WebSphere® Portal content publishing (WPCP) is a component of IBM WebSphere Portal for Multiplatforms. WPCP is a Web content management solution which provides an interface to create, contribute and manage portal content. WPCP itself consists of separate components, each of which supports several different underlying database applications. Supporting multiple database applications posed challenges to the WPCP development team. Variations in Structured Query Language (SQL), data type restrictions, as well as other underlying database characteristics affected the design and development of WPCP. These differences also pose challenges to users of WPCP who make use of multiple database applications. This paper presents those differences and explores how they may affect users of WPCP version 5.0.

Introduction

Figure 1: Database applications supported by WPCP component

Figure 1 depicts the basic component structure of WPCP. Its two main components are its author-time server and its run-time server. The author-time server consists of the tools and work area for configuring, modifying, and previewing a Web application prior to publishing it to a run-time environment. The WPCP author-time component supports the following database applications:
The run-time environment is the production server with which the Web site visitor interfaces. It contains the engines required to deliver Web content, to personalize the Web content, and to record site metrics. The WPCP run-time component supports the following database applications:

- IBM DB2 Universal Database for Linux, Unix, and Windows (DB2)
- Oracle
- Informix Dynamic Server (Informix)
- IBM Cloudscape (Cloudscape)
- Microsoft SQL Server (SQL Server)
- IBM DB2 Universal Database DB2 for OS/390 (DB2 390)

Note: WPCP run-time server can be installed as a stand-alone component of WebSphere Portal Server. This is depicted by the dotted line box in Figure 1.

Figure 1 depicts the databases supported by WPCP. The WPCP author-time component uses a database (wpcp50) to manage authoring and configuration information as well as documents. The WPCP run-time component uses one database (wpcp50) to manage campaign and personalization information, and uses a separate database (fdbk50) to collect Web site traffic information for generating reports of site activity, including information about campaigns and personalized resources. This database is referred to as the Feedback database.

Note: wpcp50 and fdbk50 are default names used during the WebSphere Portal installation.

**WPCP Installation and Configuration for Databases**

You will most likely encounter the effects of multiple database applications during the installation and configuration of WebSphere Portal. This section highlights a number of these differences. The WebSphere Portal InfoCenter documentation contains detailed instructions on configuring WebSphere Portal (including WPCP) to each of the supported database applications.

WPCP is fully configured during the installation process to use Cloudscape by default. To transfer to another database, such as DB2, the database transfer process must be executed. Additionally, WebSphere Portal allows installation and configuration of the WPCP run-time server on a separate machine from the author-time server. The databases used by the run-time and author-time servers may run on different database applications.
The following information in this section pertains to all the supported databases except Cloudscape.

Once the database application has been installed, the necessary WPCP databases and users with the proper privileges must be created prior to performing the database transformation steps. For DB2, Informix, or DB2 390, one database user with administrative rights is needed to configure WPCP properly. If using Oracle or SQL Server, four database users are required. Table 2 lists the recommended users to simplify database administration.

<table>
<thead>
<tr>
<th>Database name</th>
<th>User name</th>
</tr>
</thead>
<tbody>
<tr>
<td>wpcp50</td>
<td>EJB, PZNADMIN, WCMDBADM</td>
</tr>
<tr>
<td>fdbk50</td>
<td>FEEDBACK</td>
</tr>
</tbody>
</table>

Table 2: WPCP recommended databases and user names

If using Oracle, the explicit user names EJB, FEEDBACK and PZNADMIN are required. If using SQL Server, the explicit user names EJB and FEEDBACK are required. Under both Oracle and SQL Server, the specific user name WCMDBADM is not mandatory.

The next section drills down further into the differences between database definitions by comparing, highlighting and showing examples of the main differences between the WPCP DB2 table definitions and the other supported WPCP databases: Oracle, Informix, Cloudscape, SQL Server and DB2 390.

**WPCP Underlying Database Differences**

As mentioned before, variations in the SQL from one database application to another, different data types supported among database applications, as well as other underlying database differences, such as database schema semantics, all affect WPCP. Using DB2 as the baseline database application, let’s start by looking at the WPCP underlying database schema definition.

**Database Schemas and Views**

A database schema can be defined as a collection of database objects that are owned by a single user and form a unique namespace to provide a logical classification of database objects. A namespace is a collection of entities such as tables, views, indexes and triggers that cannot have duplicate names. The concept of a database schema is shared among most database applications. However, the implementation may differ slightly between applications.
Typical usage of a database schema is to organize the structure of a particular database. Appropriate authorization is required for a user or group of users to use objects within a schema. A user or group is permitted to use the objects of a schema when granted access by the database administrator or owner. In Informix and SQL Server, a schema name must coincide with a valid user name. The other supported database applications allow any schema name within the respective naming guidelines. After obtaining a connection to a database, the current user name is used as the schema name if the schema name was not set explicitly. WPCP uses schema names to provide more structured maintenance of WPCP database objects and to avoid object naming collisions.

A database uses the view construct to represent data from any number of tables as a single logical table. It is not a defined table but a logical table that facilitates access to the underlying data. WPCP uses a view to join columns from multiple tables.

WPCP creates the view **WPCP_TAB_COL_VIEW** to facilitate retrieving table column names. The following is the **CREATE VIEW** statement for SQL Server:

```
CREATE VIEW WCMDBADM.WPCP_TAB_COL_VIEW AS
SELECT
    SOBJ.NAME AS TABLE_NAME, SCOL.NAME AS COLUMN_NAME,
    USER_NAME(SOBJ.UID) AS SCHEMA_NAME
FROM
    SYSOBJECTS SOBJ, SYSCOLUMNS SCOL
WHERE
    SOBJ.XTYPE='U'
    AND SOBJ.NAME != 'DTPROPERTIES'
    AND SOBJ.ID=SCOL.ID
    AND USER_NAME(SOBJ.UID) = 'WCMDBADM';
```

Author-time WPCP processing uses this view in its internal processing only. A similar view is created for Informix, Cloudscape and DB2 390.

**Oracle**

A database table is the basic unit of storage in a relational database management system. It consists of one or more units of information (rows), each of which contains the same types of values (columns). Each column is given a column name and a data type. The SQL used to define the WPCP tables has syntactical differences across database applications. In some instances, WPCP uses different data types when defining the columns in its tables. The following table lists some of these differences:

<table>
<thead>
<tr>
<th>DB2 Data type</th>
<th>Oracle Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARACTER(n)</td>
<td>CHAR(n)</td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>VARCHAR(n)</td>
</tr>
<tr>
<td>LONG VARCHAR(n)</td>
<td>LONG</td>
</tr>
<tr>
<td>DECIMAL(p,s)</td>
<td>NUMBER(p,s)</td>
</tr>
<tr>
<td>INTEGER</td>
<td>NUMBER</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>
Table 3: Oracle and DB2 corresponding data types

Table 3 contains corresponding data types between DB2 and Oracle. If using the same SQL when defining a table with any of the DB2 data types listed in Table 3, Oracle will implicitly convert the data type to its equivalent data type upon execution. The following points discuss some other data type variations between DB2 and Oracle.

- As shown in the next example, a column of BIGINT data type in DB2 is not supported in Oracle but is accepted if defined as a NUMBER in Oracle.

- The following DB2 data types GRAPHIC, LONG VARGRAPHIC, VARGRAPHIC, TIME, and TIMESTAMP do not have corresponding Oracle data types. However, the data of type TIME and TIMESTAMP can also be expressed as Oracle DATE data if formatted correctly.

  Note: The Oracle DATE data type supports timestamp values down to the second.

- A couple DB2 data types require syntactical changes to become valid in Oracle, namely, Binary Large Object (BLOB) and Character Large Object (CLOB). When defining a column of the BLOB or CLOB data type in DB2, the maximum length in bytes must be specified in parentheses (up to 2 gigabytes). Oracle BLOB and CLOB data types require an initialization prior to populating the column with data. The function name is EMPTY_BLOB and EMPTY_CLOB, respectively; see example below. The maximum BLOB or CLOB size in Oracle is 4 gigabytes.

  DB2:

  ```sql
  CREATE TABLE CMPROCESS
  ( ID           VARCHAR(100) NOT NULL,
    NAME         VARCHAR(32),
    STATE        INTEGER,
    ORIGINATOR   VARCHAR(32),
    WORKFLOW     VARCHAR(255),
    SUBJECT      VARCHAR(255),
    STARTDATE    BIGINT,
    ATTRIBUTES   BLOB(8M),
    PRIMARY KEY (ID)
  );
  ```

  Oracle:

  ```sql
  CREATE TABLE CMPROCESS
  ( ID           VARCHAR(100) NOT NULL,
    NAME         VARCHAR(32),
    STATE        INTEGER,
    ORIGINATOR   VARCHAR(32),
    WORKFLOW     VARCHAR(255),
    SUBJECT      VARCHAR(255),
    STARTDATE    NUMBER,
    ATTRIBUTES   BLOB DEFAULT EMPTY_BLOB(),
    PRIMARY KEY (ID)
  );
  ```
Additionally, the Oracle **RAW** data type is intended for binary data or byte strings of a smaller scale (maximum size is 2 kilobytes) and, most importantly, does not inherit all of the large object (LOB) restrictions.

Another variation between the two database applications occurs during automatic generation of default column values. To ensure distinctiveness, WPCP often requires unique numeric column values in specific tables. In DB2, this is accomplished by appending the following SQL to the end of a numeric column definition `UNIQUE GENERATED BY DEFAULT AS IDENTITY(START WITH 1, INCREMENT BY 1, CACHE 20)`. An identity column is implicitly **NOT NULL**. Therefore, DB2 will generate a value for the column when a row is inserted into the table, unless a unique value is specified. The end of the statement `CACHE 20` pre-allocates values in memory for faster access allowing a maximum loss of 20 values in case of database deactivation.

To reproduce this in Oracle, the **NEXTVAL** function of a **SEQUENCE** is needed for the desired column. A **sequence** is a schema object that can generate unique sequential values such as primary key values. **NEXTVAL** will increment the sequence and return the next value. The SQL statement:

```
CREATE SEQUENCE FEEDBACK.KEY_ID START WITH 1 INCREMENT BY 1 CACHE 20;
```

creates a sequence. Upon insertion of a row into the table, **NEXTVAL** is qualified by the schema and sequence names (for this example, `FEEDBACK.KEY_ID.NEXTVAL`, where `FEEDBACK` is the schema and `KEY_ID` is the sequence name) to perform this operation.

**Informix**

The WPCP table definition SQL syntax for DB2 and Informix are nearly identical with the exception of a few supported data type differences. The list below describes the differences found in the WPCP schema.

- The **BIGINT** data type in DB2 is equivalent to the **INT8** data type in Informix.

- Columns defined with the **TIMESTAMP** (`ymmdhhmmssxxx`) in DB2 are replaced with the **DATETIME YEAR TO FRACTION** (`yyyy-mm-dd hh:mm:ss:xxx`) in Informix, where **YEAR** is the largest qualifier and **FRACTION** is the smallest qualifier. The table definitions below will show the syntactical differences between the two database applications.

**DB2:**

```sql
CREATE TABLE CTACAMPAIGNS
( WCPCPNNAME VARCHAR(255) NOT NULL ,
  WCPPRIORITY INT NOT NULL ,
  WCPCPNSPLIT INT NOT NULL,
  WCPCPNSTART TIMESTAMP NOT NULL ,
  WCPCPNSTOP  TIMESTAMP NOT NULL ,
);
```
The \textit{FRACTION} represents a decimal fraction-of-a-second with up to 5 digits of scale. The default scale is 3 digits.

- Informix automatically generates a sequential integer for a column value when a row is inserted into a table if the column is defined as a \texttt{SERIAL} (range from 1 to 2,147,483,647) or \texttt{SERIAL8} (range from 1 to 9,223,372,036,854,775,807) data type. A unique index or primary key constraint must be applied to this column to prevent duplicate numbers. An example is provided below:

\begin{verbatim}
DB2:
CREATE TABLE FEEDBACK.DOMAIN
  ( ID     BIGINT NOT NULL UNIQUE GENERATED BY DEFAULT AS IDENTITY(START WITH 1, INCREMENT BY 1, CACHE 20),
   NAME   VARCHAR(50) NOT NULL,
   PRIMARY KEY (ID)
  );

Informix:
CREATE TABLE FEEDBACK.DOMAIN
  ( ID     SERIAL8(1) NOT NULL PRIMARY KEY,
   NAME   CHARACTER VARYING(50) NOT NULL
  );
\end{verbatim}

Other syntactical variations that exist between DB2 and Informix pertain to the \texttt{BLOB}, \texttt{CLOB}, \texttt{VARCHAR}, and \texttt{INTEGER} data types. In contrast to DB2, the Informix \texttt{BLOB} and \texttt{CLOB} data types do not require the specification of a maximum size. The maximum length of the \texttt{BLOB} or \texttt{CLOB} data type in Informix is 4 terabytes. \texttt{INT} and \texttt{INTEGER} are
synonymous in Informix. The same is true for `VARCHAR(m)` and `CHARACTER VARYING(m)`.

**DB2:**

```
CREATE TABLE FILERESOURCE
    ( WCPURI VARCHAR(255) NOT NULL,
      PATH VARCHAR(255) NOT NULL,
      WCPMIMETYPE VARCHAR(255),
      WCPTEMPLATE CHAR(1),
      PROJECTID VARCHAR(32) NOT NULL,
      WORKSPACE VARCHAR(100) NOT NULL,
      PUBLISHABLE CHAR(1),
      CACHEABLE CHAR(1),
      WCPCONTENT BLOB(100M),
      PRIMARY KEY (PROJECTID, WORKSPACE, PATH, WCPURI) )
```

**Informix:**

```
CREATE TABLE FILERESOURCE
    ( WCPURI VARCHAR(125) NOT NULL,
      PATH VARCHAR(125) NOT NULL,
      WCPMIMETYPE VARCHAR(255),
      WCPTEMPLATE CHAR(1),
      PROJECTID VARCHAR(32) NOT NULL,
      WORKSPACE VARCHAR(100) NOT NULL,
      PUBLISHABLE CHAR(1),
      CACHEABLE CHAR(1),
      WCPCONTENT BLOB,
      PRIMARY KEY (PROJECTID, WORKSPACE, PATH, WCPURI) )
LOCK MODE ROW;
```

Notice the last line of the Informix table definition. Database locking is intended to prevent multiple users from simultaneously updating the same data. Informix uses table-level locking by default. WPCP uses row-level locking to facilitate concurrent user access.

When designing database tables, keep in mind that Informix Dynamic Server has different maximum sizes for
- The `LONG VARCHAR (LVARCHAR)` data type, which is restricted to 2 kilobytes
- The total number of bytes in a primary key (the sum of the column widths) may not exceed 390 bytes

**Cloudscape**

Similar to Informix, the WPCP table definitions for Cloudscape resemble the DB2 SQL. In the WPCP schema, there are some integer column values which only require 1 byte. In DB2, the integer data type with the smallest byte count is `SMALLINT`, which provides 2 bytes of storage. Cloudscape does support the `SMALLINT` data type however the `TINYINT` data type in Cloudscape provides only 1 byte of storage. Therefore, the applicable WPCP columns are created using the `TINYINT` data type for performance reasons. For example `FEEDBACK.LOGCONTROL` is defined as follows:
LOB support in Cloudscape also varies from DB2. The **LONG VARBINARY** data type is used to store bit strings of unlimited length in Cloudscape as opposed to the **BLOB** data type in DB2. The maximum size of the **LONG VARBINARY** value is not required when creating columns of this type. In DB2, the **CLOB** data type is used to store a character large object string. **LONG VARCHAR** is the corresponding data type in Cloudscape.

Columns defined with automatically incremented default values prohibit the insertion or modification of that column value in Cloudscape. The **BY DEFAULT** clause in DB2 indicates that DB2 will generate a value for the column when a row is inserted into the table, unless a value is specified. The following table definitions exemplify the contrast in syntax between the two database applications when creating such a column.

**DB2:**
```
CREATE TABLE FEEDBACK.BROWSERS  
(   ID    BIGINT NOT NULL GENERATED BY DEFAULT AS IDENTITY(START WITH 1, INCREMENT BY 1, CACHE 20),
   NAME  VARCHAR(250) NOT NULL,
   PRIMARY KEY (ID)
);
```

**Cloudscape:**
```
CREATE TABLE FEEDBACK.BROWSERS  
(   ID    BIGINT NOT NULL DEFAULT AUTOINCREMENT INITIAL 1 INCREMENT 1,
   NAME  VARCHAR(250) NOT NULL,
   PRIMARY KEY (ID)
);
```

**SQL Server**

Similar to Cloudscape, SQL Server includes the **TINYINT** data type. In some cases WPCP replaces **SMALLINT** or **INTEGER** with the **TINYINT** data type. Again as in Cloudscape, the **TINYINT** data type occupies 1 byte of storage. Valid values range between 0 and 255.

Another WPCP data type difference between SQL Server and DB2 is that WPCP uses the **NVARCHAR** type for unicode character data. This data type is used throughout the author-
time and run-time databases. Any character data that could be unicode has the NVARCHAR type. For example, the NAME field in FEEDBACK.KEY:

**DB2:**

```sql
CREATE TABLE FEEDBACK.KEY
    ( ID     BIGINT NOT NULL UNIQUE GENERATED BY DEFAULT AS IDENTITY(START WITH 1, INCREMENT BY 1, CACHE 20),
     NAME   VARCHAR(250) NOT NULL,
     PRIMARY KEY (ID)
    );
```

**SQL Server:**

```sql
CREATE TABLE FEEDBACK.[KEY]
    ( [ID]   BIGINT IDENTITY (1, 1) NOT NULL,
     [NAME] NVARCHAR(250) NOT NULL
    );
```

Also, the syntax for defining identity columns differs between SQL Server and DB2. In the above example, `IDENTITY (1, 1)` tells the database to generate the values for that column (the ID column of FEEDBACK.KEY here) starting at 1 and incrementing by 1 each operation.

**Note:** SQL Server does not maintain the concept of caching the identity column values like DB2.

SQL Server does not support BLOB or CLOB data types. As shown in Table 4, WPCP used the following similar data types for SQL Server:

<table>
<thead>
<tr>
<th><strong>DB2 LOB data types in WPCP</strong></th>
<th><strong>SQL Server equivalent</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOB (8k to 2Gb)</td>
<td>IMAGE</td>
</tr>
<tr>
<td>BLOB (&lt; 8k)</td>
<td>VARBINARY(n)</td>
</tr>
<tr>
<td>CLOB (n) for ascii data</td>
<td>TEXT</td>
</tr>
<tr>
<td>CLOB (n) for unicode data</td>
<td>NTEXT</td>
</tr>
</tbody>
</table>

Table 4: **SQL Server Large Object data types**

**DB2 390**

The WPCP databases for DB2 390 present a number of differences from DB2 (for Linux, Unix and Windows). Among the differences are the need to use tablespaces, differences in data types, the definition of primary keys and loading of large tables.

**Tablespaces**

For DB2 390 WPCP uses tablespaces and on DB2 it does not. A tablespace specifies where and how the actual data from a database table is stored. For DB2, “the where” is a container, which is either a file or device. Configuration options on how the tablespace is managed and stored include, memory page size, paging behavior, system or database
managed. WPCP creates tables in specific tablespaces for both the author-time and run-time tables.

WPCP organizes its tables into different schemas. On DB2 390 there is a one-to-one correspondence between database and tablespace. As shown in Table 5, the recommended configuration for author-time and run-time tables is to share the same tablespace, while Feedback tables are in a separate tablespace. This configuration allows you to tune and backup these tables separately based on the usage characteristics of the tables. Typically, the Feedback tables have a higher volume of data, whereas the author-time and run-time tables have a higher access frequency.

<table>
<thead>
<tr>
<th>Database (default database)</th>
<th>Schema</th>
<th>Database.Tablespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author-time (wpcp50)</td>
<td>WCMDBADM</td>
<td>[WCM.DB].[WCM.TS]</td>
</tr>
<tr>
<td>Run-time (wpcp50)</td>
<td>PZNADMIN</td>
<td>[PZN.DB].[PZN.TS]</td>
</tr>
<tr>
<td>Run-time (wpcp50)</td>
<td>EJB</td>
<td>[EJB.DB].[EJB.TS]</td>
</tr>
<tr>
<td>Feedback (fdbk50)</td>
<td>FEEDBACK</td>
<td>[FBK.DB].[FBK.TS]</td>
</tr>
</tbody>
</table>

Table 5: Database-Schema-Tablespace recommendations

Here is a comparison of the DB2 and DB2 390 CMCONTEXT table (author-time) definition. In this example [WCM.DB] represents the database name and [WCM.TS] represents the tablespace name.

**DB2:**
```sql
CREATE TABLE CMCONTEXT
(
  NAME          VARCHAR(100) NOT NULL,
  DESCRIPTION   VARCHAR(200),
  READONLY      CHAR(1),
  ACTIVITYID    VARCHAR(200),
  PRIMARY KEY(NAME)
);
```

**DB2 390:**
```sql
CREATE TABLE CMCONTEXT
(
  NAME          VARCHAR(100) NOT NULL,
  DESCRIPTION   VARCHAR(200),
  READONLY      CHAR(1),
  ACTIVITYID    VARCHAR(200),
  PRIMARY KEY(NAME)
) IN [WCM.DB].[WCM.TS];
CREATE UNIQUE INDEX IKEYCM ON CMCONTEXT(NAME);
```

Also, when creating the author-time tablespace, WPCP specifies the paging behavior of the tablespace as well. Here is the default specification:

```sql
CREATE TABLESPACE [WCM.TS] IN [WCM.DB] FREEPAGE 15 SEGSIZE 16 LOCKSIZE ROW;
```

On DB2 390 specific locking behavior can be specified at the page level by creating a tablespace and specifying its paging and locking attributes. For the WPCP author-time
database the tablespace is created to lock rows, meaning multiple rows can be accessed in parallel. This ensures the correct locking behavior for the author-time tables.

**Data Types**

Another WPCP table definition difference between DB2 390 and DB2 is that the \texttt{DECIMAL(13,0)} data type specification replaces \texttt{BIGINT}. This variation exists in both the author-time and Feedback databases. For example:

**DB2:**

```sql
CREATE TABLE CMACTIVITY
  ( ID          VARCHAR(200) NOT NULL,
    TIME        BIGINT NOT NULL,
    PROCESSID   VARCHAR(100),
    NAME        VARCHAR(32),
    STATE       INTEGER,
    OWNER       VARCHAR(32),
    PRIMARY KEY (ID, TIME)
 );
```

**DB2 390:**

```sql
CREATE TABLE CMACTIVITY
  ( ID          VARCHAR(200) NOT NULL,
    TIME        DECIMAL(13,0) NOT NULL,
    PROCESSID   VARCHAR(100),
    NAME        VARCHAR(32),
    STATE       INTEGER,
    OWNER       VARCHAR(32),
    PRIMARY KEY (ID, TIME) )
  IN [WCM=DB].[WCM=TS];
  CREATE UNIQUE INDEX ICMAID ON CMACTIVITY (ID, TIME);
```

**Primary Keys**

Primary keys are defined differently between DB2 390 and other databases. On DB2 390, a unique index must be explicitly created for each primary key, while other databases automatically create unique indexes for primary keys.

There are two ways to define a primary key. For both the Feedback and run-time databases, primary keys are added using the \texttt{ALTER} statements, while author-time tables specify primary keys on the \texttt{CREATE TABLE} statements. The following is an example showing the \texttt{PZNADMIN.CAMPAIGNS} (a run-time table) table creation using the \texttt{ALTER} statement syntax (see the \texttt{CMACTIVITY} table definition above for an example of creating primary keys on the \texttt{CREATE TABLE} statement):

**DB2:**

```sql
CREATE TABLE PZNADMIN.CAMPAIGNS
  ( SCOPEID           VARCHAR(32) NOT NULL,
    CAMPAIGNNAME      VARCHAR(250) NOT NULL,
    PRIORITY          INTEGER NOT NULL,
    CAMPAIGNSSPLIT    INTEGER NOT NULL,
    PRIMARY KEY (SCOPEID, CAMPAIGNNAME)
 );
```
When designing database tables, keep in mind that DB2 390 has different maximum sizes for:

- The `LONG VARCHAR` data type, the size of which is determined by a formula based on the page size of the 390 system and the other columns in the table
- The total number of bytes (the sum of the column widths) in a primary key may not exceed 255 bytes

**WPCP Usage Considerations**

Anywhere an extension of WPCP is possible there may be database application considerations. This includes managing Resources, performing custom logging and reporting. The following discussion offers some guidelines and highlights key differences when extending WPCP.

**Resources and ResourceManager Classes**

If writing a Resource, database access and processing are entirely internal to that Resource. The Resource code needs to handle all SQL syntax and database access considerations.

When using Resource and ResourceManager classes generated by the WPCP wizards, there are some design issues to be aware of. You should first create the DataSource in WebSphere Application Server (WAS) as part of deploying your application. This is the DataSource you specified when generating the Resource. This allows the generated Resource to access the database through a JNDI name lookup. If the tables for the Resource do not exist, the ResourceManager will create the tables at runtime.
Be aware that the wizards generate code for the single JDBC driver DataSource class you choose such as:

- COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource for DB2
- com.merant.sequelink.jdbcx.datasource.SequeLinkDataSource for SQL Server

Again, it is strongly recommended that you create a WAS DataSource and reference the database through its JNDI name. This recommendation applies to generated resources and custom resources that need to access a database.

**Logging**

If writing custom code to extend the WPCP logging framework, you need to handle all SQL syntax and database application differences in your custom code.

WPCP provides a set of logging functions that include both an automatic rule logging feature (when using personalization rules), as well as a programmable logging feature (logging beans). Data logged by either of these techniques is collected and stored by WPCP into the Feedback database schema. Specifically, by using either personalization rules or logging beans in a WebSphere Portal Web application, data is captured, then dispatched, in the form of Java™ event objects, to a set of registered listeners. One such listener, the WPCP Feedback Listener, stores the data in the Feedback schema for the purpose of subsequent reporting. The logging framework consists of events, listeners and the WPCP log manager.

The WPCP logging framework can be extended. One extension is to write a custom listener to intercept data as it is logged, to process and store it into a set of user-defined database tables. Database application differences such as those presented earlier may affect custom code written for a custom listener.

Figure 6 depicts the WPCP logging subsystem. In this illustration, a custom listener collects logged data and stores it into custom database tables.
Reporting

Note: If you extend the WPCP reporting framework to write custom reports, you are responsible for the applicable SQL syntax and database application differences in your custom code.

WPCP provides a reporting framework (and a set of default reports) that can be used to write custom reports against the Feedback database schema. The default reports consist of Rule, Category, Action and Rating Ranking as well as Rule and Campaign Effectiveness.

Each default report has the same basic structure. Each obtains a connection to the Feedback database followed by several SQL queries against the database to retrieve the data. In many cases, these queries differ, based on database application. When writing a custom report, users often copy a WPCP default report and modify it to meet their needs.

The following characteristics of database applications affect the contents of the default reports:

1) Table names which match certain reserved database keywords must be delimited when referenced in SQL syntax. For example:

- The Feedback table name `KEY` must be delimited by double quotes (" ") in Cloudscape, and by left and right brackets ([ ]) in SQL Server.
- Columns names `MONTH`, `DAY` and `YEAR`, must be delimited by double quotes (" ") in Cloudscape.
• The column name **ID** must be delimited by left and right brackets ([ ] ) in SQL Server.

Consequently, the SQL for the following prepared statement is slightly different under Cloudscape than under the other database applications supported by WPCP.

**Cloudscape:**

```
SELECT ID FROM FEEDBACK.CALENDAR
WHERE "MONTH"= ? AND "DAY"= ? AND "YEAR"= ?
```

**Other supported database applications:**

```
SELECT ID FROM FEEDBACK.CALENDAR
WHERE MONTH= ? AND DAY= ? AND YEAR= ?
```

Writers of custom reports need to be aware of these subtle differences when referencing the Feedback tables.

2) Database schema names can be used explicitly as part of a table name when referencing the table in SQL or the schema name can be set on the database connection. Setting the schema on the connection allows for table names to be referenced directly in SQL without the schema name. For DB2 390, Informix and SQL Server the WPCP reports specify the schema name with each table reference. For DB2, Cloudscape and Oracle the reports set the schema name on the connection. To set the schema on the connection, WPCP reports issue the following statements:

**DB2 and Cloudscape:**

```
SET CURRENT SCHEMA = FEEDBACK
```

**Oracle:**

```
ALTER SESSION SET CURRENT_SCHEMA = FEEDBACK
```

Consequently, the SQL for the following statement (made within each default report) is slightly different under DB2, Cloudscape and Oracle, than under DB2 390, Informix and SQL Server.

**DB2, Cloudscape, Oracle:**

```
SELECT SUM(HF.HITS) FROM HIT_FACTS HF
WHERE HF.GMTDATE_ID BETWEEN 1234 AND 1234
AND HF.RECORDTYPE = 32768
```

**DB2 390, Informix and SQL Server:**

```
SELECT SUM(HF.HITS) FROM FEEDBACK.HIT_FACTS HF
WHERE HF.GMTDATE_ID BETWEEN 1234 AND 1234
AND HF.RECORDTYPE = 32768
```

Again, writers of custom reports should be aware of these differences.
3) Not all SQL constructs are supported by all database applications. For example, in DB2, DB2 390 and Cloudscape, **LEFT JOIN** can be used in the main SQL query of the WPCP Rating Rank report. For Oracle, Informix and SQL Server, the Rating Rank report must use a different query. Although each WPCP default report contains a number of SQL queries, a main query retrieves most of the data. The following SQL statement is the main query in the Rating Rank report for DB2, DB2 390 and Cloudscape. This query uses **LEFT JOIN**.

Note: In the following query, table names are not fully qualified. In the Rating Rank report, when issuing this query against DB2 390, the report fully qualifies the table names. (e.g. The table name **KEY** is referenced directly, instead of **FEEDBACK.KEY**.) Additionally, there are slight differences for Cloudscape.

**DB2, DB2 390, Cloudscape:**

```sql
SELECT KVC.VALUE AS RESOURCEID, KVV.VALUE AS COLLECTION,
AVG(DOUBLE(FILTERED_PIDLIST.SCORE)) AS AVE,
MAX(FILTERED_PIDLIST.DATE_ID) AS MAXDATE,
MIN(FILTERED_PIDLIST.DATE_ID) AS MINDATE
FROM ( SELECT HP1.PARMS_ID AS PID, HF.GMTDATE_ID AS DATE_ID,
V1.VALUE AS SCORE
FROM HITPARMS HP1, VALUE V1, PARMS P1, KEY_VALUE_COMBO KC1,
CALENDAR C1, KEY_VALUE_PAIR KP1, KEY K1,
HIT_FACTS HF
WHERE HP1.HIT_ID = HF.ID AND HP1.PARMS_ID = P1.ID
AND KC1.PARMS_ID = P1.ID AND KC1.KEYVALUEPAIR_ID = KP1.ID
AND KP1.KEY_ID = K1.ID AND KP1.VALUE_ID = V1.ID
AND HF.GMTDATE_ID = C1.ID AND HF.GMTDATE_ID BETWEEN 1234
AND 1234 AND HF.RECORDTYPE = 32768 AND K1.NAME = 'wcpRating'
) AS FILTERED_PIDLIST
LEFT JOIN
( SELECT J1_KC.PARMS_ID, J1_K.NAME, J1_V.VALUE
FROM KEY J1_K, VALUE J1_V, KEY_VALUE_PAIR J1_KP, KEY_VALUE_COMBO J1_KC
WHERE J1_KC.KEYVALUEPAIR_ID=J1_KP.ID AND J1_KP.KEY_ID=J1_K.ID
AND J1_KP.VALUE_ID=J1_V.ID AND J1_K.NAME='wcpRatingResId'
) KVC ON FILTERED_PIDLIST.PID=KVC.PARMS_ID
LEFT JOIN
( SELECT J2_KC.PARMS_ID, J2_K.NAME, J2_V.VALUE
FROM KEY J2_K, VALUE J2_V, KEY_VALUE_PAIR J2_KP, KEY_VALUE_COMBO J2_KC
WHERE J2_KC.KEYVALUEPAIR_ID=J2_KP.ID AND J2_KP.KEY_ID=J2_K.ID
AND J2_KP.VALUE_ID=J2_V.ID AND J2_K.NAME='wcpRatingCollection'
) KVV ON FILTERED_PIDLIST.PID=KVV.PARMS_ID
GROUP BY KVC.VALUE, KVV.VALUE
ORDER BY AVE DESC
```

The following is the version of the main query of Rating Rank for Oracle, Informix and SQL Server.

Note: Note variations in the syntax of the line beginning with **AVG**, due to the conversion of character data to numeric value. Also note that table names are not
fully qualified as required for Informix and SQL Server. (e.g. The table name `KEY` is referenced directly, instead of `FEEDBACK.KEY`.)

**Oracle, Informix, SQL Server:**

```sql
SELECT V.VALUE AS RESID, R.VALUE AS COLID,
AVG(TO_NUMBER(V1.VALUE)) AS AVE,  //Oracle
//AVG((V1.VALUE)::FLOAT) AS AVE,  //Informix
//AVG(CAST(V1.VALUE AS FLOAT)) AS AVE,  //SQL Server
MIN(HF.GMTDATE_ID) MN, MAX(HF.GMTDATE_ID) MX
FROM HIT_FACTS HF, VALUE V, VALUE R, VALUE V1, PARMS P,
KEY_VALUE_COMBO KC, KEY_VALUE_PAIR KP, KEY K,
KEY_VALUE_COMBO KC2, KEY_VALUE_PAIR KP2, KEY K2
WHERE P.ID IN (SELECT HP1.PARMS_ID
FROM HITPARMS HP1, PARMS P1, KEY_VALUE_COMBO KC1,
CALENDAR C1, KEY_VALUE_PAIR KP1, KEY K1
WHERE HP1.HIT_ID = HF.ID AND HP1.PARMS_ID = P1.ID
AND KC1.PARMS_ID = P1.ID AND KC1.KEYVALUEPAIR_ID = KP1.ID
AND KP1.KEY_ID = K1.ID AND KP1.VALUE_ID = V1.ID
AND HF.GMTDATE_ID = C1.ID AND HF.RECORDTYPE = 32768
AND K1.NAME = 'wcpRating')
AND ( (KC.PARMS_ID = P.ID AND KC.KEYVALUEPAIR_ID = KP.ID
AND KP.KEY_ID = K.ID AND KP.VALUE_ID = V.ID
AND K.NAME = 'wcpRatingResId'
AND KC2.PARMS_ID = P.ID
AND KC2.KEYVALUEPAIR_ID = KP2.ID
AND KP2.KEY_ID = K2.ID AND KP2.VALUE_ID = R.ID
AND K2.NAME = 'wcpRatingCollection' ) )
GROUP BY V.VALUE, R.VALUE
ORDER BY AVE DESC
```

**Summary**

WPCP supports DB2, Oracle, Informix, Cloudscape, SQL Server, and DB2 390, a variety of database applications, to provide a flexible content management solution. Although this effort poses many developmental challenges, it conceals many of the underlying database differences from WPCP end users. Customers are welcome to extend the current WPCP framework to cater to their business needs, however, some important provisions are recommended to assist in this process. We hope the information presented here is beneficial when using WPCP.

**Resources**

- Visit the [WebSphere Portal Zone](#) for more information about WPCP.
- Here is the [DB2 for Windows SQL Reference](#). DB2 is our baseline database.
- Recent Oracle database documentation is available [here](#).
- Informix is one of DB2’s Information Management Database Servers.
• **IBM Cloudscape** is now part of the DB2 Family.

• Learn more about Microsoft SQL Server [here](#).

• **DB2 390** has several implementation differences from DB2.

Special thanks to Mary Streble and Carol Newbold for their technical contributions.

**Trademarks**

The following are trademarks of International Business Machines Corporation: IBM, DB2, OS/390, Cloudscape, Informix and WebSphere.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product and service names may be trademarks or service marks of others.

© IBM Corporation 2003. All rights reserved.