IBM DB2 for i porting guide

DB2 for Linux, UNIX and Windows (LUW) to the IBM i operating system

.

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

Table of contents

Abstract

In this paper, the reader will learn about the various processes and considerations regarding porting a DB2 for Linux, UNIX and Microsoft (LUW) database application to run under 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 IBM ® DB2 product will be referred to in this paper as DB2 for LUW (Linux™, UNIX®, Microsoft® Windows®). This set of products is also known as DB2 Distributed. This set of products was previously known as DB2 Universal Database.

NOTE: Because this document only focuses on porting from DB2 for LUW to DB2 for i (formerly known as DB2 for i5/OS), this document does **NOT** contain a list of the DB2 for i features that are missing from DB2 for LUW.

The audience for this document

This document is written for application developers and database administrators who want to convert a DB2 for LUW databases and applications to DB2 for i. The document covers the most common issues and inconsistencies encountered by developers porting from DB2 for LUW to DB2 for i, as well as support and administration topics that are relevant to database administrators.

This document concentrates primarily on the database differences between the DB2 for LUW 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. Refer to **Appendix D** for a Web site listing that provides a more general perspective on porting applications to the IBM Power System and IBM i™ (IBM i5™, eServer™ i5, and iSeries™) platforms.

Assumptions

The paper assumes you are working with DB2 10.5 for LUW (although many of the same issues apply to DB2 for z/OS™) and are porting to DB2 for i 7.2 release.

The document also assumes the readers are familiar with the concepts of RDBMS and DB2 SQL. We also assume readers have easy access to DB2 for i 7.2 documentation. Refer to that documentation for detailed information about the actual SQL statement syntax. Refer to the **Resources** section for DB2 for i documentation.

Preparing to port

Before diving into a detailed comparison of the differences between DB2 for i and DB2 for LUW, 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 application will enable you to be more productive as you use this document to assess the port to DB2 for i.

Here are some questions that you need to answer about your application and database:

- Does the application use standard SQL or proprietary features?
- 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?
- Does the business logic reside in procedural database objects (that is, triggers and procedures) or within the application?

DB2 product family

This section addresses why a porting guide is necessary when porting between DB2 products.

The DB2 product line is composed of three members: DB2 for LUW, DB2 for z/OS, and DB2 for i. DB2 for LUW is the single product available across all Linux, UNIX, and Windows platforms. DB2 for z/OS was actually the first available DB2 product. DB2 for i was the last member to join the DB2 product family when it adopted the DB2 branding in 1995.

The fact that there are three products does correctly imply that each member of the DB2 product line has its own unique code base. No single code base runs across all of the platforms and operating systems mentioned previously; hence, the need for this porting guide. Even though each code version is unique and developed by different IBM laboratories, a tremendous amount of technology sharing takes place continually at various levels across the DB2 brand.

To learn more about technology sharing between the DB2 products, you can read a white paper that is entitled, **Understanding DB2 for i and the IBM DB2 Product Family** You can find this white paper at
ibm.com/partnerworld/wps/servlet/ContentHandler/SROY-6UZDN4

NOTE: The **DB2 Family SQL Reference for Cross-Platform Development** is a good complement to this porting guide, and can be found at **ibm.com**/partnerworld/i/db2porting

A high-level matrix showing SQL features that are common across the DB2 products is available at **ibm.com**/developerworks/db2/library/techarticle/db2common/.

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 for i. 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 for i?” 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 for i can accommodate 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 will require more effort and investment, but will better position your application for expansion to other platforms and databases in the future. In addition, as you enhance your core application, this design will make if faster to deploy these enhancements to all of your supported platforms.

If the maintainability of source code that must also remain portable is a top requirement, a good application design is to isolate all of the database run-time calls from the application. This isolation is usually accomplished by having the application create its own set of database methods or calls that are, then, passed 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 database interface) that is 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 will be 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 that are made 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 have a common set of single source code, then platform specific features are avoided in the conversion, but the tradeoff is usually performance when the developer codes to the lowest common denominator. For example, DB2 for LUW has a scalar function called **X** and DB2 for i has an equivalent function called **Y**. The developer converting to DB2 for i has two choices: change the application to call function **Y** (which yields the best performance) or code a DB2 for i user-defined function (UDF) called **X** that just calls function **Y** (This allows the application code to remain unchanged, but results in slower performance).

The design of applications using Dynamic SQL interfaces also should be reviewed and analyzed. Some applications have an inefficient design that blindly prepares and runs the same SQL statement repeatedly within a single program call, even though it would be more efficient to prepare that SQL statement one time. Or, the application will use the ODBC **SQLExecDirect** function or the JDBC **Statement** class to run the same SQL request multiple times within a connection. Time should be allocated for testing and analyzing the performance of these inefficient program models running on DB2 for i. The DB2 for i engine employs caching and other techniques to mitigate the performance impacts of this inefficient model, however, the best SQL performance is usually achieved by using a design paradigm of preparing a unique statement only once and running it many times. This design methodology is the most efficient approach on all database servers.

On the topic of performance, 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**. This course teaches the developer the proper way to design and implement a high-performing DB2 solution. You can find this paper at **ibm.com**/systems/power/software/i/db2/education/performance.html

Porting tools

IBM is not aware of any porting tools that assist in porting databases between DB2 products.

Sizing the port

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

Architecture

This section provides an overview of the DB2 for i architecture and a comparison of that architecture with the DB2 for LUW database product. In addition, the interfaces used with DB2 for LUW and DB2 for i databases are also 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 (DB2 UDB) 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 customers will tell you that they are not using DB2.

Since 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 the native database interface is quite similar to indexed sequential access method (ISAM).

Because of this native, non-SQL interface, some IBM i customers and consultants will use terminology that is not familiar to those coming from an SQL background. Table 1, below, 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 for i is integrated, most SQL-based applications will run on any server that can run IBM i without requiring additional products to be purchased for the client or server. The following DB2 capabilities are included with the operating system at no additional charge:

SQL parser and run-time support
SQL interfaces (server side)
- CLI (call level interface)
- 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)

PHP ibm_db2 extension (Zend Server for IBM i)
Ruby ibm_db2 project (PowerRuby)
Open Group™ DRDA application requester and server

SQL Statement Processors (**RUNSQLSTM** and **RUNSQL** CL commands)
Qshell DB2 utility
Performance tuning tools

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

If your application already supports DB2 access through the middleware provided by DB2 Connect, another option is DB2 Connect Unlimited Edition for System i. This product is not included with the base operating system and must be purchased separately. For more information on this product, see the following Web site: **ibm.com**/software/products/en/db2connunlieditforsysti.

As with the other DB2 products, DB2 for i also enables the building of Web-based applications with some of the IBM WebSphere® suite of products. In addition, some of the IBM InfoSphere products also support DB2 for i.

Many vendors prefer using a native database interface over an industry standard interface, such as ODBC or CLI, because of performance concerns. DB2 for i does not have a lower-level SQL-based interface available; instead, the ODBC, JDBC, and CLI interfaces are treated as the lower-level SQL-based interface. These interfaces provide performance similar to the native interfaces offered by other database products. One word of caution, some programmers use the term **native** to refer to the non-SQL interfaces that exist for DB2 for i.

The one SQL-based interface that is 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 can only effectively by used with a thorough understanding of the DB2 for i SQL engine. That knowledge can be obtained in the **DB2 for i SQL Performance Workshop**. Refer to the **Resources** section of this paper for a Web site listing for this workshop.

Physical model

DB2 for i does support the notion of independent, isolated databases. However, the default configuration for DB2 for i is as a single system-wide database. For example, DRDA access of a DB2 for LUW server requires a database name to be specified on the **CONNECT** statement. With DB2 for i, this database name, by default, is the system name. When connected, the client can access any database object for which they have been authorized. Schemas are the logical containers for related database objects on the IBM i platform. Any database object in the schema can be accessed by any user as long as they have the proper authorization (without a **CONNECT** statement). In addition, database relationships, such as a referential constraint, can easily be defined across objects that are stored in different schemas.

If you want to create independent, isolated databases on the IBM i platform, configure a new independent auxiliary storage group instead of using the **CREATE DATABASE** statement. A

relational database directory entry is then created for each independent auxiliary storage group.

Storage model

The physical storage models for DB2 for LUW and DB2 for i are drastically different. With DB2 for i, all of the 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 for i object is automatically partitioned (or striped) across the disk devices to eliminate contention.

DB2 for i supports partitioned tables assuming that the DB2 Multisystem product has been purchased and installed. Partitioned tables are primarily used on DB2 for i to overcome the size limits for a single table and to improve the performance of bulk data operations such as massive deletes. Before implementing partitioned tables, an analysis of the application and performance requirements should be completed. The *Table partitioning strategies for DB2 for i* white paper contains implementation details and recommendations that need to be considered during the analysis process. This white paper is found on the IBM **i Education Curriculum** Web site at **ibm.com**/partnerworld/wps/whitepaper/i5os.

Metadata

Metadata provides the roadmap to interpret the information stored in the database. All of the DB2 products store metadata in the System Catalog. The catalogs supported by the DB2 products are similar, but there are specific differences for which there needs to be an accounting.

The preferred way to reference metadata in DB2 for i is through the catalog views, such as **SYSIBM.TABLES**, not the underlying tables. Detailed descriptions of the catalog views can be found in an appendix of the **DB2 for i SQL Reference Guide**. DB2 for LUW and DB2 for i both support a common set of catalog views for ODBC and JDBC clients in the **SYSIBM** schema.

Tolerance for unsupported syntax

Often, the SQL database-creation scripts for other DB2 databases contain low-level configuration statements (for example, creating a tablespace) or syntax on the SQL data-definition language (DDL) statements that prevent the scripts from running on DB2 for i until the unsupported syntax is removed from the database scripts. As a result, beginning with the 6.1 release, DB2 for i attempts to ignore statements and syntax that will never be supported by this product. This administrative syntax is not needed because the DB2 for i database engine and operating system automatically handle many low-level database administration tasks that require manual configuration with the other DB2 products.

The toleration enables faster porting, because unnecessary syntax does not have to be removed from database creation scripts. A warning SQLSTATE is returned to flag whenever syntax is ignored on DB2 for i. Here is an example of statements and syntax that are ignored and tolerated by DB2 for i. The tolerated syntax is shown in bold.

CREATE TABLESPACE TS1 MANAGED BY DATABASE USING (device '/dev/rcont $N' 20000)

SQLSTATE: 01505

CREATE TABLE newtab1 (c1 INT) **INDEX IN ts1**

SQLSTATE: 01680

Data types

There are only minor differences between the *basic* data types that are supported by DB2 for LUW and DB2 for i. DB2 for i does not support the abstract data types.

DB2 for LUW supports a **LONG VARCHAR** and **LONG VARGRAPHIC** data type that, in most cases, can be converted to the LONG **VARCHAR** and LONG **VARGRAPHIC** data types that are supported by DB2 for i. The lengths of LONG VARCHAR and LONG VARGRAPHIC columns on DB2 for i are limited by the 32-kilobyte maximum record length. If the length limitations are encountered with the DB2 for i LONG VARCHAR and LONG VARGRAPHIC data types, then you should consider the usage of CLOB or DBCLOB data types. It should be noted that DB2 LUW has deprecated support for the LONG VARCHAR and LONG VARGRAPHIC data type, so support might be removed in the future.

Even though both products support the **FLOAT(n)** definition, you might want to consider converting any **FLOAT(n)** definitions to the **DECFLOAT** data type for maximum portability with other DB2 products.

The **NUMERIC** type is implemented as a zoned decimal type on DB2 for i and packed decimal type by the other DB2 products. Semantically, these are the same and should require no application changes.

Unicode® data is supported with the **GRAPHIC** and **VARGRAPHIC** data types with a CCSID value of 13488 for the UCS-2 encoding or a CCSID value of 1200 for the UTF-16 encoding. The national-character data types (NCHAR, NVARCHAR and NCLOB) can also be used to create a column with the UTF-16 encoding. The UTF-8 Unicode encoding is also available with the **CHAR** and **VARCHAR** data types with a CCSID value of 1208. DB2 for i allows the CCSID value to be specified at a column level; however, DB2 for LUW only allows a CCSID setting to be specified at a database level.

VARCHAR performance considerations

Many DB2 for LUW applications use **VARCHAR** types for almost all character string values. In these circumstances, it is better to port it to the fixed-length DB2 data type **CHAR(n)**, as it is more efficient and takes less storage than the **VARCHAR** data type. A general rule of thumb is to use the **CHAR** data type for columns of 40 bytes or less. Before making a decision 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**, **VARGRAPHIC**, **BLOB**, **CLOB**, and **DBCLOB**) should only be used for long-memo or text-description columns that are referenced infrequently. DB2 variable-length types usually cause additional disk I/O (unless the **ALLOCATE** keyword is used), 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 will allocate **n** bytes in the fixed portion of the row and will only store the column in the overflow area when the column value exceeds **N**. Setting **N** so that almost all of the column values will be 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 could 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; therefore, 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 happens to be in the same overflow storage area as three 2-megabyte **BLOB** values, then the application will wait 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 must be set to the maximum to ensure that the value is always stored in the fixed portion of the row.

Another option is to move the **LOB** column to a different table.

LOB types

LOB columns are always journaled on DB2 for i; conditional logging of **LOB** values, which are offered by DB2 for LUW, are not directly supported. The IBM i journal minimal data option can reduce the overhead of logging **LOB** values by only logging **LOB** values when the data has actually changed. Logging for the DB2 for i table can also be disabled.

Date and timestamp differences

DB2 for LUW and DB2 for i use different internal representations of date and timestamp values, but the internal representation does not require any application-level changes.

XML considerations

Starting with the 7.1 release, DB2 for i provides an XML data type. DB2 for i stores XML values as a character string instead of the parsed hierarchical format used by the pureXML

support in DB2 for LUW. The DB2 for i XML functionality in 7.1 does not include an XQuery search interface that's available on DB2 for LUW XMLEXISTS and XMLTABLE functions. As an alternative, applications can employ leverage the IBM OmniFind Text Search Server to index and search the stored XML values.

For a deeper understanding of the XML support available on prior releases, the following IBM Redbooks® document should be referenced: *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

The IBM i platform supports two types of SQL identifiers: ordinary identifiers and delimited identifiers. 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. DB2 for i does not support double-byte table names.

It is strongly 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 that are using non-English CCSID values. The _ (underscore) character is not considered to be a variant character.

In general, the DB2 for i SQL identifier-length maximums are not an issue when porting databases. DB2 supports identifiers with a length up to 128 characters for columns, tables, indexes, procedures, and constraints. Authorization names are the identifier types with the lowest maximum length at 10 characters. All of the SQL limits are documented in the **DB2 for i SQL Reference**. A listing for this online manual is provided in the **Resources** section of this paper.

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 will automatically generate a short identifier of 10 characters in length 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 DB2 for i support for long and short SQL identifiers is very analogous to the file name restrictions when trying to use Windows and DOS utilities on the a Windows object created with a long name.

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** will have 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 will instead assign 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 in length.

Naming conventions and formats

Almost all of the 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 the traditional, native database access on the IBM i platform. This naming mode is not supported by DB2 for LUW. System naming mode dictates that the traditional OS/400 slash (/) is used when explicitly qualifying SQL objects with a schema name (for example, mylib/mytablename). Furthermore, when a schema is not specified, then system naming will search through the libraries defined in the job's library list looking for the unqualified object.

***SQL** (SQL naming) is based on the SQL standard. SQL naming requires the schema and object name to be separated with a period (**.**) (for example, mylib.mytablename). With the SQL naming convention, the only library (or schema) that is searched for unqualified objects is the current schema; the current schema value usually defaults to the library that matches 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 for i will expect to find that object in a schema named JOHNDOE.

There are other settings that 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. (See the **Resources** section of this paper.)

Functions

The portability of functions between DB2 products is quite high. The DB2 for LUW functions that are not yet supported by DB2 for i can usually be implemented as a user-defined function (UDF).

UDFs also have many similarities on DB2 for LUW and DB2 for i. Both products allow UDFs to be written in SQL, C, COBOL, and Java. DB2 for i supports both user-defined table functions and user-defined scalar functions. DB2 for LUW row functions are not yet supported by DB2 for i, but it might be possible to implement the row function as a table function.

Here is a summary of the DB2 for i UDF support.

Sourced UDF: A sourced UDF is a reuse of the implementation of an existing function, usually with some attributes changed, such as input data types. They are primarily used with user-defined data types.

SQL-bodied UDF: This type of UDF can be fully implemented with the SQL procedural language (PSM).

External UDF: This type of UDF can be used when it is easier to implement the function with a high-level programming language. An external UDF can be written in any high-level programming language supported on the IBM i operating system, including Java.

UDF performance

If performance is more important than maintaining a single code base, then the developer might want to avoid using UDFs for all DB2 servers in a porting exercise, especially when the same function can be implemented inline with a series of calls to IBM i system-supplied functions. Specifying the **NOT FENCED** and **DETERMINISTIC** attributes when creating functions can improve the performance of DB2 for i UDFs. Unlike DB2 for LUW, the **NOT FENCED** attribute will not cause database corruption or database control structures to be damaged.

UDFs and Check Constraints

DB2 for i does not allow a UDF to be referenced on a check constraint definition. This invocation method is supported on DB2 for LUW. One possible circumvention on DB2 for i would be implementing the UDF-based check constraint logic inside of an SQL Trigger.

Oracle Compatibility functions

DB2 for i does not support the all of the Oracle capability functions such as DECODE and NVL that are available on DB2 for LUW.

CASE expression considerations

DB2 for i only supports basic predicate comparisons. If a DB2 for LUW CASE expression contains an **EXISTS** or **IN** predicate, it will need to be converted to use one of the basic predicates. Here is an example of a **CASE** expression that is not yet supported on the DB2 for i:

```
SELECT articleID, (CASE WHEN EXISTS(SELECT * FROM returns)
                   THEN 1 ELSE 0 END) AS second FROM article
```

Here is an example of the converted **CASE** expression that is supported on DB2 for i:

```
SELECT articleID, (CASE WHEN 1 = (SELECT DISTINCT 1 FROM returns)
                   THEN 1 ELSE 0 END) AS second FROM from article
```

Joins

DB2 for i supports a superset of the join operators available on DB2 for LUW.

Data change table reference

DB2 for LUW allows data change table references for INSERT, UPDATE, and DELETE statements. The DB2 for i support only allows data change table references for INSERT statements.

FETCH FIRST n ROWS clause

Although DB2 for i allows the specification of the **FETCH FIRST n ROWS** clause to limit the number of results returned by a **SELECT** statement, it does not yet allow that clause to be included on updateable requests; nor can it be specified on an **UPDATE** statement. For example, the following cursor declaration is not supported on DB2 for i because the **FETCH FIRST** clause is not compatible with the update clause.

```
DECLARE c1 CURSOR for SELECT ordid, ordqty FROM orders
  FETCH FIRST 10 ROWS ONLY
  FOR UPDATE
```

Thus, the **FETCH FIRST** clause would need to be eliminated when porting this type of cursor declaration to DB2 for i. The application would need to employ a counter variable that tracks and limits the number of rows in the result set to best simulate the processing of the **FETCH FIRST** clause.

GROUP BY considerations

When the **GROUP BY** clause contains an expression, the exact same expression is required on the **SELECT** list. For example, the following statement is legal on DB2 for LUW, but not on DB2 for i:

```
SELECT quarter(shipdate),
   CASE WHEN quarter(shipdate)=1 THEN 5 END AS "Q1"
 FROM item
 GROUP BY quarter(shipdate)
```

The statement must be converted as follows for DB2 for i:

```
SELECT quarter(shipdate),
   CASE WHEN quarter(shipdate)=1 THEN 5 END AS "Q1"
 FROM item
 GROUP BY quarter(shipdate),
          CASE WHEN quarter(shipdate)=1 THEN 5 END
```

OLAP specification considerations

The DB2 for i support for OLAP specification is limited compared to the DB2 LUW support. DB2 for I supports the ROW_NUMBER function on numbering specification. In addition, the RANK and DENSE_RANK functions are supported for ordered OLAP specifications. However, there is no support for aggregate function specifications such as the following: AVG(CLOSEPRICE) OVER (PARTITION BY SYMBOL ORDER BY TRADINGDATE).

If the aggregate function window definition only contains a partition clause and not an order clause, then it is possible to rewrite the query. For example, the following statement is supported by DB2 for I, but not DB2 for LUW.

```
  SELECT  ACCTID, CLEAREDBAL,
      SUM(BALANCE) OVER(PARTITION BY ACCTID, CLEAREDBAL),
      AVG(BALANCE) OVER(PARTITION BY ACCTID, CLEAREDBAL),
      ROW_NUMBER() OVER (ORDER BY ACCTID)
    FROM accounts
```

The query can be rewritten as follows for DB2 for i:

```
WITH aggcte(ACCTID, CLEAREDBAL,  SUMBAL, AVGBAL) AS (
  SELECT  ACCTID, CLEAREDBAL,
      SUM(BALANCE)
      AVG(BALANCE)
     FROM accounts
   GROUP BY ACCTID, CLEAREDBAL)
SELECT ACCTID, CLEAREDBAL,
     SUMBAL,
     AVGBAL,
     ROW_NUMBER() OVER (ORDER BY ACCTID)
FROM aggcte
```

Stored procedures

All of the DB2 family members support both external and SQL stored procedures. External stored procedures are essentially programs written in a high-level language, such as C++, and then registered as stored procedures with the **CREATE PROCEDURE** statement. External procedures developed in C, C++, Java, and COBOL are portable across the DB2 Family. There are some operating system specific attributes on the **CREATE PROCEDURE** statement that may need to be altered.

The SQL stored procedure language is supported by all of the DB2 Family and is based on the ANSI/ISO Persistent Stored Module (PSM) standard specification. DB2 for i was the first member to support the SQL procedural language and also makes this language available for the development of SQL UDFs and SQL triggers.

DB2 for i also does not support the DB2 for LUW GET and PUT ROUTINE commands for deployment of SQL procedures. DB2 for i SQL procedures can be deployed in a similar manner by saving the procedure objects off of a development system and then using the IBM i Restore commands to deploy the procedures on the user's system. The IBM Data Studio is a common tool that can be used to deploy stored procedures to both DB2 for i and DB2 for LUW servers.

Triggers

DB2 for i supports SQL triggers, including **Instead Of** triggers, just as DB2 for LUW does. The DB2 for i SQL trigger support is actually a superset of the trigger support that is

available in DB2 for LUW. DB2 for i allows full usage of the SQL procedural language within the body of an SQL trigger allowing for more complex and sophisticated trigger definitions. One incompatibility with the DB2 for LUW trigger support is the DB2 for i inability to pass transition variables as parameters on a stored procedure or function call. For example, the following statement is legal on DB2 for LUW, but not on DB2 for i:

```
CREATE TRIGGER testtrig
  AFTER INSERT ON request
  REFERENCING NEW as new
  FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    CALL process_request(1
                , new.Prod_State
                , new.Rec_Type
                , new.Rec_Vrs)
                );
     END
```

This trigger definition must be converted as follows on DB2 for i to wrapper each of the transition variables with the casting function associated the column data type:

```
CREATE TRIGGER testtrig
  AFTER INSERT ON request
  REFERENCING NEW as new
  FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    CALL process_request(1
                , VARCHAR(new.Prod_State)
                , CHAR( new.Rec_Type)
                , SMALLINT(new.Rec_Vrs)
                );
     END
```

Constraints

The constraint support for DB2 for i and DB2 for LUW is also almost identical. One difference is that DB2 for i supports the **Delete** rule of **Set Default** on referential constraint definitions; this is not supported by DB2 for LUW. DB2 for i constraint names also have to be unique within a schema. Thus, two tables in the same schema cannot use the same constraint name.

For check constraint definitions, there are a different set of restrictions for DB2 for i and DB2 for LUW. For example, a DB2 for i check condition cannot reference a UDF. Consult the documentation for a complete list of the restrictions.

Views

DB2 for i does not yet allow Views containing **Union All** operators to be updateable DB2 for LUW does.

Oracle Compatibility Mode

DB2 for i does not include support for the Oracle Compatibility mode supported by DB2 for LUW. That being stated, DB2 for i has been enhanced over the last two release with several features to improve Oracle compatibility such as the Use Currently Committed clause and

Array support. The DB2 for i porting guide for Oracle databases should be referenced for additional details.

Unsupported features

There are some DB2 for LUW features or capabilities that are not yet supported or are extremely difficult to support or come up with a DB2 for i circumvention. The features listed here might require portions of the application to be rewritten or redesigned:

Generated columns
Regression aggregate functions
Label-Based Access Control
DB2 Trusted Connection
BLU columnar database capability

The following DB2 for LUW features are also not yet supported by DB2 for i. Although these missing performance features will probably not require a change to the application, they might cause performance impacts on the ported application and require additional performance tuning.

DB2 Cube views
INCLUDES and **CLUSTER** on **CREATE INDEX**

DB2 for i supports the creation of Materialized Query Tables (MQTs) with awareness of the MQTs by the query optimizer. There are differences with the DB2 for i query optimizer's awareness and usage of MQTs. One of the biggest differences is that DB2 for i doesn't support system-maintained MQTs. The *Creating and using materialized query table in IBM DB2 for i* white paper contains all the details on the DB2 for i MQT support. This paper can be accessed online at:

ibm.com/partnerworld/wps/servlet/ContentHandler/SROY-6UZ536

Application development

Most DB2 for LUW applications being ported are programmed in either C++ or Java. The primary data access interface in these ported applications is ODBC, CLI, and JDBC.

Java

Each member of the DB2 product family has fully embraced Java by supporting both the JDBC and SQLJ data access interfaces. In addition, each DB2 product also supports Java as a development language for stored procedures and user-defined functions.

Middleware

Many of the ported DB2 applications that support networked clients use the middleware (for example, JDBC and ODBC drivers) provided by DB2 Connect Unlimited Edition for System i. Although DB2 Connect also supports distributed access of DB2 for i, it might be preferable to switch to the middleware included with the IBM i operating system for a couple of reasons. First, the IBM i ODBC and JDBC drivers are customized specifically for the DB2 for i

database, making it easier for the application to exploit its strengths. Second, many customers already have the IBM i ODBC and JDBC drivers installed on their clients and are not going to be enthusiastic about installing another piece of middleware and paying for the license charges. Furthermore, DB2 Connect does not provide any performance or functional benefits over the IBM i ODBC and JDBC drivers.

Because of different interpretations of the ODBC and JDBC standards, there might be some minor differences in the behavior of the IBM i ODBC and JDBC drivers when compared to the DB2 Connect ODBC and JDBC drivers.

The DB2 for i SQL CLI is not identical to DB2 for LUW CLI. Here is a list of the most important differences:

- The **SQLSetStmtOption** function has been deprecated and replaced with **SQLSetStmtAttr. SQLSetSmtOption** is still supported, but IBM recommends using SQLSetStmtAttr for standards compatibility.
- The **SQLSetStmtAttr** function does not support the **SQL_ATTR_CONCURRENCY** argument.
- IBM i pointers cannot be used interchangeably with integers.
- The **SQLGetData** function does not support the return of a timestamp value in the **TIMESTAMP_STRUCT** parameter. Timestamp values are always returned as strings.
- The **SQLSetPos** function is not supported
- Key-set driven cursors are not supported
- The **SQLFetchScroll** and **SQLExtendedFetch** functions require the column values and column indicator values to be copied to two different storage areas. The indicator values cannot be intermixed with the column values.
- Additional details and circumventions (Refer to the **Resources** part of this paper for a link for more information, found in the porting overview.)

Cursors

In general, DB2 for i cursor support is more robust than DB2 for LUW from a function perspective, so cursor processing will be easy to migrate from DB2 for LUW to DB2 for i. DB2 for i supports static and dynamic scrollable cursors with the following fetch options: **First**, **Last**, **Next**, **Prior**, and **Relative**.

Concurrent environments

Both DB2 for LUW and DB2 for i support the same cursor sensitivity settings: **ASENSITIVE**, **SENSITIVE**, and **INSENSITIVE.** The default setting is **ASENSITIVE** for all of the DB2 products. The **ASENSITIVE** usually is the best performing option since it gives the query optimizer the flexibility of choosing the setting. Since the DB2 query optimizers are not identical, there may be times where the optimizers choose a different implementation for the exact same SQL statement. If the behavior is different and the application requires a specific cursor sensitivity behavior, the setting can be changed to **SENSITIVE** or **INSENSITIVE**.

Blocked operations

Blocked inserts and fetches with an array are supported on all of the DB2 for i application interfaces (embedded SQL, CLI, and others.). In contrast, DB2 for LUW only supports blocked operations with CLI (this includes ODBC and JDBC). DB2 for i supports blocked updates, deletes, and merge operations on the IBM i CLI, ODBC, and ADO.NET interfaces.

Case sensitivity

By default, DB2 for LUW performs case-sensitive searches. Some DB2 for LUW applications perform case-insensitive searches by using the **UPPER** scalar function. For example, **`UPPER(statecolumn)='NY'`** finds all occurrences of the state of New York, regardless of the case used in entering the state value (**ny**, **Ny**,and others). The **UPPER** function can be used on DB2 for i in the same manner.

In order for an index to be used by the DB2 for i query optimizer with the **UPPER** function, there are two options available. A derived (functional) SQL index can be created or the application can use the IBM i national-language sort-sequence support.

The derived SQL-index support enables SQL functions, such as **UPPER**, to be included in the key definition on an index, as shown in the following example:

```
CREATE INDEX ix_uStateName ON customers(UPPER(statecolumn))
```

The query optimizer has the ability to use derived indexes for search predicates that reference SQL functions (**`UPPER(statecolumn)='NY' ).`**

With the sort sequence approach, an index must be created with a sort sequence (a translate table that maps lowercase values to uppercase values ; for English, the table is Q037 and is found in the QUSRSYS library).

If this usage of the **UPPER** function is found in the application being ported, then you might want to 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. This sort sequence will cause all character comparisons to be translated into uppercase before comparing the actual character string values. This eliminates the need for the **UPPER** function. 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 when using 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 (**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.

Concurrency and recovery

DB2 for LUW and DB2 for i have implementations for database concurrency and recovery that are very similar.

Concurrency and locks

DB2 for LUW and DB2 for i both acquire row-level locks as part of the transaction processing. These locks ensure the integrity of the transaction and control concurrent access to the data through the usage of read and update row-level locks.

DB2 for LUW might convert row locks to table locks automatically without an explicit user request. DB2 for LUW performs lock escalation by converting many row locks to a table lock if it perceives that there are too many locks held at that time. Table lock escalation is probably done more on DB2 for LUW than it is on DB2 for i. However, there are some situations where DB2 for i will implicitly acquire an exclusive table-level lock.

DB2 for LUW and DB2 for i are enabled to resolve deadlocks automatically, without any user intervention.

A more detailed explanation of the locking employed by DB2 for i can be found in the **SQL Programming Concepts Guide**.

Isolation levels

DB2 for i supports the same four isolation levels that are available with DB2 for LUW. DB2 for i also supports one additional isolation level, **No Commit**. (This level is described later in this section.)

The DB2 for i default isolation level can be interface dependent, so it is best for the application to explicitly set the isolation level. DB2 for i 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 procedure, function, module, or program 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 for a specific statement or cursor temporarily.

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

NOTE: In the list below, the DB2 isolation levels are highlighted in **bold** and the ANS 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 to 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 while 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 were to roll back, the **UR** transaction would have read data that was never committed.

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.

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 **No Commit** or **Uncommitted Read** on lookup or work tables where concurrency and recovery are not an issue because of their static nature.

Concurrent Access Resolution

When a transaction encounters incompatible row locks held by other transactions, the DB2 for i database manage by default waits for the incompatible row lock to be released. If the transaction waits for that lock longer than the lock timeout threshold (default is 60 seconds), the DB2 database manager will return a lock timeout error to the application. This concurrent access resolution behavior is associated with the WAIT FOR OUTCOME clause.

If the application being ported needs a different behavior when incompatible row locks are encountered, DB2 for i also supports the following concurrent access resolution behaviors:

Skip Locked Data: This value causes the transaction to skip over any rows with incompatible locks that it encounters.

Use Currently Commited: This values allows the database manager to use the currently committed version of the data for read-only scans that encounter an incompatible lock on a row being updated or deleted. Rows in the process of being inserted are skipped. DB2 for i attempts to retrieve the currently committed version of the data by scanning the associated journal receiver.

These concurrent access resolution behaviors are not available on all of the isolation levels, please consult the **DB2 for i SQL Reference Guide** for additional details.

AutoCommit

Like the isolation levels, the **AutoCommit** default setting is dependent on the application interface. It is recommended that the ported application be changed to explicitly specify an **AutoCommit** setting.

Journaling and recovery

DB2 for LUW and DB2 for i both implement write-ahead logging (journaling), in which the changed data is always written to the log (journal) before the change is committed. DB2 for i logs changes in its journal and journal receiver objects, including the old version of the data.

DB2 for i journals are more usable than DB2 for LUW logs. DB2 for i 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. In addition, an IBM i application or administrator can actually view and use the data stored in the journal.

However, database objects involved in a transaction can be logged to a different journal. 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 for i journaling and recovery can be found in the **SQL Programming Concepts** and **Backup and Recovery Guides**.

After the port

Just as with any software project, database porting is not complete at the point that all the code has been converted and compiled successfully. Thorough testing and performance tuning need to be performed 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 running 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 does make IBM Power Systems hardware available for IBM i functional testing to developers and integrators through its Power Development Platform. Visit the following Web site for details: **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 Partnerword. 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 DB2 for LUW query optimizer might work great with a specific set of indexes over the database although the DB2 for i optimizer might require additional indexes to be added to that base set of indexes. The opposite holds true as well. The DB2 for i cost-based query optimizer does not require the collection of database statistics. 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 statistics for some SQL requests. For more information on this capability of the SQL Query Engine (SQE), visit the following Web site: **ibm.com**/systems/i/db2/sqe.html.

Each DB2 product features an advanced, cost-based query optimizer. Because each DB2 query optimizer tries to exploit its target system and hardware fully, each database engine has its own unique performance personality. Therefore, it might be necessary to tune applications as they are moved to a different DB2 product. As mentioned earlier, however, there is a large amount of common, patented optimization technology and algorithms across the different query optimizers.

Also as mentioned earlier, no low-level performance-tuning tasks need to be performed on the IBM i platform. An administrator does not need to worry about striping tables across disk drives or configuring bufferpools and caches.

An inefficiency sometimes found in the design of ported applications is that the application programs will blindly prepare and run the same SQL statement repeatedly within a single

program call, even though it would be more efficient to prepare that SQL statement one time. Or, the application will use the ODBC **SQLExecDirect** function or the JDBC **Statement** class to run the same SQL request multiple times within a connection. Performance should be analyzed to ensure that these inefficient program models perform optimally with DB2 for i. The best performance is generally achieved when the application is designed to prepare a single statement once and execute that statement many times.

DB2 for i includes several database performance tools, including the System i Navigator, SQL Plan Cache, SQL Performance Monitor, and Visual Explain tools (see a description 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 for i, you must attend the **DB2 for i SQL Performance Workshop** (**ibm.com**/systems/power/software/i/db2/education/performance.html). Some performance-tuning information can be found in the **Database Performance and Query Optimization** book. Refer to the **Resources** section of this paper for more information.

In addition to the IBM performance tools, more-advanced DB2 for i performance tuning and management tools are available from Centerfield Technology. (A link to this site is found in the **Resources** section of this paper.)

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

The IBM i Performance and Scalability Center is a great resource for helping performance test and size the hardware required by your application. Visit this Web site for more details:

ibm.com/systems/services/labservices/psscontact.html.

You will find information on sizing tools on the following website, but none of the sizing tools are specifically designed for DB2 for i workloads.

ibm.com/systems/support/tools/estimator/index.html

An engagement with the Performance and Scalability Center is the safest and most accurate approach.

Administration

As discussed in the **Architecture** section, administrative controls and tasks are quite different across the DB2 family. Many of the differences in this area are because of the administrative requirements of the underlying the operating systems. Several administrative functions are unnecessary on DB2 for i because the database manager and operating system automatically handle the tasks. For instance, DB2 for i does not provide a **RUNSTATS** utility for optimizer statistics, because its database manager always keeps these statistics current. Similarly, there also is no concept of table spaces or low-level storage management (for example, partitioning data across drives) in DB2 for i.

For these reasons, a slightly different skill set is needed to support the various DB2 product members. IBM Data Studio does have limited support for the IBM i platform. Thus, the System i

Navigator tool (formerly, Operations Navigator) should be used for managing DB2 for i databases instead of the IBM Data Studio or InfoSphere Optim tools.

System i Navigator

System i Navigator is the graphical interface into DB2 for i for database administrators and application programmers. From the System i Navigator interface, a developer can perform almost all database administrative duties ranging from performance tuning to journal management. 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.
Index Advisor: Provide access to indexes advised by query optimizer and the ability to create the advised indexes.
Health Center: Allows user to track and monitor their databases and environment against DB2 limits.
Index and MQT Evaluators: Enables users to analyze how often an index or MQT is being used by the query optimizer. Evaluators are accessed with the Show Indexes & Show MQT task.
Journal Management: Start and end logging and swapping of journal receivers.
Database Navigator: Perform a basic graphical modeling of existing database definitions. The IBM InfoSphere Data Architect product also provides full modeling support for DB2 for i databases.
Note: this collection of tools is sometimes referred to as the OnDemand Performance Center.

DB2 for LUW CLP provides command line access to the database server. Although CLP can be used with DB2 for i, the System i Navigator SQL Script Center or interactive SQL utility (STRSQL) are the recommended choices.

The System i Navigator SQL Script Center offers no scheduling capabilities (as does the Script Center that is part of Data Studio). The IBM i job scheduler can be used for required scheduling activities.

Utilities

This section provides a discussion and comparison of various database utilities.

IBM Data Studio

For DB2 for i databases, the IBM Data Studio (formerly known as DB2 Developer Workbench) can be used to create and deploy Java and SQL stored procedures as well as SQL functions. More details on using this tool can be found in the **IBM DB2 for i and IBM**

Data Studio white paper
(**ibm.com**/partnerworld/wps/whitepaper/i5os/db2_workbench/support).

Load and import

The IBM i platform has an import command just as DB2 for LUW does. DB2 for i imports are normally done with the **CPYFRMIMPF** CL command. This command can import files containing delimited data or fixed-format data. The command assumes that the specified input file does not contain any column-definition information, the RMVCOLNAM parameter can be used to skip column name information in the first record. The System i Navigator toolset also provides a graphical interface to this command.

Here is an example of the utility being used to load a DB2 for LUW table (staff) where the column data is delimited with a column. The data is first exported and then copied into stream-based flat files on a DB2 for i server.

On DB2 for LUW CLP, type the following:

DB2 EXPORT TO staff.del OF DEL SELECT * FROM userid.staff

FTP **staff.del** to the DB2 for i server.

Use the **CPYFRMIMPF** CL command with the staff.del file, as follows:

CPYFRMIMPF FROMSTMF('~mydir/staff.del') TOFILE(MYLIB/STAFF)

DTAFMT(*DLM)

FLDDLM(',')

The previous steps are the most efficient way to move data from DB2 for LUW to DB2 for i. For someone who is not as comfortable with the IBM i platform, the data transfer can be done completely from the DB2 for LUW CLP interface using the following steps:

1. DB2 EXPORT TO staff.ixf OF IXF SELECT * FROM userid.staff

DB2 CONNECT TO myserver USER myid USING mypasswd
DB2 IMPORT FROM staff.ixf OF ixf INSERT INTO mylib.staff

The DB2 for LUW **Export** and **Import** commands can also be used to move tables with **LOB** columns to DB2 for i.

LOB example:

1. CONNECT TO db2luw user db2id using db2pwd
2. EXPORT TO myfile.ixf OF IXF LOBS TO D:\test\LOBexp\mylobs\ lobfile lobs1
   MODIFIED BY lobsinfile SELECT * FROM LOBTEXTTBL
3. CONNECT RESET
4. CONNECT TO idb2 USER idb2id USING idb2pwd
5. IMPORT FROM "D:\test\LOBexp\myfile.ixf" OF IXF LOBS FROM
   "D:\test\LOBexp\mylobs" MODIFIED BY LOBSINFILE METHOD P(1, 2)
   MESSAGES "D:\test\LOBexp\log.txt"
   INSERT INTO myschema.myLOBtbl(CUSTOMER_ID, CUSTOMER_DES)
6. CONNECT RESET

Both methods require the target table to be created in DB2 for i already. The table definition can be easily migrated by using the DB2 for LUW **db2look** command or IBM Data Studio.

If the database parallelism feature (DB2 SMP) has been installed and activated, then the database manager will break the input file into logical partitions and load those partitions into the target table in parallel. Data exports to delimited and fixed-format files can be performed with the **CPYTOIMPF** CL command. If date values exist in the table that is being exported, then the DATFMT parameter (for example, DATFMT(*YYMD)) needs to be specified on the CPYTOIMPF CL command.

ALTER TABLE

Although DB2 for LUW and DB2 for i both have an **ALTER TABLE** statement, there are differences in what can be changed. The DB2 for i alter capabilities are more advanced than DB2 for LUW. For example, DB2 for i allows more attributes of an existing column to be changed.

Constraint administration

The primary consideration for DB2 for i is always to ensure full data integrity. In certain situations, however, preserving data integrity on a continuous basis can have a significant performance impact. For example, bulk loading of data via a load utility is slower when, for every row, a detour is taken to ensure that all of the pertinent constraints are satisfied for that row.

The mechanism that DB2 for LUW uses to balance integrity and performance is the **Set Integrity** statement, which is used to disable a constraint and then re-enable the constraint. On the IBM i platform, 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 for i equivalent of the DB2 **Set Integrity** statement.

Summary

The purpose of this document is to highlight potential issues in porting databases and applications from DB2 for LUW to DB2 for i. It is a living document. As there is more experience with different porting situations, the document will grow accordingly.

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

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)
- Preparing for and Tuning the SQL Query Engine on DB2 for i (SG24-6598)
- 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

DB2 for i white papers
ibm.com/partnerworld/wps/whitepaper/i5os

DB2 Family Common Features Matrix and SQL Reference
ibm.com/developerworks/db2/library/techarticle/db2common/

IBM Education Resources (classroom and online)
ibm.com/systems/power/software/i/db2/education/index.html
ibm.com/partnerworld/wps/training/i5os/courses

IBM DB2 for i and IBM Data Studio
ibm.com/partnerworld/wps/whitepaper/i5os/db2_workbench/support

Porting Central: CLI differences for DB2 for i
ibm.com/partnerworld/pwhome.nsf/weblook/cli2b.html

What does DB2 on i really mean?
ibm.com/partnerworld/wps/servlet/ContentHandler/SROY-6UZDN4

IBM i Performance and Scalability 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

Conversion services

IBM Systems and Technology Group Lab Services

ibm.com/systems/services/labservices/

Online forum

IBM developerWorks: ibm.biz/db2iforum

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)

Third-pary performance tools

Centerfield Technology

www.insuresql.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.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.