Operation and recovery: Describes the steps in normal day-to-day operation and discusses the steps one should take to prepare for recovery in the event of some failure.

*DB2 Performance Monitoring and Tuning Guide* explains how to monitor the performance of the DB2 system and its parts. It also lists things that can be done to make some parts run faster.

If you will be using the RACF access control module for DB2 authorization checking, you will need *DB2 RACF Access Control Module Guide*.

If you are involved with DB2 only to design the database, or plan operational procedures, you need *DB2 Administration Guide*. If you also want to carry out your own plans by creating DB2 objects, granting privileges, running utility jobs, and so on, you also need:

- *DB2 SQL Reference*, which describes the SQL statements you use to create, alter, and drop objects and grant and revoke privileges
- *DB2 Utility Guide and Reference*, which explains how to run utilities

- *DB2 Command Reference*, which explains how to run commands

If you will be using data sharing, you need *DB2 Data Sharing: Planning and Administration*, which describes how to plan for and implement data sharing.

Additional information about system and database administration can be found in *DB2 Messages* and *DB2 Codes*, which list messages and codes issued by DB2, with explanations and suggested responses.

## **Diagnosis**

Diagnostics detect and describe errors in the DB2 program. They might also recommend or apply a remedy. The documentation for this task is in *DB2 Diagnosis Guide and Reference*, *DB2 Messages*, and *DB2 Codes*.



---

## Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106-0032, Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation  
J46A/G4  
555 Bailey Avenue  
San Jose, CA 95141-1003  
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

---

## Programming Interface Information

This information is intended to help you to plan for and administer DB2 Version 9.1 for z/OS. This information also documents General-use Programming Interface and Associated Guidance Information and Product-sensitive Programming Interface and Associated Guidance Information provided by DB2 Version 9.1 for z/OS.

### General-use Programming Interface and Associated Guidance Information

General-use Programming Interfaces allow the customer to write programs that obtain the services of DB2 Version 9.1 for z/OS.

General-use Programming Interface and Associated Guidance Information is identified where it occurs by the following markings:



General-use Programming Interface and Associated Guidance Information...



### Product-sensitive Programming Interface and Associated Guidance Information

Product-sensitive Programming Interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of this IBM software product. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive Programming Interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

Product-sensitive Programming Interface and Associated Guidance Information is identified where it occurs by the following markings:



Product-sensitive Programming Interface and Associated Guidance Information...



---

## Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com)<sup>®</sup> are trademarks or registered marks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (<sup>®</sup> or <sup>™</sup>), indicating trademarks that were owned by IBM at the time this information was published. A complete and current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of other companies, and have been used at least once in the DB2 for z/OS library:

- Microsoft, Windows, Windows NT<sup>®</sup>, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
- Intel<sup>®</sup>, Intel logo, Intel Inside<sup>®</sup>, Intel Inside logo, Intel Centrino<sup>®</sup>, Intel Centrino logo, Celeron<sup>®</sup>, Intel Xeon<sup>®</sup>, Intel SpeedStep<sup>®</sup>, Itanium<sup>®</sup>, and Pentium<sup>®</sup> are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Adobe<sup>®</sup>, the Adobe logo, Postscript, and the Postscript log are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Other company, product, or service names may be trademarks or service marks of others.

---

## Glossary

**abend** See abnormal end of task.

**abend reason code**

A 4-byte hexadecimal code that uniquely identifies a problem with DB2.

**abnormal end of task (abend)**

Termination of a task, job, or subsystem because of an error condition that recovery facilities cannot resolve during execution.

**access method services**

The facility that is used to define, alter, delete, print, and reproduce VSAM key-sequenced data sets.

**access path**

The path that is used to locate data that is specified in SQL statements. An access path can be indexed or sequential.

**active log**

The portion of the DB2 log to which log records are written as they are generated. The active log always contains the most recent log records. See also archive log.

**address space**

A range of virtual storage pages that is identified by a number (ASID) and a collection of segment and page tables that map the virtual pages to real pages of the computer's memory.

**address space connection**

The result of connecting an allied address space to DB2. See also allied address space and task control block.

**address space identifier (ASID)**

A unique system-assigned identifier for an address space.

**AFTER trigger**

A trigger that is specified to be activated after a defined trigger event (an insert, update, or delete operation on the table that is specified in a trigger definition). Contrast with BEFORE trigger and INSTEAD OF trigger.

**agent** In DB2, the structure that associates all processes that are involved in a DB2 unit of work. See also allied agent and system agent.

**aggregate function**

An operation that derives its result by using values from one or more rows. Contrast with scalar function.

**alias**

An alternative name that can be used in SQL statements to refer to a table or view in the same or a remote DB2 subsystem. An alias can be qualified with a schema qualifier and can thereby be referenced by other users. Contrast with synonym.

**allied address space**

An area of storage that is external to DB2 and that is connected to DB2. An allied address space can request DB2 services. See also address space.

**allied agent**

An agent that represents work requests that originate in allied address spaces. See also system agent.

**allied thread**

A thread that originates at the local DB2 subsystem and that can access data at a remote DB2 subsystem.

**allocated cursor**

A cursor that is defined for a stored procedure result set by using the SQL ALLOCATE CURSOR statement.

**ambiguous cursor**

A database cursor for which DB2 cannot determine whether it is used for update or read-only purposes.

**APAR** See authorized program analysis report.

**APF** See authorized program facility.

**API** See application programming interface.

**APPL** A VTAM network definition statement that is used to define DB2 to VTAM as an application program that uses SNA LU 6.2 protocols.

**application**

A program or set of programs that performs a task; for example, a payroll application.

**application plan**

The control structure that is produced during the bind process. DB2 uses the

application plan to process SQL statements that it encounters during statement execution.

**application process**

The unit to which resources and locks are allocated. An application process involves the execution of one or more programs.

**application programming interface (API)**

A functional interface that is supplied by the operating system or by a separately orderable licensed program that allows an application program that is written in a high-level language to use specific data or functions of the operating system or licensed program.

**application requester**

The component on a remote system that generates DRDA requests for data on behalf of an application.

**application server**

The target of a request from a remote application. In the DB2 environment, the application server function is provided by the distributed data facility and is used to access DB2 data from remote applications.

**archive log**

The portion of the DB2 log that contains log records that have been copied from the active log. See also active log.

**ASCII** An encoding scheme that is used to represent strings in many environments, typically on PCs and workstations. Contrast with EBCDIC and Unicode.

**ASID** See address space identifier.

**attachment facility**

An interface between DB2 and TSO, IMS, CICS, or batch address spaces. An attachment facility allows application programs to access DB2.

**attribute**

A characteristic of an entity. For example, in database design, the phone number of an employee is an attribute of that employee.

**authorization ID**

A string that can be verified for connection to DB2 and to which a set of privileges is allowed. An authorization ID can represent an individual or an organizational group.

**authorized program analysis report (APAR)**

A report of a problem that is caused by a suspected defect in a current release of an IBM supplied program.

**authorized program facility (APF)**

A facility that allows an installation to identify system or user programs that can use sensitive system functions.

**automatic bind**

(More correctly *automatic rebind*.) A process by which SQL statements are bound automatically (without a user issuing a BIND command) when an application process begins execution and the bound application plan or package it requires is not valid.

**automatic query rewrite**

A process that examines an SQL statement that refers to one or more base tables or materialized query tables, and, if appropriate, rewrites the query so that it performs better.

**auxiliary index**

An index on an auxiliary table in which each index entry refers to a LOB or XML document.

**auxiliary table**

A table that contains columns outside the actual table in which they are defined. Auxiliary tables can contain either LOB or XML data.

**backout**

The process of undoing uncommitted changes that an application process made. A backout is often performed in the event of a failure on the part of an application process, or as a result of a deadlock situation.

**backward log recovery**

The final phase of restart processing during which DB2 scans the log in a backward direction to apply UNDO log records for all aborted changes.

**base table**

A table that is created by the SQL CREATE TABLE statement and that holds persistent data. Contrast with clone table, materialized query table, result table, temporary table, and transition table.

**base table space**

A table space that contains base tables.

## basic row format

A row format in which values for columns are stored in the row in the order in which the columns are defined by the CREATE TABLE statement.

basic sequential access method (BSAM)

An access method for storing or retrieving data blocks in a continuous sequence, using either a sequential-access or a direct-access device.

**BEFORE** trigger

A trigger that is specified to be activated before a defined trigger event (an insert, an update, or a delete operation on the table that is specified in a trigger definition). Contrast with AFTER trigger and INSTEAD OF trigger.

binary large object (BLOB)

A binary string data type that contains a sequence of bytes that can range in size from 0 bytes to 2 GB, less 1 byte. This string does not have an associated code page and character set. BLOBs can contain, for example, image, audio, or video data. In general, BLOB values are used whenever a binary string might exceed the limits of the VARBINARY type.

binary string

A sequence of bytes that is not associated with a CCSID. Binary string data type can be further classified as BINARY, VARBINARY, or BLOB.

**bind** A process by which a usable control structure with SQL statements is generated; the structure is often called an access plan, an application plan, or a package. During this bind process, access paths to the data are selected, and some authorization checking is performed. See also automatic bind.

## bit data

- Data with character type CHAR or VARCHAR that is defined with the FOR BIT DATA clause. Note that using BINARY or VARBINARY rather than FOR BIT DATA is highly recommended.
- Data with character type CHAR or VARCHAR that is defined with the FOR BIT DATA clause.

- A form of character data. Binary data is generally more highly recommended than character-for-bit data.

**BLOB** See binary large object.

**block fetch**

A capability in which DB2 can retrieve, or fetch, a large set of rows together. Using block fetch can significantly reduce the number of messages that are being sent across the network. Block fetch applies only to non-rowset cursors that do not update data.

## bootstrap data set (BSDS)

A VSAM data set that contains name and status information for DB2 and RBA range specifications for all active and archive log data sets. The BSDS also contains passwords for the DB2 directory and catalog, and lists of conditional restart and checkpoint records.

## BSAM

See basic sequential access method.

**BSDS** See bootstrap data set.

buffer pool

An area of memory into which data pages are read, modified, and held during processing.

**built-in data type**

A data type that IBM supplies. Among the built-in data types for DB2 for z/OS are string, numeric, XML, ROWID, and datetime. Contrast with *distinct* type.

## built-in function

A function that is generated by DB2 and that is in the SYSIBM schema. Contrast with user-defined function. See also function, cast function, external function, sourced function, and SQL function.

**business dimension**

A category of data, such as products or time periods, that an organization might want to analyze.

## cache structure

A coupling facility structure that stores data that can be available to all members of a Sysplex. A DB2 data sharing group uses cache structures as group buffer pools.

**CAF** See call attachment facility.

**call attachment facility (CAF)**

A DB2 attachment facility for application programs that run in TSO or z/OS batch. The CAF is an alternative to the DSN command processor and provides greater control over the execution environment. Contrast with Recoverable Resource Manager Services attachment facility.

**call-level interface (CLI)**

A callable application programming interface (API) for database access, which is an alternative to using embedded SQL.

**cascade delete**

A process by which DB2 enforces referential constraints by deleting all descendent rows of a deleted parent row.

**CASE expression**

An expression that is selected based on the evaluation of one or more conditions.

**cast function**

A function that is used to convert instances of a (source) data type into instances of a different (target) data type.

**castout**

The DB2 process of writing changed pages from a group buffer pool to disk.

**castout owner**

The DB2 member that is responsible for casting out a particular page set or partition.

**catalog**

In DB2, a collection of tables that contains descriptions of objects such as tables, views, and indexes.

**catalog table**

Any table in the DB2 catalog.

**CCSID**

See coded character set identifier.

**CDB**

See communications database.

**CDRA**

See Character Data Representation Architecture.

**CEC**

See central processor complex.

**central electronic complex (CEC)**

See central processor complex.

**central processor complex (CPC)**

A physical collection of hardware that consists of main storage, one or more central processors, timers, and channels.

**central processor (CP)**

The part of the computer that contains the sequencing and processing facilities for instruction execution, initial program load, and other machine operations.

**CFRM** See coupling facility resource management.

**CFRM policy**

The allocation rules for a coupling facility structure that are declared by a z/OS administrator.

**character conversion**

The process of changing characters from one encoding scheme to another.

**Character Data Representation Architecture (CDRA)**

An architecture that is used to achieve consistent representation, processing, and interchange of string data.

**character large object (CLOB)**

A character string data type that contains a sequence of bytes that represent characters (single-byte, multibyte, or both) that can range in size from 0 bytes to 2 GB, less 1 byte. In general, CLOB values are used whenever a character string might exceed the limits of the VARCHAR type.

**character set**

A defined set of characters.

**character string**

A sequence of bytes that represent bit data, single-byte characters, or a mixture of single-byte and multibyte characters. Character data can be further classified as CHARACTER, VARCHAR, or CLOB.

**check constraint**

A user-defined constraint that specifies the values that specific columns of a base table can contain.

**check integrity**

The condition that exists when each row in a table conforms to the check constraints that are defined on that table.

**check pending**

A state of a table space or partition that prevents its use by some utilities and by some SQL statements because of rows that violate referential constraints, check constraints, or both.

**checkpoint**

A point at which DB2 records status information on the DB2 log; the recovery process uses this information if DB2 abnormally terminates.

**child lock**

For explicit hierarchical locking, a lock that is held on either a table, page, row, or a large object (LOB). Each child lock has a parent lock. See also parent lock.

**CI** See control interval.

**CICS** Represents (in this information): CICS Transaction Server for z/OS: Customer Information Control System Transaction Server for z/OS.

**CICS attachment facility**

A facility that provides a multithread connection to DB2 to allow applications that run in the CICS environment to execute DB2 statements.

**claim** A notification to DB2 that an object is being accessed. Claims prevent drains from occurring until the claim is released, which usually occurs at a commit point. Contrast with drain.

**claim class**

A specific type of object access that can be one of the following isolation levels:

- Cursor stability (CS)
- Repeatable read (RR)
- Write

**class of service**

A VTAM term for a list of routes through a network, arranged in an order of preference for their use.

**clause** In SQL, a distinct part of a statement, such as a SELECT clause or a WHERE clause.

**CLI** See call-level interface.

**client** See requester.

**CLOB** See character large object.

**clone object**

An object that is associated with a clone table, including the clone table itself and check constraints, indexes, and BEFORE triggers on the clone table.

**clone table**

A table that is structurally identical to a base table. The base and clone table each

have separate underlying VSAM data sets, which are identified by their data set instance numbers.

**closed application**

An application that requires exclusive use of certain statements on certain DB2 objects, so that the objects are managed solely through the external interface of that application.

**clustering index**

An index that determines how rows are physically ordered (*clustered*) in a table space. If a clustering index on a partitioned table is not a partitioning index, the rows are ordered in cluster sequence within each data partition instead of spanning partitions.

**CM** See compatibility mode.

**CM\*** See compatibility mode\*.

**C++ member**

A data object or function in a structure, union, or class.

**C++ member function**

An operator or function that is declared as a member of a class. A member function has access to the private and protected data members and to the member functions of objects in its class. Member functions are also called methods.

**C++ object**

A region of storage. An object is created when a variable is defined or a new function is invoked.

An instance of a class.

**coded character set**

A set of unambiguous rules that establish a character set and the one-to-one relationships between the characters of the set and their coded representations.

**coded character set identifier (CCSID)**

A 16-bit number that uniquely identifies a coded representation of graphic characters. It designates an encoding scheme identifier and one or more pairs that consist of a character set identifier and an associated code page identifier.

**code page**

A set of assignments of characters to code points. Within a code page, each code

point has only one specific meaning. In EBCDIC, for example, the character *A* is assigned code point X'C1', and character *B* is assigned code point X'C2'.

**code point**

In CDRA, a unique bit pattern that represents a character in a code page.

**code unit**

The fundamental binary width in a computer architecture that is used for representing character data, such as 7 bits, 8 bits, 16 bits, or 32 bits. Depending on the character encoding form that is used, each code point in a coded character set can be represented by one or more code units.

**coexistence**

During migration, the period of time in which two releases exist in the same data sharing group.

**cold start**

A process by which DB2 restarts without processing any log records. Contrast with warm start.

**collection**

A group of packages that have the same qualifier.

**column**

The vertical component of a table. A column has a name and a particular data type (for example, character, decimal, or integer).

**column function**

See aggregate function.

**"come from" checking**

An LU 6.2 security option that defines a list of authorization IDs that are allowed to connect to DB2 from a partner LU.

**command**

A DB2 operator command or a DSN subcommand. A command is distinct from an SQL statement.

**command prefix**

A 1- to 8-character command identifier. The command prefix distinguishes the command as belonging to an application or subsystem rather than to z/OS.

**command recognition character (CRC)**

A character that permits a z/OS console

operator or an IMS subsystem user to route DB2 commands to specific DB2 subsystems.

**command scope**

The scope of command operation in a data sharing group.

**commit**

The operation that ends a unit of work by releasing locks so that the database changes that are made by that unit of work can be perceived by other processes. Contrast with rollback.

**commit point**

A point in time when data is considered consistent.

**common service area (CSA)**

In z/OS, a part of the common area that contains data areas that are addressable by all address spaces. Most DB2 use is in the extended CSA, which is above the 16-MB line.

**communications database (CDB)**

A set of tables in the DB2 catalog that are used to establish conversations with remote database management systems.

**comparison operator**

A token (such as =, >, or <) that is used to specify a relationship between two values.

**compatibility mode\* (CM\*)**

A stage of the version-to-version migration process that applies to a DB2 subsystem or data sharing group that was in enabling-new-function mode (ENFM), enabling-new-function mode\* (ENFM\*), or new-function mode (NFM) at one time. Fallback to a prior version is not supported. When in compatibility mode\*, a DB2 data sharing group cannot coexist with members that are still at the prior version level. Contrast with compatibility mode, enabling-new-function mode, enabling-new-function mode\*, and new-function mode.

**compatibility mode (CM)**

The first stage of the version-to-version migration process. In a DB2 data sharing group, members in compatibility mode can coexist with members that are still at the prior version level. Fallback to the prior version is also supported. When in compatibility mode, the DB2 subsystem

cannot use any new functions of the new version. Contrast with compatibility mode\*, enabling-new-function mode, enabling-new-function mode\*, and new-function mode.

**composite key**

An ordered set of key columns or expressions of the same table.

**compression dictionary**

The dictionary that controls the process of compression and decompression. This dictionary is created from the data in the table space or table space partition.

**concurrency**

The shared use of resources by more than one application process at the same time.

**conditional restart**

A DB2 restart that is directed by a user-defined conditional restart control record (CRCR).

**connection**

In SNA, the existence of a communication path between two partner LUs that allows information to be exchanged (for example, two DB2 subsystems that are connected and communicating by way of a conversation).

**connection context**

In SQLJ, a Java object that represents a connection to a data source.

**connection declaration clause**

In SQLJ, a statement that declares a connection to a data source.

**connection handle**

The data object containing information that is associated with a connection that DB2 ODBC manages. This includes general status information, transaction status, and diagnostic information.

**connection ID**

An identifier that is supplied by the attachment facility and that is associated with a specific address space connection.

**consistency token**

A timestamp that is used to generate the version identifier for an application. See also version.

**constant**

A language element that specifies an unchanging value. Constants are classified

as string constants or numeric constants. Contrast with variable.

**constraint**

A rule that limits the values that can be inserted, deleted, or updated in a table. See referential constraint, check constraint, and unique constraint.

**context**

An application's logical connection to the data source and associated DB2 ODBC connection information that allows the application to direct its operations to a data source. A DB2 ODBC context represents a DB2 thread.

**contracting conversion**

A process that occurs when the length of a converted string is smaller than that of the source string. For example, this process occurs when an EBCDIC mixed-data string that contains DBCS characters is converted to ASCII mixed data; the converted string is shorter because the shift codes are removed.

**control interval (CI)**

- A unit of information that VSAM transfers between virtual and auxiliary storage.
- In a key-sequenced data set or file, the set of records that an entry in the sequence-set index record points to.

**conversation**

Communication, which is based on LU 6.2 or Advanced Program-to-Program Communication (APPC), between an application and a remote transaction program over an SNA logical unit-to-logical unit (LU-LU) session that allows communication while processing a transaction.

**coordinator**

The system component that coordinates the commit or rollback of a unit of work that includes work that is done on one or more other systems.

**coprocessor**

See SQL statement coprocessor.

**copy pool**

A collection of names of storage groups that are processed collectively for fast replication operations.

**copy target**

A named set of SMS storage groups that are to be used as containers for copy pool volume copies. A copy target is an SMS construct that lets you define which storage groups are to be used as containers for volumes that are copied by using FlashCopy functions.

**copy version**

A point-in-time FlashCopy copy that is managed by HSM. Each copy pool has a version parameter that specifies the number of copy versions to be maintained on disk.

**correlated columns**

A relationship between the value of one column and the value of another column.

**correlated subquery**

A subquery (part of a WHERE or HAVING clause) that is applied to a row or group of rows of a table or view that is named in an outer subselect statement.

**correlation ID**

An identifier that is associated with a specific thread. In TSO, it is either an authorization ID or the job name.

**correlation name**

An identifier that is specified and used within a single SQL statement as the exposed name for objects such as a table, view, table function reference, nested table expression, or result of a data change statement. Correlation names are useful in an SQL statement to allow two distinct references to the same base table and to allow an alternative name to be used to represent an object.

**cost category**

A category into which DB2 places cost estimates for SQL statements at the time the statement is bound. The cost category is externalized in the COST\_CATEGORY column of the DSN\_STATEMENT\_TABLE when a statement is explained.

**coupling facility**

A special PR/SM logical partition (LPAR) that runs the coupling facility control program and provides high-speed caching, list processing, and locking functions in a Parallel Sysplex.

**coupling facility resource management (CFRM)**

A component of z/OS that provides the

services to manage coupling facility resources in a Parallel Sysplex. This management includes the enforcement of CFRM policies to ensure that the coupling facility and structure requirements are satisfied.

**CP** See central processor.

**CPC** See central processor complex.

**CRC** See command recognition character.

**created temporary table**

A persistent table that holds temporary data and is defined with the SQL statement CREATE GLOBAL TEMPORARY TABLE. Information about created temporary tables is stored in the DB2 catalog and can be shared across application processes. Contrast with declared temporary table. See also temporary table.

**cross-system coupling facility (XCF)**

A component of z/OS that provides functions to support cooperation between authorized programs that run within a Sysplex.

**cross-system extended services (XES)**

A set of z/OS services that allow multiple instances of an application or subsystem, running on different systems in a Sysplex environment, to implement high-performance, high-availability data sharing by using a coupling facility.

**CS** See cursor stability.

**CSA** See common service area.

**CT** See cursor table.

**current data**

Data within a host structure that is current with (identical to) the data within the base table.

**current status rebuild**

The second phase of restart processing during which the status of the subsystem is reconstructed from information on the log.

**cursor** A control structure that an application program uses to point to a single row or multiple rows within some ordered set of rows of a result table. A cursor can be used to retrieve, update, or delete rows from a result table.

**cursor sensitivity**

The degree to which database updates are visible to the subsequent FETCH statements in a cursor.

**cursor stability (CS)**

The isolation level that provides maximum concurrency without the ability to read uncommitted data. With cursor stability, a unit of work holds locks only on its uncommitted changes and on the current row of each of its cursors. See also read stability, repeatable read, and uncommitted read.

**cursor table (CT)**

The internal representation of a cursor.

**cycle** A set of tables that can be ordered so that each table is a descendent of the one before it, and the first table is a descendent of the last table. A self-referencing table is a cycle with a single member. See also referential cycle.

**database**

A collection of tables, or a collection of table spaces and index spaces.

**database access thread (DBAT)**

A thread that accesses data at the local subsystem on behalf of a remote subsystem.

**database administrator (DBA)**

An individual who is responsible for designing, developing, operating, safeguarding, maintaining, and using a database.

**database alias**

The name of the target server if it is different from the location name. The database alias is used to provide the name of the database server as it is known to the network.

**database descriptor (DBD)**

An internal representation of a DB2 database definition, which reflects the data definition that is in the DB2 catalog. The objects that are defined in a database descriptor are table spaces, tables, indexes, index spaces, relationships, check constraints, and triggers. A DBD also contains information about accessing tables in the database.

**database exception status**

In a data sharing environment, an indication that something is wrong with a database.

**database identifier (DBID)**

An internal identifier of the database.

**database management system (DBMS)**

A software system that controls the creation, organization, and modification of a database and the access to the data that is stored within it.

**database request module (DBRM)**

A data set member that is created by the DB2 precompiler and that contains information about SQL statements. DBRMs are used in the bind process.

**database server**

The target of a request from a local application or a remote intermediate database server.

**data currency**

The state in which the data that is retrieved into a host variable in a program is a copy of the data in the base table.

**data dictionary**

A repository of information about an organization's application programs, databases, logical data models, users, and authorizations.

**data partition**

A VSAM data set that is contained within a partitioned table space.

**data-partitioned secondary index (DPSI)**

A secondary index that is partitioned according to the underlying data. Contrast with nonpartitioned secondary index.

**data set instance number**

A number that indicates the data set that contains the data for an object.

**data sharing**

A function of DB2 for z/OS that enables applications on different DB2 subsystems to read from and write to the same data concurrently.

**data-sharing group**

A collection of one or more DB2

subsystems that directly access and change the same data while maintaining data integrity.

**data sharing member**

A DB2 subsystem that is assigned by XCF services to a data sharing group.

**data source**

A local or remote relational or non-relational data manager that is capable of supporting data access via an ODBC driver that supports the ODBC APIs. In the case of DB2 for z/OS, the data sources are always relational database managers.

**data type**

An attribute of columns, constants, variables, parameters, special registers, and the results of functions and expressions.

**data warehouse**

A system that provides critical business information to an organization. The data warehouse system cleanses the data for accuracy and currency, and then presents the data to decision makers so that they can interpret and use it effectively and efficiently.

**DBA** See database administrator.

**DBAT** See database access thread.

**DB2 catalog**

A collection of tables that are maintained by DB2 and contain descriptions of DB2 objects, such as tables, views, and indexes.

**DBCLOB**

See double-byte character large object.

**DB2 command**

An instruction to the DB2 subsystem that a user enters to start or stop DB2, to display information on current users, to start or stop databases, to display information on the status of databases, and so on.

**DBCS** See double-byte character set.

**DBD** See database description.

**DB2I** See DB2 Interactive.

**DBID** See database identifier.

**DB2 Interactive (DB2I)**

An interactive service within DB2 that

facilitates the execution of SQL statements, DB2 (operator) commands, and programmer commands, and the invocation of utilities.

**DBMS**

See database management system.

**DBRM**

See database request module.

**DB2 thread**

The database manager structure that describes an application's connection, traces its progress, processes resource functions, and delimits its accessibility to the database manager resources and services. Most DB2 for z/OS functions execute under a thread structure.

**DCLGEN**

See declarations generator.

**DDF** See distributed data facility.

**deadlock**

Unresolvable contention for the use of a resource, such as a table or an index.

**declarations generator (DCLGEN)**

A subcomponent of DB2 that generates SQL table declarations and COBOL, C, or PL/I data structure declarations that conform to the table. The declarations are generated from DB2 system catalog information.

**declared temporary table**

A non-persistent table that holds temporary data and is defined with the SQL statement DECLARE GLOBAL TEMPORARY TABLE. Information about declared temporary tables is not stored in the DB2 catalog and can be used only by the application process that issued the DECLARE statement. Contrast with created temporary table. See also temporary table.

**default value**

A predetermined value, attribute, or option that is assumed when no other value is specified. A default value can be defined for column data in DB2 tables by specifying the DEFAULT keyword in an SQL statement that changes data (such as INSERT, UPDATE, and MERGE).

**deferred embedded SQL**

SQL statements that are neither fully static nor fully dynamic. These statements

are embedded within an application and are prepared during the execution of the application.

**deferred write**

The process of asynchronously writing changed data pages to disk.

**degree of parallelism**

The number of concurrently executed operations that are initiated to process a query.

**delete hole**

The location on which a cursor is positioned when a row in a result table is refetched and the row no longer exists on the base table. See also update hole.

**delete rule**

The rule that tells DB2 what to do to a dependent row when a parent row is deleted. Delete rules include CASCADE, RESTRICT, SET NULL, or NO ACTION.

**delete trigger**

A trigger that is defined with the triggering delete SQL operation.

**delimited identifier**

A sequence of characters that are enclosed within escape characters.

**delimiter token**

A string constant, a delimited identifier, an operator symbol, or any of the special characters that are shown in DB2 syntax diagrams.

**denormalization**

The intentional duplication of columns in multiple tables to increase data redundancy. Denormalization is sometimes necessary to minimize performance problems. Contrast with normalization.

**dependent**

An object (row, table, or table space) that has at least one parent. The object is also said to be a dependent (row, table, or table space) of its parent. See also parent row, parent table, and parent table space.

**dependent row**

A row that contains a foreign key that matches the value of a primary key in the parent row.

**dependent table**

A table that is a dependent in at least one referential constraint.

**descendent**

An object that is a dependent of an object or is the dependent of a descendent of an object.

**descendent row**

A row that is dependent on another row, or a row that is a descendent of a dependent row.

**descendent table**

A table that is a dependent of another table, or a table that is a descendent of a dependent table.

**deterministic function**

A user-defined function whose result is dependent on the values of the input arguments. That is, successive invocations with the same input values produce the same answer. Sometimes referred to as a *not-variant* function. Contrast with nondeterministic function (sometimes called a *variant function*).

**dimension**

A data category such as time, products, or markets. The elements of a dimension are referred to as members. See also dimension table.

**dimension table**

The representation of a dimension in a star schema. Each row in a dimension table represents all of the attributes for a particular member of the dimension. See also dimension, star schema, and star join.

**directory**

The DB2 system database that contains internal objects such as database descriptors and skeleton cursor tables.

**disk**

A direct-access storage device that records data magnetically.

**distinct type**

A user-defined data type that is represented as an existing type (its source type), but is considered to be a separate and incompatible type for semantic purposes.

**distributed data**

Data that resides on a DBMS other than the local system.

**distributed data facility (DDF)**

A set of DB2 components through which DB2 communicates with another relational database management system.

**Distributed Relational Database Architecture (DRDA)**

A connection protocol for distributed relational database processing that is used by IBM relational database products. DRDA includes protocols for communication between an application and a remote relational database management system, and for communication between relational database management systems. See also DRDA access.

**DNS** See domain name server.

**DOCID**

See document ID.

**document ID**

A value that uniquely identifies a row that contains an XML column. This value is stored with the row and never changes.

**domain**

The set of valid values for an attribute.

**domain name**

The name by which TCP/IP applications refer to a TCP/IP host within a TCP/IP network.

**domain name server (DNS)**

A special TCP/IP network server that manages a distributed directory that is used to map TCP/IP host names to IP addresses.

**double-byte character large object (DBCLOB)**

A graphic string data type in which a sequence of bytes represent double-byte characters that range in size from 0 bytes to 2 GB, less 1 byte. In general, DBCLOB values are used whenever a double-byte character string might exceed the limits of the VARCHAR type.

**double-byte character set (DBCS)**

A set of characters, which are used by national languages such as Japanese and Chinese, that have more symbols than can be represented by a single byte. Each character is 2 bytes in length. Contrast with single-byte character set and multibyte character set.

**double-precision floating point number**

A 64-bit approximate representation of a real number.

**DPSI** See data-partitioned secondary index.

**drain** The act of acquiring a locked resource by quiescing access to that object. Contrast with claim.

**drain lock**

A lock on a claim class that prevents a claim from occurring.

**DRDA**

See Distributed Relational Database Architecture.

**DRDA access**

An open method of accessing distributed data that you can use to connect to another database server to execute packages that were previously bound at the server location.

**DSN**

- The default DB2 subsystem name.
- The name of the TSO command processor of DB2.
- The first three characters of DB2 module and macro names.

**dynamic cursor**

A named control structure that an application program uses to change the size of the result table and the order of its rows after the cursor is opened. Contrast with static cursor.

**dynamic dump**

A dump that is issued during the execution of a program, usually under the control of that program.

**dynamic SQL**

SQL statements that are prepared and executed at run time. In dynamic SQL, the SQL statement is contained as a character string in a host variable or as a constant, and it is not precompiled.

**EA-enabled table space**

A table space or index space that is enabled for extended addressability and that contains individual partitions (or pieces, for LOB table spaces) that are greater than 4 GB.

**EB** See exabyte.

## EBCDIC

Extended binary coded decimal interchange code. An encoding scheme that is used to represent character data in the z/OS, VM, VSE, and iSeries environments. Contrast with ASCII and Unicode.

## embedded SQL

SQL statements that are coded within an application program. See static SQL.

## enabling-new-function mode\* (ENFM\*)

A transitional stage of the version-to-version migration process that applies to a DB2 subsystem or data sharing group that was in new-function mode (NFM) at one time. When in enabling-new-function mode\*, a DB2 subsystem or data sharing group is preparing to use the new functions of the new version but cannot yet use them. A data sharing group that is in enabling-new-function mode\* cannot coexist with members that are still at the prior version level. Fallback to a prior version is not supported. Contrast with compatibility mode, compatibility mode\*, enabling-new-function mode, and new-function mode.

## enabling-new-function mode (ENFM)

A transitional stage of the version-to-version migration process during which the DB2 subsystem or data sharing group is preparing to use the new functions of the new version. When in enabling-new-function mode, a DB2 data sharing group cannot coexist with members that are still at the prior version level. Fallback to a prior version is not supported, and new functions of the new version are not available for use in enabling-new-function mode. Contrast with compatibility mode, compatibility mode\*, enabling-new-function mode\*, and new-function mode.

## enclave

In Language Environment, an independent collection of routines, one of which is designated as the main routine. An enclave is similar to a program or run unit. See also WLM enclave.

## encoding scheme

A set of rules to represent character data (ASCII, EBCDIC, or Unicode).

ENFM See enabling-new-function mode.

## ENFM\*

See enabling-new-function mode\*.

**entity** A person, object, or concept about which information is stored. In a relational database, entities are represented as tables. A database includes information about the entities in an organization or business, and their relationships to each other.

## enumerated list

A set of DB2 objects that are defined with a LISTDEF utility control statement in which pattern-matching characters (\*, %, \_ or ?) are not used.

## environment

A collection of names of logical and physical resources that are used to support the performance of a function.

## environment handle

A handle that identifies the global context for database access. All data that is pertinent to all objects in the environment is associated with this handle.

## equijoin

A join operation in which the join-condition has the form *expression* = *expression*. See also join, full outer join, inner join, left outer join, outer join, and right outer join.

## error page range

A range of pages that are considered to be physically damaged. DB2 does not allow users to access any pages that fall within this range.

## escape character

The symbol, a double quotation (") for example, that is used to enclose an SQL delimited identifier.

## exabyte

A unit of measure for processor, real and virtual storage capacities, and channel volume that has a value of 1 152 921 504 606 846 976 bytes or 2<sup>60</sup>.

## exception

An SQL operation that involves the EXCEPT set operator, which combines two result tables. The result of an exception operation consists of all of the rows that are in only one of the result tables.

**exception table**

A table that holds rows that violate referential constraints or check constraints that the CHECK DATA utility finds.

**exclusive lock**

A lock that prevents concurrently executing application processes from reading or changing data. Contrast with share lock.

**executable statement**

An SQL statement that can be embedded in an application program, dynamically prepared and executed, or issued interactively.

**execution context**

In SQLJ, a Java object that can be used to control the execution of SQL statements.

**exit routine**

A user-written (or IBM-provided default) program that receives control from DB2 to perform specific functions. Exit routines run as extensions of DB2.

**expanding conversion**

A process that occurs when the length of a converted string is greater than that of the source string. For example, this process occurs when an ASCII mixed-data string that contains DBCS characters is converted to an EBCDIC mixed-data string; the converted string is longer because shift codes are added.

**explicit hierarchical locking**

Locking that is used to make the parent-child relationship between resources known to IRLM. This kind of locking avoids global locking overhead when no inter-DB2 interest exists on a resource.

**explicit privilege**

A privilege that has a name and is held as the result of an SQL GRANT statement and revoked as the result of an SQL REVOKE statement. For example, the SELECT privilege.

**exposed name**

A correlation name or a table or view name for which a correlation name is not specified.

**expression**

An operand or a collection of operators and operands that yields a single value.

**Extended Recovery Facility (XRF)**

A facility that minimizes the effect of failures in z/OS, VTAM, the host processor, or high-availability applications during sessions between high-availability applications and designated terminals. This facility provides an alternative subsystem to take over sessions from the failing subsystem.

**Extensible Markup Language (XML)**

A standard metalanguage for defining markup languages that is a subset of Standardized General Markup Language (SGML).

**external function**

A function that has its functional logic implemented in a programming language application that resides outside the database, in the file system of the database server. The association of the function with the external code application is specified by the EXTERNAL clause in the CREATE FUNCTION statement. External functions can be classified as external scalar functions and external table functions. Contrast with sourced function, built-in function, and SQL function.

**external procedure**

A procedure that has its procedural logic implemented in an external programming language application. The association of the procedure with the external application is specified by a CREATE PROCEDURE statement with a LANGUAGE clause that has a value other than SQL and an EXTERNAL clause that implicitly or explicitly specifies the name of the external application. Contrast with external SQL procedure and native SQL procedure.

**external routine**

A user-defined function or stored procedure that is based on code that is written in an external programming language.

**external SQL procedure**

An SQL procedure that is processed using a generated C program that is a representation of the procedure. When an external SQL procedure is called, the C program representation of the procedure is executed in a stored procedures address

space. Contrast with external procedure and native SQL procedure.

**failed member state**

A state of a member of a data sharing group in which the member's task, address space, or z/OS system terminates before the state changes from active to quiesced.

**fallback**

The process of returning to a previous release of DB2 after attempting or completing migration to a current release. Fallback is supported only from a subsystem that is in compatibility mode.

**false global lock contention**

A contention indication from the coupling facility that occurs when multiple lock names are hashed to the same indicator and when no real contention exists.

**fan set**

A direct physical access path to data, which is provided by an index, hash, or link; a fan set is the means by which DB2 supports the ordering of data.

**federated database**

The combination of a DB2 server (in Linux, UNIX, and Windows environments) and multiple data sources to which the server sends queries. In a federated database system, a client application can use a single SQL statement to join data that is distributed across multiple database management systems and can view the data as if it were local.

**fetch orientation**

The specification of the desired placement of the cursor as part of a FETCH statement. The specification can be before or after the rows of the result table (with BEFORE or AFTER). The specification can also have either a single-row fetch orientation (for example, NEXT, LAST, or ABSOLUTE *n*) or a rowset fetch orientation (for example, NEXT ROWSET, LAST ROWSET, or ROWSET STARTING AT ABSOLUTE *n*).

**field procedure**

A user-written exit routine that is designed to receive a single value and transform (encode or decode) it in any way the user can specify.

**file reference variable**

A host variable that is declared with one of the derived data types (BLOB\_FILE, CLOB\_FILE, DBCLOB\_FILE); file reference variables direct the reading of a LOB from a file or the writing of a LOB into a file.

**filter factor**

A number between zero and one that estimates the proportion of rows in a table for which a predicate is true.

**fixed-length string**

A character, graphic, or binary string whose length is specified and cannot be changed. Contrast with varying-length string.

**FlashCopy**

A function on the IBM Enterprise Storage Server that can, in conjunction with the BACKUP SYSTEM utility, create a point-in-time copy of data while an application is running.

**foreign key**

A column or set of columns in a dependent table of a constraint relationship. The key must have the same number of columns, with the same descriptions, as the primary key of the parent table. Each foreign key value must either match a parent key value in the related parent table or be null.

**forest** An ordered set of subtrees of XML nodes.

**forward log recovery**

The third phase of restart processing during which DB2 processes the log in a forward direction to apply all REDO log records.

**free space**

The total amount of unused space in a page; that is, the space that is not used to store records or control information is free space.

**full outer join**

The result of a join operation that includes the matched rows of both tables that are being joined and preserves the unmatched rows of both tables. See also join, equijoin, inner join, left outer join, outer join, and right outer join.

**fullselect**

A subselect, a fullselect in parentheses, or

| a number of both that are combined by  
| set operators. Fullselect specifies a result  
| table. If a set operator is not used, the  
| result of the fullselect is the result of the  
| specified subselect or fullselect.

**fully escaped mapping**

A mapping from an SQL identifier to an XML name when the SQL identifier is a column name.

**function**

A mapping, which is embodied as a program (the function body) that is invocable by means of zero or more input values (arguments) to a single value (the result). See also aggregate function and scalar function.

Functions can be user-defined, built-in, or generated by DB2. (See also built-in function, cast function, external function, sourced function, SQL function, and user-defined function.)

**function definer**

The authorization ID of the owner of the schema of the function that is specified in the CREATE FUNCTION statement.

**function package**

A package that results from binding the DBRM for a function program.

**function package owner**

The authorization ID of the user who binds the function program's DBRM into a function package.

**function signature**

The logical concatenation of a fully qualified function name with the data types of all of its parameters.

**GB** Gigabyte. A value of (1 073 741 824 bytes).

**GBP** See group buffer pool.

**GBP-dependent**

The status of a page set or page set partition that is dependent on the group buffer pool. Either read/write interest is active among DB2 subsystems for this page set, or the page set has changed pages in the group buffer pool that have not yet been cast out to disk.

**generalized trace facility (GTF)**

A z/OS service program that records significant system events such as I/O

interrupts, SVC interrupts, program interrupts, or external interrupts.

**generic resource name**

A name that VTAM uses to represent several application programs that provide the same function in order to handle session distribution and balancing in a Sysplex environment.

**geographic feature**

An object on the surface of the Earth (such as a city or river), a space (such as a safety zone around a hazardous site), or an event that occurs at a location (such as an auto accident that occurred at a particular intersection).

**geographic information system**

A complex of objects, data, and applications that is used to create and analyze spatial information about geographic features.

**getpage**

An operation in which DB2 accesses a data page.

**global lock**

A lock that provides concurrency control within and among DB2 subsystems. The scope of the lock is across all DB2 subsystems of a data sharing group.

**global lock contention**

Conflicts on locking requests between different DB2 members of a data sharing group when those members are trying to serialize shared resources.

**governor**

See resource limit facility.

**graphic string**

A sequence of DBCS characters. Graphic data can be further classified as GRAPHIC, VARGRAPHIC, or DBCLOB.

**GRECP**

See group buffer pool recovery pending.

**gross lock**

The *shared*, *update*, or *exclusive* mode locks on a table, partition, or table space.

**group buffer pool duplexing**

The ability to write data to two instances of a group buffer pool structure: a primary group buffer pool and a secondary group buffer pool. z/OS publications refer to these instances as the

“old” (for primary) and “new” (for secondary) structures.

**group buffer pool (GBP)**

A coupling facility cache structure that is used by a data sharing group to cache data and to ensure that the data is consistent for all members.

**group buffer pool recovery pending (GRECP)**

The state that exists after the buffer pool for a data sharing group is lost. When a page set is in this state, changes that are recorded in the log must be applied to the affected page set before the page set can be used.

**group level**

The release level of a data sharing group, which is established when the first member migrates to a new release.

**group name**

The z/OS XCF identifier for a data sharing group.

**group restart**

A restart of at least one member of a data sharing group after the loss of either locks or the shared communications area.

**GTF** See generalized trace facility.

**handle**

In DB2 ODBC, a variable that refers to a data structure and associated resources. See also statement handle, connection handle, and environment handle.

**help panel**

A screen of information that presents tutorial text to assist a user at the workstation or terminal.

**heuristic damage**

The inconsistency in data between one or more participants that results when a heuristic decision to resolve an indoubt LUW at one or more participants differs from the decision that is recorded at the coordinator.

**heuristic decision**

A decision that forces indoubt resolution at a participant by means other than automatic resynchronization between coordinator and participant.

**histogram statistics**

A way of summarizing data distribution. This technique divides up the range of

possible values in a data set into intervals, such that each interval contains approximately the same percentage of the values. A set of statistics are collected for each interval.

**hole** A row of the result table that cannot be accessed because of a delete or an update that has been performed on the row. See also delete hole and update hole.

**home address space**

The area of storage that z/OS currently recognizes as *dispatched*.

**host** The set of programs and resources that are available on a given TCP/IP instance.

**host expression**

A Java variable or expression that is referenced by SQL clauses in an SQLJ application program.

**host identifier**

A name that is declared in the host program.

**host language**

A programming language in which you can embed SQL statements.

**host program**

An application program that is written in a host language and that contains embedded SQL statements.

**host structure**

In an application program, a structure that is referenced by embedded SQL statements.

**host variable**

In an application program written in a host language, an application variable that is referenced by embedded SQL statements.

**host variable array**

An array of elements, each of which corresponds to a value for a column. The dimension of the array determines the maximum number of rows for which the array can be used.

**IBM System z9 Integrated Processor (zIIP)**

A specialized processor that can be used for some DB2 functions.

**IDCAMS**

An IBM program that is used to process access method services commands. It can

be invoked as a job or jobstep, from a TSO terminal, or from within a user's application program.

#### **IDCAMS LISTCAT**

A facility for obtaining information that is contained in the access method services catalog.

#### **identity column**

| A column that provides a way for DB2 to  
| automatically generate a numeric value  
| for each row. Identity columns are  
| defined with the AS IDENTITY clause.  
| Uniqueness of values can be ensured by  
| defining a unique index that contains  
| only the identity column. A table can  
| have no more than one identity column.

**IFCID** See instrumentation facility component identifier.

**IFI** See instrumentation facility interface.

#### **IFI call**

An invocation of the instrumentation facility interface (IFI) by means of one of its defined functions.

#### **image copy**

An exact reproduction of all or part of a table space. DB2 provides utility programs to make full image copies (to copy the entire table space) or incremental image copies (to copy only those pages that have been modified since the last image copy).

#### **IMS attachment facility**

A DB2 subcomponent that uses z/OS subsystem interface (SSI) protocols and cross-memory linkage to process requests from IMS to DB2 and to coordinate resource commitment.

#### **in-abort**

A status of a unit of recovery. If DB2 fails after a unit of recovery begins to be rolled back, but before the process is completed, DB2 continues to back out the changes during restart.

#### **in-commit**

A status of a unit of recovery. If DB2 fails after beginning its phase 2 commit processing, it "knows," when restarted, that changes made to data are consistent. Such units of recovery are termed *in-commit*.

#### **independent**

An object (row, table, or table space) that is neither a parent nor a dependent of another object.

**index** A set of pointers that are logically ordered by the values of a key. Indexes can provide faster access to data and can enforce uniqueness on the rows in a table.

#### **index-controlled partitioning**

A type of partitioning in which partition boundaries for a partitioned table are controlled by values that are specified on the CREATE INDEX statement. Partition limits are saved in the LIMITKEY column of the SYSIBM.SYSINDEXPART catalog table.

#### **index key**

The set of columns in a table that is used to determine the order of index entries.

#### **index partition**

A VSAM data set that is contained within a partitioning index space.

#### **index space**

A page set that is used to store the entries of one index.

#### **indicator column**

A 4-byte value that is stored in a base table in place of a LOB column.

#### **indicator variable**

A variable that is used to represent the null value in an application program. If the value for the selected column is null, a negative value is placed in the indicator variable.

#### **indoubt**

A status of a unit of recovery. If DB2 fails after it has finished its phase 1 commit processing and before it has started phase 2, only the commit coordinator knows if an individual unit of recovery is to be committed or rolled back. At restart, if DB2 lacks the information it needs to make this decision, the status of the unit of recovery is *indoubt* until DB2 obtains this information from the coordinator. More than one unit of recovery can be *indoubt* at restart.

#### **indoubt resolution**

The process of resolving the status of an *indoubt* logical unit of work to either the committed or the rollback state.

**inflight**

A status of a unit of recovery. If DB2 fails before its unit of recovery completes phase 1 of the commit process, it merely backs out the updates of its unit of recovery at restart. These units of recovery are termed *inflight*.

**inheritance**

The passing downstream of class resources or attributes from a parent class in the class hierarchy to a child class.

**initialization file**

For DB2 ODBC applications, a file containing values that can be set to adjust the performance of the database manager.

**inline copy**

A copy that is produced by the LOAD or REORG utility. The data set that the inline copy produces is logically equivalent to a full image copy that is produced by running the COPY utility with read-only access (SHRLEVEL REFERENCE).

**inner join**

The result of a join operation that includes only the matched rows of both tables that are being joined. See also join, equijoin, full outer join, left outer join, outer join, and right outer join.

**inoperative package**

A package that cannot be used because one or more user-defined functions or procedures that the package depends on were dropped. Such a package must be explicitly rebound. Contrast with invalid package.

**insensitive cursor**

A cursor that is not sensitive to inserts, updates, or deletes that are made to the underlying rows of a result table after the result table has been materialized.

**insert trigger**

A trigger that is defined with the triggering SQL operation, an insert.

**install** The process of preparing a DB2 subsystem to operate as a z/OS subsystem.

**INSTEAD OF trigger**

A trigger that is associated with a single view and is activated by an insert, update, or delete operation on the view and that can define how to propagate the

insert, update, or delete operation on the view to the underlying tables of the view.

**instrumentation facility component identifier (IFCID)**

A value that names and identifies a trace record of an event that can be traced. As a parameter on the START TRACE and MODIFY TRACE commands, it specifies that the corresponding event is to be traced.

**instrumentation facility interface (IFI)**

A programming interface that enables programs to obtain online trace data about DB2, to submit DB2 commands, and to pass data to DB2.

**Interactive System Productivity Facility (ISPF)**

An IBM licensed program that provides interactive dialog services in a z/OS environment.

**inter-DB2 R/W interest**

A property of data in a table space, index, or partition that has been opened by more than one member of a data sharing group and that has been opened for writing by at least one of those members.

**intermediate database server**

The target of a request from a local application or a remote application requester that is forwarded to another database server.

**internal resource lock manager (IRLM)**

A z/OS subsystem that DB2 uses to control communication and database locking.

**internationalization**

The support for an encoding scheme that is able to represent the code points of characters from many different geographies and languages. To support all geographies, the Unicode standard requires more than 1 byte to represent a single character. See also Unicode.

**intersection**

An SQL operation that involves the INTERSECT set operator, which combines two result tables. The result of an intersection operation consists of all of the rows that are in both result tables.

**invalid package**

A package that depends on an object (other than a user-defined function) that

is dropped. Such a package is implicitly rebound on invocation. Contrast with inoperative package.

**IP address**

A value that uniquely identifies a TCP/IP host.

**IRLM** See internal resource lock manager.

**isolation level**

The degree to which a unit of work is isolated from the updating operations of other units of work. See also cursor stability, read stability, repeatable read, and uncommitted read.

**ISPF** See Interactive System Productivity Facility.

**iterator**

In SQLJ, an object that contains the result set of a query. An iterator is equivalent to a cursor in other host languages.

**iterator declaration clause**

In SQLJ, a statement that generates an iterator declaration class. An iterator is an object of an iterator declaration class.

**JAR** See Java Archive.

**Java Archive (JAR)**

A file format that is used for aggregating many files into a single file.

**JDBC** A Sun Microsystems database application programming interface (API) for Java that allows programs to access database management systems by using callable SQL.

**join** A relational operation that allows retrieval of data from two or more tables based on matching column values. See also equijoin, full outer join, inner join, left outer join, outer join, and right outer join.

**KB** Kilobyte. A value of 1024 bytes.

**Kerberos**

A network authentication protocol that is designed to provide strong authentication for client/server applications by using secret-key cryptography.

**Kerberos ticket**

A transparent application mechanism that transmits the identity of an initiating principal to its target. A simple ticket contains the principal's identity, a session

key, a timestamp, and other information, which is sealed using the target's secret key.

**key**

A column, an ordered collection of columns, or an expression that is identified in the description of a table, index, or referential constraint. The same column or expression can be part of more than one key.

**key-sequenced data set (KSDS)**

A VSAM file or data set whose records are loaded in key sequence and controlled by an index.

**KSDS** See key-sequenced data set.

**large object (LOB)**

A sequence of bytes representing bit data, single-byte characters, double-byte characters, or a mixture of single- and double-byte characters. A LOB can be up to 2 GB minus 1 byte in length. See also binary large object, character large object, and double-byte character large object.

**last agent optimization**

An optimized commit flow for either presumed-nothing or presumed-abort protocols in which the last agent, or final participant, becomes the commit coordinator. This flow saves at least one message.

**latch**

A DB2 mechanism for controlling concurrent events or the use of system resources.

**LCID** See log control interval definition.

**LDS** See linear data set.

**leaf page**

An index page that contains pairs of keys and RIDs and that points to actual data. Contrast with nonleaf page.

**left outer join**

The result of a join operation that includes the matched rows of both tables that are being joined, and that preserves the unmatched rows of the first table. See also join, equijoin, full outer join, inner join, outer join, and right outer join.

**limit key**

The highest value of the index key for a partition.

**linear data set (LDS)**

A VSAM data set that contains data but

- no control information. A linear data set can be accessed as a byte-addressable string in virtual storage.
- linkage editor**  
A computer program for creating load modules from one or more object modules or load modules by resolving cross references among the modules and, if necessary, adjusting addresses.
- link-edit**  
The action of creating a loadable computer program using a linkage editor.
- list**  
A type of object, which DB2 utilities can process, that identifies multiple table spaces, multiple index spaces, or both. A list is defined with the LISTDEF utility control statement.
- list structure**  
A coupling facility structure that lets data be shared and manipulated as elements of a queue.
- L-lock** See logical lock.
- load module**  
A program unit that is suitable for loading into main storage for execution. The output of a linkage editor.
- LOB** See large object.
- LOB locator**  
A mechanism that allows an application program to manipulate a large object value in the database system. A LOB locator is a fullword integer value that represents a single LOB value. An application program retrieves a LOB locator into a host variable and can then apply SQL operations to the associated LOB value using the locator.
- LOB lock**  
A lock on a LOB value.
- LOB table space**  
A table space that contains all the data for a particular LOB column in the related base table.
- local**  
A way of referring to any object that the local DB2 subsystem maintains. A *local table*, for example, is a table that is maintained by the local DB2 subsystem. Contrast with remote.
- locale**  
The definition of a subset of a user's environment that combines a CCSID and characters that are defined for a specific language and country.
- local lock**  
A lock that provides intra-DB2 concurrency control, but not inter-DB2 concurrency control; that is, its scope is a single DB2.
- local subsystem**  
The unique relational DBMS to which the user or application program is directly connected (in the case of DB2, by one of the DB2 attachment facilities).
- location**  
The unique name of a database server. An application uses the location name to access a DB2 database server. A database alias can be used to override the location name when accessing a remote server.
- location alias**  
Another name by which a database server identifies itself in the network. Applications can use this name to access a DB2 database server.
- lock**  
A means of controlling concurrent events or access to data. DB2 locking is performed by the IRLM.
- lock duration**  
The interval over which a DB2 lock is held.
- lock escalation**  
The promotion of a lock from a row, page, or LOB lock to a table space lock because the number of page locks that are concurrently held on a given resource exceeds a preset limit.
- locking**  
The process by which the integrity of data is ensured. Locking prevents concurrent users from accessing inconsistent data. See also claim, drain, and latch.
- lock mode**  
A representation for the type of access that concurrently running programs can have to a resource that a DB2 lock is holding.
- lock object**  
The resource that is controlled by a DB2 lock.

**lock promotion**

The process of changing the size or mode of a DB2 lock to a higher, more restrictive level.

**lock size**

The amount of data that is controlled by a DB2 lock on table data; the value can be a row, a page, a LOB, a partition, a table, or a table space.

**lock structure**

A coupling facility data structure that is composed of a series of lock entries to support shared and exclusive locking for logical resources.

**log**

A collection of records that describe the events that occur during DB2 execution and that indicate their sequence. The information thus recorded is used for recovery in the event of a failure during DB2 execution.

**log control interval definition**

A suffix of the physical log record that tells how record segments are placed in the physical control interval.

**logical claim**

A claim on a logical partition of a nonpartitioning index.

**logical index partition**

The set of all keys that reference the same data partition.

**logical lock (L-lock)**

The lock type that transactions use to control intra- and inter-DB2 data concurrency between transactions. Contrast with physical lock (P-lock).

**logically complete**

A state in which the concurrent copy process is finished with the initialization of the target objects that are being copied. The target objects are available for update.

**logical page list (LPL)**

A list of pages that are in error and that cannot be referenced by applications until the pages are recovered. The page is in *logical error* because the actual media (coupling facility or disk) might not contain any errors. Usually a connection to the media has been lost.

**logical partition**

A set of key or RID pairs in a

nonpartitioning index that are associated with a particular partition.

**logical recovery pending (LRECP)**

The state in which the data and the index keys that reference the data are inconsistent.

**logical unit (LU)**

An access point through which an application program accesses the SNA network in order to communicate with another application program. See also LU name.

**logical unit of work**

The processing that a program performs between synchronization points.

**logical unit of work identifier (LUWID)**

A name that uniquely identifies a thread within a network. This name consists of a fully-qualified LU network name, an LUW instance number, and an LUW sequence number.

**log initialization**

The first phase of restart processing during which DB2 attempts to locate the current end of the log.

**log record header (LRH)**

A prefix, in every log record, that contains control information.

**log record sequence number (LRSN)**

An identifier for a log record that is associated with a data sharing member. DB2 uses the LRSN for recovery in the data sharing environment.

**log truncation**

A process by which an explicit starting RBA is established. This RBA is the point at which the next byte of log data is to be written.

**LPL** See logical page list.

**LRECP**

See logical recovery pending.

**LRH** See log record header.

**LRSN** See log record sequence number.

**LU** See logical unit.

**LU name**

Logical unit name, which is the name by which VTAM refers to a node in a network.

**LUW** See logical unit of work.

**LUWID**

See logical unit of work identifier.

**mapping table**

A table that the REORG utility uses to map the associations of the RIDs of data records in the original copy and in the shadow copy. This table is created by the user.

**mass delete**

The deletion of all rows of a table.

**materialize**

- The process of putting rows from a view or nested table expression into a work file for additional processing by a query.
- The placement of a LOB value into contiguous storage. Because LOB values can be very large, DB2 avoids materializing LOB data until doing so becomes absolutely necessary.

**materialized query table**

A table that is used to contain information that is derived and can be summarized from one or more source tables. Contrast with base table.

**MB** Megabyte (1 048 576 bytes).

**MBCS** See multibyte character set.

**member name**

The z/OS XCF identifier for a particular DB2 subsystem in a data sharing group.

**menu** A displayed list of available functions for selection by the operator. A menu is sometimes called a *menu panel*.

**metalanguage**

A language that is used to create other specialized languages.

**migration**

The process of converting a subsystem with a previous release of DB2 to an updated or current release. In this process, you can acquire the functions of the updated or current release without losing the data that you created on the previous release.

**mixed data string**

A character string that can contain both single-byte and double-byte characters.

**mode name**

A VTAM name for the collection of physical and logical characteristics and attributes of a session.

**modify locks**

An L-lock or P-lock with a MODIFY attribute. A list of these active locks is kept at all times in the coupling facility lock structure. If the requesting DB2 subsystem fails, that DB2 subsystem's modify locks are converted to retained locks.

**multibyte character set (MBCS)**

A character set that represents single characters with more than a single byte. UTF-8 is an example of an MBCS. Characters in UTF-8 can range from 1 to 4 bytes in DB2. Contrast with single-byte character set and double-byte character set. See also Unicode.

**multidimensional analysis**

The process of assessing and evaluating an enterprise on more than one level.

**Multiple Virtual Storage (MVS)**

An element of the z/OS operating system. This element is also called the Base Control Program (BCP).

**multisite update**

Distributed relational database processing in which data is updated in more than one location within a single unit of work.

**multithreading**

Multiple TCBs that are executing one copy of DB2 ODBC code concurrently (sharing a processor) or in parallel (on separate central processors).

**MVS** See Multiple Virtual Storage.

**native SQL procedure**

An SQL procedure that is processed by converting the procedural statements to a native representation that is stored in the database directory, as is done with other SQL statements. When a native SQL procedure is called, the native representation is loaded from the directory, and DB2 executes the procedure.

**nested table expression**

A fullselect in a FROM clause (surrounded by parentheses).

**network identifier (NID)**

The network ID that is assigned by IMS or CICS, or if the connection type is RRSAF, the RRS unit of recovery ID (URID).

**new-function mode (NFM)**

The normal mode of operation that exists after successful completion of a version-to-version migration. At this stage, all new functions of the new version are available for use. A DB2 data sharing group cannot coexist with members that are still at the prior version level, and fallback to a prior version is not supported. Contrast with compatibility mode, compatibility mode\*, enabling-new-function mode, and enabling-new-function mode\*.

**NFM** See new-function mode.

**NID** See network identifier.

**node ID index**

See XML node ID index.

**nondeterministic function**

A user-defined function whose result is not solely dependent on the values of the input arguments. That is, successive invocations with the same argument values can produce a different answer. This type of function is sometimes called a *variant* function. Contrast with deterministic function (sometimes called a *not-variant function*).

**nonleaf page**

A page that contains keys and page numbers of other pages in the index (either leaf or nonleaf pages). Nonleaf pages never point to actual data. Contrast with leaf page.

**nonpartitioned index**

An index that is not physically partitioned. Both partitioning indexes and secondary indexes can be nonpartitioned.

**nonpartitioned secondary index (NPSI)**

An index on a partitioned table space that is not the partitioning index and is not partitioned. Contrast with data-partitioned secondary index.

**nonpartitioning index**

See secondary index.

**nonscrollable cursor**

A cursor that can be moved only in a

forward direction. Nonscrollable cursors are sometimes called forward-only cursors or serial cursors.

**normalization**

A key step in the task of building a logical relational database design. Normalization helps you avoid redundancies and inconsistencies in your data. An entity is normalized if it meets a set of constraints for a particular normal form (first normal form, second normal form, and so on). Contrast with denormalization.

**not-variant function**

See deterministic function.

**NPSI** See nonpartitioned secondary index.

**NUL** The null character ('\0'), which is represented by the value X'00'. In C, this character denotes the end of a string.

**null** A special value that indicates the absence of information.

**null terminator**

In C, the value that indicates the end of a string. For EBCDIC, ASCII, and Unicode UTF-8 strings, the null terminator is a single-byte value (X'00'). For Unicode UTF-16 or UCS-2 (wide) strings, the null terminator is a double-byte value (X'0000').

**ODBC**

See Open Database Connectivity.

**ODBC driver**

A dynamically-linked library (DLL) that implements ODBC function calls and interacts with a data source.

**OLAP** See online analytical processing.

**online analytical processing (OLAP)**

The process of collecting data from one or many sources; transforming and analyzing the consolidated data quickly and interactively; and examining the results across different dimensions of the data by looking for patterns, trends, and exceptions within complex relationships of that data.

**Open Database Connectivity (ODBC)**

A Microsoft database application programming interface (API) for C that allows access to database management systems by using callable SQL. ODBC

does not require the use of an SQL preprocessor. In addition, ODBC provides an architecture that lets users add modules called *database drivers*, which link the application to their choice of database management systems at run time. This means that applications no longer need to be directly linked to the modules of all the database management systems that are supported.

#### **ordinary identifier**

An uppercase letter followed by zero or more characters, each of which is an uppercase letter, a digit, or the underscore character. An ordinary identifier must not be a reserved word.

#### **ordinary token**

A numeric constant, an ordinary identifier, a host identifier, or a keyword.

#### **originating task**

In a parallel group, the primary agent that receives data from other execution units (referred to as *parallel tasks*) that are executing portions of the query in parallel.

#### **outer join**

The result of a join operation that includes the matched rows of both tables that are being joined and preserves some or all of the unmatched rows of the tables that are being joined. See also join, equijoin, full outer join, inner join, left outer join, and right outer join.

#### **overloaded function**

A function name for which multiple function instances exist.

#### **package**

An object containing a set of SQL statements that have been statically bound and that is available for processing. A package is sometimes also called an *application package*.

#### **package list**

An ordered list of package names that may be used to extend an application plan.

#### **package name**

The name of an object that is used for an application package or an SQL procedure package. An application package is a bound version of a database request module (DBRM) that is created by a

BIND PACKAGE or REBIND PACKAGE command. An SQL procedural language package is created by a CREATE or ALTER PROCEDURE statement for a native SQL procedure. The name of a package consists of a location name, a collection ID, a package ID, and a version ID.

#### **page**

A unit of storage within a table space (4 KB, 8 KB, 16 KB, or 32 KB) or index space (4 KB, 8 KB, 16 KB, or 32 KB). In a table space, a page contains one or more rows of a table. In a LOB or XML table space, a LOB or XML value can span more than one page, but no more than one LOB or XML value is stored on a page.

#### **page set**

Another way to refer to a table space or index space. Each page set consists of a collection of VSAM data sets.

#### **page set recovery pending (PSRCP)**

A restrictive state of an index space. In this case, the entire page set must be recovered. Recovery of a logical part is prohibited.

#### **panel**

A predefined display image that defines the locations and characteristics of display fields on a display surface (for example, a *menu panel*).

#### **parallel complex**

A cluster of machines that work together to handle multiple transactions and applications.

#### **parallel group**

A set of consecutive operations that execute in parallel and that have the same number of parallel tasks.

#### **parallel I/O processing**

A form of I/O processing in which DB2 initiates multiple concurrent requests for a single user query and performs I/O processing concurrently (in *parallel*) on multiple data partitions.

#### **parallelism assistant**

In Sysplex query parallelism, a DB2 subsystem that helps to process parts of a parallel query that originates on another DB2 subsystem in the data sharing group.

**parallelism coordinator**

In Sysplex query parallelism, the DB2 subsystem from which the parallel query originates.

**Parallel Sysplex**

A set of z/OS systems that communicate and cooperate with each other through certain multisystem hardware components and software services to process customer workloads.

**parallel task**

The execution unit that is dynamically created to process a query in parallel. A parallel task is implemented by a z/OS service request block.

**parameter marker**

A question mark (?) that appears in a statement string of a dynamic SQL statement. The question mark can appear where a variable could appear if the statement string were a static SQL statement.

**parameter-name**

An SQL identifier that designates a parameter in a routine that is written by a user. Parameter names are required for SQL procedures and SQL functions, and they are used in the body of the routine to refer to the values of the parameters. Parameter names are optional for external routines.

**parent key**

A primary key or unique key in the parent table of a referential constraint. The values of a parent key determine the valid values of the foreign key in the referential constraint.

**parent lock**

For explicit hierarchical locking, a lock that is held on a resource that might have child locks that are lower in the hierarchy. A parent lock is usually the table space lock or the partition intent lock. See also child lock.

**parent row**

A row whose primary key value is the foreign key value of a dependent row.

**parent table**

A table whose primary key is referenced by the foreign key of a dependent table.

**parent table space**

A table space that contains a parent table. A table space containing a dependent of that table is a dependent table space.

**participant**

An entity other than the commit coordinator that takes part in the commit process. The term participant is synonymous with agent in SNA.

**partition**

A portion of a page set. Each partition corresponds to a single, independently extendable data set. The maximum size of a partition depends on the number of partitions in the partitioned page set. All partitions of a given page set have the same maximum size.

**partition-by-growth table space**

A table space whose size can grow to accommodate data growth. DB2 for z/OS manages partition-by-growth table spaces by automatically adding new data sets when the database needs more space to satisfy an insert operation.

**partitioned data set (PDS)**

A data set in disk storage that is divided into partitions, which are called members. Each partition can contain a program, part of a program, or data. A program library is an example of a partitioned data set.

**partitioned index**

An index that is physically partitioned. Both partitioning indexes and secondary indexes can be partitioned.

**partitioned page set**

A partitioned table space or an index space. Header pages, space map pages, data pages, and index pages reference data only within the scope of the partition.

**partitioned table space**

A table space that is based on a single table and that is subdivided into partitions, each of which can be processed independently by utilities.

**partitioning index**

An index in which the leftmost columns are the partitioning columns of the table. The index can be partitioned or nonpartitioned.

**partner logical unit**

An access point in the SNA network that is connected to the local DB2 subsystem by way of a VTAM conversation.

**path** See SQL path.

**PDS** See partitioned data set.

**physical consistency**

The state of a page that is not in a partially changed state.

**physical lock (P-lock)**

A type of lock that DB2 acquires to provide consistency of data that is cached in different DB2 subsystems. Physical locks are used only in data sharing environments. Contrast with logical lock (L-lock).

**physically complete**

The state in which the concurrent copy process is completed and the output data set has been created.

**piece** A data set of a nonpartitioned page set.

**plan** See application plan.

**plan allocation**

The process of allocating DB2 resources to a plan in preparation for execution.

**plan member**

The bound copy of a DBRM that is identified in the member clause.

**plan name**

The name of an application plan.

**P-lock** See physical lock.

**point of consistency**

A time when all recoverable data that an application accesses is consistent with other data. The term point of consistency is synonymous with sync point or commit point.

**policy** See CFRM policy.

**postponed abort UR**

A unit of recovery that was inflight or in-abort, was interrupted by system failure or cancellation, and did not complete backout during restart.

**precision**

In SQL, the total number of digits in a decimal number (called the *size* in the C language). In the C language, the number of digits to the right of the decimal point

(called the *scale* in SQL). The DB2 information uses the SQL terms.

**precompilation**

A processing of application programs containing SQL statements that takes place before compilation. SQL statements are replaced with statements that are recognized by the host language compiler. Output from this precompilation includes source code that can be submitted to the compiler and the database request module (DBRM) that is input to the bind process.

**predicate**

An element of a search condition that expresses or implies a comparison operation.

**prefix** A code at the beginning of a message or record.

**preformat**

The process of preparing a VSAM linear data set for DB2 use, by writing specific data patterns.

**prepare**

The first phase of a two-phase commit process in which all participants are requested to prepare for commit.

**prepared SQL statement**

A named object that is the executable form of an SQL statement that has been processed by the PREPARE statement.

**primary authorization ID**

The authorization ID that is used to identify the application process to DB2.

**primary group buffer pool**

For a duplexed group buffer pool, the structure that is used to maintain the coherency of cached data. This structure is used for page registration and cross-invalidation. The z/OS equivalent is *old* structure. Compare with secondary group buffer pool.

**primary index**

An index that enforces the uniqueness of a primary key.

**primary key**

In a relational database, a unique, nonnull key that is part of the definition of a table. A table cannot be defined as a parent unless it has a unique key or primary key.

**principal**

An entity that can communicate securely with another entity. In Kerberos, principals are represented as entries in the Kerberos registry database and include users, servers, computers, and others.

**principal name**

The name by which a principal is known to the DCE security services.

**privilege**

The capability of performing a specific function, sometimes on a specific object. See also explicit privilege.

**privilege set**

- | • For the installation SYSADM ID, the set of all possible privileges.
- | • For any other authorization ID, including the PUBLIC authorization ID, the set of all privileges that are recorded for that ID in the DB2 catalog.

**process**

In DB2, the unit to which DB2 allocates resources and locks. Sometimes called an application process, a process involves the execution of one or more programs. The execution of an SQL statement is always associated with some process. The means of initiating and terminating a process are dependent on the environment.

**program**

A single, compilable collection of executable statements in a programming language.

**program temporary fix (PTF)**

A solution or bypass of a problem that is diagnosed as a result of a defect in a current unaltered release of a licensed program. An authorized program analysis report (APAR) fix is corrective service for an existing problem. A PTF is preventive service for problems that might be encountered by other users of the product. A PTF is *temporary*, because a permanent fix is usually not incorporated into the product until its next release.

**protected conversation**

A VTAM conversation that supports two-phase commit flows.

**PSRCP**

See page set recovery pending.

**PTF**

See program temporary fix.

**QSAM**

See queued sequential access method.

**query** A component of certain SQL statements that specifies a result table.

**query block**

The part of a query that is represented by one of the FROM clauses. Each FROM clause can have multiple query blocks, depending on DB2 processing of the query.

**query CP parallelism**

Parallel execution of a single query, which is accomplished by using multiple tasks. See also Sysplex query parallelism.

**query I/O parallelism**

Parallel access of data, which is accomplished by triggering multiple I/O requests within a single query.

**queued sequential access method (QSAM)**

An extended version of the basic sequential access method (BSAM). When this method is used, a queue of data blocks is formed. Input data blocks await processing, and output data blocks await transfer to auxiliary storage or to an output device.

**quiesce point**

A point at which data is consistent as a result of running the DB2 QUIESCE utility.

**RACF** Resource Access Control Facility. A component of the z/OS Security Server.

**range-partitioned table space**

A type of universal table space that is based on partitioning ranges and that contains a single table.

**RBA** See relative byte address.

**RCT** See resource control table.

**RDO** See resource definition online.

**read stability (RS)**

An isolation level that is similar to repeatable read but does not completely isolate an application process from all other concurrently executing application processes. See also cursor stability, repeatable read, and uncommitted read.

**rebind**

The creation of a new application plan for

an application program that has been bound previously. If, for example, you have added an index for a table that your application accesses, you must rebind the application in order to take advantage of that index.

**rebuild**

The process of reallocating a coupling facility structure. For the shared communications area (SCA) and lock structure, the structure is repopulated; for the group buffer pool, changed pages are usually cast out to disk, and the new structure is populated only with changed pages that were not successfully cast out.

**record** The storage representation of a row or other data.

**record identifier (RID)**

A unique identifier that DB2 uses to identify a row of data in a table. Compare with row identifier.

**record identifier (RID) pool**

An area of main storage that is used for sorting record identifiers during list-prefetch processing.

**record length**

The sum of the length of all the columns in a table, which is the length of the data as it is physically stored in the database. Records can be fixed length or varying length, depending on how the columns are defined. If all columns are fixed-length columns, the record is a fixed-length record. If one or more columns are varying-length columns, the record is a varying-length record.

**Recoverable Resource Manager Services attachment facility (RRSAF)**

A DB2 subcomponent that uses Resource Recovery Services to coordinate resource commitment between DB2 and all other resource managers that also use RRS in a z/OS system.

**recovery**

The process of rebuilding databases after a system failure.

**recovery log**

A collection of records that describes the events that occur during DB2 execution and indicates their sequence. The

recorded information is used for recovery in the event of a failure during DB2 execution.

**recovery manager**

A subcomponent that supplies coordination services that control the interaction of DB2 resource managers during commit, abort, checkpoint, and restart processes. The recovery manager also supports the recovery mechanisms of other subsystems (for example, IMS) by acting as a participant in the other subsystem's process for protecting data that has reached a point of consistency.

A coordinator or a participant (or both), in the execution of a two-phase commit, that can access a recovery log that maintains the state of the logical unit of work and names the immediate upstream coordinator and downstream participants.

**recovery pending (RECP)**

A condition that prevents SQL access to a table space that needs to be recovered.

**recovery token**

An identifier for an element that is used in recovery (for example, NID or URID).

**RECP** See recovery pending.

**redo** A state of a unit of recovery that indicates that changes are to be reapplied to the disk media to ensure data integrity.

**reentrant code**

Executable code that can reside in storage as one shared copy for all threads. Reentrant code is not self-modifying and provides separate storage areas for each thread. See also threadsafe.

**referential constraint**

The requirement that nonnull values of a designated foreign key are valid only if they equal values of the primary key of a designated table.

**referential cycle**

A set of referential constraints such that each base table in the set is a descendent of itself. The tables that are involved in a referential cycle are ordered so that each table is a descendent of the one before it, and the first table is a descendent of the last table.

**referential integrity**

The state of a database in which all

values of all foreign keys are valid. Maintaining referential integrity requires the enforcement of referential constraints on all operations that change the data in a table on which the referential constraints are defined.

**referential structure**

A set of tables and relationships that includes at least one table and, for every table in the set, all the relationships in which that table participates and all the tables to which it is related.

**refresh age**

The time duration between the current time and the time during which a materialized query table was last refreshed.

**registry**

See registry database.

**registry database**

A database of security information about principals, groups, organizations, accounts, and security policies.

**relational database**

A database that can be perceived as a set of tables and manipulated in accordance with the relational model of data.

**relational database management system (RDBMS)**

A collection of hardware and software that organizes and provides access to a relational database.

**relational schema**

See SQL schema.

**relationship**

A defined connection between the rows of a table or the rows of two tables. A relationship is the internal representation of a referential constraint.

**relative byte address (RBA)**

The offset of a data record or control interval from the beginning of the storage space that is allocated to the data set or file to which it belongs.

**remigration**

The process of returning to a current release of DB2 following a fallback to a previous release. This procedure constitutes another migration process.

**remote**

Any object that is maintained by a remote DB2 subsystem (that is, by a DB2 subsystem other than the local one). A *remote view*, for example, is a view that is maintained by a remote DB2 subsystem. Contrast with local.

**remote subsystem**

Any relational DBMS, except the local subsystem, with which the user or application can communicate. The subsystem need not be remote in any physical sense, and might even operate on the same processor under the same z/OS system.

**reoptimization**

The DB2 process of reconsidering the access path of an SQL statement at run time; during reoptimization, DB2 uses the values of host variables, parameter markers, or special registers.

**reordered row format**

A row format that facilitates improved performance in retrieval of rows that have varying-length columns. DB2 rearranges the column order, as defined in the CREATE TABLE statement, so that the fixed-length columns are stored at the beginning of the row and the varying-length columns are stored at the end of the row.

**REORG pending (REORP)**

A condition that restricts SQL access and most utility access to an object that must be reorganized.

**REORP**

See REORG pending.

**repeatable read (RR)**

The isolation level that provides maximum protection from other executing application programs. When an application program executes with repeatable read protection, rows that the program references cannot be changed by other programs until the program reaches a commit point. See also cursor stability, read stability, and uncommitted read.

**repeating group**

A situation in which an entity includes multiple attributes that are inherently the same. The presence of a repeating group violates the requirement of first normal

form. In an entity that satisfies the requirement of first normal form, each attribute is independent and unique in its meaning and its name. See also normalization.

**replay detection mechanism**

A method that allows a principal to detect whether a request is a valid request from a source that can be trusted or whether an untrustworthy entity has captured information from a previous exchange and is replaying the information exchange to gain access to the principal.

**request commit**

The vote that is submitted to the prepare phase if the participant has modified data and is prepared to commit or roll back.

**requester**

The source of a request to access data at a remote server. In the DB2 environment, the requester function is provided by the distributed data facility.

**resource**

The object of a lock or claim, which could be a table space, an index space, a data partition, an index partition, or a logical partition.

**resource allocation**

The part of plan allocation that deals specifically with the database resources.

**resource control table**

A construct of previous versions of the CICS attachment facility that defines authorization and access attributes for transactions or transaction groups. Beginning in CICS Transaction Server Version 1.3, resources are defined by using resource definition online instead of the resource control table. See also resource definition online.

**resource definition online (RDO)**

The recommended method of defining resources to CICS by creating resource definitions interactively, or by using a utility, and then storing them in the CICS definition data set. In earlier releases of CICS, resources were defined by using the resource control table (RCT), which is no longer supported.

**resource limit facility (RLF)**

A portion of DB2 code that prevents dynamic manipulative SQL statements

from exceeding specified time limits. The resource limit facility is sometimes called the governor.

**resource limit specification table (RLST)**

A site-defined table that specifies the limits to be enforced by the resource limit facility.

**resource manager**

- A function that is responsible for managing a particular resource and that guarantees the consistency of all updates made to recoverable resources within a logical unit of work. The resource that is being managed can be physical (for example, disk or main storage) or logical (for example, a particular type of system service).
- A participant, in the execution of a two-phase commit, that has recoverable resources that could have been modified. The resource manager has access to a recovery log so that it can commit or roll back the effects of the logical unit of work to the recoverable resources.

**restart pending (RESTP)**

A restrictive state of a page set or partition that indicates that restart (backout) work needs to be performed on the object.

**RESTP**

See restart pending.

**result set**

The set of rows that a stored procedure returns to a client application.

**result set locator**

A 4-byte value that DB2 uses to uniquely identify a query result set that a stored procedure returns.

**result table**

The set of rows that are specified by a SELECT statement.

**retained lock**

A MODIFY lock that a DB2 subsystem was holding at the time of a subsystem failure. The lock is retained in the coupling facility lock structure across a DB2 for z/OS failure.

**RID**

See record identifier.

**RID pool**

See record identifier pool.

**right outer join**

The result of a join operation that includes the matched rows of both tables that are being joined and preserves the unmatched rows of the second join operand. See also join, equijoin, full outer join, inner join, left outer join, and outer join.

**RLF** See resource limit facility.

**RLST** See resource limit specification table.

**role** A database entity that groups together one or more privileges and that can be assigned to a primary authorization ID or to PUBLIC. The role is available only in a trusted context.

**rollback**

The process of restoring data that was changed by SQL statements to the state at its last commit point. All locks are freed. Contrast with commit.

**root page**

The index page that is at the highest level (or the beginning point) in an index.

**routine**

A database object that encapsulates procedural logic and SQL statements, is stored on the database server, and can be invoked from an SQL statement or by using the CALL statement. The main classes of routines are procedures and functions.

**row** The horizontal component of a table. A row consists of a sequence of values, one for each column of the table.

**row identifier (ROWID)**

A value that uniquely identifies a row. This value is stored with the row and never changes.

**row lock**

A lock on a single row of data.

**row-positioned fetch orientation**

The specification of the desired placement of the cursor as part of a FETCH statement, with respect to a single row (for example, NEXT, LAST, or ABSOLUTE *n*). Contrast with rowset-positioned fetch orientation.

**rowset**

A set of rows for which a cursor position is established.

**rowset cursor**

A cursor that is defined so that one or more rows can be returned as a rowset for a single FETCH statement, and the cursor is positioned on the set of rows that is fetched.

**rowset-positioned fetch orientation**

The specification of the desired placement of the cursor as part of a FETCH statement, with respect to a rowset (for example, NEXT ROWSET, LAST ROWSET, or ROWSET STARTING AT ABSOLUTE *n*). Contrast with row-positioned fetch orientation.

**row trigger**

A trigger that is defined with the trigger granularity FOR EACH ROW.

**RRSAF**

See Recoverable Resource Manager Services attachment facility.

**RS**

See read stability.

**savepoint**

A named entity that represents the state of data and schemas at a particular point in time within a unit of work.

**SBCS** See single-byte character set.

**SCA** See shared communications area.

**scalar function**

An SQL operation that produces a single value from another value and is expressed as a function name, followed by a list of arguments that are enclosed in parentheses.

**scale**

In SQL, the number of digits to the right of the decimal point (called the precision in the C language). The DB2 information uses the SQL definition.

**schema**

The organization or structure of a database.

A collection of, and a way of qualifying, database objects such as tables, views, routines, indexes or triggers that define a database. A database schema provides a logical classification of database objects.

**scrollability**

The ability to use a cursor to fetch in either a forward or backward direction. The FETCH statement supports multiple fetch orientations to indicate the new position of the cursor. See also fetch orientation.

**scrollable cursor**

A cursor that can be moved in both a forward and a backward direction.

**search condition**

A criterion for selecting rows from a table. A search condition consists of one or more predicates.

**secondary authorization ID**

An authorization ID that has been associated with a primary authorization ID by an authorization exit routine.

**secondary group buffer pool**

For a duplexed group buffer pool, the structure that is used to back up changed pages that are written to the primary group buffer pool. No page registration or cross-invalidation occurs using the secondary group buffer pool. The z/OS equivalent is *new* structure.

**secondary index**

A nonpartitioning index that is useful for enforcing a uniqueness constraint, for clustering data, or for providing access paths to data for queries. A secondary index can be partitioned or nonpartitioned. See also data-partitioned secondary index (DPSI) and nonpartitioned secondary index (NPSI).

**section**

The segment of a plan or package that contains the executable structures for a single SQL statement. For most SQL statements, one section in the plan exists for each SQL statement in the source program. However, for cursor-related statements, the DECLARE, OPEN, FETCH, and CLOSE statements reference the same section because they each refer to the SELECT statement that is named in the DECLARE CURSOR statement. SQL statements such as COMMIT, ROLLBACK, and some SET statements do not use a section.

**security label**

A classification of users' access to objects or data rows in a multilevel security environment."

**segment**

A group of pages that holds rows of a single table. See also segmented table space.

**segmented table space**

A table space that is divided into equal-sized groups of pages called segments. Segments are assigned to tables so that rows of different tables are never stored in the same segment.

**self-referencing constraint**

A referential constraint that defines a relationship in which a table is a dependent of itself.

**self-referencing table**

A table with a self-referencing constraint.

**sensitive cursor**

A cursor that is sensitive to changes that are made to the database after the result table has been materialized.

**sequence**

A user-defined object that generates a sequence of numeric values according to user specifications.

**sequential data set**

A non-DB2 data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape. Several of the DB2 database utilities require sequential data sets.

**sequential prefetch**

A mechanism that triggers consecutive asynchronous I/O operations. Pages are fetched before they are required, and several pages are read with a single I/O operation.

**serialized profile**

A Java object that contains SQL statements and descriptions of host variables. The SQLJ translator produces a serialized profile for each connection context.

**server** The target of a request from a remote requester. In the DB2 environment, the server function is provided by the distributed data facility, which is used to access DB2 data from remote applications.

**service class**

An eight-character identifier that is used by the z/OS Workload Manager to associate user performance goals with a particular DDF thread or stored procedure. A service class is also used to classify work on parallelism assistants.

**service request block**

A unit of work that is scheduled to execute.

**session**

A link between two nodes in a VTAM network.

**session protocols**

The available set of SNA communication requests and responses.

**set operator**

The SQL operators UNION, EXCEPT, and INTERSECT corresponding to the relational operators union, difference, and intersection. A set operator derives a result table by combining two other result tables.

**shared communications area (SCA)**

A coupling facility list structure that a DB2 data sharing group uses for inter-DB2 communication.

**share lock**

A lock that prevents concurrently executing application processes from changing data, but not from reading data. Contrast with exclusive lock.

**shift-in character**

A special control character (X'0F') that is used in EBCDIC systems to denote that the subsequent bytes represent SBCS characters. See also shift-out character.

**shift-out character**

A special control character (X'0E') that is used in EBCDIC systems to denote that the subsequent bytes, up to the next shift-in control character, represent DBCS characters. See also shift-in character.

**sign-on**

A request that is made on behalf of an individual CICS or IMS application process by an attachment facility to enable DB2 to verify that it is authorized to use DB2 resources.

**simple page set**

A nonpartitioned page set. A simple page

set initially consists of a single data set (page set piece). If and when that data set is extended to 2 GB, another data set is created, and so on, up to a total of 32 data sets. DB2 considers the data sets to be a single contiguous linear address space containing a maximum of 64 GB. Data is stored in the next available location within this address space without regard to any partitioning scheme.

**simple table space**

A table space that is neither partitioned nor segmented. Creation of simple table spaces is not supported in DB2 Version 9.1 for z/OS. Contrast with partitioned table space, segmented table space, and universal table space.

**single-byte character set (SBCS)**

A set of characters in which each character is represented by a single byte. Contrast with double-byte character set or multibyte character set.

**single-precision floating point number**

A 32-bit approximate representation of a real number.

**SMP/E**

See System Modification Program/Extended.

**SNA** See Systems Network Architecture.

**SNA network**

The part of a network that conforms to the formats and protocols of Systems Network Architecture (SNA).

**socket** A callable TCP/IP programming interface that TCP/IP network applications use to communicate with remote TCP/IP partners.

**sourced function**

A function that is implemented by another built-in or user-defined function that is already known to the database manager. This function can be a scalar function or an aggregate function; it returns a single value from a set of values (for example, MAX or AVG). Contrast with built-in function, external function, and SQL function.

**source program**

A set of host language statements and SQL statements that is processed by an SQL precompiler.

**source table**

A table that can be a base table, a view, a table expression, or a user-defined table function.

**source type**

An existing type that DB2 uses to represent a distinct type.

**space** A sequence of one or more blank characters.

**spatial column**

A column in a table that is defined using one of the spatial data types provided by IBM Spatial Support for DB2 for z/OS.

**spatial data**

Data that is made up of coordinates that identify a geographic location or geographic region.

**spatial function**

A function provided by IBM Spatial Support for DB2 for z/OS that performs various operations on spatial data.

**spatial reference system**

A set of parameters that includes coordinates that define the maximum possible extent of space that is referenced by a given range of coordinates, an identifier of the coordinate system from which the coordinates are derived, and numbers that convert coordinates into positive integers to improve performance when the coordinates are processed.

**special register**

A storage area that DB2 defines for an application process to use for storing information that can be referenced in SQL statements. Examples of special registers are SESSION\_USER and CURRENT DATE.

**specific function name**

A particular user-defined function that is known to the database manager by its specific name. Many specific user-defined functions can have the same function name. When a user-defined function is defined to the database, every function is assigned a specific name that is unique within its schema. Either the user can provide this name, or a default name is used.

**SPUFI** See SQL Processor Using File Input.

**SQL** See Structured Query Language.

**SQL authorization ID (SQL ID)**

The authorization ID that is used for checking dynamic SQL statements in some situations.

**SQLCA**

See SQL communication area.

**SQL communication area (SQLCA)**

A structure that is used to provide an application program with information about the execution of its SQL statements.

**SQL connection**

An association between an application process and a local or remote application server or database server.

**SQLDA**

See SQL descriptor area.

**SQL descriptor area (SQLDA)**

A structure that describes input variables, output variables, or the columns of a result table.

**SQL escape character**

The symbol that is used to enclose an SQL delimited identifier. This symbol is the double quotation mark ("). See also escape character.

**SQL function**

A user-defined function in which the CREATE FUNCTION statement contains the source code. The source code is a single SQL expression that evaluates to a single value. The SQL user-defined function can return the result of an expression. See also built-in function, external function, and sourced function.

**SQL ID**

See SQL authorization ID.

**SQLJ** Structured Query Language (SQL) that is embedded in the Java programming language.

**SQL path**

An ordered list of schema names that are used in the resolution of unqualified references to user-defined functions, distinct types, and stored procedures. In dynamic SQL, the SQL path is found in the CURRENT PATH special register. In static SQL, it is defined in the PATH bind option.

**SQL procedure**

A user-written program that can be

invoked with the SQL CALL statement.  
An SQL procedure is written in the SQL procedural language. Two types of SQL procedures are supported: external SQL procedures and native SQL procedures. See also external procedure and native SQL procedure.

### **SQL processing conversation**

Any conversation that requires access of DB2 data, either through an application or by dynamic query requests.

### **SQL Processor Using File Input (SPUFI)**

A facility of the TSO attachment subcomponent that enables the DB2I user to execute SQL statements without embedding them in an application program.

### **SQL return code**

Either SQLCODE or SQLSTATE.

### **SQL routine**

A user-defined function or stored procedure that is based on code that is written in SQL.

### **SQL schema**

A collection of database objects such as tables, views, indexes, functions, distinct types, schemas, or triggers that defines a database. An SQL schema provides a logical classification of database objects.

### **SQL statement coprocessor**

An alternative to the DB2 precompiler that lets the user process SQL statements at compile time. The user invokes an SQL statement coprocessor by specifying a compiler option.

### **SQL string delimiter**

A symbol that is used to enclose an SQL string constant. The SQL string delimiter is the apostrophe ('), except in COBOL applications, where the user assigns the symbol, which is either an apostrophe or a double quotation mark (").

**SRB** See service request block.

### **stand-alone**

An attribute of a program that means that it is capable of executing separately from DB2, without using DB2 services.

### **star join**

A method of joining a dimension column of a fact table to the key column of the

corresponding dimension table. See also join, dimension, and star schema.

### **star schema**

The combination of a fact table (which contains most of the data) and a number of dimension tables. See also star join, dimension, and dimension table.

### **statement handle**

In DB2 ODBC, the data object that contains information about an SQL statement that is managed by DB2 ODBC. This includes information such as dynamic arguments, bindings for dynamic arguments and columns, cursor information, result values, and status information. Each statement handle is associated with the connection handle.

### **statement string**

For a dynamic SQL statement, the character string form of the statement.

### **statement trigger**

A trigger that is defined with the trigger granularity FOR EACH STATEMENT.

### **static cursor**

A named control structure that does not change the size of the result table or the order of its rows after an application opens the cursor. Contrast with dynamic cursor.

### **static SQL**

SQL statements, embedded within a program, that are prepared during the program preparation process (before the program is executed). After being prepared, the SQL statement does not change (although values of variables that are specified by the statement might change).

### **storage group**

A set of storage objects on which DB2 for z/OS data can be stored. A storage object can have an SMS data class, a management class, a storage class, and a list of volume serial numbers.

### **stored procedure**

A user-written application program that can be invoked through the use of the SQL CALL statement. Stored procedures are sometimes called procedures.

**string** See binary string, character string, or graphic string.

**strong typing**

A process that guarantees that only user-defined functions and operations that are defined on a distinct type can be applied to that type. For example, you cannot directly compare two currency types, such as Canadian dollars and U.S. dollars. But you can provide a user-defined function to convert one currency to the other and then do the comparison.

**structure**

- A name that refers collectively to different types of DB2 objects, such as tables, databases, views, indexes, and table spaces.
- A construct that uses z/OS to map and manage storage on a coupling facility. See also cache structure, list structure, or lock structure.

**Structured Query Language (SQL)**

A standardized language for defining and manipulating data in a relational database.

**structure owner**

In relation to group buffer pools, the DB2 member that is responsible for the following activities:

- Coordinating rebuild, checkpoint, and damage assessment processing
- Monitoring the group buffer pool threshold and notifying castout owners when the threshold has been reached

**subcomponent**

A group of closely related DB2 modules that work together to provide a general function.

**subject table**

The table for which a trigger is created. When the defined triggering event occurs on this table, the trigger is activated.

**subquery**

A SELECT statement within the WHERE or HAVING clause of another SQL statement; a nested SQL statement.

**subselect**

That form of a query that includes only a SELECT clause, FROM clause, and optionally a WHERE clause, GROUP BY clause, HAVING clause, ORDER BY clause, or FETCH FIRST clause.

**substitution character**

A unique character that is substituted during character conversion for any characters in the source program that do not have a match in the target coding representation.

**subsystem**

A distinct instance of a relational database management system (RDBMS).

**surrogate pair**

A coded representation for a single character that consists of a sequence of two 16-bit code units, in which the first value of the pair is a high-surrogate code unit in the range U+D800 through U+DBFF, and the second value is a low-surrogate code unit in the range U+DC00 through U+DFFF. Surrogate pairs provide an extension mechanism for encoding 917 476 characters without requiring the use of 32-bit characters.

**SVC dump**

A dump that is issued when a z/OS or a DB2 functional recovery routine detects an error.

**sync point**

See commit point.

**syncpoint tree**

The tree of recovery managers and resource managers that are involved in a logical unit of work, starting with the recovery manager, that make the final commit decision.

**synonym**

In SQL, an alternative name for a table or view. Synonyms can be used to refer only to objects at the subsystem in which the synonym is defined. A synonym cannot be qualified and can therefore not be used by other users. Contrast with alias.

**Sysplex**

See Parallel Sysplex.

**Sysplex query parallelism**

Parallel execution of a single query that is accomplished by using multiple tasks on more than one DB2 subsystem. See also query CP parallelism.

**system administrator**

The person at a computer installation who designs, controls, and manages the use of the computer system.

**system agent**

A work request that DB2 creates such as prefetch processing, deferred writes, and service tasks. See also allied agent.

**system authorization ID**

The primary DB2 authorization ID that is used to establish a trusted connection. A system authorization ID is derived from the system user ID that is provided by an external entity, such as a middleware server.

**system conversation**

The conversation that two DB2 subsystems must establish to process system messages before any distributed processing can begin.

**System Modification Program/Extended (SMP/E)**

A z/OS tool for making software changes in programming systems (such as DB2) and for controlling those changes.

**Systems Network Architecture (SNA)**

The description of the logical structure, formats, protocols, and operational sequences for transmitting information through and controlling the configuration and operation of networks.

**table**

A named data object consisting of a specific number of columns and some number of unordered rows. See also base table or temporary table. Contrast with auxiliary table, clone table, materialized query table, result table, and transition table.

**table-controlled partitioning**

A type of partitioning in which partition boundaries for a partitioned table are controlled by values that are defined in the CREATE TABLE statement.

**table function**

A function that receives a set of arguments and returns a table to the SQL statement that references the function. A table function can be referenced only in the FROM clause of a subselect.

**table locator**

A mechanism that allows access to trigger tables in SQL or from within user-defined functions. A table locator is a fullword integer value that represents a transition table.

**table space**

A page set that is used to store the records in one or more tables. See also partitioned table space, segmented table space, and universal table space.

**table space set**

A set of table spaces and partitions that should be recovered together for one of the following reasons:

- Each of them contains a table that is a parent or descendent of a table in one of the others.
- The set contains a base table and associated auxiliary tables.

A table space set can contain both types of relationships.

**task control block (TCB)**

A z/OS control block that is used to communicate information about tasks within an address space that is connected to a subsystem. See also address space connection.

**TB**

Terabyte. A value of 1 099 511 627 776 bytes.

**TCB**

See task control block.

**TCP/IP**

A network communication protocol that computer systems use to exchange information across telecommunication links.

**TCP/IP port**

A 2-byte value that identifies an end user or a TCP/IP network application within a TCP/IP host.

**template**

A DB2 utilities output data set descriptor that is used for dynamic allocation. A template is defined by the TEMPLATE utility control statement.

**temporary table**

A table that holds temporary data. Temporary tables are useful for holding or sorting intermediate results from queries that contain a large number of rows. The two types of temporary table, which are created by different SQL statements, are the created temporary table and the declared temporary table. Contrast with result table. See also created temporary table and declared temporary table.

**thread** See DB2 thread.

**threadsafe**

A characteristic of code that allows multithreading both by providing private storage areas for each thread, and by properly serializing shared (global) storage areas.

**three-part name**

The full name of a table, view, or alias. It consists of a location name, a schema name, and an object name, separated by a period.

**time** A three-part value that designates a time of day in hours, minutes, and seconds.

**timeout**

Abnormal termination of either the DB2 subsystem or of an application because of the unavailability of resources. Installation specifications are set to determine both the amount of time DB2 is to wait for IRLM services after starting, and the amount of time IRLM is to wait if a resource that an application requests is unavailable. If either of these time specifications is exceeded, a timeout is declared.

**Time-Sharing Option (TSO)**

An option in z/OS that provides interactive time sharing from remote terminals.

**timestamp**

A seven-part value that consists of a date and time. The timestamp is expressed in years, months, days, hours, minutes, seconds, and microseconds.

**trace** A DB2 facility that provides the ability to monitor and collect DB2 monitoring, auditing, performance, accounting, statistics, and serviceability (global) data.

**transaction**

An atomic series of SQL statements that make up a logical unit of work. All of the data modifications made during a transaction are either committed together as a unit or rolled back as a unit.

**transaction lock**

A lock that is used to control concurrent execution of SQL statements.

**transaction program name**

In SNA LU 6.2 conversations, the name of

the program at the remote logical unit that is to be the other half of the conversation.

**transition table**

A temporary table that contains all the affected rows of the subject table in their state before or after the triggering event occurs. Triggered SQL statements in the trigger definition can reference the table of changed rows in the old state or the new state. Contrast with auxiliary table, base table, clone table, and materialized query table.

**transition variable**

A variable that contains a column value of the affected row of the subject table in its state before or after the triggering event occurs. Triggered SQL statements in the trigger definition can reference the set of old values or the set of new values.

**tree structure**

A data structure that represents entities in nodes, with a most one parent node for each node, and with only one root node.

**trigger**

A database object that is associated with a single base table or view and that defines a rule. The rule consists of a set of SQL statements that run when an insert, update, or delete database operation occurs on the associated base table or view.

**trigger activation**

The process that occurs when the trigger event that is defined in a trigger definition is executed. Trigger activation consists of the evaluation of the triggered action condition and conditional execution of the triggered SQL statements.

**trigger activation time**

An indication in the trigger definition of whether the trigger should be activated before or after the triggered event.

**trigger body**

The set of SQL statements that is executed when a trigger is activated and its triggered action condition evaluates to true. A trigger body is also called triggered SQL statements.

**trigger cascading**

The process that occurs when the

triggered action of a trigger causes the activation of another trigger.

#### **triggered action**

The SQL logic that is performed when a trigger is activated. The triggered action consists of an optional triggered action condition and a set of triggered SQL statements that are executed only if the condition evaluates to true.

#### **triggered action condition**

An optional part of the triggered action. This Boolean condition appears as a WHEN clause and specifies a condition that DB2 evaluates to determine if the triggered SQL statements should be executed.

#### **triggered SQL statements**

The set of SQL statements that is executed when a trigger is activated and its triggered action condition evaluates to true. Triggered SQL statements are also called the trigger body.

#### **trigger granularity**

In SQL, a characteristic of a trigger, which determines whether the trigger is activated:

- Only once for the triggering SQL statement
- Once for each row that the SQL statement modifies

#### **triggering event**

The specified operation in a trigger definition that causes the activation of that trigger. The triggering event is comprised of a triggering operation (insert, update, or delete) and a subject table or view on which the operation is performed.

#### **triggering SQL operation**

The SQL operation that causes a trigger to be activated when performed on the subject table or view.

#### **trigger package**

A package that is created when a CREATE TRIGGER statement is executed. The package is executed when the trigger is activated.

#### **trust attribute**

An attribute on which to establish trust. A trusted relationship is established based on one or more trust attributes.

#### **trusted connection**

A database connection whose attributes match the attributes of a unique trusted context defined at the DB2 database server.

#### **trusted connection reuse**

The ability to switch the current user ID on a trusted connection to a different user ID.

#### **trusted context**

A database security object that enables the establishment of a trusted relationship between a DB2 database management system and an external entity.

#### **trusted context default role**

A role associated with a trusted context. The privileges granted to the trusted context default role can be acquired only when a trusted connection based on the trusted context is established or reused.

#### **trusted context user**

A user ID to which switching the current user ID on a trusted connection is permitted.

#### **trusted context user-specific role**

A role that is associated with a specific trusted context user. It overrides the trusted context default role if the current user ID on the trusted connection matches the ID of the specific trusted context user.

#### **trusted relationship**

A privileged relationship between two entities such as a middleware server and a database server. This relationship allows for a unique set of interactions between the two entities that would be impossible otherwise.

**TSO** See Time-Sharing Option.

#### **TSO attachment facility**

A DB2 facility consisting of the DSN command processor and DB2I. Applications that are not written for the CICS or IMS environments can run under the TSO attachment facility.

#### **typed parameter marker**

A parameter marker that is specified along with its target data type. It has the general form:

CAST(? AS data-type)

#### **type 2 indexes**

Indexes that are created on a release of

DB2 after Version 7 or that are specified as type 2 indexes in Version 4 or later.

**UCS-2** Universal Character Set, coded in 2 octets, which means that characters are represented in 16-bits per character.

**UDF** See user-defined function.

**UDT** User-defined data type. In DB2 for z/OS, the term distinct type is used instead of user-defined data type. See distinct type.

**uncommitted read (UR)**

The isolation level that allows an application to read uncommitted data. See also cursor stability, read stability, and repeatable read.

**underlying view**

The view on which another view is directly or indirectly defined.

**undo** A state of a unit of recovery that indicates that the changes that the unit of recovery made to recoverable DB2 resources must be backed out.

**Unicode**

A standard that parallels the ISO-10646 standard. Several implementations of the Unicode standard exist, all of which have the ability to represent a large percentage of the characters that are contained in the many scripts that are used throughout the world.

**union** An SQL operation that involves the UNION set operator, which combines the results of two SELECT statements. Unions are often used to merge lists of values that are obtained from two tables.

**unique constraint**

An SQL rule that no two values in a primary key, or in the key of a unique index, can be the same.

**unique index**

An index that ensures that no identical key values are stored in a column or a set of columns in a table.

**unit of recovery (UOR)**

A recoverable sequence of operations within a single resource manager, such as an instance of DB2. Contrast with unit of work.

**unit of work (UOW)**

A recoverable sequence of operations within an application process. At any

time, an application process is a single unit of work, but the life of an application process can involve many units of work as a result of commit or rollback operations. In a multisite update operation, a single unit of work can include several *units of recovery*. Contrast with unit of recovery.

**universal table space**

A table space that is both segmented and partitioned.

**unlock**

The act of releasing an object or system resource that was previously locked and returning it to general availability within DB2.

**untyped parameter marker**

A parameter marker that is specified without its target data type. It has the form of a single question mark (?).

**updatability**

The ability of a cursor to perform positioned updates and deletes. The updatability of a cursor can be influenced by the SELECT statement and the cursor sensitivity option that is specified on the DECLARE CURSOR statement.

**update hole**

The location on which a cursor is positioned when a row in a result table is fetched again and the new values no longer satisfy the search condition. See also delete hole.

**update trigger**

A trigger that is defined with the triggering SQL operation update.

**UR** See uncommitted read.

**user-defined data type (UDT)**

See distinct type.

**user-defined function (UDF)**

A function that is defined to DB2 by using the CREATE FUNCTION statement and that can be referenced thereafter in SQL statements. A user-defined function can be an external function, a sourced function, or an SQL function. Contrast with built-in function.

**user view**

In logical data modeling, a model or representation of critical information that the business requires.

## UTF-16

Unicode Transformation Format, 16-bit encoding form, which is designed to provide code values for over a million characters and a superset of UCS-2. The CCSID value for data in UTF-16 format is 1200. DB2 for z/OS supports UTF-16 in graphic data fields.

## UTF-8

Unicode Transformation Format, 8-bit encoding form, which is designed for ease of use with existing ASCII-based systems. The CCSID value for data in UTF-8 format is 1208. DB2 for z/OS supports UTF-8 in mixed data fields.

**value** The smallest unit of data that is manipulated in SQL.

## variable

A data element that specifies a value that can be changed. A COBOL elementary data item is an example of a host variable. Contrast with constant.

## variant function

See nondeterministic function.

## varying-length string

A character, graphic, or binary string whose length varies within set limits. Contrast with fixed-length string.

## version

A member of a set of similar programs, DBRMs, packages, or LOBs.

- **A version of a program** is the source code that is produced by precompiling the program. The program version is identified by the program name and a timestamp (consistency token).
- **A version of an SQL procedural language routine** is produced by issuing the CREATE or ALTER PROCEDURE statement for a native SQL procedure.
- **A version of a DBRM** is the DBRM that is produced by precompiling a program. The DBRM version is identified by the same program name and timestamp as a corresponding program version.
- **A version of an application package** is the result of binding a DBRM within a particular database system. The application package version is identified by the same program name and consistency token as the DBRM.

- **A version of a LOB** is a copy of a LOB value at a point in time. The version number for a LOB is stored in the auxiliary index entry for the LOB.
- **A version of a record** is a copy of the record at a point in time.

## view

A logical table that consists of data that is generated by a query. A view can be based on one or more underlying base tables or views, and the data in a view is determined by a SELECT statement that is run on the underlying base tables or views.

## Virtual Storage Access Method (VSAM)

An access method for direct or sequential processing of fixed- and varying-length records on disk devices.

## Virtual Telecommunications Access Method (VTAM)

An IBM licensed program that controls communication and the flow of data in an SNA network (in z/OS).

## volatile table

A table for which SQL operations choose index access whenever possible.

## VSAM

See Virtual Storage Access Method.

## VTAM

See Virtual Telecommunications Access Method.

## warm start

The normal DB2 restart process, which involves reading and processing log records so that data that is under the control of DB2 is consistent. Contrast with cold start.

## WLM application environment

A z/OS Workload Manager attribute that is associated with one or more stored procedures. The WLM application environment determines the address space in which a given DB2 stored procedure runs.

## WLM enclave

A construct that can span multiple dispatchable units (service request blocks and tasks) in multiple address spaces, allowing them to be reported on and managed by WLM as part of a single work request.

**write to operator (WTO)**

An optional user-coded service that allows a message to be written to the system console operator informing the operator of errors and unusual system conditions that might need to be corrected (in z/OS).

**WTO** See write to operator.

**WTOR**

Write to operator (WTO) with reply.

**XCF** See cross-system coupling facility.

**XES** See cross-system extended services.

**XML** See Extensible Markup Language.

**XML attribute**

A name-value pair within a tagged XML element that modifies certain features of the element.

**XML column**

A column of a table that stores XML values and is defined using the data type XML. The XML values that are stored in XML columns are internal representations of well-formed XML documents.

**XML data type**

A data type for XML values.

**XML element**

A logical structure in an XML document that is delimited by a start and an end tag. Anything between the start tag and the end tag is the content of the element.

**XML index**

An index on an XML column that provides efficient access to nodes within an XML document by providing index keys that are based on XML patterns.

**XML lock**

A column-level lock for XML data. The operation of XML locks is similar to the operation of LOB locks.

**XML node**

The smallest unit of valid, complete structure in a document. For example, a node can represent an element, an attribute, or a text string.

**XML node ID index**

An implicitly created index, on an XML table that provides efficient access to XML

documents and navigation among multiple XML data rows in the same document.

**XML pattern**

A slash-separated list of element names, an optional attribute name (at the end), or kind tests, that describe a path within an XML document in an XML column. The pattern is a restrictive form of path expressions, and it selects nodes that match the specifications. XML patterns are specified to create indexes on XML columns in a database.

**XML publishing function**

A function that returns an XML value from SQL values. An XML publishing function is also known as an XML constructor.

**XML schema**

In XML, a mechanism for describing and constraining the content of XML files by indicating which elements are allowed and in which combinations. XML schemas are an alternative to document type definitions (DTDs) and can be used to extend functionality in the areas of data typing, inheritance, and presentation.

**XML schema repository (XSR)**

A repository that allows the DB2 database system to store XML schemas. When registered with the XSR, these objects have a unique identifier and can be used to validate XML instance documents.

**XML serialization function**

A function that returns a serialized XML string from an XML value.

**XML table**

An auxiliary table that is implicitly created when an XML column is added to a base table. This table stores the XML data, and the column in the base table points to it.

**XML table space**

A table space that is implicitly created when an XML column is added to a base table. The table space stores the XML table. If the base table is partitioned, one partitioned table space exists for each XML column of data.

**X/Open**

An independent, worldwide open systems organization that is supported by most of

the world's largest information systems suppliers, user organizations, and software companies. X/Open's goal is to increase the portability of applications by combining existing and emerging standards.

**XRF** See Extended Recovery Facility.

| **XSR** See XML schema repository.

| **zIIP** See IBM System z9 Integrated Processor.

**z/OS** An operating system for the System z product line that supports 64-bit real and virtual storage.

**z/OS Distributed Computing Environment (z/OS DCE)** A set of technologies that are provided by the Open Software Foundation to implement distributed computing.

---

# Index

## A

- access method services
  - commands
    - DEFINE CLUSTER 20
- access path
  - affects lock attributes 369
  - direct row access 679
  - hints 311
  - index access 731
  - low cluster ratio
    - effects of 731
    - suggests table space scan 664
  - multiple index access
    - description 672
    - disabling 75
  - selection
    - influencing with SQL 293
    - Visual Explain 299, 421
  - table space scan 664
  - unique index with matching value 678
- Access path
  - join types 707
    - pair-wise 709
  - star schema 707
    - pair-wise 709
- access path selection 165
- accessibility
  - keyboard xiv
  - shortcut keys xiv
- accounting
  - elapsed times 424
  - trace
    - description 431
- Accounting trace
  - performance implications
    - processor resources 46
- ACQUIRE
  - option of BIND PLAN subcommand
    - locking tables and table spaces 353
    - thread creation 138
- active log
  - data set
    - placement 85
  - size
    - determining 104
    - tuning considerations 104
- allocating space
  - effect on INSERT operations 21, 87
  - preformatting 21, 87
- ambiguous cursor 364, 407
- application plan
  - monitoring 601
- application program
  - suspension
    - description 320
    - timeout periods 349
- ARCHIVE LOG FREQ field of panel DSNTIPL 103
- ASUTIME column
  - resource limit specification table (RLST) 35
- asynchronous data from IFI 464
- audit trace
  - description 433
- auditing
  - data 433
  - trace data through IFI 473
- authorization ID
  - checking during thread creation 138
- automatic query rewrite
  - description of process 117
  - determining occurrence of 124
  - enabling query optimization 124
  - examples 119
  - exploiting 116
  - introduction 116
  - query requirements 117
- auxiliary table
  - LOCK TABLE statement 380, 384
- AVGKEYLEN column
  - SYSINDEXES catalog table 157
  - SYSINDEXPART catalog table 157
- AVGROWLEN column
  - SYSTABLEPART catalog table 162
  - SYSTABLES catalog table
    - data collected by RUNSTATS utility 162
  - SYSTABLES\_HIST catalog table 171
  - SYSTABLESPACE catalog table 163
- AVGSIZE column
  - SYSLOBSTATS catalog table 161

## B

- BIND PACKAGE subcommand of DSN
  - options
    - RELEASE 353
    - REOPT(ALWAYS) 279
    - REOPT(NONE) 279
    - REOPT(ONCE) 279
- BIND PLAN subcommand of DSN
  - options
    - ACQUIRE 353
    - RELEASE 353
    - REOPT(ALWAYS) 279
    - REOPT(NONE) 279
    - REOPT(ONCE) 279
- block fetch
  - description 404
  - enabling 407
  - LOB data impact 406
  - scrollable cursors 406
- buffer information area used in IFI 443
- buffer pool
  - advantages of large pools 65
  - advantages of multiple pools 479
  - allocating storage 67
  - altering attributes 479
  - available pages 56
  - displaying current status 479
  - hit ratio 65
  - immediate writes 481
  - in-use pages 56
  - long-term page fix option 69

- buffer pool (*continued*)
  - monitoring 422, 481
  - page-stealing algorithm 69
  - read operations 57
  - reading
    - dynamic prefetch 57
    - normal read 57
    - sequential prefetch 57
  - size 64, 65, 139
  - statistics 481
  - thresholds 59, 481
  - update efficiency 481
  - updated pages 56
  - write efficiency 481
  - write operations 57

- buffer pools
  - I/O 18
  - size recommendations 18
- BUFFERPOOL clause
  - ALTER INDEX statement 58
  - ALTER TABLESPACE statement 58
  - CREATE DATABASE statement 58
  - CREATE INDEX statement 58
  - CREATE TABLESPACE statement 58

## C

- cache 84
- Cache 53
  - sort performance 84
- CAF (call attachment facility)
  - DSNALI language interface module 442
- CARDF column
  - SYSOLDIST catalog table
    - access path selection 154
    - data collected by RUNSTATS utility 154
  - SYSOLDIST\_HIST catalog table 167
  - SYSINDEXPART catalog table
    - data collected by RUNSTATS utility 158
  - SYSINDEXPART\_HIST catalog table 168
  - SYSKEYTARGETS\_HIST catalog table 169
  - SYSKEYTGTDIST catalog table
    - access path selection 160
    - data collected by RUNSTATS utility 160
  - SYSKEYTGTDIST\_HIST catalog table 169
  - SYSTABLEPART catalog table
    - data collected by RUNSTATS utility 162
  - SYSTABLEPART\_HIST catalog table 170
  - SYSTABLES catalog table
    - data collected by RUNSTATS utility 162
  - SYSTABLES\_HIST catalog table 171
  - SYSTABSTATS catalog table
    - data collected by RUNSTATS utility 163
  - SYSTABSTATS\_HIST catalog table 171
- cardinality 165
- CARDINALITY column of SYSROUTINES catalog table 161
- cardinality of user-defined table function
  - improving query performance 304
- Cartesian join 694
- catalog statistics
  - history 167, 500
  - influencing access paths 308
- catalog tables
  - historical statistics 167, 500
  - SYSOLDIST
    - data collected by RUNSTATS utility 153
  - SYSOLDIST\_HIST 167

- catalog tables (*continued*)
  - SYSOLDISTSTATS
    - data collected by RUNSTATS utility 154, 160
  - SYSOLSTATS
    - data collected by RUNSTATS utility 155
  - SYSOLUMNS
    - data collected by RUNSTATS utility 156
  - SYSOLUMNS\_HIST 168
  - SYSINDEXES
    - access path selection 731
    - data collected by RUNSTATS utility 157
  - SYSINDEXES\_HIST 168
  - SYSINDEXPART\_HIST 168
  - SYSINDEXSTATS
    - data collected by RUNSTATS utility 158
  - SYSINDEXSTATS\_HIST 169
  - SYSKEYTARGETS catalog table 159
  - SYSKEYTARGETS\_HIST catalog table 169
  - SYSKEYTARGETSTATS catalog table 159
  - SYSKEYTGTDIST
    - data collected by RUNSTATS utility 160
  - SYSKEYTGTDIST\_HIST catalog table 169
  - SYSLOBSTATS
    - data collected by RUNSTATS utility 159
  - SYSLOBSTATS\_HIST 170
  - SYSPLANAUTH
    - checked during thread creation 138
  - SYSROUTINES
    - data collected by RUNSTATS utility 161
  - SYSTABLEPART
    - data collected by RUNSTATS utility 162
    - PAGESAVE column 94
    - updated by LOAD and REORG utilities for data compression 94
  - SYSTABLEPART\_HIST 170
  - SYSTABLES
    - data collected by RUNSTATS utility 162
    - updated by LOAD and REORG for data compression 94
  - SYSTABLES\_HIST 171
  - SYSTABLESPACE
    - data collected by RUNSTATS utility 163
  - SYSTABSTATS
    - data collected by RUNSTATS utility 163
    - PCTROWCOMP column 94
  - SYSTABSTATS\_HIST 171
- catalog, DB2
  - locks 337
  - statistics
    - production system 172
    - querying the catalog 497
  - tuning 86
- CD-ROM, books on 755
- Channels
  - ESCON 53
  - FICON 53
  - Performance
    - channels 53
    - I/O parallelism 53
- CHECKPOINT FREQ field of panel DSNTIPN 104
- CICS
  - correlating DB2 and CICS accounting records 620
  - facilities
    - monitoring facility (CMF) 419, 639
    - tools 426
  - language interface module (DSNCLI)
    - IFI entry point 442

- CICS (*continued*)
    - statistics 419
  - claim
    - class 386
    - definition 385
  - Class 1 elapsed time 638
  - CLOSE
    - clause of CREATE TABLESPACE statement
    - deferred close 81
  - cluster ratio
    - description 731
    - effects
      - low cluster ratio 731
      - table space scan 664
  - CLUSTERED column of SYSINDEXES catalog table
    - data collected by RUNSTATS utility 157
  - CLUSTERING column
    - SYSINDEXES\_HIST catalog table 168
  - CLUSTERING column of SYSINDEXES catalog table
    - access path selection 157
  - CLUSTERRATIO column
    - SYSINDEXSTATS\_HIST catalog table 169
  - CLUSTERRATIOOF column
    - SYSINDEXES catalog table
      - data collected by RUNSTATS utility 157
    - SYSINDEXES\_HIST catalog table 168
    - SYSINDEXSTATS catalog table
      - access path selection 159
  - coding 8
  - Coding
    - performance planning 8
  - COLCARD column of SYSCOLSTATS catalog table
    - data collected by RUNSTATS utility 155
    - updating 740
  - COLCARDATA column of SYSCOLSTATS catalog table 155
  - COLCARDF column
    - SYSOLUMNS catalog table 156
    - SYSOLUMNS\_HIST catalog table 168
  - COLCARDF column of SYSCOLUMNS catalog table
    - statistics not exact 164
    - updating 740
  - COLGROUPCOLNO column
    - SYSOLDIST catalog table
      - access path selection 154
    - SYSOLDIST\_HIST catalog table 167
    - SYSOLDISTSTATS catalog table
      - data collected by RUNSTATS utility 155
  - column
    - SYSKEYTARGETSTATS catalog table
      - data collected by RUNSTATS utility 159
  - COLVALUE column
    - SYSOLDIST catalog table
      - access path selection 154
    - SYSOLDIST\_HIST catalog table 167
    - SYSOLDISTSTATS catalog table
      - data collected by RUNSTATS utility 155
  - commands
    - concurrency 319, 385
    - issuing DB2 commands from IFI 443
  - compatibility
    - locks 335
  - complex trigger
    - WHEN clause 683
  - Compressed data
    - performance 94
  - compressing 96
  - Compressing
    - Indexes 96
  - compressing data 91
  - concurrency
    - commands 319, 385
    - contention independent of databases 338
    - control by drains and claims 385
    - control by locks 319
    - description 319
    - effect of
      - ISOLATION options 360, 361
      - lock escalation 344
      - LOCKSIZE options 372
      - row locks 372
    - recommendations 323
    - utilities 319, 385
    - utility compatibility 388
    - with real-time statistics 231
  - Concurrency
    - reports 485
    - scenario
      - reports 485
  - connection
    - DB2
      - thread 151
  - continuous block fetch 404
  - Copies
    - package 295
  - COPY utility
    - effect on real-time statistics 227
  - COST\_CATEGORY\_B column of RLST 36
  - CP processing, disabling parallel operations 61
  - CPU
    - reducing 45
  - created temporary table
    - table space scan 664
  - CS (cursor stability)
    - claim class 386
    - drain lock 387
    - optimistic concurrency control 357
    - page and row locking 357
  - CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION special register 124
  - CURRENT REFRESH AGE special register 124
  - CURRENTDATA option
    - BIND PACKAGE subcommand
      - enabling block fetch 407
    - BIND PLAN subcommand 407
  - CURRENTDATA option of BIND
    - plan and package options differ 365
  - cursor
    - ambiguous 364, 407
    - defined WITH HOLD, subsystem parameter to release
      - locks 376
- ## D
- data
    - compression 91
    - effect of locks on integrity 319
    - improving access 421
    - understanding access 421
  - data compression
    - determining effectiveness 94
    - DSN1COMP utility 94
    - performance considerations 91
    - performance implications 92

- data set
  - allocation and extension 139
  - closing 81
  - DSMAX value 79
  - limit 79
  - monitoring I/O activity 479
  - open 79, 139
- data sets
  - frequently used 18
- data sharing
  - real-time statistics 230
  - using IFI 472
- database
  - access thread
    - differences from allied threads 143
- database descriptor (DBD) 70
- DATA REPEAT FACTOR column
  - SYSINDEXES catalog table
    - data collected by RUNSTATS utility 157
  - SYSINDEXES\_HIST catalog table 168, 169
  - SYSINDEXSTATS catalog table
    - data collected by RUNSTATS utility 159
- DB2 books online 755
- DB2 Buffer Pool Analyzer
  - description 422
- DB2 commands
  - issuing from IFI 439, 443
- DB2 data set statistics
  - obtaining through IFCID 0199 460
- DB2 Information Center for z/OS solutions 755
- DB2 performance 415
- DB2 Performance Expert
  - description 422
- DB2 Performance Monitor (DB2 PM) 637
- DB2 PM (DB2 Performance Monitor)
  - EXPLAIN 172
- DB2 private protocol access
  - description 401
  - resource limit facility 41
- DBD (database descriptor)
  - EDM pool 70
  - freeing 140
  - load
    - in EDM pool 139
    - using ACQUIRE(ALLOCATE) 138
  - locks on 339
  - use count 140
- DBD01 directory table space
  - placement of data sets 85
- DBFULTA0 (Fast Path Log Analysis Utility) 419
- DDF (distributed data facility)
  - block fetch 404
- deadlock
  - description 321
  - detection scenarios 493
  - example 321
  - indications
    - in CICS 323
    - in IMS 322
    - in TSO 322
  - recommendation for avoiding 325
  - row vs. page locks 372
  - wait time calculation 349
  - with RELEASE(DEALLOCATE) 325
  - X'00C9088' reason code in SQLCA 321
- DEADLOCK TIME field of panel DSNTIPJ 348
- DEADLOCK option of START irlmproc command 347
- DECFLOAT data type 275
- deferred close 78
- deferred write threshold (DWQT)
  - description 61
  - recommendation for LOBs 63
- DEFINE CLUSTER command of access method services 20
- Design
  - external
    - performance 8
  - internal
    - performance 8
- DFSII000 (IMS language interface module) 442
- direct row access 679
- Direct row access
  - incompatible access methods 681
- disability xiv
- disk 83
  - data set, allocation and extension 85
  - improving use 85
- DISPLAY THREAD command
  - shows parallel tasks 602
- displaying
  - buffer pool information 479
- distributed data
  - DB2 private protocol access 401
  - DRDA protocol 401
  - operating
    - displaying status 460
    - in an overloaded network 7
  - performance considerations 403
  - programming
    - block fetch 404
    - FOR FETCH ONLY 407
    - FOR READ ONLY 407
  - resource limit facility 41
  - server-elapsed time monitoring 609
  - tuning 403
- distributed data facility (DDF) 404
- distribution statistics 741
- DPSI
  - performance considerations 290
- drain
  - definition 386
  - DRAIN ALL 389
  - wait calculation 352
- drain lock
  - description 319, 387
  - types 387
  - wait calculation 352
- DRDA access
  - description 401
  - resource limit facility 41
- DSMAX
  - DSMAX
    - calculating 79
  - formula for 79
  - initial value 79
  - limit on open data sets 79
  - modifying 79
- DSN\_FUNCTION\_TABLE
  - column descriptions 658
  - format 561
- DSN\_PREDICAT\_TABLE 503
- DSN\_STATEMENT\_CACHE\_TABLE
  - column descriptions 661
- DSN\_STATEMENT\_CACHE\_TABLE format 563

- DSN\_STATEMNT\_TABLE
  - column descriptions 659
  - format 562
- DSN1COMP 96
- DSN1COMP utility
  - description 94
- DSN6SPRM macro
  - RELCURHL parameter 376
- DSN6SYSP macro
  - PCLOSEN parameter 82
  - PCLOSET parameter 82
- DSNACCOR stored procedure
  - description 199
  - example call 212
  - option descriptions 201
  - output 216
  - syntax diagram 200
- DSNACCOX stored procedure
  - description 176
  - option descriptions 178
  - output 191
  - syntax diagram 177
- DSNAEXP stored procedure
  - description 567
  - example call 568
  - option descriptions 568
  - output 569
  - syntax diagram 567
- DSNALI (CAF language interface module)
  - inserting 442
- DSNCLI (CICS language interface module)
  - entry point 442
- DSNDIFCA mapping macro 466
- DSNDQWIW mapping macro 645
- DSNDWBUF mapping macro 443
- DSNDWQAL mapping macro 448
- DSNELI (TSO language interface module) 442
- DSNTESP data set 501, 733
- DSNTIJSJ job
  - installation 33
- DSNUM column
  - SYSINDEXPART catalog table
    - data collected by RUNSTATS utility 158
  - SYSINDEXPART\_HIST catalog table 168
  - SYSTABLEPART catalog table
    - data collected by RUNSTATS utility 162
  - SYSTABLEPART\_HIST catalog table 170
- duration of locks
  - controlling 353
  - description 334
- DWQT option of ALTER BUFFERPOOL command 61
- dynamic prefetch
  - description 686
- dynamic SQL
  - skeletons, EDM pool 70

## E

- EDM pool
  - DBD freeing 140
  - description 70
- EDPROC column of SYSTABLES catalog table 163
- error
  - IFI (instrumentation facility interface) 474
- escalation, lock 344
- EVALUATE UNCOMMITTED field of panel DSNTIP4 377

- EXCLUSIVE
  - lock mode
    - effect on resources 334
    - LOB 379
    - page 334
    - row 334
    - table, partition, and table space 334
    - XML 383
- EXPLAIN 503, 649
  - creating tables 558
  - report of outer join 691
  - SQL statements
    - analyzing 557
  - statement
    - alternative using IFI 441
    - description 421
    - executing under DB2 QMF 571
    - index scans 670
    - interpreting output 649
    - investigating SQL processing 421
  - tables 558
- EXPLAIN tables 503, 526, 527, 658, 659, 661
  - creating 561, 562, 563
  - Creating 559
  - DSN\_DETCOST\_TABLE 516
  - DSN\_FILTER\_TABLE 514
  - DSN\_FUNCTION\_TABLE 561
  - DSN\_PGROUPTABLE 510
  - DSN\_PTASK\_TABLE 513
  - DSN\_QUERY\_TABLE 528
  - DSN\_SORT\_TABLE 522
  - DSN\_SORTKEY\_TABLE 524
  - DSN\_STATEMENT\_CACHE\_TABLE 563
  - DSN\_STATEMNT\_TABLE 562
  - DSN\_STRUCT\_TABLE 508
  - PLAN\_TABLE 559, 649
- EXTENTS column
  - SYSINDEXPART catalog table
    - data collected by RUNSTATS utility 158
  - SYSINDEXPART\_HIST catalog table 168
  - SYSTABLEPART catalog table
    - data collected by RUNSTATS utility 162
  - SYSTABLEPART\_HIST catalog table 170

## F

- FARINDREF column
  - SYSTABLEPART\_HIST catalog table 170
- FARINDREF column of SYSTABLEPART catalog table
  - data collected by RUNSTATS utility 162
- FAROFFPOSF column
  - SYSINDEXPART\_HIST catalog table 168
- FAROFFPOSF column of SYSINDEXPART catalog table
  - data collected by RUNSTATS utility 158
- Fast Path Log Analysis Utility 419
- FETCH FIRST n ROW ONLY clause
  - effect on distributed performance 410
  - effect on OPTIMIZE clause 410
- FETCH FIRST n ROWS ONLY clause
  - effect on OPTIMIZE clause 300
- filter factor
  - catalog statistics used for determining 164
  - predicate 260
- FIRSTKEYCARD column
  - SYSINDEXSTATS catalog table
    - recommendation for updating 740

- FIRSTKEYCARDF column
  - SYSINDEXES catalog table
    - data collected by RUNSTATS utility 157
    - recommendation for updating 740
  - SYSINDEXES\_HIST catalog table 168
  - SYSINDEXSTATS catalog table
    - data collected by RUNSTATS utility 159
  - SYSINDEXSTATS\_HIST catalog table 169
- free space
  - advantages and disadvantages 89
  - recommendations 89
- Free space
  - compressed data 94
  - FREEPAGE 89, 91
  - PCTFREE 89, 90
  - reserving 89
  - single-row pages 91
- FREESPACE column
  - SYSLOBSTATS catalog table 161
  - SYSLOBSTATS\_HIST catalog table 170
- frequency statistics 165
- FREQUENCYF column
  - SYSCOLDIST catalog table
    - access path selection 154
  - SYSCOLDIST\_HIST catalog table 167
  - SYSCOLDISTSTATS catalog table 155
  - SYSKEYTGTDIST catalog table
    - access path selection 160
  - SYSKEYTGTDIST\_HIST catalog table 170
  - SYSKEYTGTDISTSTATS catalog table
    - data collected by RUNSTATS utility 161
- full image copy
  - use after LOAD 105
  - use after REORG 105
- FULLKEYCARDDATA column
  - SYSINDEXSTATS catalog table 159
- FULLKEYCARDF column
  - SYSINDEXES catalog table
    - data collected by RUNSTATS utility 157
  - SYSINDEXES\_HIST catalog table 168
  - SYSINDEXSTATS catalog table 159
  - SYSINDEXSTATS\_HIST catalog table 169
- function
  - column
    - when evaluated 665
- Function table
  - creating 561

## G

- general-use programming information, described 765
- generalized trace facility (GTF)
  - event identifiers 437
- Global trace
  - performance implications
    - processor resources 46
- governor (resource limit facility) 28
- GROUP BY clause
  - effect on OPTIMIZE clause 302
- GTF (generalized trace facility)
  - format of trace records 621
  - interpreting trace records 626
  - recording trace records 437
- GUI symbols 765

## H

- HIGH2KEY column
  - SYSCOLSTATS catalog table 156
  - SYSCOLUMNS catalog table
    - access path selection 156
    - recommendation for updating 740
  - SYSCOLUMNS\_HIST catalog table 168
  - SYSKEYTARGETS catalog table
    - data collected by RUNSTATS utility 159
  - SYSKEYTARGETS\_HIST catalog table 169
- HIGHKEY column
  - SYSKEYTARGETSTATS catalog table
    - data collected by RUNSTATS utility 159
- HIGHKEY column of SYSCOLSTATS catalog table 156
- HIGHVALUE 165
- HIGHVALUE column
  - SYSCOLDIST catalog table
    - access path selection 154, 155
  - SYSCOLDIST\_HIST catalog table 168
  - SYSKEYTGTDIST catalog table
    - access path selection 160, 161
  - SYSKEYTGTDIST\_HIST catalog table 170
- hints, optimization 311
- Histogram statistics 165
- host variable
  - in equal predicate 297
- hybrid join
  - description 700
  - disabling 75

## I

- I/O
  - address space 50
  - distributing 18
    - data clustering 19
    - partitioning 19
  - priorities 50
- I/O activity, monitoring by data set 479
- I/O processing
  - minimizing contention 20, 84
  - parallel
    - disabling 61
- IFCA (instrumentation facility communication area)
  - command request 443
  - description 466
  - field descriptions 466
  - IFI READS request 447
  - READA request of IFI 464
  - WRITE request of IFI 466
- IFCID (instrumentation facility component identifier)
  - 0199 479
  - area
    - description 471
    - READS request of IFI 447
    - WRITE request of IFI 466
  - description 621
  - identifiers by number
    - 0001 431, 459, 605
    - 0002 459
    - 0015 138
    - 0021 140
    - 0032 140
    - 0033 140
    - 0038 140
    - 0039 140

IFCID (instrumentation facility component identifier)  
(continued)

identifiers by number (continued)

0058 139  
0070 140  
0073 138  
0084 140  
0088 141  
0089 141  
0104 459  
0106 459  
0124 459  
0147 433, 460  
0148 460  
0149 460  
0150 460  
0185 460  
0199 460  
0202 460  
0217 460  
0221 604  
0222 604  
0225 460  
0230 460  
0234 460  
0254 460  
0258 85  
0306 460  
0316 460  
0317 461  
0345 461  
0346 461

SMF type 431

IFI (instrumentation facility interface)

asynchronous data 464  
auditing data 473  
authorization 447  
buffer information area 443  
collecting trace data, example 440  
command request, output example 647  
commands  
    READA 464  
    READS 447  
data integrity 473  
data sharing group, in a 472  
dynamic statement cache information 461  
errors 474  
issuing DB2 commands  
    example 439, 443  
    syntax 443  
locking 474  
output area  
    command request 443  
    description 471  
    example 439, 443  
passing data to DB2, example 441  
qualification area 448  
READS output 645  
READS request 448  
recovery considerations 474  
return area  
    command request 443  
    description 470  
    READA request 464  
    READS request 447  
storage requirements 447, 464  
summary of functions 441

IFI (instrumentation facility interface) (continued)

synchronous data 459  
using stored procedures 443  
writer header 645  
image copy  
    full  
        use after LOAD 105  
immediate write threshold (IWTH) 59  
improving 83  
IMS  
    facilities  
        Fast Path 150  
        processing limit 27  
        regions 150  
        tools 426  
    language interface module (DFSLI000)  
        IFI applications 442  
    planning  
        design recommendations 150  
IMS Performance Analyzer (IMS PA)  
    description 419  
    IMS transit times 639  
inactive connections 144  
index 96  
    access methods  
        access path selection 668  
        by nonmatching index 671  
        description 666  
        IN-list index scan 672  
        matching index columns 670  
        matching index description 668  
        multiple 672  
        one-fetch index scan 676  
    costs 666  
    evaluating effectiveness 95  
    locking 337  
    reasons for using 666  
Index  
    NOT PADDED 96  
    Performance 96  
Indexes 96  
INITIAL\_INSTS column of SYSROUTINES catalog table 161  
INITIAL\_IOS column of SYSROUTINES catalog table 161  
INLISTP 310  
INSERT processing, effect of MEMBER CLUSTER option of  
    CREATE TABLESPACE 324  
instrumentation facility communication area (IFCA) 443  
instrumentation facility interface (IFI) 439  
INSTS\_PER\_INVOC column of SYSROUTINES catalog  
    table 162  
integrity  
    IFI data 473  
INTENT EXCLUSIVE lock mode 334, 379  
INTENT SHARE lock mode 334, 379  
IOS\_PER\_INVOC column of SYSROUTINES catalog table 162  
IRLM (internal resource lock manager)  
    IFI trace records 459  
    OMEGAMON locking report 488  
    startup procedure options 347  
    workload manager 49  
isolation level  
    control by SQL statement  
        example 365  
    recommendations 325  
Isolation option  
    choosing 357  
    default 357

Isolation option (*continued*)

ISOLATION

(CS) 357

(RR) 357

(RS) 357

(UR) 357

preferred 357

IWITH (immediate write threshold) 59

IXNAME column

SYSKEYTARGETS catalog table

data collected by RUNSTATS utility 159

## J

Job 26

join operation

Cartesian 694

description 691

hybrid

description 700

disabling 75

merge scan 698

nested loop 694

star join 702

Join operations

star schema 702

join sequence

definition 249

JOIN\_TYPE

'P' 709

'S' 707

## K

KEYCARDATA column

SYSOLDIST catalog table

data collected by RUNSTATS utility 155

KEYCOUNTF column

SYSINDEXSTATS catalog table 159

SYSINDEXSTATS\_HIST catalog table 169

KEYGROUPKEYNO column

SYSKEYTGTDIST catalog table

access path selection 160

SYSKEYTGTDIST\_HIST catalog table 170

SYSKEYTGTDISTSTATS catalog table

data collected by RUNSTATS utility 161

KEYVALUE column

SYSKEYTGTDIST catalog table

access path selection 160

SYSKEYTGTDIST\_HIST catalog table 170

SYSKEYTGTDISTSTATS catalog table

data collected by RUNSTATS utility 161

## L

language interface modules

DFSLI000 442

DSNALI 442

DSNCLI

description 442

DSNELI 442

LEAFDIST column

SYSINDEXPART catalog table

data collected by RUNSTATS utility 158

SYSINDEXPART\_HIST catalog table 168

LEAFFAR column

SYSINDEXPART catalog table 158

example 734

SYSINDEXPART\_HIST catalog table 169

LEAFNEAR column

SYSINDEXPART catalog table 158

SYSINDEXPART\_HIST catalog table 169

level of a lock 329

library 755

limited block fetch 404

limited partition scan 682

limiting resources 26

LIMITKEY column

SYSINDEXPART catalog table 158

list prefetch

disabling 75

thresholds 687

LOAD utility

effect on real-time statistics 221

LOB

lock

description 378

LOB (large object)

block fetching 406

lock duration 379

LOCK TABLE statement 380

locking 378

LOCKSIZE clause of CREATE or ALTER

TABLESPACE 381

modes of LOB locks 379

modes of table space locks 379

when to reorganize 736

lock

avoidance 363, 377

benefits 319

class

drain 319

transaction 319

compatibility 335

DB2 installation options 348

description 319

drain

description 387

types 387

wait calculation 352

duration

controlling 353

description 334

LOBs 379

page locks 140

XML data 383

effects

deadlock 321

deadlock wait calculation 349

suspension 320

timeout 320

timeout periods 349

escalation

description 344

OMEGAMON reports 490

hierarchy

description 330

LOB locks 378

LOB table space, LOCKSIZE clause 381

maximum number 372

mode 334

modes for various processes 346

- lock (*continued*)
  - object
    - DB2 catalog 337
    - DBD 339
    - description 336
    - indexes 337
    - LOCKMAX clause 374
    - LOCKSIZE clause 372
    - SKCT (skeleton cursor table) 339
    - SKPT (skeleton package table) 339
  - options affecting
    - access path 369
    - bind 353
    - cursor stability 357
    - IFI (instrumentation facility interface) 474
    - IRLM 347
    - program 353
    - read stability 360
    - repeatable read 361
  - page locks
    - commit duration 140
    - CS, RS, and RR compared 361
    - description 329
    - performance 490
  - promotion 344
  - recommendations for concurrency 323
  - row locks
    - compared to page 372
  - size
    - controlling 372, 374
    - page 329
    - partition 329
    - table 329
    - table space 329
    - storage needed 348
    - suspension time 487
    - table of modes acquired 340
    - trace records 139
  - XML locks 381
  - XML table space, LOCKSIZE clause 385
- LOCK TABLE 366
- LOCK TABLE statement
  - effect on auxiliary tables 380, 384
- lock/latch suspension time 638
- LOCKMAX clause
  - effect of options 374
- LOCKPART clause of CREATE and ALTER TABLESPACE
  - effect on locking 332
- LOCKS PER TABLE(SPACE) field of panel DSNTIPJ 375
- LOCKS PER USER field of panel DSNTIPJ 372
- LOCKSIZE clause
  - effect of options 372, 381, 385
- log
  - buffer
    - size 99
  - determining size of active logs 104
  - performance
    - considerations 99
    - recommendations 99
  - use
    - monitoring 99
  - write threshold 99, 100
- Log
  - reads 101
- log write, forced at commit 100
- Logging
  - utilities 105

- LOW2KEY column
  - SYSOLSTATS catalog table 156
- SYSOLSTATS catalog table 156
- SYSOLSTATS catalog table
  - access path selection 157
  - recommendation for updating 740
- SYSOLSTATS\_HIST catalog table 168
- SYSKEYTARGETS catalog table
  - data collected by RUNSTATS utility 159
- SYSKEYTARGETS\_HIST catalog table 169
- SYSKEYTARGETSTATS catalog table
  - data collected by RUNSTATS utility 159
- LOWKEY column
  - SYSKEYTARGETSTATS catalog table 159
- LOWKEY column of SYSOLSTATS catalog table 156
- LOWVALUE 165
- LOWVALUE column
  - SYSOLDIST catalog table
    - access path selection 154, 155
  - SYSOLDIST\_HIST catalog table 168
  - SYSKEYTGTDIST catalog table
    - access path selection 160, 161
  - SYSKEYTGTDIST\_HIST catalog table 170

## M

- mapping macro
  - DSNDIFCA 466
  - DSNDQWIW 645
  - DSNDWBUF 443
  - DSNDWQAL 448
- materialization
  - outer join 691
  - views and nested table expressions 716
- materialized query table 128
  - altering 129
  - creating 126
  - defining 133
  - design guidelines 133
  - enabling for automatic query rewrite 124
  - examples in automatic query rewrite 119, 134
  - introduction 116
  - maintaining 130
  - populating 130
  - refresh age 124
  - refreshing 130
  - statistics 132
  - system-maintained 126
  - use in automatic query rewrite 117
  - user-maintained 126
- MAX BATCH CONNECT field of panel DSNTIPE 151
- MAX REMOTE ACTIVE field of panel DSNTIPE 143
- MAX TSO CONNECT field of panel DSNTIPE 151
- MAXCSA option of START irlmproc command 347
- MAXROWS clause 91
- MEMBER CLUSTER option of CREATE TABLESPACE 324
- merge processing
  - views or nested table expressions 715
- MERGE statement 697
- message by identifier
  - DSNBB440I 602
  - DSNI103I 344
- mode of a lock 334
- monitor program
  - using IFI 439
  - using OMEGAMON performance monitor 422
- monitoring 415
  - application packages 601

- monitoring (*continued*)
  - application plans 601
  - CICS 426
  - DB2 426
  - IMS 426
  - server-elapsed time for remote requests 609
  - tools
    - DB2 trace 429
    - monitor trace 433
    - performance 419
  - using IFI 439
- multiple allegiance 84

## N

- NACTIVE column
  - SYSTABSTATS catalog table 163
- NACTIVEF column of SYSTABLESPACE catalog table
  - data collected by RUNSTATS utility 163
- NEARINDREF column
  - SYSTABLEPART catalog table 162
  - SYSTABLEPART\_HIST catalog table 170
- NEAROFFPOSF column
  - SYSINDEXPART catalog table
    - data collected by RUNSTATS utility 158
  - SYSINDEXPART\_HIST catalog table 169
- nested table expression
  - processing 714
- NLEAF column
  - SYSINDEXES catalog table
    - data collected by RUNSTATS utility 157
  - SYSINDEXES\_HIST catalog table 168
  - SYSINDEXSTATS catalog table 159
  - SYSINDEXSTATS\_HIST catalog table 169
- NLEVELS column
  - SYSINDEXES catalog table
    - data collected by RUNSTATS utility 157
  - SYSINDEXES\_HIST catalog table 168
  - SYSINDEXSTATS catalog table 159
  - SYSINDEXSTATS\_HIST catalog table 169
- non-DB2 utilities
  - effect on real-time statistics 228
- nonsegmented table space
  - dropping 105
  - scan 664
- NOT PADDED 96
- NPAGES column
  - SYSTABLES catalog table 163
  - SYSTABSTATS catalog table 163
  - SYSTABSTATS\_HIST catalog table 171
- NPAGESF column
  - SYSTABLES catalog table
    - data collected by RUNSTATS utility 163
  - SYSTABLES\_HIST catalog table 171
- NUMBER OF LOGS field of panel DSNTIPL 103
- NUMCOLUMNS column
  - SYSOLDIST catalog table
    - access path selection 154
  - SYSOLDIST\_HIST catalog table 168
  - SYSOLDISTSTATS catalog table 155
- NUMKEYS column
  - SYSKEYTGTDIST catalog table
    - access path selection 160
  - SYSKEYTGTDIST\_HIST catalog table 170

## O

- object of a lock 336
- objectives 13
- OMEGAMON 151, 485
  - accounting report
    - concurrency scenario 486
    - overview 637
  - description 419
  - statistics report
    - buffer pools 481
    - DB2 log 102
- OMEGAMONstatistics report
  - EDM pool 70
- online 755
- online books 755
- online monitor program using IFI 439
- OPEN
  - statement
    - performance 691
- optimistic concurrency control 325, 357
- optimization hints 311
- Optimization Service Center 421
- Optimization tools
  - tables used by 503, 577, 579, 580, 581, 582, 586
- OPTIMIZE FOR n ROWS clause 301
  - effect on distributed performance 409, 410
  - interaction with FETCH FIRST clause 300, 410
- ORDER BY clause
  - effect on OPTIMIZE clause 302
- ORGRATIO column
  - SYSLOBSTATS catalog table 161
  - SYSLOBSTATS\_HIST catalog table 170
- originating task 391
- OSC 421
- outer join
  - EXPLAIN report 691
  - materialization 691
- output area used in IFI
  - command request 443
  - description 471
  - example 439, 443
  - WRITE request 466

## P

- package
  - accounting trace 431
  - binding
    - EXPLAIN option for remote 570
  - monitoring 601
  - RLFPKG column of RLST 36
  - SKPT (skeleton package table) 70
- Package
  - copies 293, 295
- Package copies
  - current 295
  - original 295
  - previous 295
- page
  - buffer pool 56
  - locks
    - description 329
    - in OMEGAMON reports 490
- PAGE\_RANGE column of PLAN\_TABLE 682

- PAGESAVE column
  - SYSTABLEPART catalog table
    - data collected by RUNSTATS utility 162
    - updated by LOAD and REORG utilities for data compression 94
  - SYSTABLEPART\_HIST catalog table 170
- pair-wise join 702
- Parallel Access Volumes (PAV) 84
- parallel processing
  - description 391
  - disabling using resource limit facility 44
  - enabling 398
  - monitoring 601
  - related PLAN\_TABLE columns 683
  - tuning 743
- Parallel processing
  - parallel CP 391
  - parallel I/O 391
  - sequential 391
- parallelism
  - modes 44
- partition
  - compressing data 91
  - partition scan, limited 682
- partitioned table space
  - locking 332
- PAV (Parallel Access Volumes) 84
- PC option of START irlmproc command 347
- PCLOSEN subsystem parameter 82
- PCLOSET subsystem parameter 82
- PCTPAGES column
  - SYSTABLES catalog table 163
  - SYSTABLES\_HIST catalog table 171
  - SYSTABSTATS catalog table 163
- PCTROWCOMP column
  - SYSTABLES catalog table 94
    - data collected by RUNSTATS utility 163
  - SYSTABLES\_HIST catalog table 171
  - SYSTABSTATS catalog table 94, 163
  - updated by LOAD and REORG for data compression 94
- PERCACTIVE column
  - SYSTABLEPART catalog table
    - data collected by RUNSTATS utility 162
  - SYSTABLEPART\_HIST catalog table 171
- PERCDROP column
  - SYSTABLEPART catalog table
    - data collected by RUNSTATS utility 162
  - SYSTABLEPART\_HIST catalog table 171
- performance 53, 96
  - affected by
    - data set distribution 19
    - extents 22, 88
    - PRIQTY clause 22, 88
  - goals 5
  - monitoring
    - tools 419
    - trace 433
    - using OMEGAMON performance monitor 422
    - with EXPLAIN 421
  - planning 1
    - objectives 5
  - programming applications 235
  - storage devices 83
  - system design 17
- Performance
  - objectives
    - validating 13
- Performance (*continued*)
  - overview of managing 3
  - process for managing 3
  - reviewing
    - questions 11
  - storage servers 83
- performance considerations
  - DPSI 290
  - scrollable cursor 289
- performance implications
  - processor resource 47
  - processor resources 47
- performance planning
  - design
    - internal 8
- Performance planning 8, 13
  - design
    - external 8
- Performance regression
  - REBIND 293
- Performance Reporter for z/OS 423
- Performance trace 47
- PIECESIZE clause
  - ALTER INDEX statement
    - recommendations 20
    - relation to PRIQTY 22, 88
  - CREATE INDEX statement
    - recommendations 20
    - relation to PRIQTY 22, 88
- PLAN\_TABLE
  - column descriptions 649
  - Creating 559
  - Format 559
- PLAN\_TABLE table
  - report of outer join 691
- PLANMGMT option
  - BASIC 293
  - EXTENDED 293
  - OFF 293
- planning
  - performance 1
- Planning
  - workload 6
- pool, inactive connections 144
- PQTY column
  - SYSINDEXPART catalog table
    - data collected by RUNSTATS utility 158
  - SYSINDEXPART\_HIST catalog table 169
  - SYSTABLEPART catalog table
    - data collected by RUNSTATS utility 162
  - SYSTABLEPART\_HIST catalog table 171
- predicate
  - description 245
  - filter factor 260
  - generation 272
  - impact on access paths 245
  - indexable 248
  - join 247
  - local 247
  - modification 272
  - properties 247
  - stage 1 (sargable) 249
  - stage 2
    - evaluated 249
    - influencing creation 305
  - subquery 247

- PREFORMAT
  - option of LOAD utility 21, 87
  - option of REORG TABLESPACE utility 21, 87
- preformatting space for data sets 21, 87
- PRIMARY\_ACCESTYPE column of PLAN\_TABLE 679
- prioritizing resources 26
- problem determination
  - using OMEGAMON performance monitor 422
- processing speed
  - processor resources consumed
    - accounting trace 429, 638
    - buffer pool 65
    - thread reuse 45
    - traces 46
    - transaction manager 428
  - RMF reports 426
  - time needed to perform I/O operations 18
- Processor resource
  - performance trace 47
- processor resources 45
  - audit trace 47
- Processor resources
  - accounting trace 46
  - global trace 46
  - statistics trace 47
- PROCLIM option of IMS TRANSACTION macro 150
- product-sensitive programming information, described 765
- Profile tables 577
  - DSN\_OBJECT\_RUNTIME\_INFO 586
  - DSN\_PROFILE\_ATTRIBUTES 581
  - DSN\_PROFILE\_ATTRIBUTES\_HISTORY 581
  - DSN\_PROFILE\_HISTORY 580
  - DSN\_PROFILE\_TABLE 579
  - DSN\_STATEMENT\_RUNTIME\_INFO 582
  - DSN\_VIRTUAL\_INDEXES 577
- programming applications
  - performance 235
- programming interface information, described 765
- PSEUDO\_DELETED\_ENTRIES column
  - SYSINDEXPART catalog table 158
  - SYSINDEXPART\_HIST catalog table 169
- PSPI symbols 765

## Q

- QMF (Query Management Facility)
  - options 152
  - performance 152
- qualification area used in IFI
  - description of fields 448
  - READS request 448
  - restricted IFCIDs 448
  - restrictions 458
- QUANTILENO 165
- QUANTILENO column
  - SYSOLDIST\_HIST catalog table 168
  - SYSKEYTGTDIST catalog table
    - access path selection 160
  - SYSKEYTGTDIST\_HIST catalog table 170
  - SYSKEYTGTDISTWSTATS catalog table
    - access path selection 154, 155, 161
- Query Management Facility (QMF) 141
- query parallelism 391
- QUERYNO clause
  - reasons to use 315

## R

- READA (read asynchronously) 464
- reading
  - dynamic prefetch 57
  - normal read 57
  - sequential prefetch 57
- READS (read synchronously) 447
- real storage 55
- REAL TIME STATS
  - field of panel DSNTIPO 175
- real-time statistics
  - accuracy 231
  - for DEFINE NO objects 228
  - for read-only objects 229
  - for TEMP table spaces 228
  - for work file table spaces 228
  - improving concurrency 231
  - in data sharing 230
  - when DB2 externalizes 220
- Real-time statistics
  - clone tables 231
  - EXCHANGE command 231
- real-time statistics tables
  - contents 175
  - effect of dropping objects 229
  - effect of mass delete operations 230
  - effect of SQL operations 229
  - effect of updating partitioning keys 229
  - recovering 231
  - setting up 175
  - setting update interval 175
  - starting 175
- reason code
  - X'00C90088' 321
  - X'00C9008E' 320
- REBIND
  - Performance regression 293
- REBIND PACKAGE subcommand of DSN
  - options
    - RELEASE 353
- REBIND PLAN subcommand of DSN
  - options
    - ACQUIRE 353
    - RELEASE 353
- REBUILD INDEX utility
  - effect on real-time statistics 225
- recovery
  - IFI calls 474
  - real-time statistics tables 231
- reducing 45
- refresh age 124
- REFRESH TABLE statement 130
- registering a base table as 128
- RELCURHL subsystem parameter
  - recommendation 376
- RELEASE
  - option of BIND PLAN subcommand
    - combining with other options 353
- RELEASE LOCKS field of panel DSNTIP4
  - recommendation 376
- reoptimizing access path 279
- REORG utility
  - effect on real-time statistics 223
- Reserving
  - space 89
- resource allocation 139
  - performance factors 139

- resource limit facility (governor)
  - calculating service units 42
  - description 28
  - distributed environment 28
  - governing by plan or package 31, 32
  - stopping and starting 40
- Resource Measurement Facility (RMF) 419, 426
- resource objectives 25
- RESOURCE TIMEOUT field of panel DSNTIPI 348
- resources
  - job 26
  - limiting 26
- response time 424
- RETAINED LOCK TIMEOUT field of installation panel DSNTIPI 349
- RETLWAIT subsystem parameter 349
- RID (record identifier) pool
  - size 75
  - storage
    - allocation 75
    - estimation 75
- RLFASUERR column of RLST 36
- RLFASUWARN column of RLST 36
- RLMT (resource limit table)
  - columns 37
- RLST (resource limit specification table)
  - columns 34
  - precedence of entries 28
  - RLMT (resource limit table for middleware) 28
- RMF (Resource Measurement Facility) 419, 426
- RO SWITCH CHKPTS field of installation panel DSNTIPL 82
- RO SWITCH TIME field of installation panel DSNTIPL 82
- rollback
  - effect on performance 101
- ROW CHANGE TIMESTAMP 325
- row change token 325
- ROWID
  - coding example 681
- RR (repeatable read)
  - claim class 386
  - drain lock 387
  - how locks are held (figure) 361
  - page and row locking 361
- RRSAF (Recoverable Resource Manager Services attachment facility)
  - transactions
    - using global transactions 325
- RS (read stability)
  - claim class 386
  - page and row locking (figure) 360
- RUNSTATS utility
  - aggregate statistics 499
  - effect on real-time statistics 226
  - timestamp 741
  - use
    - tuning DB2 17
    - tuning queries 499

## S

- scope of a lock 329
- SCOPE option
  - START irlmproc command 347
- scrollable cursor
  - block fetching 406
  - optimistic concurrency control 357
  - performance considerations 289

- SCT02 table space
  - placement of data sets 85
- SECQTY1 column
  - SYSINDEXPART\_HIST catalog table 169
- SECQTYI column
  - SYSINDEXPART catalog table 158
  - SYSTABLEPART catalog table 162
  - SYSTABLEPART\_HIST catalog table 171
- segmented table space
  - locking 332
  - scan 664
- SEGSIZE clause of CREATE TABLESPACE
  - recommendations 664
- sequences
  - improving concurrency 325
- Sequential
  - Parallel CP
    - processing 391
  - Parallel I/O
    - processing 391
  - processing 391
- sequential detection 687
- sequential prefetch
  - bind time 686
  - description 686
- sequential prefetch threshold (SPTH) 59
- service level agreements 5
- SET CURRENT DEGREE statement 398
- SHARE
  - INTENT EXCLUSIVE lock mode 334, 379
  - lock mode
    - LOB 379
    - page 334
    - row 334
    - table, partition, and table space 334
    - XML 383
- shortcut keys
  - keyboard xiv
- simple table space
  - locking 332
- SKCT (skeleton cursor table)
  - EDM pool 70
  - EDM pool efficiency 70
  - locks on 339
- skeleton cursor table (SKCT) 70
- SKIP LOCKED DATA 370
  - examples 371
  - isolation levels 370
  - lock mode compatibility 371
  - lock sizes 371
- Skipping locked pages 370
- Skipping locked rows 370
- SKPT (skeleton package table)
  - EDM pool 70
  - locks on 339
- SMF (System Management Facility)
  - buffers 435
  - record types 431
  - trace record
    - accounting 431
    - format 621
    - lost records 435
    - recording 435
    - statistics 431
- softcopy publications 755
- sort
  - description 76

- sort (*continued*)
  - pool 76
  - program
    - RIDs (record identifiers) 691
    - when performed 691
  - shown in PLAN\_TABLE 689
- Sort performance 84
- SORT POOL SIZE field of panel DSNTIPC 76
- SPACE column
  - SYSTABLEPART catalog table 162
- SPACE column of SYSTABLESPACE catalog table
  - data collected by RUNSTATS utility 163
- SPACEF column
  - SYSINDEXES catalog table
    - data collected by RUNSTATS utility 157
  - SYSINDEXPART catalog table 158
  - SYSINDEXPART\_HIST catalog table 169
  - SYSTABLEPART catalog table 162
  - SYSTABLEPART\_HIST catalog table 171
  - SYSTABLES catalog table 163
- SPACEF column of SYSTABLESPACE catalog table
  - data collected by RUNSTATS utility 163
- SPTH (sequential prefetch threshold) 59
- SPUFI
  - resource limit facility (governor) 31
- SQL (Structured Query Language)
  - performance trace 139
  - statement cost 139
  - statements 139
    - performance factors 139
- SQL statements
  - DECLARE CURSOR
    - to ensure block fetching 407
  - EXPLAIN
    - monitor access paths 421
  - RELEASE 404
  - SET CURRENT DEGREE 398
- SQLCA (SQL communication area)
  - reason code for deadlock 321
  - reason code for timeout 320
- SQLCODE
  - 510 364
  - 905 28
- SQLSTATE
  - '57014' 28
- SQTY column
  - SYSINDEXPART catalog table 158
  - SYSTABLEPART catalog table 162
- SSM (subsystem member)
  - thread reuse 150
- star schema
  - data caching 711
  - defining indexes for 306
- Star schema
  - 'P' 709
  - access 707, 709
  - JOIN\_TYPE 707, 709
- star-join queries
  - processing
    - pair-wise join 307
    - star join 307
- state
  - of a lock 334
- Statement 366
- Statement cache table
  - creating 563

- Statement table
  - creating 562
- statistics
  - aggregate 499
  - created temporary tables 166
  - distribution 741
  - filter factor 164
  - history catalog tables 167, 500
  - materialized query tables 132
  - partitioned table spaces 166
  - trace
    - class 4 605
    - description 431
- statistics report
  - thread queuing 151
- Statistics trace
  - performance implications
    - processor resources 47
- STATS\_FORMAT column
  - SYSKEYTARGETS catalog table
    - data collected by RUNSTATS utility 159
  - SYSKEYTARGETS\_HIST catalog table 169
  - SYSKEYTARGETSTATS catalog table
    - data collected by RUNSTATS utility 159
- STATSTIME column
  - use by RUNSTATS 153
- STDDEV function
  - when evaluation occurs 665
- STOP DATABASE command
  - timeout 320
- storage
  - calculating
    - locks 348
  - controller cache 83
  - hierarchy 53
  - IFI requirements
    - READA 464
    - READS 447
  - real 55
- Storage 83
- storage controller 53
- storage controller cache 83
- Storage devices
  - performance 83
- Storage servers
  - advanced features 84
  - DB2 performance 83
  - Performance
    - Storage servers 84
- stored procedure
  - DSNACCOR 199
  - DSNACCOX 176
  - DSNAEXP 567
  - limiting resources 27
  - running concurrently 109
- subquery
  - join transformation 285
  - tuning examples 288
- subsystem member (SSM) 150
- synchronous data from IFI 459
- synchronous write
  - analyzing accounting report 638
  - data management threshold (DMTH) 59
  - DMTH (data management threshold) 59
  - immediate 59, 481
- Sysplex query parallelism
  - disabling Sysplex query parallelism 745

- Sysplex query parallelism (*continued*)
  - disabling using buffer pool threshold 61
  - processing across a data sharing group 395
  - splitting large queries across DB2 members 391
- SYSTABLESPACESTATS
  - contents 175
- system design
  - performance 17
- System Management Facility (SMF). 435
- System monitor 45
- system monitoring
  - monitoring tools
    - DB2 trace 429
- System z9 Integrated Information Processor 477

## T

- table
  - expression, nested
    - processing 714
  - locks 329
- table expressions, nested
  - materialization 716
- table space
  - compressing data 91
  - locks
    - control structures 139
    - description 329
  - scans
    - access path 664
- tables 503, 649
- Tables 649
- TCP/IP
  - keep\_alive interval 146
- temporary table
  - monitoring 484
- temporary work file 77
- testing 8
- Testing
  - performance planning 8
- thread
  - creation
    - connections 151
    - description 138
    - IMS 150
    - performance factors 138
  - distributed
    - active 146
    - inactive vs. active 144
    - maximum number 143
    - pooling of inactive threads 144
- MAX REMOTE CONNECTED field of panel
  - DSNTIPE 143
- queuing 151
- reuse
  - description 138
  - effect on processor resources 45
  - IMS 150
  - remote connections 147
  - TSO 141
- steps in creation and termination 138
- termination
  - description 140
  - IMS 150
- time out for idle distributed threads 146
- timeout
  - changing multiplier
    - DL/I BATCH TIMEOUT field of installation panel
      - DSNTIPI 349
    - IMS BMP and DL/I batch 349
    - IMS BMP TIMEOUT field of panel DSNTIPI 349
    - utilities 351
  - description 320
  - idle thread 146
  - indications in IMS 321
  - multiplier values 349
  - row vs. page locks 372
  - X'00C9008E' reason code in SQLCA 320
- Tivoli Decision Support for z/OS 423
- TOKENE option of DSNCRCT macro 620
- trace
  - accounting 431
  - audit 433
  - description 419, 429
  - distributed data 605
  - effect on processor resources 46
  - interpreting output 621
  - monitor 433
  - performance 433
  - recommendation 605
  - record descriptions 621
  - record processing 621
  - statistics
    - description 431
- transaction
  - IMS
    - using global transactions 325
- transaction lock
  - description 319
- TSO
  - connections
    - tuning 151
  - DSNELI language interface module
    - IFI 442
  - foreground 141
  - resource limit facility (governor) 27
  - running SQL 141
- tuning
  - DB2
    - active log size 104
    - catalog location 86
    - catalog size 86
    - disk space 85
    - virtual storage utilization 53
- TYPE column
  - SYSCOLDIST catalog table
    - access path selection 154
  - SYSCOLDIST\_HIST catalog table 168
  - SYSCOLDISTSTATS catalog table 155
  - SYSKEYTGTDIST catalog table
    - access path selection 160
  - SYSKEYTGTDIST\_HIST catalog table 170

## U

- UNION clause
  - effect on OPTIMIZE clause 302
- UPDATE
  - lock mode
    - page 334
    - row 334
  - table, partition, and table space 334

- update efficiency 481
- UR (uncommitted read)
  - claim class 386
  - effect on reading LOBs 378
  - effect on reading XML data 382
  - recommendation 325
- USE AND KEEP EXCLUSIVE LOCKS option of WITH clause 365
- USE AND KEEP SHARE LOCKS option of WITH clause 365
- USE AND KEEP UPDATE LOCKS option of WITH clause 365
- user-defined function
  - providing access cost 729
- user-defined table function
  - improving query performance 304
- Using 366
- utilities
  - compatibility 388
  - concurrency 319, 385
  - effect on real-time statistics 221
  - timeout multiplier 351
  - types
    - RUNSTATS 499
- Utilities
  - logging 105
  - Performance 105
- UTILITY TIMEOUT field of panel DSNTIPI 351
- UTSERIAL lock 387

## V

- VARIANCE function
  - when evaluation occurs 665
- VDWQT option of ALTER BUFFERPOOL command 61
- vertical deferred write threshold (VDWQT) 61
- view
  - EXPLAIN 719, 720
  - processing
    - view merge 714
- virtual buffer pool assisting parallel sequential threshold (VPXPSEQT) 61
- virtual buffer pool parallel sequential threshold (VPPSEQT) 61
- virtual buffer pool sequential steal threshold (VPSEQT) 61
- virtual storage
  - buffer pools 53
  - improving utilization 53
  - IRLM 53
- Visual Explain 172, 299
- volatile table 303
- VPPSEQT option of ALTER BUFFERPOOL command 61
- VPSEQT option of ALTER BUFFERPOOL command 61
- VPXPSEQT option of ALTER BUFFERPOOL command 61

## W

- WBUFxxx field of buffer information area 443
- WITH clause
  - isolation level 365
  - specifies isolation level 365
- WLM
  - I/O request 50
- work file
  - monitoring 484
  - table space
    - minimize I/O contention 20

- work file (*continued*)
  - used by sort 77
- work file database
  - minimizing I/O contention 20
- Workload
  - defining 6
- WQAxix fields of qualification area 448
- write claim class 386
- write drain lock 387
- write efficiency 481

## X

- XLKUPDLT subsystem parameter 376
- XML
  - lock
    - description 381
    - locks 381
- XML (extensible markup language)
  - lock duration 383
  - LOCK TABLE statement 384
  - locking 381
  - LOCKSIZE clause of CREATE or ALTER TABLESPACE 385
- XML data
  - best practices 237
  - performance 237
- XML locks
  - controlling 381

## Z

- z/OS
  - performance options
    - WLM 49
  - WLM 49
  - workload management 49
- zIIP 477





Program Number: 5635-DB2

Printed in USA

SC18-9851-03



Spine information:

DB2 Version 9.1 for z/OS

Performance Monitoring and Tuning Guide

