IBM DB2 for i porting guide

Oracle to IBM i platform

.

Database technology team
ISV Business Strategy and Enablement
October 2014
Version 8.1

Table of contents

Abstract

This paper explains various processes and considerations regarding porting an Oracle database application to the IBM i platform to run with DB2 for i. Discussions include: assessing the differences between the source and target databases, porting approaches and administrative differences.

Introduction

This is a working document; new topics will be added as they appear in more and more porting situations. Meanwhile, depending on the type of application, not all topics discussed in this paper are relevant to a particular situation.

The audience for this document

This document is written for application developers and database administrators who want to convert an Oracle® application to IBM® DB2 for i®. The document covers the most common issues and inconsistencies encountered by developers when porting from Oracle to DB2 for i. It also covers DB2 support and administration topics that are relevant to database administrators.

This document concentrates primarily on the database differences between the Oracle and DB2 for i database servers. It does not focus on the differences between the underlying operating systems. Application development issues are also only covered from a database perspective.

Assumptions

The paper assumes readers are working with Oracle 11g Release and are porting to the DB2 for i 7.2 release. For simplicity, this paper uses DB2 to represent DB2 for i.

The document also assumes readers are familiar with the concepts of RDBMS and with Oracle SQL and PL/SQL. It is also assumed that readers have easy access to DB2 for i documentation. Refer to that documentation for detailed information about the actual SQL statement syntax. DB2 for i documentation can be found online at: **ibm.com**/systems/power/software/i/db2/docs/books.html.

Preparing to port

Before diving into a detailed comparison of DB2 and Oracle, the first step in considering a port to DB2 is reviewing the source database. A firm understanding of the technologies and interfaces used by your Oracle application makes you more productive as you use this document to assess the port to DB2. Here are some questions that you need to be able to answer about your application and database:

- Does the application use standard or proprietary SQL such as the **Connect By** statement?
- What programming interfaces does the application use for data access?
- Does the application rely on any platform-specific middleware or technology?
- Are there any known performance bottlenecks?
- Is the business logic in procedural database objects (triggers and procedures) or the application?

To request a formal DB2 porting assessment document, send an e-mail to: rchudb@us.ibm.com

Porting approaches

After you have a thorough understanding of the source database, it is time to consider the approach that you want to take when porting your application to DB2. Any solution can be ported, given enough time and money. However, the question to ask is: “What is the end goal for the application being ported? Does it need to be a portable solution that can easily support multiple DBMS products or a solution that is tightly integrated and tuned for DB2?” This porting issue needs to be discussed before the database porting project starts to make sure that all of the parties on both sides agree on the priority. If the solution already supports multiple DBMS products, then this is a relatively easy question to answer. With its support for industry-standard interfaces, DB2 allows this application design with minor changes.

Design trade-offs

However, if your application currently supports just a single DBMS product, you have some application design issues to investigate further. The first alternative to consider is redesigning your application with an abstraction layer to support database servers more easily with a single code base. This design approach requires more effort and investment, but also better positions your application for expansion to other platforms and databases in the future. In addition, as you enhance your core application, this design makes if faster to deploy these enhancements to all of your supported platforms.

If it is important to maintain the source code to remain portable, a good application design is to isolate all database runtime calls from the application. This is usually accomplished by having the application create its own set of database methods or calls to pass to a single procedure or module. That procedure or module then turns the application database request into the specific format or API (ODBC, JDBC or other interfaces) supported by the target database.

Another alternative is to create a separate version of your application specifically for DB2 for i. Although this can require less up-front investment than the previous approach, the long-term expenses are greater because of having a second code base to maintain, enhance and test. The benefit of this approach is that the best performance is usually obtained by changing the application and database to exploit the target server. However, the more changes you make to exploit the target, the harder it is to have a single set of application source code that can run against multiple database servers. If the goal is to have a common set of single-source code, then avoid platform-specific features in the conversion. Performance is usually the tradeoff when you code to the lowest common denominator. For example, Oracle has a scalar function called **X** and DB2 has an equivalent function called **Y**. The developer who is converting to DB2 has two choices: change the application to call function **Y** (which yields the best performance) or code a DB2 user-defined function (UDF), called **X**, that does nothing other than call function **Y**. (This allows the application code to remain unchanged, but results in slower performance.)

This tradeoff is especially critical with DB2 for i because its implementation of dynamic SQL **prepare** and **execute** statements has some nuances. Many times, an application designed for other database servers blindly prepares and runs the same SQL statement repeatedly within a single program call, even though it is more efficient to prepare that SQL statement only one time. Or the application uses the ODBC **SQLExecDirect** function or the JDBC **statement** class to run the same SQL request multiple times within a connection. DB2 is not suited for either of these inefficient program models. If the application cannot be changed to a model where an SQL statement is prepared once and run many times, then most likely, that application will not perform well on the IBM i platform.

On the performance topic, it is strongly recommended that database administrators or SQL developers who are new to the IBM i platform attend the **DB2 for i SQL Performance Workshop** (**ibm.com**/systems/power/software/i/db2/education/performance.html). This course teaches the developer the proper way to design and implement a high-performing DB2 solution.

Misused porting approach

Because DB2 is also available for Linux™ and Microsoft® Windows® workstations, many developers wonder if they can perform their porting and associated testing on the workstation and then move it to the IBM i platform after the testing on the workstation has been completed. Does that work? The short answer is, “Probably not.” Although DB2 for i does belong to DB2 Family, there are differences in the SQL syntax and features supported by each DB2 product. There is not a master version of DB2 and each product has features that are not supported by one of the other database products. In addition, DB2 for i does not include support for the Oracle Compatibility Mode first delivered with the DB2 9.7 release.

This approach only works if, prior to using any SQL statement on the workstation version, the developer first verifies that the SQL statement and syntax is supported by DB2 for i. An online white paper can be referenced for a high-level examination of DB2 for i and the IBM DB2 Family at the IBM PartnerWorld® (**ibm.com**/partnerworld/wps/servlet/ContentHandler/SROY-6UZDN4)**.**

Porting tools

Porting tools are available from IBM and other providers to automate some of the porting process.

The IBM DB2 Migration Toolkit for Oracle is designed to automate many of the steps involved in porting of a database from Oracle to DB2 for i. This no-charge download automates the migration of tables, views, indexes, triggers and stored procedures, as well as other Oracle objects. This toolkit can be accessed at the following web site: **ibm.com**/i/partnerworld/db2porting.

Although the DB2 Migration Toolkit can greatly reduce the amount of time it takes to convert from Oracle to DB2, it does not completely automate the database conversion process. For example, the toolkit does not convert references to the Oracle catalogs to the corresponding DB2 catalog references. The programmer has to perform some manual work. However, the toolkit does flag items that have to be converted manually by the programmer. The converted programming objects (triggers, functions and stored procedures) need to be reviewed to ensure they are semantically equivalent. In addition, the SQL code generated by the DB2 Migration Toolkit might not result in the best-performing SQL statements. This means that some performance tuning also needs to be done after using the toolkit.

There are also a couple of tools that can be purchased from other software providers (SwisSQL and Ispirer) online at: www.swissql.com and www.ispirer.com. These conversion tools might not generate SQL that is specifically supported by DB2 for i.

If your application depends on proprietary features that are not supported by existing migration tools, then it can be prudent to consider building your own migration tool. This approach involves a significant up-front investment and require an in-depth knowledge of both the source and target databases.

Sizing the port

Although the design approach and usage of porting tools definitely affect the costs required to complete the port, the effort is also dependent on the features used in the source database. This section highlights and compares some of the key differences encountered between Oracle and DB2 databases during application ports to help assess the difficulty of your porting project.

Architecture

This section provides an overview of the DB2 architecture and a comparison of that architecture with the Oracle database product. In addition, the interfaces used with Oracle and DB2 databases are compared.

Brief history

An integrated, fully relational database has shipped with every IBM i machine as far back as the inception of the IBM AS/400® systems in the late 1980s. Many users did not realize they had a database because this integrated relational database originally had no name. In 1995, the database joined the DB2 brand by adopting the name DB2/400. Then, in 1999, the DB2 Universal Database branding was added. DB2 for i is still running on the original database engine; it has more features and functions with each new release of the IBM i operating system. In fact, the database is integrated so well that some IBM i users say that they are not using DB2.

Because the AS/400 family of systems was developed before SQL was widely used, a proprietary language and high-level language extensions (think of them as APIs) were made available for relational database creation and data access. Data Definition Specification (DDS) is the language used for creating DB2 objects. The language extensions (or APIs). known as the record level I/O interfaces. are available in most of the languages supported by the IBM i platform with the most usage in RPG and COBOL programs. These extensions and DDS are also known as the native database interface and are quite similar to indexed sequential access method (ISAM).

Because of this native, non-SQL interface, some IBM i users and consultants use terminology that is unfamiliar to those coming from an SQL background. Table 1 provides a mapping of that terminology:

SQL terms	IBM i terms
TABLE	PHYSICAL FILE
ROW	RECORD
COLUMN	FIELD
INDEX	KEYED LOGICAL FILE, ACCESS PATH
VIEW	NON-KEYED LOGICAL FILE
SCHEMA	LIBRARY, COLLECTION, SCHEMA
LOG	JOURNAL
ISOLATION LEVEL	COMMITMENT CONTROL LEVEL

Table 1: Mapping of SQL and IBM i terms

Because of the integrated nature of the IBM i database, both the native and SQL interfaces are almost completely interchangeable. Objects created with DDS can be accessed with SQL statements. And, objects created with SQL can be accessed with the native record level access APIs. The DB2 SQL interface is compliant with the core level of the SQL 2011 standard.

Interfaces and packaging

Because DB2 is integrated, most SQL-based applications run on any IBM i hardware without requiring additional products to be purchased for the client or server. The following DB2 capabilities are included with the IBM i operating system at no additional charge:

- SQL parser and runtime support
- SQL interfaces (server side)
  - Call level interface (CLI)
  - Native JDBC driver (Version 4.0)
  - SQLJ
- IBM i Access for Windows (formerly IBM Client Access Express and IBM iSeries Access)
  - ODBC (Version 3.5)
  - JDBC (Version 4.0)
  - OLE DB provider
  - .NET provider (Version 2.0)
- Open Group™ DRDA application requester and server
- PHP ibm_db2 extension
- SQL Statement Processors (**RUNSQLSTM & RUNSQL** CL command)
- Qshell DB2 utility
- Performance tuning tools
- System i Navigator

The DB2 for i SQL Development Kit and Query Manager (5722-ST1) product must be purchased if you need to embed SQL in a high-level language (C, C++, RPG, or COBOL). This product only needs to be installed on your development system.

If an application already supports DB2 access through DB2 Connect middleware, another option is DB2 Connect Unlimited Edition for System i (**ibm.com**/software/data/db2/db2connect/edition-uei.html). This product is not included with the base operating system; it must be purchased separately.

Many vendors prefer a native database interface (for example, Oracle Call Interface [OCI]) instead of an industry-standard interface, such as ODBC or CLI, to improve performance. DB2 does not really have a native SQL-based interface, instead the ODBC and CLI interfaces are treated as such for DB2 for I, providing performance similar to the native interfaces offered by other database products.

The one SQL-based interface similar to a native interface for DB2 is the Extended Dynamic interface (through the **QSQPRCED** API or **XDA** API set). However, this interface is proprietary and is used effectively only by thoroughly understanding the DB2 for i SQL engine. That knowledge is obtained from the **DB2 for i SQL Performance Workshop** (**ibm.com**/systems/power/software/i/db2/education/performance.html).

Storage model

The physical storage model for DB2 and Oracle differ drastically. With DB2, all storage allocation and management is handled by the database manager and operating system. Thus, there are no tablespaces to create or manage on the IBM i platform. In addition, the data for a DB2 object is automatically partitioned (or striped) across the disk devices to eliminate contention.

DB2 does support the notion of independent, isolated databases. However, its default configuration is as a single, system-wide database. If developers want to create independent, isolated databases on the IBM i platform, they must configure a new, independent auxiliary storage group.

DB2 for i supports partitioned tables. However, partitioned tables should only be used when the row count or size limit for a single table is about to be exceeded. For more details on the DB2 for i partitioned table implementation, consult the white papers listed on the **IBM i Education Curriculum** at **ibm.com**/partnerworld/wps/whitepaper/i5os.

Metadata

Metadata provides the roadmap to interpret information stored in the database. In Oracle, metadata is stored in the Data Dictionary; in DB2, metadata is stored in the System Catalog. The preferred way to reference metadata in DB2 is with catalog views, such as **SYSIBM.TABLES**, not the underlying tables. The structure and names of these views are different from the Oracle metadata tables, so you must understand the views to map the system information from Oracle to DB2. Catalog-view descriptions are in an appendix of the *DB2 for i SQL Reference Guide*. DB2 for Linux, UNIX and Windows, as well as DB2 for i support a common set of catalog views for ODBC and JDBC clients in the **SYSIBM** schema.

Data types

The following table (Table 2) summarizes the mapping from the Oracle data types to corresponding DB2 data types. The mapping is one-to-many and depends on the actual usage of the data.

Oracle data type	**Notes**	**DB2 data type**	**Notes**
DATE		TIMESTAMP(0)	— Use Oracle **TO_CHAR()** function to format a **DATE** for subsequent DB2 load. Note that the Oracle default **DATE** format is DD-MON-YY
TIMESTAMP		TIMESTAMP	
VARCHAR2(n)	n <= 4000	CHAR(n) VARCHAR(n)	n <= 32766, CHAR n <= 32740, VARCHAR
LONG	n <= 2 GB	VARCHAR (n) CLOB(n)	— if n <= ~32KB, use **CHAR** or **VARCHAR** —if 32 KB <= n <= 2 GB, use **CLOB**
RAW and LONG RAW	n <= 255	BINARY(n) VARBINARY(n) BLOB(n)	— if n <= 32 KB, use **CHAR(n)** for **BIT DATA** or **VARCHAR(n)** for **BIT DATA** — if n<= 2 GB, use **BLOB(n)**
BLOB	n <= 4 GB	BLOB(n)	— if n <= 2 GB, use **BLOB(n)**
CLOB	n <= 4 GB	CLOB(n)	— if n <= 2 GB, use **CLOB(n)**
NCLOB	n <= 4 GB	NCLOB	DBCLOB can also be used
NUMBER		SMALLINT / INTEGER / BIGINT DECIMAL(p,s)/NUMERIC(p,s) DECFLOAT	— if Oracle decl is **NUMBER(p)** or **NUMBER(p,0)**, use **SMALLINT, INTEGER**, or **BIGINT** — if Oracle decl is **NUMBER(p,s)**, use **DECIMAL(p,s)** —if Oracle decl is **NUMBER**, use **DECFLOAT**
XMLType		XML	See the **XML considerations** that follow next

Table 2: Mapping of Oracle data types to DB2

Null indicators

Null indicators are defined as *SMALLINT* or *short* in C/C++. When fetching null values from a column, the column's host variable is unchanged and the null-indicator variable is set to a negative value.

DATE and TIME

The Oracle **DATE** data type indicates year, month, day, hour, minute and second. It does not correspond to the DB2 **DATE** data type, which contains only the year, month and day. In DB2, the **TIMESTAMP** data type by default contains information from the year through to the seconds and microseconds. Specifying a precision of 0 for a DB2 TIMESTAMP column will result in a value that matches an Oracle DATE value. The Oracle **sysdate** data type is equivalent to the DB2 **CURRENT TIMESTAMP** special register.

NUMBER

The Oracle **NUMBER** data type can be mapped to many DB2 types, depending on whether the NUMBER data type is used to store an integer or a real number with a decimal portion. Another consideration is the space usage. Each DB2 type requires a different amount of space: the **SMALLINT** data type uses two bytes, the **INTEGER** data type uses four bytes, and the **BIGINT** data type uses eight bytes. The space usage for the Oracle **NUMBER** type depends on the parameter used in the declaration. The **NUMBER** data type, with the default precision of 38 significant digits, uses 20 bytes of storage. Mapping the **NUMBER** data type to the **SMALLINT** data type, for example, can save 18 bytes per column. An Oracle **NUMBER** data type with nonzero scale (decimal places) can be mapped to the DB2 **DECIMAL** or **NUMERIC** data types. The **DECIMAL** data type is stored as packed decimal in DB2, and the **NUMERIC** data type is stored in a zoned-decimal format. If no precision or scale is specified with the Oracle NUMBER data type, then DB2 DECFLOAT data type should be used.

Character types (including variable-length types)

DB2 supports the **CHAR** and **VARCHAR** data types for single-byte character strings and the **NCHAR**, **NVARCHAR**, b, **GRAPHIC and VARGRAPHIC** data types for double-byte character strings. Unicode data is supported with the **GRAPHIC** and **VARGRAPHIC** data types with a CCSID value of 1200 for the UTF-16 encoding. As of IBM i V6R1, the national-character data types (NCHAR, NVARCHAR and NCLOB) are also used to create a column with UTF-16 encoding. UTF-8 Unicode encoding is also available with the **CHAR** and **VARCHAR** data types with a CCSID value of 1208.

Many Oracle applications use the **VARCHAR2** data type for very small character strings. In these circumstances, it is better to port it to the fixed-length DB2 data type **CHAR(n)**, which is more efficient and takes less storage than **VARCHAR**. A general guideline is to use the **CHAR** data type for columns of 40 bytes or fewer. Before deciding on the column type based on performance, carefully read the DB2 for i paging and I/O behaviors for variable-length and LOB columns described in this section.

In general, the DB2 variable-length data types (**VARCHAR** and **VARGRAPHIC**) should only be used for long memo or text description columns that are not referenced very frequently. DB2 variable-length types usually cause additional disk I/O because they can be stored in a different data segment (auxiliary overflow storage) from the rest of the columns.

If it is not possible to change the column definition from a variable-length column to a fixed-length column, the **ALLOCATE** keyword can be used to improve performance. **ALLOCATE(0)** is the default and causes column value to be stored in the auxiliary overflow part of the row; this is the best value if the goal is to save space. The **ALLOCATE(N)** keyword allocates **n** bytes in the fixed portion of the row and only stores the column in the overflow area when the column value exceeds **N**. Setting **N** so that almost all of the column values are stored in the fixed portion of the row can improve performance by avoiding the extra I/O operation from the auxiliary overflow area.

Here is an example of changing the **ALLOCATE** value in an effort to improve performance. An address column is initially defined as **VARCHAR(60)** with the default **ALLOCATE** value of **0**, meaning that the data is stored in the auxiliary overflow section. Performance analysis has shown that this address column is retrieved frequently, so you want to move most of the address values back into the fixed portion of the row. If most of the address values are 40 bytes or less in length, then you can use the **ALTER TABLE** statement to change the **ALLOCATE** value to **40**. Now, all address values that are 40 bytes or less are stored in the fixed portion of the row, thus, eliminating the extra I/O operation on the auxiliary overflow area.

The problem is further magnified when both **LOB** and **VARCHAR** values are stored in the auxiliary overflow area. **LOB** storage is almost identical to **VARCHAR** column storage, so it is possible for both large **LOB** values and smaller **VARCHAR** columns to share the same overflow area. The sharing is not a problem, but when a single **VARCHAR** column that sits in the overflow area is referenced, the entire overflow area for that row is paged into main memory, including all BLOB and VARCHAR values. The application might only retrieve a **VARCHAR(50)** column, but if that column is in the same overflow storage area as three 2-megabyte **BLOB** values, then the application waits until the **VARCHAR** and all three large **BLOB** values are paged into memory. If a row contains both **BLOB** and **VARCHAR** columns, then most likely, the **VARCHAR** column's **ALLOCATE** value should be set to the maximum to ensure that the value is always stored in the fixed portion of the row.

To simulate the older Oracle **RAW** and **LONG RAW** data types, DB2 provides the **BINARY** and **VARBINARY** data types.

Oracle also supports three types of large objects: **BLOB**, **CLOB** and **NCLOB**. The Oracle **BLOB** and **CLOB** data types can store up to four gigabytes of data and map directly to the DB2 **BLOB** and **CLOB** types, which have a maximum size of two gigabytes. The DB2 **DBCLOB** type maps to the Oracle **NCLOB** data type for double-byte character sets.

The Oracle **BFILE** data type lets you manage binary data stored outside the database. A table in the database has a pointer to the externally stored **BFILE**. This is useful if data stored elsewhere must be shared with another system. DB2 has similar function with its **DATALINK** type.

ROWID

The Oracle **ROWID** pseudocolumn lets you uniquely identify or locate a row in the table. DB2 does support a **ROWID** data type that allows you to create a column with similar function to the Oracle **ROWID** pseudo column. The DB2 **ROWID** type stores a 40-byte unique value that does not contain any information about the data file, block, or row. Although DB2 allows direct access to a row through its **ROWID** value, the access of this row is no faster than a **WHERE** clause that is implemented with

an index scan. DB2 automatically creates a unique constraint over **ROWID** columns. Some DB2 for i solutions can also be used the **RRN** scalar function to simulate the **ROWID** value.

XML considerations

As of the 7.1 release, DB2 for i provides an XML data type similar to the Oracle XMLtype data type using the unstructured storage model. The DB2 for i XML functionality in 7.1 does not include an XQuery search interface. As an alternative, applications can employ the IBM OmniFind Text Search Server to index and search the stored XML values. (**Note:** For details on available XML support in prior IBM i releases, see this IBM Redbooks® document: *The Ins and Outs of XML and DB2 for i* (SG24-7258).

SQL language elements

This section covers the most common differences with SQL syntax and semantics.

Identifiers

DB2 for i supports two types of SQL identifiers: *ordinary* and *delimited*. Ordinary identifiers begin with an uppercase letter and are converted to all uppercase. A delimited identifier is a sequence of one or more characters enclosed within SQL escape characters (**..**). With delimited identifiers, leading blanks are significant and letters in the identifier are not converted to uppercase. The length of a delimited identifier includes the two SQL escape characters for column names, but not for other SQL names.

It is recommended that you not use variant characters (such as **$**, @, #, **^** and **~**) in SQL identifiers. This is because of the fact that the underlying code points of these characters can vary depending on the CCSID or language that is active on the system. If variant characters are used in your identifiers, then unpredictable results can occur when using these characters, especially on systems with nonEnglish CCSID values. The _ (underscore) character is not considered a variant character.

In general, the DB2 for i SQL identifier-length maximums are no problem when porting databases. DB2 supports identifiers with up to 128 characters for columns, tables, indexes, procedures and constraints. Authorization names, with a maximum length of 10 characters, are probably the only identifier that might cause issues. All SQL limits are explained in the *DB2 for i SQL Reference*.

Although DB2 supports long SQL identifiers, the commands and interfaces native to the IBM i operating system, there is a hard maximum of 10 characters for object names. For example, the **Save** commands that are used to back up database objects such as tables and indexes only support a 10-character identifier. Therefore, how do you back up an SQL object that has an identifier longer than 10 characters? When an SQL identifier longer than 10 characters is specified, DB2 automatically generates a short identifier of 10 characters that can be used with operating system commands and native interfaces. This short name is generated for all SQL object names, including column identifiers. These shorter SQL identifiers with the 10-character maximum are also known as system names.

The system-generated short names do have a downside because they are generated by the system. First, they are not user-friendly. DB2 appends a five-digit unique number to the first five characters of the SQL identifier. For instance, **CUSTOMER_MASTER** has a short name of **CUSTO00001**. Second and more importantly, these short names are not guaranteed to be consistent across systems or repeated creations of the same SQL object because of the dependencies on creation order and other

identifiers that share the same first five characters. Thus, special SQL syntax was added that allows the short name to be controlled by the developer.

The SQL **FOR SYSTEM NAME** clause was recently made available to allow a short name to be assigned to tables, indexes and views. The FOR COLUMN clause can be used on the **CREATE TABLE** and **ALTER TABLE** statements to assign a short column name. The **SPECIFIC** clause can be used to assign a short name when creating procedures and functions. Here are some examples of using this special SQL syntax to assign your own short system names:

```
CREATE TABLE dbtest/customer_master
  FOR SYSTEM NAME cusmst
    (customer_name FOR COLUMN cusnam CHAR (20),
     customer_city FOR COLUMN cuscty CHAR(40))
```

To overwrite the system-generated name for **customer_master** (**CUSTO0001**), the **FOR SYSTEM NAME** clause instead assigns a short name of **cusmst**. After running this statement, the SQL table name is **customer_master** and the system table name is **cusmst**. Either name can be referenced on other SQL statements.

The **FOR SCHEMA** clause can be used on the **CREATE SCHEMA** statement to assign a short system name for schema names that are longer than 10 characters.

Naming conventions and formats

Almost all DB2 for i SQL interfaces allow you to specify an SQL Naming Mode or Format. This option controls the syntax when qualifying an object with a schema (or library) name as well as defining the default search path used when unqualified SQL object names have been referenced.

***SYS** (System naming) mode is based on native-database access on the IBM i platform. This mode dictates using the traditional OS/400 slash (/) to explicitly qualify SQL objects with a schema name (for example, mylib/mytablename). Also, if a schema is unspecified, system naming searches the libraries defined in the job's library list looking for the unqualified object.

***SQL** (SQL naming) is based on the SQL standard and requires a period (.) to separate the schema and object name (for example, mylib.mytablename). With this convention, the only library (or schema) searched for unqualified objects is the library matching the name of the current SQL authorization name (or user-profile name). If the authorization name is JOHNDOE and an unqualified object is referenced, DB2 expects to find the object in a JOHNDOE schema.

Other settings can impact the default search path, but this section describes the default behavior of each of the naming conventions. Consult the **SQL Reference Guide** for a full explanation.

Three-part names

The SQL naming convention on DB2 for i does support full three-part names (for example, myschema.mytable.mycolumn) for those applications referencing columns in that manner. These full names cannot be used with the system naming convention (***SYS**).

Synonyms

In Oracle, you can create a synonym as an alternative name for an object. This includes a table, view, sequence, procedure, stored function, package, snapshot or another synonym. DB2

supports this concept with an ALIAS, but with some differences. You can only create an ALIAS on a table or view and the Oracle public keyword is not supported.

Functions

Many Oracle functions are supplied with DB2 with the same name and behavior, requiring no porting effort. Other Oracle functions have a different DB2 name but the same purpose and are also easily ported as sourced UDFs (see next section). The remaining functions have no DB2 equivalent or are user-written. Both are converted as UDFs on DB2. There are several scalar and table UDF types in DB2, all created with the **CREATE FUNCTION** statement. The basic UDF types are:

- **Sourced UDF:** This is a reuse of the implementation of an existing function, usually with some attributes changed, such as input data types. A sourced UDF is also used as an alias. If there is a function in Oracle, such as **ORAFUNC**, that has an equivalent called **DB2FUNC** in DB2, then a sourced UDF, called **ORAFUNC**, can be created in DB2, invoking the **DB2FUNC** function, and Oracle statements calling **ORAFUNC** need not be changed.
- **SQL-bodied UDF:** This can be fully implemented with the SQL procedural language (PSM).
- **External UDF:** This is used when it is easier to implement the function with any high-level programming language supported on IBM i, including Java™.

UDF performance

If performance is more critical than a single code base, avoid DB2 UDFs (if the same function can be performed inline with calls to IBM i functions). UDFs on IBM i are implemented with external program calls, and more performance overhead than system-supplied calls. Using the **NOT FENCED** and **DETERMINISTIC** attributes when creating functions inhances performance. See UDF examples at: **ibm.com**/developerworks/db2/library/samples/db2/0205udfs/index.html. (**Note:** Not all these UDF examples work on DB2 for i.)

NVL

The Oracle proprietary **NVL** function converts null fields to specified values. **NVL(MANAGER_ID, 'No Manager')** converts null values in the manager_id column to the string **No Manager**. In DB2, use the **COALESCE** function to convert nulls, as in **COALESCE(MANAGER_ID,'No Manager')**.

DECODE

DB2 does not provide the **DECODE** function. Instead, it supports the **CASE** expression. The mapping of **DECODE** to a **CASE** expression is very direct:

```
DECODE (condition, case1, assign1, case2, assign 2....,default)
```

Is equivalent to:

```
CASE condition
WHEN case1 THEN assign 1
WHEN case2 THEN assign 2
....
ELSE default
END
```

INSTR

The Oracle **INSTR** function maps most closely to the DB2 **LOCATE** function. If the **INSTR** function is used to look for the nth occurrence of a string instead of the first occurrence, the **start position** parameter needs to be used iteratively on the DB2 **LOCATE** function.

TO_CHAR

Here is a DB2 user-defined function that simulates the Oracle **TO_CHAR** function:

```
CREATE FUNCTION to_char ( t1 TIMESTAMP, format VARCHAR(32) )
  RETURNS VARCHAR(26)
  LANGUAGE SQL
  READS SQL DATA
  DETERMINISTIC
  NOT FENCED
  NO EXTERNAL ACTION
BEGIN
DECLARE chs_tmstmp CHAR( 26 );
DECLARE retval VARCHAR( 26 );

SET chs_tmstmp = CHAR( t1 );
CASE TRIM(format)
     WHEN 'DD'
       THEN SET RETVAL = SUBSTR( chs_tmstmp, 9, 2);
     WHEN 'MM'
       THEN SET RETVAL = SUBSTR( chs_tmstmp, 6, 2);
     WHEN 'YYYY'
       THEN SET RETVAL = SUBSTR( chs_tmstmp, 1, 4);
     WHEN 'YYYY-MM-DD'
       THEN SET RETVAL = SUBSTR( chs_tmstmp, 1, 10);
       ELSE SIGNAL SQLSTATE '38Z01'
            SET MESSAGE_TEXT = 'INVALID FORMAT SPECIFIED.';
 END CASE;
 RETURN retval;
 END;
```

Figure 1.

ROWNUM

The Oracle **ROWNUM** function has different syntax on DB2. Here are two examples of the DB2 **RowNumber** OLAP expression that provide capabilities similar to the Oracle **ROWNUM** function.

```
SELECT ROW_NUMBER() OVER() as rownum, name, dept
FROM employee

SELECT id_range, name, dept
FROM (SELECT ROWNUMBER() OVER(ORDER BY id) as id_range,
      name, dept FROM staff ) as id_list
WHERE id_range > 10 AND id_range < 20
```

Figure 2.

Considerations for null and empty strings

Oracle Database treats empty strings (zero-length character strings) as NULL values. This proprietary Oracle behavior can cause equivalent DB2 for i functions to return different values then the Oracle application expects. Here are the circumventions for some of the DB2 functions that behave differently than Oracle when empty-string and null values are processed.

- Concat(arg1,arg2) circumvention: NULLIF(CONCAT(coalesce(arg1,''), coalesce(arg2,'')), '')
- Trim(arg1) circumvention (also use for LTRIM and RTRIM): NULLIF(TRIM(arg1) , '')

- Substr(arg1,start,length) circumvention:
  CASE LENGTH(SUBSTRING(arg1,start,length)) WHEN 0
  THEN NULL ELSE SUBSTRING(arg1,start,length) END

Case-sensitivity considerations

Oracle applications also sometimes contain an expression on the **ORDER BY** clause [for example, **ORDER BY UPPER(col2)**] that is used merely to enforce an uppercase sorting of the data. If that is true of the application being ported, then one might consider employing a sort sequence to override the EBCDIC-based ordering of the data. This is done when creating tables (and indexes) by specifying ***LANGIDSHR** for the **Sort Sequence** parameter and **ENU** (English upper case) for the **Language Identifier** parameter. All of the IBM i SQL programming interfaces have connection attributes and parameters for specifying these settings.

The **SET OPTION** statement can also be used within the source of a program with embedded SQL. Usage of an uppercase sort sequence also makes all character comparisons case-insensitive for the specified table. When an application is run with a sort sequence, that sequence is applied to the ordering of the data (that is, **ORDER BY**) and to all character-value comparisons. These comparisons include basic predicates (**=**, **<** and others) as well as the **MIN** and **MAX** functions. Consult the **SQL Reference Guide** for additional details on sort sequences.

If case-sensitivity searches are implemented by specifying the **UPPER** function, then creating an index with a specific **sort** (or **translate**) table is an alternative to sort sequences. For instance, an index created with the QUSRSYS/Q037 table maps lowercase English letters into uppercase. This translation allows the index to be used on a search condition such as **WHERE UPPER(col2)='ABC'** to implement an index scan instead of a full-table scan.

SQL-derived indexes are an alternative to sort sequences that also allow the query optimizer to use index scans. The derived SQL-index support added in the IBM i 6.1 release enables SQL functions, such as Upper, to be included in the key definition on an index, as shown in the following example:

```
CREATE INDEX ixcol2 ON tab1(UPPER(col2))
```

Joins

Oracle supports a proprietary syntax for three types of outer joins: right, left and full. DB2 supports the standard SQL syntax for **Right** and **Left Outer Joins**. (See Table 3.)

Oracle Outer Join	DB2 Outer Join
SELECT A.last_name, A_id,B.name FROM emp A, Customer B WHERE A.id (+) = B.sales_rep_id;	SELECT A.last_name,A.id,B.name FROM emp A **RIGHT OUTER JOIN** customer B ON A.id = B.sales_rep_id;
SELECT A.last_name, A_id,B.name FROM emp A, Customer B WHERE A.id = B.sales_rep_id (+);	SELECT A.last_name,A.id,B.name FROM emp A **LEFT OUTER JOIN** customer B ON A.id = B.sales_rep_id;

Table 3: Oracle and DB2 Joins

The proprietary outer join syntax is also used in Oracle to replace the **NOT IN** predicate as follows:

```
SELECT Tbl1.col1 FROM Tbl1, Tbl2
WHERE Tbl1.col1 = Tbl2.col2(+) AND 'CHAR' = Tbl2.col2(+)
```

The DB2 equivalent of that statement is:

```
SELECT Tbl1.col1 FROM Tbl1
```

```
WHERE Tbl1.col1 NOT IN
      (SELECT Tbl2.col2 FROM Tbl2 WHERE 'CHAR' = Tbl2.col2)
```

MINUS set operator

Oracle supports a non-standard set operator that returns the difference of two sets. Here is an example of an SQL request utilizing the Oracle **MINUS** operator:

```
SELECT b.order_step_id, b.cust_orderid FROM event a, wrkcntrct b
WHERE a.cntrct_id = b.cntrct_id AND qty_type='D' AND qty > 0
   MINUS
SELECT b.order_step_id, b.cust_orderid FROM wrkordr a, wrkcntrct b
WHERE a.step_id = b.step_id AND status = 'D'
```

Here is the equivalent ported request on DB2 by employing the standard **EXCEPT** set operator:

```
SELECT b.order_step_id, b.cust_orderid FROM event a, wrkcntrct b
WHERE a.cntrct_id = b.cntrct_id AND qty_type='D' AND qty > 0
   EXCEPT DISTINCT
SELECT b.order_step_id, b.cust_orderid FROM wrkordr a, wrkcntrct b
WHERE a.step_id = b.step_id AND status = 'D'
```

CONNECT BY

Starting with the IBM i 7.1 release, DB2 for i includes support for the **CONNECT BY** clause which is used in **SELECT** statements to select rows from a table with hierarchical data (such as tree navigation).

Stored procedures

In Oracle, stored procedures must be programmed in PL/SQL. In DB2, SQL stored procedures can be written. The language used for them is a subset of the ANSI/ISO Persistent Stored Modules (PSM) specification, and is similar to PL/SQL and other stored procedure languages. This same PSM language is available for the development of SQL triggers and SQL UDFs on DB2.

DB2 also allows stored procedures to be written in any language supported by the DB2 precompilers; including Java, C, C++, RPG and COBOL. Stored procedures can use any database interface the IBM i platform supports (embedded SQL, CLI, JDBC and others). Regardless of language, a DB2 stored procedure is treated and behaves the same as any IBM i program object. SQL procedures are automatically converted to C code and compiled when the **Create Procedure** statement runs.

When Oracle stored procedures are migrated to DB2, the most appropriate approach is to use SQL procedures, because of their similarity to PL/SQL. SQL procedures are automatically converted to C code and compiled when the **Create Procedure** statement runs. Because the SQL procedures are implemented with generated C code, there are cases where the generated C does not perform as efficiently as a C program created by an application developer. In some porting exercises, because of performance requirements, the Oracle PL/SQL stored procedures have been implemented as external stored procedures (C with embedded SQL) on the IBM i platform. The performance of PSM stored procedures improved significantly with IBM i V5R4. Tips and techniques for coding procedural SQL with good performance are documented in the *Improving SQL Procedural Performance* white paper (**ibm.com**/partnerworld/wps/whitepaper/ibmi/sql_sql_v6r1/procedure).

The Oracle package object that allows multiple procedures to be stored in the same object and share global variables is not supported on DB2. One alternative is to create a schema and then place all of the procedures from the Oracle package into that schema. DB2 does include support for global variables to allow variable values to be shared across function and procedure calls.

With Oracle, unqualified names in both static and dynamic statements within procedures are, by default, resolved to the creator's authorization ID (user profile). The DB2 default behavior is different for static and dynamic SQL with ***SQL** naming. The **DYNDFTCOL** and **DFTRDBCOL** attributes can be specified on the **SET OPTION** clause of a DB2 SQL procedure to mimic the Oracle authorization behavior. To learn more about this approach, you can reference the article, **Improve Your Productivity with Significant Enhancements in DB2 SQL**, which is found in the Resources section. In addition, the IBM Redbooks document, **Stored Procedures, Triggers and User-Defined Functions on DB2 for i** can be referenced for more details.

You can see some of the PL/SQL constructs compared to the SQL procedural language in Table 4:

PL/SQL statement	SQL procedure language statement
CREATE PROCEDURE OR REPLACE p1 (v1 IN VARCHAR2(8), v2 IN OUT NUMBER(3)) AS BEGIN INSERT INTO t1 VALUES(v1,v2); END	CREATE OR REPLACE PROCEDURE p1 (IN v1 VARCHAR(8), INOUT v2 INTEGER) BEGIN INSERT INTO t1 VALUES(v1,v2); END

DECLARE
rowtype1 table1%ROWTYPE; [columns a,b,c];
Variable9 datatype9 := value9;
BEGIN
SELECT * INTO rowtype1 FROM table1
WHERE c = xxxx;
END;
	BEGIN
	DECLARE variablea datatype1;
	DECLARE variableb datatype2;
	DECLARE variablec datatype3;
	DECLARE variable9 datatype9 DEFAULT value9;
	SELECT a, b, c INTO variablea, variableb, variablec
	FROM table1 WHERE c = xxxx;
	END;

variable1 := value1;	set variable1 = value1;
SELECT a INTO variable1 FROM table1 WHERE b=yyyy;	set variable1 = (SELECT a FROM table1 WHERE b=yyyy);

IF condition1
THEN
statement1; statement2;
ELSIF condition2
THEN
statement3; statement4;
ELSE
statement5;
END IF;
	IF condition1
	THEN
	statement1; statement2;
	ELSEIF condition2
	THEN
	statement3; statement4;
	ELSE
	statement5;
	END IF;

LOOP statement1; statement2; EXIT WHEN condition; [or: if condition then exit; end if;] END LOOP;	myloop1: LOOP statement1; statement2; IF condition THEN LEAVE myloop1; END LOOP myloop1;
WHILE condition LOOP statement1; statement2; END LOOP;	WHILE condition DO statement1; statement2; END WHILE;

FOR index IN lower_bound..upper_bound
LOOP
statement1; statement2;
END LOOP;
	SET index = lower_bound;
	WHILE index <= upper_bound DO
	statement1; statement2;
	SET index = index + 1;
	END WHILE;

DECLARE CURSOR mycursor IS
SELECT a, b FROM table1 WHERE b=yyyy;
BEGIN
FOR table1_rec IN mycursor LOOP
a_total := a_total + table1_rec.a;
END LOOP;
	FOR my_loop AS
	SELECT a, b FROM table1 WHERE b=yyyy DO
	set a_total = a_total + a;
	END FOR;

Table 4: PL/SQL constructs compared to the SQL procedural language

Result set differences

Both Oracle and DB2 stored procedures can return result sets, but the syntax for doing this in a procedure is quite different. Here is a comparison of the Oracle and DB2 result set constructs.

Oracle:

```
CREATE OR REPLACE PROCEDURE sp_listemp RETURN types.cursortype
    As   l_cursor    types.cursorType;
    BEGIN
       OPEN l_cursor FOR SELECT ename, empno FROM emp ORDER by ename;
       RETURN l_cursor;
    END;
```

DB2:

```
CREATE OR REPLACE PROCEDURE sp_listemp ()
        RESULT SET 1
        LANGUAGE SQL
        BEGIN
           DECLARE l_cursor CURSOR WITH RETURN TO CALLER FOR
        SELECT ename, empno FROM emp ORDER BY ename;
           OPEN l_cursor;
         END;
```

In more recent versions of Oracle, support for cursor variables was added. These cursor variables can then be specified in the procedure parameter list as the following example shows. DB2 and the SQL standard do not allow cursors to be passed as parameters. Thus, this type of Oracle procedure has to be changed to return result sets using the DB2 constructs shown in the previous example.

Oracle:

```
CREATE PACKAGE Department AS
   TYPE  cursor_type IS REF CURSOR;
   PROCEDURE get_emps(l_cursor CURSOR_TYPE);
END;

CREATE PACKAGE BODY department AS
        PROCEDURE get_emps(l_cursor CURSOR_TYPE)
        AS BEGIN
          OPEN l_cursor FOR SELECT ename, empno FROM emp ORDER BY ename;
        END;
```

DB2:

```
CREATE PROCEDURE get_emps()
        RESULT SET 1
        LANGUAGE SQL
        BEGIN
           DECLARE l_cursor CURSOR WITH RETURN TO CLIENT FOR
        SELECT ename, empno FROM emp ORDER BY ename;
           OPEN l_cursor;
         END;
```

Exception processing

Exception handling is an area where the SQL and PL/SQL have some significant differences. PL/SQL can use a single **EXCEPTION** statement to catch multiple exceptions. However, the DB2

for i SQL procedural language requires a handler for each exception type. The following is an example of transforming the PL/SQL EXCEPTION block into handlers.

Oracle PL/SQL:

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN
     ROLLBACK;
  WHEN OTHERS THEN
     ROLLBACK;
```

DB2:

```
DECLARE UNDO  HANDLER FOR NOT FOUND, SQLEXCEPTION
     SET var='DoNothing';
DECLARE UNDO  HANDLER FOR SQLEXCEPTION SET var='DoNothing';
```

The PL/SQL **RAISE** statement is not available, but the SQL does include the **SIGNAL** and **RESIGNAL** statements for emulation of raised errors. The following examples contain a sample of that emulation, a user-defined exception named **out_of_stock** is raised in a PL/SQL stored procedure:

Oracle PL/SQL:

```
...
DECLARE
out_of_stock EXCEPTION;
number_on_hand NUMBER(4);
BEGIN
...
IF number_on_hand < 1 THEN
RAISE out_of_stock;
END IF;

EXCEPTION
WHEN out_of_stock THEN ...
-- handle the error
END;
```

In the equivalent DB2 example, the **SIGNAL** statement returns a user-defined exception:

DB2:

```
...
DECLARE number_on_hand int;
DECLARE out_of_stock CONDITION FOR 'STK01';
DECLARE EXIT HANDLER FOR out_of_stock ...
-- handle the error

BEGIN
...
IF number_on_hand < 1 THEN
  SIGNAL SQLSTATE 'STK01';
END IF;

END
```

The Oracle exception handlers have the same behavior as the DB2 exit handlers, which means that when the handler completes, the stored procedure terminates. Unlike Oracle, DB2 also enables processing to continue after an exception is encountered. In this case, when the handler is invoked, the execution continues with the next statement following the one that raised the

exception. In Oracle, the programmer needs to emulate this behavior by placing the statement along with its exception handlers in a PL/SQL sub-block.

PUT_LINE

Oracle supplies a range of proprietary packages that are quite frequently used by PL/SQL developers. For instance, the **dbms_output** package contains a **put_line()** procedure that is often used for debugging purposes. Calls to the **put_line** procedure are typically scattered throughout a stored procedure body so that it is possible to trace which sections of code have already been run. The output created by the **put_line** procedure is buffered and is not available until the PL/SQL program completes. When used from a client utility such as **SQL*Plus**, the content of the **put_line** buffer is typically displayed on a client workstation.

DB2 does not provide any built-in functions with capabilities similar to the **put_line** procedure. One of the common workarounds is to use a temporary table to store debug messages. The content of such a debug table can then be easily queried and displayed on a client workstation. However, some applications require that the messages are saved in a stream file (rather than a database table) to be subsequently processed by other application modules. Such applications can take advantage of the DB2 **put_line** workaround that has been published on the IBM developerWorks™ web site. For more information, refer to **ibm.com**/developerworks/db2/library/techarticle/0302izuha/0302izuha.html.

NOTE: The version of the source available on the developerWorks site does not support DB2 for i. Refer to **APPENDIX B** for a version of the workaround that works with DB2 for i.

The workaround allows you to write messages from a DB2 procedure directly into a stream file. Note that the behavior of the **put_line** workaround differs in certain areas from Oracle's implementation, as explained here:

- The workaround has been implemented as a set of overloaded user defined functions (UDFs) rather than procedures. For this reason, the invocation requires a function call.
- The output is not buffered and is immediately written to a stream file on the database server rather than to a standard output device on the client workstation.

Triggers

DB2 supports SQL and external triggers (implemented in high-level languages). SQL triggers are the most likely implementation because of the similarity with Oracle triggers. DB2 for i SQL triggers support **BEFORE ROW**, **AFTER STATEMENT** and **AFTER ROW** triggers; the **BEFORE STATEMENT** trigger is not supported. DB2 also supports transition variables. **OLD** and **NEW** column values, as well as access to logical tables containing the affected rows with the **OLD TABLE** and **NEW TABLE** clauses, are supported. As mentioned, PSM is available for implementing business logic in the SQL trigger.

In general, the SQL trigger body can contain any SQL statement. However, **BEFORE** triggers can alter data only with the **SET** statement; other SQL statements that insert, create or alter data are prohibited. See the **SQL Reference** for details of the IBM i SQL **CREATE TRIGGER** statement.

Because DB2 does not have the Oracle concept of versioning, it is not possible cannot get mutating errors with DB2 triggers; there is only one version of a row at any one time.

In Oracle, the trigger body consists of an anonymous PL/SQL block. In DB2, the trigger body consists of one or more SQL statements. If there is more than one SQL statement in a DB2 trigger, they must be enclosed by **BEGIN** and **END** clauses. DB2 triggers can invoke both stored procedures and functions.

Constraints

Both DB2 and Oracle support **primary key**, **foreign key** (referential integrity), **unique**, **NOT NULL** and **check** constraints. The constraint support is quite similar with some differences with referential integrity constraints. Oracle constraint support has a proprietary behavior of allowing constraint enforcement to be deferred until the end of the transaction. With DB2 for i, constraints are always enforced at the end of each statement.

DB2 automatically creates an index for the database manager's usage in enforcing the constraint when the following types of constraints are created:

- Primary key
- Foreign key
- Unique key

Referential integrity

Both DB2 and Oracle support foreign keys where a parent table's primary key is referenced by the child table. However, DB2 allows referential integrity (RI) to apply for any unique constraints, not just the primary key. This can reduce checking in the application code to enforce more complex dependencies between tables automatically.

DB2 provides the **SET NULL**, **SET DEFAULT** and **NO ACTION** options on a Delete operation, in addition to the **CASCADE** and **RESTRICT** options that Oracle also provides. The default for Oracle is **DELETE RESTRICT** and the default for DB2 and the SQL standard is **DELETE NO ACTION**. These are virtually identical in meaning. The approach used in porting is to use the same option in DB2 as is used in Oracle.

Sequence

An Oracle **Sequence** is a database object that can be used by multiple users to generate unique integers. Typically, the numbers drawn from a **Sequence** object are used as a primary key. For example, a **Sequence** object can be used to generate the next employee number or the next invoice order number. DB2 also supports a **Sequence** object with the **PREVIOUS VALUE** and **NEXT VALUE** operators mapping to the Oracle **CURRVAL** and **NEXTVAL** keywords respectively.

Miscellaneous differences

Here are a few additional SQL language differences between Oracle and DB2 for i.

SYS.DUAL

SYS.DUAL is a dummy table supplied by Oracle to let SQL requests get current IBM information. The DB2 family also provides a dummy table with a different name, **SYSIBM.SYSDUMMY1**.

Date and time arithmetic

Oracle allows you to add and subtract constants from dates without specifying a unit. For example, **DateCol + 2** adds two days to the value in that column. DB2 also supports a more robust set of date and time arithmetic operations, but requires the unit to be specified. The previous Oracle example must be coded as follows: **DateCol + 2 DAYS**

TO_DATE and TO_CHAR

DB2 does not have equivalent functions to Oracle **TO_CHAR** and **TO_DATE**, which will display date values in different formats. The DB2 **VARCHAR_FORMAT** and **TIMESTAMP_FORMAT** functions are the closest equivalents, but not all of the Oracle datetime formats are supported.

Unsupported features

Some Oracle capabilities are extremely difficult to support or come up with a DB2 circumvention. The following features might require portions of the application to be rewritten totally or redesigned:

- Oracle forms
- Materialized views
- Object-oriented and object-relational support
- Optimizer hints (**Note:** The DB2 optimizer is cost-based and does not give programmers any control. ISVs need to have someone attend the **DB2 for i SQL Performance Workshop** (**ibm.com**/systems/i/db2/db2performance.html)

Application development

Most Oracle applications being ported are programmed in either PL/SQL, C, or C++ with a little bit of COBOL. Most of the Oracle applications being ported by software vendors access the database with API-based interfaces such as OCI; not many of the ported applications are using embedded SQL. As discussed in earlier sections, the Oracle PL/SQL programming extension is the exclusive language for development of procedures, functions and triggers.

DB2 implements two standards-based Java programming APIs: JDBC and SQLJ. Because the JDBC and SQLJ implementations in DB2 are closely based on the standards, there is typically very little to do in converting Java applications from Oracle to DB2. Stored procedures and user-defined functions can be implemented with Java, as well.

Both DB2 and Oracle support dynamic SQL and embedded SQL interfaces.

Dynamic SQL

The Oracle Call Interface (OCI) is a proprietary Oracle programming interface for the C language. DB2 does not have a direct mapping for OCI applications, but when ported to DB2, they are usually converted to use ANSI and ISO standard CLI. This conversion is relatively straightforward and the flow of execution is the same, but the OCI call names differ from the CLI function names. To assist developers in the conversion, the most commonly used OCI calls are mapped to CLI in a table that is provided in **Appendix C**.

Embedded SQL

When porting Oracle applications with embedded SQL, it is helpful to use the flagging options on the Oracle precompiler to identify ANSI extensions (that is, proprietary features). It is recommended that the existing Oracle embedded SQL applications are precompiled with the following options (**FIPS=YES**; **MODE=ANSI**; **DBMS=V7**; and **ORACA=NO**) to facilitate migration to DB2.

The syntax for static embedded SQL (in addition to the language element differences previously discussed) is almost identical between DB2 and Oracle. Data is passed between the embedded SQL and the host language through host variables. All DB2 embedded SQL, except SQLJ, must start with the phrase **EXEC SQL**. DB2 supports C, C++, COBOL, RPG and Java as host languages for embedded static SQL. It is also important to remember that the DB2 procedural SQL statements cannot be embedded in a host language (only in the body of a stored procedure, function, or trigger).

After compilation, DB2 SQL programs do not need to be bound. DB2 automatically rebinds the access plans for any SQL statement that is impacted by a server environment change or a change in the table statistics.

The syntax and methodology for embedded dynamic SQL (**Prepare**, **Execute** or **Execute Immediate**) is also very similar. The fundamental difference when embedding dynamic SQL between corresponding Oracle and DB2 applications often is just the variation in the SQLDA structure.

Oracle Forms

Oracle Forms provide a GUI that is commonly used in Oracle applications, but there is no equivalent in DB2. It is best to rework the application interface to use a Java framework. The following companies provide conversion tools that can help with the conversion to a Java-based solution: Quintessence, CipherSoft, Inc, Kumaran and InformatikAteleier.

Cursors

Cursors in embedded SQL are very similar in Oracle and DB2, as demonstrated in Table 5:

Oracle embedded SQL cursor	DB2 embedded SQL cursor
EXEC SQL DECLARE cursor_name CURSOR FOR SELECT col1 FROM table WHERE col2='123';	EXEC SQL DECLARE cursor_name CURSOR FOR SELECT col1 FROM table WHERE col2='123';
EXEC SQL OPEN cursor_name	EXEC SQL OPEN cursor_name;
EXEC SQL FETCH cursor_name INTO :host_var1;	EXEC SQL FETCH cursor_name INTO :host_var1;
EXEC SQL CLOSE cursor_name;	EXEC SQL CLOSE cursor_name;

Table 5: Using cursors in embedded SQL

Oracle includes the concept of cursor attributes, which can be mapped to DB2 return codes. Another option is using the **GET DIAGNOSTICS** statement. DB2 does not support the **%ROWCOUNT** attribute. The application must have its own explicit indexing or use other methods, such as **FETCH FIRST N ROWS ONLY** to control the number of rows returned. (See Table 6.)

Oracle cursor attribute	DB2 mapping	Description

%ISOPEN	EXEC SQL OPEN CURSOR cursor_name;
	if (SQLCODE == -502) /* already open */
	OR
	EXEC SQL FETCH cursor_name INTO var1;
	if (SQLCODE == -501) /* not open */
		Boolean attribute. Returns TRUE if the cursor is already open, but requires the (extra) OPEN to be issued. Or, you can assume the cursor is open, and deal with the error if it is not.

%NOTFOUND	if (SQLCODE == 100)	Boolean attribute. Returns TRUE if the fetch did not return a row.
%FOUND	if (SQLCODE == 0)	Boolean attribute. Returns TRUE if the fetch returned a row.
%ROWCOUNT	Use a counter variable or FETCH FIRST N ROWS ONLY	Numeric attribute. Returns the number of rows returned so far.

Table 6: Oracle-to-DB2 cursor mapping

DB2 and Oracle handle cursor exit conditions identically, except for the variable and **#DEFINE** name differences highlighted in Table 7:

Oracle embedded SQL return codes	DB2 SQL return codes

status = SUCCESS;
while (status == SUCCESS)
{
EXEC SQL FETCH cursor_name
INTO :var1;
status = ORC_CODE;
if (status == SUCCESS)
{ ... }
else if (status==NO_DATA_FOUND)
{EXEC SQL CLOSE cursor_name}
}
	status = SUCCESS;
	while (status == SUCCESS)
	{
	EXEC SQL FETCH cursor_name
	INTO :var1;
	status = SQLCODE;
	if (status == SUCCESS)
	{....}
	else if (status == SQL_NO_DATA_FOUND)
	{ EXEC SQL CLOSE cursor_name}
	}

Table 7: Oracle and DB2 return codes

DB2 supports scrollable cursors through all of its application interfaces (embedded SQL, ODBC and other similar interfaces). It allows the cursor to scroll forward and backward.

Blocked inserts, fetches and updates

Blocked inserts and fetched with an array are supported on all of the DB2 application interfaces (embedded SQL, CLI and other similar interfaces). Blocked updates are not yet supported.

DB2 blocked inserts can be performed with **row-wise** or **column-wise** binding. In addition, a **? ROWS** clause is required by some interfaces when coding blocked inserts. Blocked insert coding examples can be found at the following URL: **ibm.com**/systems/power/software/i/db2/support/code/bi_odbc.html.

Error handling

Part of porting from Oracle to DB2 involves mapping the error conditions. The mapping is not direct; therefore, each application needs to map out the conversion from Oracle to DB2.

Oracle supports user-defined errors and an application can raise exceptions. DB2 embedded SQL does not support user-defined errors in the SQL error handling structure. However, DB2 SQL procedures, triggers and functions can define a user-defined **SQLSTATE** value, then use the **SIGNAL** statement to indicate to the application that the condition has occurred.

DB2 exceptions are communicated through the **SQLSTATE** and **SQLCODE** values. The **SQLSTATE** value is a more platform-independent diagnostic vehicle and must be used when possible. The DB2 GET **DIAGNOSTIC** statement can also be considered.

The ***EXEC SQL WHENEVER*** clause is used to indicate what action to take for each of the three supported conditions: **SQLERROR**, **SQLWARNING** and **NOT FOUND**. The syntax is the same as for Oracle embedded SQL. To map the handling of specific Oracle errors in DB2, use the **WHENEVER SQLERROR** condition to pass control to an error routine that checks for and acts on specific errors.

Concurrency and recovery

One of the differences users notice when they port from Oracle to DB2 is the difference in concurrency control between the two databases. This section addresses the locking behaviors of each database and explains how to map from Oracle to DB2.

When ported to DB2, some Oracle applications appear to behave identically and the topic of concurrency can be ignored. However, if the applications involve frequent accesses to the same tables, the applications might behave differently.

To get the best results, it can be worth redesigning applications to achieve the best concurrency in DB2. After you understand how concurrency control in DB2 works, it is obvious how to rework the application.

Concurrency and locks

For both Oracle and DB2, a transaction is an atomic unit of work, in which all changes are either committed or rolled back. Both Oracle and DB2 also support savepoints. Although both DB2 and Oracle have row-level locking as part of the transaction processing, there are differences in when the locks are acquired.

DB2 typically only acquires row-level locks to ensure integrity. Generally, there are exclusive *(*update) locks and share (read) locks. To update a row, the database server acquires an exclusive lock on that row first. When a share lock is acquired for a table on behalf of an application, other applications can also acquire a share lock on the table, but requests for exclusive locks are denied. However, an exclusive lock blocks other applications from acquiring locks, even share locks, on the table object.

In Oracle, when an application requests a row to be fetched, no locks are acquired. In DB2, when an application issues a read request, DB2 attempts to acquire a share lock on the row. Thus, a DB2 application might acquire more share locks based on the isolation level of the application and the access plan of the query. For example, for a table scan, a share lock can be acquired for each row touched by the table scan, which might contain more rows than the result set. These share row-level locks obtained by DB2 guarantee that the data an application reads does not change under it. The

application can trust the data it fetched. This behavior can simplify the application design. On the other hand, because DB2 requests an exclusive lock on behalf of the application during an update, no other applications can read the row. This can reduce concurrency in the system if many applications are attempting to access the same data at the same time.

Because of these differences, ported applications that have updaters and readers accessing the same data concurrently from the same table can experience slightly longer wait times, but with a more consistent view of the data.

DB2 resolves deadlocks automatically, without any user intervention. A more detailed explanation of the locking employed by DB2 can be found in the **SQL Programming Concepts Guide**.

Isolation levels

Prior to looking at how to improve the concurrency of ported applications, it is useful to have a quick description of differences between the DB2 and Oracle implementation of concurrency control.

Oracle implements an optimistic view of locking. The Oracle assumption is that, in most cases, the data fetched by an application is unlikely to be changed by another application. It is up to the application to handle the situation in which the data is modified by another concurrent application.

For example, when an Oracle application starts an update transaction, the old data version is kept in the rollback segment. When another application makes a read request for the data, it gets the version from the rollback segment. After the update transaction commits, the rollback segment version is erased and all other applications see the new version of the data. Different readers of the data can hold a different value for the same row, depending on whether the data is fetched before or after the update commits. Hence, it is also called the *Oracle versioning technique*. Applications dependent on this behavior can use the USE CURRENTLY COMMITTED concurrent-access resolution support in DB2 to mimic this behavior. To ensure read consistency in Oracle, the application must issue a **SELECT FOR UPDATE** condition, which blocks all other readers and updaters.

DB2 has a suite of concurrency control schemes to suit the needs of applications. An application can select the level of isolation to provide the proper level of concurrency. Here is a brief description of each isolation level. For more details, consult the **DB2 SQL Reference Guide**.

The default isolation level used is interface-dependent; therefore, it is best for the application to set the isolation level explicitly. The isolation level is specified as an attribute of an SQL program or SQL package and applies to the activation groups that use the SQL package or SQL program. DB2 provides several ways to specify the isolation level:

- Use the **COMMIT** parameter on the **CRTSQLxxx** and **RUNSQLSTM** commands to specify the default isolation level.
- Use the **SET OPTION** statement to specify the default isolation level within the source of a module, program, SQL procedure or SQL function that contains embedded SQL.
- Use the **SET TRANSACTION** statement to override the default isolation level within a unit of work temporarily. When the unit of work ends, the isolation level returns to the value it had at the beginning of the unit of work.
- Use the isolation clause on the **SELECT**, **SELECT INTO**, **INSERT**, **UPDATE**, **DELETE,** **PREPARE** and **DECLARE CURSOR** statements to override the default isolation level temporarily for a specific statement or cursor.

- Use the connection attributes or data source settings for ODBC, JDBC and CLI applications.

DB2 supports five isolation levels. In the following list, the DB2 isolation levels are highlighted in bold and the ANSI and ISO term for that level is in italics.

- **Repeatable Read/Serializable (RR):** This is the highest level of isolation. It blocks other applications from changing data that has been read in the RR transaction until the transaction commits or rolls back. If you fetch from a cursor twice in the same transaction, you are guaranteed to get the same answer set. This level is supported through the acquisition of either an exclusive lock or shared-no update lock on the table containing rows that are read or updated; therefore, it is not recommended that you use this isolation level with tables or applications that need to support a high degree of concurrent access.
- **Read Stability/Repeatable Read (RS):** As with RR, this level of isolation guarantees that the rows read by the application remain unchanged in a transaction. However, it does not prevent the introduction of phantom rows.
- **Cursor Stability/Read Committed (CS):** This level guarantees only that a row of a table cannot be changed by another application when your cursor is positioned on that row. This means the application can trust the data it reads by fetching from the cursor and updating it.
- **Uncommitted Read/***Read Uncommitted* **(UR):** This level is commonly known as dirty read. When a UR transaction issues a read, it reads the only version of the data that DB2 has, even through the data might have been read, inserted, or updated by another transaction. The data is labeled as dirty because, if the updater rolls back, the **UR** transaction reads data that has not yet been committed. Unlike Oracle, DB2 does not use the rollback segment to store the old version of the data. Access to the data is controlled using locks.
- **No Commit (NC):** For all operations, the rules of the **UR** level apply except that commit and rollback operations have no effect on SQL statements. Any changes are effectively committed at the end of each successful change operation.

The Oracle implementation most resembles the Read Stability (**RS**) level of isolation in DB2 for writers and most resembles the Uncommitted Read (**UR**) level of isolation for readers. Note that DB2 Uncommitted Read (**UR**) level is not the same as the Oracle versioning. The **UR** level of isolation reads uncommitted data if there is a transaction in progress, as opposed to the last version of the data read by Oracle. Suppose an application reads a column from a row that has been modified by another transaction from a value of **3** to a value of **5**. In Oracle, the application reads **3** and the DB2 **UR** level reads **5**. If the update transaction commits, then DB2 has the right data. If the update transaction rolls back, then Oracle has the right data.

As mentioned, applications running against DB2 can attempt to mimic the Oracle reader behavior with the USE CURRENTLY COMMITTED support. Transactions using the CS isolation level can specify a USE CURRENTLY COMMITTED behavior at a system, connection or statement level. When a reader encounters a row locked for update or delete by another transaction, the CURRENTLY COMMITTED concurrent access resolution behavior causes DB2 to attempt to find a previously committed version of the row for the reader to use instead of waiting on the row-level lock. DB2 scans the journal receiver associated with the table to find a previously committed version of the row. If a previously committed version of the row is not found in the journal receiver, the reader must wait for the update lock to be released. Rows in the process of being inserted by other transactions are skipped when the CURRENTLY COMMITTED resolution is employed. ,

DB2 also supports the SKIP LOCKED DATA concurrent-access resolution behavior to give developers another method to increase the concurrency of their applications. When the SKIP LOCKED DATA clause is specified, then DB2 skips over the candidate row instead of waiting for the share-row lock to be acquired. The skipped row is not included in the final result set. The SKIP LOCKED DATA support is available for isolation levels except for RR.

To increase the concurrency of the system, commit your transactions often, including read-only transactions. For the best overall performance, use the lower isolation level settings whenever possible. However, it is not recommended that you frequently change the isolation level in an effort to improve performance and concurrency. One recommended performance technique is to use the No Commit (NC) or Uncommitted Read (UR) level of isolation on lookup or work tables where concurrency and recovery are not an issue because of their static nature.

AutoCommit

As with the isolation levels, the **AutoCommit** default setting is dependent on the application interface.

Optimistic locking

One locking situation that requires special treatment in DB2 is when an application reads a row from a table, presents data from the row to a user, waits for the user to change values, then updates the row. If two or more users process the same row at the same time, either of the following is possible:

- Both users can read the row in share mode (if the row is not read **FOR UPDATE**), and a deadlock results when either tries to update it.
- One user's read waits until the other user commits (if the row is read **FOR UPDATE**). If this type of application must have minimal lock-wait time and the same locking behavior when ported from Oracle, it can use the optimistic-locking scheme.

This technique requires the table to have a column that can identify when the row's last change took place. A timestamp column is the most typical example. Read the row and commit right away to release the lock. This allows other applications to have write access to the row (as in Oracle). When it is time to update the row, the application can retrieve the row again to see if the row has changed. If the timestamp column has not changed since the last fetch, the application can update the row. If the timestamp has changed, which is relatively rare, the application must start the process again by rereading the row and presenting the latest version of its contents to the user.

Optimistic locking pseudo-code example.

```
Fetch row, including the timestamp column
commit; (release the lock)

... think time when the user makes changes ...

[start transaction]
fetch the row again and compare its timestamp with the timestamp in the previous fetch
if timestamps match
        do the work; (update the row, including the timestamp column)
else
        handle the exception situation; (usually, go back to the start)
commit;
```

Journaling and recovery

All RDBMSs require that changes to the database are logged to perform database recovery. However, the implementations for DB2 and Oracle are quite different. Oracle uses redo logs and rollback segments for database recovery. Redo logs record the transaction changes and rollback segments are used to store the previous version of the data in a table when the table is being updated or mutated.

DB2 implements write-ahead journaling (logging), in which the changed data is always written to the journal before the change is committed. DB2 logs all changes in its journal and journal receiver objects, including the old version of the data. It does not have rollback segments.

DB2 journals can be used to recover changes made to a specific table, a specific schema, or all of the objects involved in the logged transaction. Database objects involved in a transaction can be logged to a different journal. However, to facilitate easy recovery, it is recommended that all of the objects in a transaction be logged to the same journal.

The system **ENDJRNPF** CL command can be used to turn off journaling for database objects that do not need recovery capabilities. Turning off journaling does improve performance, but it is at the expense of transaction recovery.

More detailed information on DB2 journaling and recovery can be found in the **SQL Programming Concepts and Backup and Recovery** guides.

After the port

As with any software project, database porting is not complete merely because all the code has been converted and compiled successfully. Thorough testing and performance tuning need to be done after the database has been moved over to DB2 for i.

Functional testing

Functional testing needs to be performed to make sure that objects and requests on the source and target are returning equivalent results. Even when the SQL syntax is exactly the same, the semantics and run behavior of an SQL statement can be different.

This functional testing phase needs to include error-recovery testing. When comparing two DBMS products, there are often subtle differences in the resolution of lock conflicts, the timing of when error conditions arise, and the value of the error condition returned to the application.

IBM makes IBM Power Systems hardware available for IBM i functional testing to developers and integrators through its Power Development Platform (**ibm.com**/partnerworld/pdp). Testing engagements can also be performed remotely or onsite at one of the IBM Innovation Centers for those companies that are members of IBM PartnerWorld®. Refer to **ibm.com**/isv/iic for more details.

Performance tuning and sizing

Many times, the biggest hurdle to overcome in database projects is the tuning of the application with the new database. Each database engine has its strength and weaknesses from a performance point of view, especially when comparing query optimizers. The Oracle query optimizer might work great with a specific

set of indexes over the database; at the same time, the DB2 optimizer might require additional indexes to be added to that base set of indexes. The opposite holds true, as well. The DB2 cost-based query optimizer does not support hints or require the collection of database statistics (that is, Oracle Analyze). The database manager automatically maintains and updates statistics (inside the table and index objects) as changes are made to the objects. The optimizer also automatically collects stand-alone column statistics for some SQL requests.

A common problem found in the design of applications ported from other databases is that the application programs blindly prepare and run the same SQL statement repeatedly within a single program call, even though it is more efficient to prepare that SQL statement one time. Or, the application uses the ODBC **SQLExecDirect** function or the JDBC statement class to run the same SQL request multiple times within a connection. DB2 typically does not perform well with either of these inefficient program models.

DB2 includes several database performance tools, including System i Navigator SQL Performance Monitor and Visual Explain tools (see descriptions of these tools in the **Administration** section) of this paper. However, in general, to use these tools effectively and to be efficient at tuning DB2, you must attend the **DB2 for i SQL Performance Workshop** (information is found at **ibm.com**/systems/power/software/i/db2/education/performance.html). Some performance tuning information can also be found in the **Database Performance and Query Optimization** book.

A key factor in analyzing and tuning the performance of your application is making sure that you are using a IBM i machine that is properly sized and configured for your application (in terms of processor, memory, and the number of disk arms). To be successful with your solution in the marketplace, you need to know the minimum server configuration required to deliver acceptable performance.

The IBM i Performance and Scalability Center is a great resource for helping performance test and size the hardware required by your application. Visit the following web site for more details: **ibm.com**/systems/ services/labservices/psscontact.html.

For general IBM i performance information, see: **ibm.com**/systems/i/solutions/perfmgmt. This online resource does include information on sizing tools, but none of the sizing tools are specifically designed for DB2 for i. An engagement with the Performance and Scalability Center is the safest and most accurate approach.

Administrative tasks

As discussed in the **Architecture** section of this paper, many low-level database administration tasks are automated on DB2. For example, no low-level performance tuning tasks need to be performed on the IBM i platform. An administrator does not worry about striping tables across disk drives or configuring bufferpools and caches. Thus, the database-administration tasks and tools are quite different.

System i Navigator

System i Navigator (formerly iSeries Navigator) is the DB2 GUI for database administrators and application programmers. It is the rough equivalent to the Enterprise Manager in Oracle.

From the System i Navigator interface, almost all database administrative duties (from performance tuning to journal management) can be performed. Here is a quick overview of some of the System i Navigator capabilities:

- **SQL Script Center:** Develop and run SQL scripts.
- **SQL Performance Monitors:** Collect and analyze database performance monitor data for query optimization issues and performance of specific SQL requests.
- **Visual Explain:** Analyze a graphical representation of the access plan generated by query optimizer.
- **SQL Plan Cache:** Provide a live analysis of the access plans for the most current and frequently run SQL requests.
- **Journal Management:** Start and end logging and swapping of journal receivers.
- **Database Navigator:** Performa basic graphical modeling of existing database definitions.

The SQL*Plus utility provides Oracle command-line access to the database server. DB2 does not have an analogous tool. Instead, it is recommended that you use the System i Navigator SQL Script Center to perform some of the scripting and use the IBM i job scheduler for any scheduling activities. The SQL Script center does not have any support for variables in the script. The db2 **QSHELL** command does provide some variable support The **STRSQL** command can also be loaded, but it does not have a graphical interface.

In addition to the IBM performance tools, more-advanced DB2 performance tuning and management tools are available from Centerfield Technology® (www.insuresql.com).

Utilities

There are several utilities that can help with the porting process from Oracle to DB2 for i.

Load and import

DB2 imports are done with the **CPYFRMIMPF** CL command, which can import files containing delimited data or fixed-format data. The command does not FTP the import file or create the target table. In addition, it assumes that the target table is created and that the specified input file does not contain any column-definition information.

Here is an example of the utility being used to load a stream-based flat file where the column data is delimited with a comma:

```
CPYFRMIMPF FROMSTMF('~mydir/myimport.txt') TOFILE(MYLIB/MYTABLE) DTAFMT(*DLM) FLDDLM(',')
```

If the database parallelism feature (DB2 SMP) has been installed and activated, the database manager breaks the input file into logical partitions and load those partitions into the target table in parallel.

The DB2 Migration Toolkit uses the **CPYTOIMPF** CL command for its data movement from an Oracle database to DB2 for i.

Importing Oracle **DATE** values can be a challenge because the Oracle and DB2 default formats are not the same. It is possible to use the Oracle **TO_CHAR** function to get the value into a format supported by DB2 when writing data into the Oracle export file. Data exports to delimited and fixed format files can be performed with the **CPYTOIMPF** CL command.

ALTER TABLE

Although DB2 and Oracle support an **ALTER TABLE** statement, there are differences in what can be changed. Table 8 gives a quick summary of these differences. (**NOTE:** Not all of the Oracle and DB2 **ALTER** capabilities are shown.)

Function	Oracle	DB2
Add column	yes	yes
Drop column	yes	yes
Increase column size	yes	yes
Decrease column size	yes	yes
Change column type	yes	yes
Add NOT NULL constraints	yes	yes
Add/drop constraints	yes	yes
Enable/disable constraints	yes	no, see next section

Table 8: ALTER TABLE functional differences

Constraint administration

The primary consideration for DB2 is always to ensure full data integrity. However, in certain situations, preserving data integrity on a continuous basis can have a significant performance impact. For example, bulk loading of data via the **Load** command is slower if, for every row, a detour is taken to ensure that all of the pertinent constraints are satisfied for that row.

The mechanism that DB2 uses to balance integrity and performance is to place a table in the **CHECK PENDING** state when an operation is performed that can lead to integrity exposures (for example, during a **Load** task). When the table is in that state, only limited operations can be performed against it (no **Select** operations, for example). To make the table fully accessible again (to take it out of the **CHECK PENDING** state), DB2 must verify that all constraints are satisfied.

The **CHGPFCST** CL command is used both to put a table into the **CHECK PENDING** state and to take it out again after an operation is performed. The **CHGPFCST** command is the DB2 equivalent of the Oracle **set constraint <name> deferred/immediate** statement.

Summary

The purpose of this document is to highlight potential issues in porting databases and applications from Oracle to DB2. This is a living document. As there is more experience with different porting situations, the document will grow accordingly. If the reader encounters an issue that needs to be addressed but which is not covered in this document, contact the owner, Kent Milligan, at rchudb@us.ibm.com. Future readers will benefit from the new information. Thank you.

NOTE: The DB2 for i 7.1 version of this paper can be obtained by sending an e-mail request to: rchudb@us.ibm.com.

Appendix A: Rough sizing estimates for conversions

This appendix includes some rough conversion costs to help estimate (size) how long it might take to convert an Oracle-based application to the IBM i platform. These estimates are provided AS IS without warranty of any kind. The amount of time to perform the conversion varies widely based on the experience level of the programmer and the complexity of the SQL.

- **Join syntax changes:** Five minutes to one hour for query
- **MINUS**: Ten minutes to two hours for query depending on complexity
- **NVL:** Five minutes for each query
- **DECODE:** Five minutes to one hour for each query
- **Date/Time arithmetic changes:** Five minutes to two hours
- **Stored procedure Result Sets in Parameter list:** One day
- **Triggers and stored procedures:** Difficult to estimate, depends on the complexity of the procedure and trigger logic

These estimates do not take into account the usage of the DB2 Migration Toolkit.

Appendix B: PUT_LINE

This appendix covers the implementation details of a solution for simulating the Oracle **PUT_LINE** support. The application source code (**put_line.c** and **crtfputlin.sql**) can be downloaded at:
ibm.com/partnerworld/wps/servlet/ContentHandler/pw_com_db2_oraclecode
The following procedure outlines the steps required to install the DB2 for i version of the **put_line** procedure:

1. FTP **put_line.c** source to a source physical file on your IBM i hardware, for example:

   ```
   put put_line.c /qsys.lib/mylib.lib/qcsrc.file/put_line.mbr
   ```

2. FTP **crtfputlin.sql** source to a source physical file on your IBM i hardware:

   ```
   put crtfputlin.sql /qsys.lib/mylib.lib/qsqlsrc.file/crtfputlin.mbr
   ```

3. Compile the C source using the following CL command:

   ```
   CRTCMOD MODULE(MYLIB/PUT_LINE) SRCFILE(MYLIB/QCSRC) DEFINE('os400')
   SYSIFCOPT(*IFSIO)
   ```

4. Create an IBM i service program (a dynamic library) using the following CL command:

   ```
   CRTSRVPGM SRVPGM(MYLIB/PUT_LINE) EXPORT(*ALL)
   ```

5. Register the UDFs using the **Run SQL Statements** utility:

   ```
   RUNSQLSTM SRCFILE(MYLIB/QSQLSRC) SRCMBR(CRTFPUTLIN) COMMIT(*NONE) NAMING(*SQL)
   ```

Note: Because of the ***SQL** naming convention, the functions are registered in a schema that has the same name as the user profile that runs the **RUNSQLSTM** command.

After successful compilation and registration, you can use the functions from any SQL interface. Here is an example:

```
select put_line('Current user profile : ' || USER || ' Current Path : ' || CURRENT
PATH || '  current schema : ' || CURRENT SCHEMA)
from sysibm.sysdummy1;
```

On the first run, the statement creates a stream file in the /tmp directory named **sqlproc_debug_file.*pid*.txt** where **pid** is the current process ID.

Appendix C: Mapping of Oracle OCI to DB2 CLI

The purpose of this appendix is to give a mapping of the most important Oracle OCI (Oracle Call Interface) calls to the closest DB2 CLI (call level interface) equivalents (Tables 9-14). Refer to the DB2 SQL Call Level Interface Guide and the online FAQs (**ibm.com**/eserver/iseries/db2/clifaq.htm) for details on the CLI calls. Numbers in parentheses refer to the corresponding notes that follow the tables. **N/A** means that the OCI call has no equivalent in CLI.

Connect / initialize / authorize	
OCIInitialize	N/A
OCIEnvInit	SQLAllocHandle (1)
OCIServerAttach	SQLConnect (2), (3)
OCIServerDetach	SQLDisconnect
OCISessionBegin	N/A (3)
OCISessionEnd	N/A
OCILogon	SQLConnect (2)
OCILogoff	SQLDisconnect

Table 9: Connect, initialize and authorize mappings

Handles / descriptors	
OCIHandleAlloc	SQLAllocHandle (1)
OCIHandleFree	SQLFreeHandle
OCIAttrGet	SQLGet__Attr (4)
OCIParamGet	N/A (5)
OCIParamSet	N/A (6)
OCIAttrSet	SQLSet__Attr (7)
OCIDescriptorAlloc	SQLSetStmtAttr (5)
OCIDescriptorFree	SQLFreeHandle (8)

Table 10: Handle and descriptor mappings

Bind / define / describe	
OCIBindDynamic	SQL__Data (11)
OCIBindByName	SQLBindParameter
OCIBindByPos	SQLBindParameter
OCIBindObject	N/A (12)
OCIBindArrayOfStruct	SQLBindParameter (13)
OCIStmtGetBindInfo	N/A (14)
OCIDefineArrayOfStruct	N/A (13)
OCIDefineDynamic	N/A
OCIDefineByPos	SQLBindCol (15)
OCIDefineObject	N/A (12)
OCIDescribeAny	(many) (16)

Table 11: Bind, define and describe mappings

Prepare / execute / fetch	
OCIStmtPrepare	SQLPrepare
OCIStmtExecute	SQLExecute
OCIStmtFetch	SQLFetch (17)

Table 12: Prepare, run and fetch mappings

Transaction management	
OCITransCommit	SQLEndTran (9)
OCITransDetach	N/A
OCITransRollback	SQLEndTran (9)
OCITransStart	N/A
OCITransPrepare	N/A (10)
OCITransForget	N/A (10)

Table 13:Transaction management mappings

Miscellaneous	
OCIBreak	SQLCancel
OCIServerVersion	SQLGetInfo (18)
OCIPasswordChange	N/A
OCIErrorGet	SQLGetDiagRec
OCIStmtGetPieceInfo	SQLGetData (11)
OCIStmtSetPieceInfo	SQL__Data (11)
OCILdaToSvcCtx	N/A
OCISvcCtxToLda	N/A

Table 14: Miscellaneous mappings

Notes for Tables 9-14:

- **SQLAllocHandle** is passed the desired handle type: environment, connection, statement, or descriptor.
- **SQLDriverConnect** is an alternative to **SQLConnect**, providing additional parameters.
- **OCISessionBegin** has no CLI equivalent. To establish multiple database connections in CLI, the server mode must first be enabled and then multiple connection handles must be allocated. **OCIServerAttach** calls are replaced by **SQLConnect** calls. More information can be found on enabling DB2 server mode at: **ibm.com**/systems/i/db2/clifaq.htm
- **OCIAttrGet** can be replaced by **SQLGetConnectAttr**, **SQLGetEnvAttr**, or **SQLGetStmtAttr**, depending on the type of handle that the attribute value is wanted for.
- **SQLSetStmtAttr** must be called with an Attribute value of **SQL_ATTR_APP_PARAM_DESC** or **SQL_ATTR_APP_ROW_DESC**. However, descriptors can be allocated implicitly instead. The function of **OCIParamGet** is performed by **SQLSetStmtAttr** (or implicitly).
- CLI does not have complex object retrieval (COR) descriptors or handles.
- **OCIAttrSet** can be replaced by **SQLSetConnectAttr**, **SQLSetEnvAttr**, or **SQLSetStmtAttr**, depending on the type of handle for which an attribute value is to be set.
- **SQLFreeHandle** must be called with a HandleType of **SQL_HANDLE_DESC** (or the connection can be freed).

- **SQLEndTran** does either a **Commit** or a **Rollback**, depending on the completion type parameter value.
- There are no CLI calls specifically for two-phase **Commit**, but it is supported.
- For piecewise operations, there is not a direct replacement for **OCIBindDynamic**, but for inserts, **SQLParamData** and **SQLPutData** must be called, and for selects, **SQLGetData** must be called.
- CLI does not have any special support for user-defined types. CLI treats one as it does the underlying built-in type.
- To use array inserts, **SQLBindParameter** must be called. **SQLParamOptions** need to be used to set the number of rows in a array. **OCIDefineArrayOfStruct** calls can be ignored.
- There is no direct equivalent, but **SQLDescribeParam** is the closest.
- To use array fetches, **SQLBindCol** must be called. In addition, calls to **SQLSetStmtAttr** are needed to set the following attributes: **SQL_ATTR_ROWSET_SIZE** (array size) and **SQL_ATTR_BIND_TYPE**
- An **OCIDescribeAny** call needs to be replaced by the appropriate call from among **SQLColAttribute**, **SQLColumns**, **SQLDescribeCol**, **SQLForeignKeys**, **SQLGetFunctions**, **SQLPrimaryKeys**, **SQLProcedures** and other similar calls.
- **SQLFetch** fetches a single row. Use **SQLExtendedFetch** or **SQLFetchScroll** to return a rowset; the simplest type of usage is a basic array fetch.
- **SQLGetInfo** provides much more than the server version. One call is needed per type of information wanted. However, **SQLGetInfo** calls should be made sparingly.

Other notes:

Most OCI calls include an error handle, which does not exist in CLI. When porting OCI calls with an error handle, the error handle parameter must be removed from the call, and a check of the return code from the CLI call must be added. When a return code other than **SQL_SUCCESS** occurs, a call to **SQLGetDiagRec** must be made.

The following classes of OCI functions do not have equivalents in DB2 CLI. The function must be implemented either in SQL or in C (or C++) directly.

- Navigational functions: **OCIObject**__ and **OCICache**__
- Data type mapping and manipulation functions: **OCIColl**__, **OCIDate**__, **OCINumber**__, **OCIString**__ and other similar functions. However, CLI performs conversion of data between data types wherever possible.
- External procedure functions: **OCIExtProc**__

Appendix D: Resources

These web sites provide useful references to supplement the information contained in this document:

- IBM i Knowledge Center
  ibm.com/support/knowledgecenter/ssw_ibm_i_72
- IBM i on IBM PartnerWorld®
  ibm.com/partnerworld/i
- DB2 for i online manuals
  ibm.com/systems/power/software/i/db2/docs/books.html
- IBM Redbooks
  ibm.com/redbooks
  - Stored Procedures, Triggers and User-Defined Functions on DB2 for i, SG24-6503
  - Advanced Database Functions and Administration on DB2 for i, SG24-4249-03
  - Diagnosing SQL Performance on DB2 for i, SG24-6654
  - The Ins and Outs of XML and DB2 for i *(SG24-7258)*
  - OnDemand SQL Performance Analysis … in V5R4 (SG24-7326)
  - DB2 for AS/400 Object Relational Support, SG24-5409
- DB2 for i home page
  ibm.com/systems/i/db2
- DB2 for i Performance Workshop
  ibm.com/systems/power/software/i/db2/education/performance.html
- Understanding DB2 for i and the DB2 Family
  ibm.com/partnerworld/wps/servlet/ContentHandler/SROY-6UZDN4
- Improving SQL Procedural Performance white paper
  ibm.com/partnerworld/wps/whitepaper/ibmi/sql_sql_v6r1/procedure
- DB2 online white papers
  ibm.com/partnerworld/wps/whitepaper/i5os
- IBM DB2 Migration Toolkit
  ibm.com/partnerworld/i/db2porting
- Sample UDFs for Migration
  ibm.com/developerworks/db2/library/samples/db2/0205udfs/index.html
- DB2 Connect Unlimited Edition for System i
  ibm.com/software/data/db2/db2connect/edition-uei.html
- Blocked Insert Coding examples
  ibm.com/systems/power/software/i/db2/support/code/bi_odbc.html
- IBM i Benchmark Center
  ibm.com/systems/services/labservices/psscontact.html

- Performance Management for IBM i
  ibm.com/systems/i/advantages/perfmgmt/
- IBM Power Development Platform
  ibm.com/partnerworld/pdp
- IBM Innovation Center
  ibm.com/isv/iic
- **Education Resources (classroom and online)**
  ibm.com/systems/power/software/i/db2/education/index.html**ibm.com**/partnerworld/wps/training/i5os/courses

Online forum

- IBM developerWorks®
  ibm.com/developerworks/forums/forum.jspa?forumID=292

Conversion services

- IBM Systems and Technology Group Lab Services
  ibm.com/systems/services/labservices

Other publications

- SQL for DB2 for i by James Cooper and Paul Conte
  (29th Street Press, ISBN 1-58304-123-0)
- SQL for eServer i5 and iSeries by Kevin Forsythe
  (MC Press, ISBN 1-58347-048-4)
- SQL Built-In Functions and Stored Procedures by Mike Faust
  (MC Press, ISBN 1-58347-054-9)
- Improve Your Productivity with Significant Enhancements in DB2 SQL Procedures
  www.mcpressonline.com/database/db2/improve-your-productivity-with-significant-enhancements-in-db2-sql.html

Other software providers

- Centerfield Technology
  www.insuresql.com
- SwisSQL: Database Migration Solution
  www.swissql.com
- Ispirer: Automated IT Migration
  www.ispirer.com

Trademarks and special notices

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

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.

Linux is a trademark of Linus Torvalds in the United States, other countries or both.

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

Information is provided "AS IS" without warranty of any kind.

Information concerning non-IBM products was obtained from a supplier of these products, published announcement material or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of the specific Statement of Direction.

Some information addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in IBM product announcements. The information is presented here to communicate IBM's current investment and development activities as a good faith effort to help with our customers' future planning.

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.