



IBM DB2 for i porting guide

SQL Server to the IBM i platform

Version 7.0

*The database technology team
ISV Business Strategy and Enablement
July 2010*



Table of contents

Abstract	1
Introduction	1
The audience for this document	1
Assumptions	1
Preparing to port	2
Porting approaches.....	2
Design trade-offs.....	2
Misused porting technique.....	3
Porting tools	4
Sizing the port	5
Architecture	5
Brief history.....	5
Interfaces and packaging	6
Storage model	7
Metadata	7
Data types.....	8
Null indicators	8
DATE and TIME types.....	8
Numeric values.....	9
Character types (including variable-length types)	9
Timestamp.....	10
Identity	11
XML considerations.....	11
SQL language elements	11
Identifiers	12
Long identifiers.....	12
Naming conventions and formats	13
Three-part names	13
Local and global variables	14
Data manipulation (DML) statements.....	14
Insert statement.....	15
Delete statement	15
Select statement.....	15
Output variable assignment	15
TOP n queries.....	15
Column labels.....	15
COMPUTE clause	16
Update statement.....	16
JOINS	16
LIKE clause	17
Case-sensitivity considerations.....	17
Functions	18
UDF performance	18
Date and time processing.....	19



Arithmetic.....	20
Other functions.....	20
Stored procedures.....	20
Performance consideration.....	21
Exception processing.....	21
Result set differences.....	22
WITH ENCRYPTION clause.....	22
Sample conversion.....	23
Triggers.....	24
Constraints.....	25
Referential integrity.....	25
Check constraints and rules.....	25
Miscellaneous differences.....	26
Computed column.....	26
CREATE DEFAULT.....	26
TRUNCATE TABLE.....	26
Unsupported features.....	27
Application development.....	27
Dynamic SQL.....	27
Cursors.....	27
Blocked inserts, fetches and updates.....	28
Built-in stored procedures.....	28
Security.....	28
Transactions, concurrency and recovery.....	29
Concurrency and locks.....	29
Isolation Levels.....	30
AutoCommit.....	31
Optimistic locking.....	31
Journaling and recovery.....	32
After the port.....	33
Functional testing.....	33
Performance tuning and sizing.....	33
Administrative tasks.....	34
IBM i Navigator.....	34
Utilities.....	35
Load and import.....	35
System stored procedures.....	35
ALTER tasks.....	35
Summary.....	36
Appendix A: Rough sizing estimates for conversions.....	37
Appendix B: Resources.....	38
Trademarks and special notices.....	40



Abstract

This paper explains various processes and considerations regarding porting a Microsoft SQL Server database application to the IBM i platform to run with IBM 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.

NOTE: Because this document only focuses on porting applications from Microsoft® SQL Server to the IBM i™ (IBM i5™, eServer™ i5 and iSeries™) platform, it does not contain a list of the IBM DB2 for i® features that are missing from SQL Server.

The audience for this document

This paper is written for application developers and database administrators who want to convert an SQL Server-based application to DB2 for i. The paper covers the most common issues and inconsistencies encountered by developers in these porting situations, as well as DB2 support and administration topics that are relevant to database administrators.

Because Sybase® SQL Server and Microsoft SQL Server have many similarities, this paper is also useful when porting Sybase solutions to DB2 for i.

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

Assumptions

This paper assumes you are working with SQL Server 2008 and are porting to the DB2 for i 7.1 release. Any references to DB2 or DB2 in this document refer to DB2 for i 7.1.

This paper also assumes readers are familiar with the concepts of RDBMS and with Microsoft SQL and Transact-SQL. Readers need to have easy access to DB2 for i documentation. For example, the reader is probably interested in referring to that documentation for detailed information about the actual SQL statement syntax. The following web sites contain DB2 for i documentation:

- publib.boulder.ibm.com/eserver/ibmi.html
- ibm.com/systems/i/db2/books.html



Preparing to port

Before diving into a detailed comparison of DB2 and SQL Server, 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 SQL Server-based application enables you to be 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 SQL or proprietary features such as Transact-SQL?
- 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?

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 for porting your application to DB2. Any solution can be ported, given enough time and money. However, the questions to ask are: “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, this question is relatively easy to answer. With its support for industry standard interfaces, DB2 accommodates 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 with a single code base more easily. This design approach requires more effort and investment, but 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 it 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 database runtime 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. The procedure or module then turns the database request into the specific format or API (ODBC, JDBC or other interface) 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 the need to maintain, enhance and test a second code base. 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. However, performance is usually the tradeoff when you code to the lowest common denominator. For example, SQL Server 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 **run** 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 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 one time and run many times (the **prepare once, run many** model), then most likely, that application does 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/i/db2/db2performance.html). This course teaches the developer the proper way to design and implement a high-performing DB2 solution.

Misused porting technique

Because DB2 is also available for workstations, many developers wonder if they can perform 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? Yes, if you are very careful as you design and code the application. Although DB2 for i does belong to the DB2 product line, 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.

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. Refer to ibm.com/partnerworld/wps/servlet/ContentHandler/SROY-6UZDN4 for a high-level examination of DB2 for i and the IBM DB2 Family.



Porting tools

IBM and other software providers have some porting tools to help you automate some of the porting process.

The IBM DB2 Migration Toolkit can be downloaded from the Internet (ibm.com/partnerworld/i/db2porting) to help convert SQL Server objects and their data into DB2 for i format. Although the DB2 Migration Toolkit can greatly reduce the amount of time it takes to convert from SQL Server to DB2, it does not completely automate the database conversion process. For example, the toolkit does not convert references to the SQL Server catalogs to the corresponding DB2 catalog references. The programmer must perform some manual work. However, the toolkit flags items that have to be converted manually by the programmer. The converted programming objects (triggers, functions and stored procedures) then need to be reviewed to ensure they are semantically equivalent. In addition, the SQL code that is generated by the DB2 Migration Toolkit might not result in the best-performing SQL statements. This means that some performance tuning must be done after using the toolkit.

There are also a couple of tools that can be purchased from other software providers to help with your porting efforts. (These tools can be downloaded at www.swissql.com and www.ispirer.com.) Be aware, though, that these conversion tools might not generate SQL statements that are specifically supported by DB2 for i.

If your application has dependencies on proprietary features that are not supported by existing migration tools, it might be prudent to build your own migration tool. This approach involves a significant upfront investment and requires an in-depth knowledge of both the source and target databases, but still might be your best alternative.



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 compare some of the key differences encountered between SQL Server and DB2 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 SQL Server database product. In addition, the interfaces used with SQL Server and DB2 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 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 tell you that they are not using DB2.

Because the IBM 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 to create DB2 objects. The language extensions (APIs), known as the record level I/O interfaces, are available in most languages supported by the IBM i platform with the most usage in RPG and COBOL programs. Together, these extensions and DDS, is also known as the native database interface and is 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 not familiar 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, the native and SQL interfaces are almost totally interchangeable. Objects created with DDS are accessible with SQL statements; objects created with SQL are accessible with native record level access APIs. The DB2 SQL interface is compliant with the SQL-92 entry-level standard and the core level of the SQL 2008 standard.

DB2 for i porting guide: SQL Server to the IBM i platform



Interfaces and packaging

Because DB2 for i is integrated, most SQL applications can run on any IBM i server 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)
 - 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 (framework 2.0)
- Open Group™ DRDA application requester and server
- SQL Statement Processor (**RUNSQLSTM** CL command)
- Qshell DB2 utility
- Performance tuning tools

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 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.

Many vendors prefer using a native database interface (for example, DB-Library) instead of an industry-standard interface, such as ODBC or CLI because of performance concerns. DB2 does not really have a lower-level SQL-based interface; instead, the ODBC and CLI interfaces are treated as the native SQL-based interface for DB2 for i and provide performance similar to the native interfaces offered by other database products.

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 be used with a thorough understanding of the DB2 for i SQL engine. That knowledge can be obtained from the **DB2 for i SQL Performance** workshop. Refer to ibm.com/systems/i/db2/db2performance.html for this workshop.



Storage model

The physical storage model for DB2 and SQL Server are drastically different. With DB2, all of the storage allocation and management is handled by the database manager and operating system. DB2 does not require table spaces as do other DB2 products. Thus, there are no filegroups or devices 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. Because low-level storage manipulation is handled by DB2, there is also no need to create objects in a specified segment or no support to create clustered indexes.

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

For more information about DB2, refer to ibm.com/partnerworld/wps/whitepaper/i5os.

Metadata

Metadata provides the roadmap to interpret the information stored in the database. In SQL Server, metadata is stored in the system catalog and DB2 metadata is stored in a similar catalog. However, DB2 has a single system catalog that is always maintained. SQL Server has a catalog in each database instance, as well as a centralized system catalog in the master database. The SQL Server master database must be up and running in order to access any SQL Server database. DB2 does not have a master database to be concerned about.

The preferred way to reference metadata in DB2 is through the catalog views, such as **SYSIBM.TABLES**, not the underlying tables. The structure and names of these views are much different from those of the SQL Server metadata tables. Therefore, the developer must be familiar with the views to map the IBM information from the SQL Server to DB2. Detailed descriptions of the catalog views can be found in the appendix of the *DB2 for i SQL Reference Guide*. DB2 for Linux™, UNIX® and Windows as well as DB2 for i each supports a common set of catalog views for ODBC and JDBC clients in the **SYSIBM** schema.



Data types

Table 2 summarizes the mapping from the SQL Server data types to corresponding DB2 data types. The mapping is one to many and depends on the actual usage of the data.

SQL Server data type	DB2 data type	Notes
tinyint	SMALLINT	SMALLINT requires 2 bytes; tinyint is 1 byte
smallmoney	NUMERIC(10,4)	Can create a user-defined type
money	NUMERIC(19,4)	Can create a user-defined type
smalldatetime	TIMESTAMP	Default timestamp format is: YYYY-MM-DD-HH.MM.SS.NNNNNN
datetime	DATE TIME TIMESTAMP	- If storing just a date value, use DATE - If storing just a time value, use TIME - If storing a timestamp value, use TIMESTAMP
timestamp	TIMESTAMP Implicitly Hidden For Each Row Update As Row Change Timestamp	Not the same as the TIMESTAMP type supported by DB2 and the SQL standard. The Implicitly hidden and row-change timestamp clauses can be used to get a behavior that is close to the SQL Server timestamp support.
decimal(n,p) numeric(p,s)	DECIMAL(p,s) NUMERIC(p,s)	-SQL Server supports a maximum of DECIMAL(38) -DB2 maximum is DECIMAL(63)
nchar(n)	NCHAR(n)	
nvarchar(n)	NVARCHAR(n)	
ntext(n)	NCLOB(n)	
text(n)	CLOB(n)	
image	BLOB(n)	
xml	XML	Reference XML considerations below
bit	BINARY(1) CHAR(1) FOR BIT DATA	Can define a Check constraint to restrict column values to 0 and 1 (CHECK(col1='1' OR col1='0'))

Table 2. Mapping of SQL Server and DB2 data types

Note: Some DB2 data types that are not available in Microsoft SQL Server are included in this table.

Null indicators

Similar to SQL Server, IBM i null indicators are defined as **SMALLINT** or **short** in C and C++. When a null value is fetched from a column, the host variable for the column remains unchanged, and the null indicator variable for the column is set to a negative value.

DATE and TIME types

There are some differences between SQL Server and DB2 in the DATE and TIME data types. The **smalldatetime** data type supports a limited calendar range (January 1, 1900 to June 6, 2079) and the values are only accurate to the minute. The **TIMESTAMP** data type for DB2 supports a wider calendar range (January 1, 0000 to December 31, 9999) and is accurate to the microsecond.

The SQL Server **datetime** column can be supported with **DATE**, **TIME** or **TIMESTAMP** data types, depending on what type of value is actually being stored in the column. The **datetime** values are only accurate to 1/300 of a second and might even contain negative values that represent dates prior to the base calendar date (January 1, 1900).



The default format for the **DATE** and **TIME** values is also different. Those format differences need to be addressed when moving SQL Server DATE and TIME values into DB2.

As mentioned, the SQL Server does not provide separate data types for DATE and TIME. If only a time is specified when setting a **datetime** or **smalldatetime** value, the date defaults to January 1, 1900. If only a date is specified, the time defaults to 12:00AM (midnight). For instance, consider the following SQL Server statement:

```
INSERT INTO test VALUES('12:19:00 AM')
```

The literal in this statement is stored as **1900-01-01 00:19:00.000**. In contrast, DB2 requires that the literal used for a timestamp represents a valid value for both the date and time portions. Here is the equivalent DB2 **Insert** statement:

```
INSERT into test VALUES('1900-01-01-00.19.00')
```

DB2 for i also supports the following additional **timestamp** formats: **yyyymmddhhmmss** and **yyyy-mm-dd hh:mm:ss.nnnnnnn**.

The SQL Server **timestamp** type does not match the SQL Standard definition for that data type. Microsoft SQL Server **timestamp** columns indicate the sequence of activity on a row. Porting considerations for this data type are discussed in a later section of this paper.

Numeric values

The SQL Server and DB2 numeric data types are fairly compatible. DB2 supports a maximum precision of 63 digits for numeric and decimal data types.

Character types (including variable-length types)

DB2 supports the **CHAR** and **VARCHAR** data types for single-byte character strings that match the SQL Server data types. DB2 supports double-byte character strings with the **NCHAR**, **NVARCHAR**, **NCLOB**, **GRAPHIC** and **VARGRAPHIC** data types. SQL Server supports double-byte data with the **nchar**, **nvarchar** and **ntext** types. All of the DB2 character types have a bigger maximum size than the SQL Server types.

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. Starting with IBM i V6R1, 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.

Many SQL Server 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)**, because it is more efficient and takes less storage than the **VARCHAR** data type. A general guideline is to use the **CHAR** data type for columns of 40 bytes or fewer. 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, it is best to use the DB2 variable-length data types (**VARCHAR**, **VARGRAPHIC**, **BLOB**, **CLOB** and **DBCLOB**) for long memo or text description columns that are not referenced frequently.

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 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; therefore, 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; thus, 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 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 is 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 **BLOB** column to a different table.

Timestamp

As mentioned, the SQL Server **TIMESTAMP** data type is not used to store timestamp values generated by an application. Instead, it is used to identify the sequence of Microsoft SQL Server activity on the row. If a row is changed, the **timestamp** column value is updated by SQL Server with an 8-byte value that is unique across the entire database. A table can only have one **timestamp** column.

The DB2 **TIMESTAMP** data type does not perform this way, by default. If the SQL Server timestamp column is primarily being used to implement optimistic locking, then the hidden column and row-change-timestamp column attributes that were made available with IBM i V6R1. Here is an example of a DB2 timestamp column that is defined with these attributes:

```
CREATE TABLE tickets(
  ticket_ord  INTEGER,
  ticket_qty  INTEGER,
  ticket_event VARCHAR(10),
  ticket_ts   TIMESTAMP NOT NULL
              IMPLICITLY HIDDEN
              FOR EACH ROW ON UPDATE AS ROW CHANGE TIMESTAMP)
```

The ticket_ts column is updated by DB2 every time that a row is inserted into the tickets table or an existing row is updated. Unlike SQL Server, the updated value of the timestamp column is not guaranteed to be unique within the table. The timestamp column, however, can be used to manually implement an optimistic-locking scheme.

If the **timestamp** column is used only to identify the rows in a particular table, the primary key of the table can be used instead. However, if an application requires a unique identifier for a row across the entire database, the **DB2 ROWID** data type, which generates a 40-byte value, can be used.

Another alternative is to use the **GENERATE_UNIQUE** function to generate a unique value and then store that value in a new column in the table. This function can then be used on an **INSERT** statement or a **Before** trigger to assign the unique value to the new column that was added to the table to simulate the **TIMESTAMP** value.

Identity

DB2 for i contains support for the **IDENTITY** column attribute that is similar to the Microsoft SQL Server IDENTITY property. The DB2 **IDENTITY_VAL_LOCAL** scalar function can be used as a replacement for the SQL Server @@IDENTITY global variable.

The identity value that is generated by DB2 can also be retrieved by specifying the INSERT on the FROM clause of a Select statement as shown here:

```
SELECT order_id FROM FINAL TABLE (
  INSERT INTO orders VALUES(DEFAULT,'11/03/2007',50,'JM12'));
```

XML considerations

Starting with the 7.1 release, DB2 for i provides an XML data type similar to the SQL Server untyped XML data type that can store well-formed XML documents as well as XML content fragments. DB2 for i does not support typed XML columns that associate an XML schema collections with the column. In contrast with DB2 for i, values stored in an XML column can be validated against an XML schema by specifying the XMLVALIDATE functions on the INSERT and UPDATE statements being used to assign values to the XML column.

The DB2 for i XML functions in 7.1 do not include an XQuery search interface. As an alternative, applications can leverage the IBM OmniFind Text Search Server to index and search the stored XML values.

For a deeper understanding of the XML support that is available in prior releases, the following IBM Redbooks® document can 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.

DB2 for i porting guide: SQL Server to the IBM i platform



Identifiers

DB2 for i 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.

It is strongly recommended that you not use variant characters (such as \$, @, #, ^ and ~) in SQL identifiers. This is because 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 use 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 that have a maximum length of 10 characters are probably the only identifier that might cause issues when porting an application. All of the SQL limits are documented in the *DB2 for i SQL Reference*.

Long identifiers

Although DB2 supports long SQL identifiers, the commands and interfaces native to the System i5 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 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 IBM i support for long and short SQL identifiers is analogous to the file name restrictions when trying to use Windows and DOS utilities on the same object.

The system-generated short names do have a down side 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 **RENAME TABLE** statement was 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
(customer_name FOR COLUMN cusnam CHAR (20),
```

DB2 for i porting guide: SQL Server to the IBM i platform



```
customer_city FOR COLUMN cuscty CHAR(40))
```

To overwrite the system-generated name for **customer_master (CUSTO0001)**, this **RENAME TABLE** statement instead assigns a short name of **cusmst**. After usage of 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.

```
RENAME TABLE dbtest/customer_master  
TO SYSTEM NAME cusmst
```

If the SQL object is created with a short name, then the **RENAME TABLE** statement can also be used as follows to assign a long SQL identifier to that object. After this **RENAME** request is completed, the result is the same as the previous example; the SQL table name is **customer_master** and the system table name is **cusmst**.

```
RENAME TABLE dbtest/cusmst TO customer_master  
FOR SYSTEM NAME cusmst
```

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. 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 searches 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 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 that object in a schema named JOHNDOE.

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

SQL Server object names have a four-part structure (see below). The first three parts of these names are optional, depending on the context in which the object is used.

```
[SQL Server Instance].[Database Name].[Owner].[object name]
```

In DB2, the majority of applications reference DB2 objects with a two-part structure (schema-name.object_name) or single part name (object_name) because the applications usually begin with an explicit connection to the database before running any SQL statements.

DB2 for i porting guide: SQL Server to the IBM i platform

The SQL naming convention on the IBM i platform 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 IBM i naming convention (*SYS).

Local and global variables

In SQL Server, proprietary identifiers starting with the @ and # symbols fall into special categories. An identifier starting with @ can only be used as a local variable. The identifier prefix of @@ is reserved for global variables that are updated by SQL Server. An identifier starting with # can be used only as temporary local object; but an identifier prefix of ## can only be a global temporary object.

Here is an example comparing DB2 and SQL Server local temporary tables:

SQL Server: CREATE TABLE #LocalTemp1 (col1 INT, col2 CHAR(2));

DB2: DECLARE GLOBAL TEMPORARY TABLE SESSION.localtemp1
(col1 INT, col2 CHAR(2));

The SQL Server system global variables are often used with **Transact-SQL (T-SQL)**. Although they are called global, they provide information about the current session. DB2 for i supports global variables, as well. However, the DB2 for i global variables do not include any system-provided global variables such as the SQL Server **@@rowcount** and **@@error** global variables. This section discusses how to simulate these two global variables in DB2.

In SQL Server, the **@@error** global variable can be queried to check whether the previous statement succeeded or not. The **@@error** global variable has the error status of the most recently run statement. In DB2, return codes from SQL statements are handled with the **SQLCODE** or **SQLSTATE** class. Either can be used in applications, but the **SQLSTATE** class is preferred, now that the **SQLCODE** class has been deprecated in the current version of the SQL standard.

The **@@rowcount** variable contains the number of rows affected by the last query. The value represents the number of rows of a cursor result set returned to the client, up to the last fetch request. In DB2, the **GET DIAGNOSTICS** statement can be used with the **ROW_COUNT** option. If the previous statement is the **PREPARE** statement, the **ROW_COUNT** value identifies the estimated number of result rows in the prepared statement. If the previous statement is the **DELETE**, **INSERT** or **UPDATE** statement, the **ROW_COUNT** value identifies the number of rows deleted, inserted or updated by that statement. After the **SELECT** or **FETCH** statement is run, SQL Server sets the number of effected rows to the **@@rowcount** variable, whereas the **ROW_COUNT** value is not affected by the **SELECT** or **FETCH** statement in DB2.

Data manipulation (DML) statements

Depending on the syntax used in the SQL Server, the developer might find that **Insert**, **Delete**, **Update** and **Select** statements port easily to DB2. However, if the SQL Server DML statements use proprietary features, the conversion process is more complex. This section discusses those features that require conversion.

Insert statement

The SQL Server **Insert** statement is almost identical, but the **INTO** clause is optional [for example, **INSERT tbl2 VALUES(33)**]. The **INTO** clause must be included on DB2 **Insert** statements.

Delete statement

The SQL Server **Delete** statement has a few major differences with the DB2 support. With the SQL Server, the **FROM** clause is optional on the **Delete** request. Here is an example of that syntax:

```
DELETE tbl2 WHERE col1=33
```

This allows tables to be joined in the request. Here is an example of that syntax:

```
DELETE stores FROM stores, closedstores WHERE stores.id = closedstores.id
```

These proprietary features are not supported by DB2.

Select statement

Here are several incompatibilities with the **SELECT** support on SQL Server to be aware of.

Output variable assignment

Both DB2 and Microsoft SQL Server support a **SELECT INTO** statement but the implementation is drastically different. The SQL Server **INTO** clause specifies a table to hold the results of the **SELECT** statement. The DB2 **INTO** clause can only reference a variable.

SQL Server does have proprietary syntax that allows variables to be assigned to the result of a **SELECT** statement. Here is an example of that syntax:

```
SELECT @ytd_sales = ytd_sales  
FROM titles WHERE title = 'DATABASE DESIGN'
```

The SQL Server support also behaves differently when the **SELECT** statement returns more than one row. If more than one row is returned, the SQL Server assigns the value from the last row to the variable(s). If the following DB2 **SELECT INTO** statement returned multiple rows, then an SQL error (**SQLSTATE 21000**) is returned, as required by the SQL Standard:

```
SELECT ytd_sales INTO out_ytd_sales  
FROM titles WHERE title = 'DATABASE DESIGN'
```

TOP n queries

SQL Server includes a **TOP** clause that is used to write reports that return only the **Top n** results for a query instead of all the results. For instance, someone wants to see only the top 10 selling products, instead of all of the products sold by the company. DB2 has a **FETCH FIRST n ROWS** clause that provides the same capability.

Column labels

There are also syntax differences when assigning column headings on a **SELECT** statement because of some proprietary Microsoft SQL Server syntax. SQL Server allows the column name to be assigned with one of the following three methods:

```
SELECT mycol = col01 FROM ...
```

DB2 for i porting guide: SQL Server to the IBM i platform

```
SELECT col01 mycol FROM ...
SELECT col01 AS mycol FROM ...
```

DB2 supports the second and third methods. The third method is the most portable option.

COMPUTE clause

Another proprietary extension in SQL Server is the **COMPUTE BY** clause, which generates summary values that appear as additional rows in the query results. This allows the developer to produce detail and summary rows with one **SELECT** statement. On DB2, the **ROLLUP** function can be used to provide similar function. Here is an example of how to convert the proprietary SQL Server syntax to DB2:

SQL Server:

```
SELECT type, price FROM titles
ORDER BY type, price
COMPUTE SUM(price) BY type
```

DB2:

```
SELECT type, SUM(price) FROM titles
GROUP BY ROLLUP (type)
ORDER BY type
```

Update statement

DB2 and SQL Server both support an **Update** statement with similar syntax. However, there is a big difference in the syntax when updated values are supplied by a **SELECT** statement. Here is an example of how to convert the proprietary SQL Server syntax to DB2:

SQL Server:

```
UPDATE VarianceItems SET CurrentQty = L.Qty
FROM VarianceItems V INNER JOIN InvLocations L
ON V.ItemNum = L.ItemNum
WHERE L.Location = 'HN'
```

DB2:

```
UPDATE VarianceItems SET CurrentQty =
(SELECT L.Qty FROM VarianceItems V INNER JOIN
InvLocations L ON V.ItemNum = L.ItemNum
WHERE L.Location='HN')
WHERE EXISTS
(SELECT * FROM InvLocations L
WHERE VarianceItems.ItemNum = L.ItemNum
AND L.Location='HN')
```

JOINS

SQL Server supports a proprietary syntax for **Left** and **Right Outer Joins** that can be found in older applications. Fortunately, more recent versions of SQL Server also support the standard **Outer Join** syntax, just as DB2 does. Thus, when SQL Server queries that use the proprietary syntax are converted, they can run on either database product (see Table 3).

SQL Server Outer Join	DB2 Outer Join
-----------------------	----------------

<pre>SELECT A.last_name, A.id,B.name FROM emp A, Customer B WHERE A.id *= B.sales_rep_id;</pre>	<pre>SELECT A.last_name,A.id,B.name FROM emp A LEFT OUTER JOIN customer B ON A.id = B.sales_rep_id;</pre>
<pre>SELECT A.last_name, A.id,B.name FROM emp A, Customer B WHERE A.id =* B.sales_rep_id;</pre>	<pre>SELECT A.last_name,A.id,B.name FROM emp A RIGHT OUTER JOIN customer B ON A.id = B.sales_rep_id;</pre>

Table 3. SQL Server and DB2 **Outer Joins**

LIKE clause

DB2 and SQL Server both support the **LIKE** clause for wild-card searches, including support for the % and _ wild-card characters. SQL Server supports additional proprietary search operations through the ^ and [] wild-card characters that are not supported by DB2.

The bracket operator ([]) can be used to represent a range ([A through F]) or a set ([ABCDEF]) of characters. The following selection clause can be used to find all of the customer names that end with **SON** and start with a letter in the **C through L** range (for example, Carson or Larson).

```
... WHERE name LIKE '[C-L]%SON' ...
```

The ^ character acts as a negation operator by searching for characters that are not in a specified range or set. The following **Selection** clause finds all names that begin with **JO** where the next letter is not **N**.

```
... WHERE name LIKE 'JO[^N]%'
```

These two wild-card characters have to be simulated on DB2 by using combinations of **LIKE** clauses and possibly the **BETWEEN** clause.

Case-sensitivity considerations

SQL Server defaults to performing case-insensitive searches. For example, the predicate, **col1='Abe'**, returns the following values from **col1** both **ABE** and **Abe**. In contrast, DB2 defaults a case-sensitive search where **col1='Abe'** returns only the value of **Abe**.

If your application relies on this case-insensitive behavior, consider employing a shared-weight sort sequence to override the default DB2 behavior. Usage of a shared-weight sort sequence effectively causes DB2 to translate a lowercase and uppercase letter to the same weight so that the character comparisons are case-sensitive. 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.

A sort sequence can be specified when creating connections, programs and DB2 objects 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 stored procedure.

Creating an index with the same sort sequence enables the query optimizer to have the option of using that index when executing queries that have an active sort sequence. A shared-weight index gives the optimizer an opportunity to use an index scan instead of a full-table scan.

Another possibility is changing the search predicate to use the UPPER function (for example, UPPER(col1)='ABE'). If this approach is used, the best performance would be obtained by creating a derived (functional) SQL index for the optimizer to apply to this search predicate. The derived SQL index support that has been added in IBM i V6R1 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))
```

Functions

Many of the functions supplied with Microsoft SQL Server are also supplied by DB2 with the same name and behavior; therefore, no effort is required to port those invocations. Other functions have a different name in DB2 than in SQL Server, but serve the same purpose; these cases can also be easily ported as Sourced UDFs (see below). The remaining functions used in an SQL Server application are functions supplied with SQL Server that have no equivalent in DB2. The unsupported SQL Server functions can be converted to run as UDFs with DB2. There are several types of scalar and table UDFs in DB2, all of which are created with the **CREATE FUNCTION** statement. The basic UDF types are:

- **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. A sourced UDF can also be used as a form of alias. If there is a function in the SQL Server, such as **SVRFUNC**, that had an equivalent called **DB2FUNC** in DB2, then a sourced UDF, called **SVRFUNC**, can be created in DB2, invoking the **DB2FUNC** function and the SQL Server statements calling **SVRFUNC** do not need to be changed.
- **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 IBM i, including Java™.

UDF performance

If performance is more important than a single code base, then you might want to avoid using DB2 UDFs in the porting exercise. This is especially helpful when the same function can be implemented inline with a series of calls to IBM i-supplied functions. UDF invocations on the IBM i platform are implemented with an external program call that has more performance overhead than a system-supplied function call. Specifying the **NOT FENCED** and **DETERMINISTIC** attributes when creating functions can improve the performance of DB2 UDFs.

Several user-defined function examples can be useful in a database conversion to DB2. For more information, go to ibm.com/developerworks/db2/library/samples/db2/0205udfs/index.html.

Note that not all of these UDF examples works on DB2 for i.

Table 4 is a listing of commonly encountered SQL Server functions that map to DB2 functions with different names. The date and time-related functions are covered in a later section.

SQL Server	DB2
CONVERT, CAST	CAST, CHAR
CHAR	CHR
CHAR_LENGTH, DATA_LENGTH	CHAR_LENGTH, LENGTH
CHARINDEX	LOCATE
PATINDEX	POSSTR, POSITION
STUFF	INSERT
REPLICATE	REPEAT
INTTOHEX	HEX
+	CONCAT,
%	MOD

Table 4. Mapping of SQL and DB2 functions

The SQL Server concatenation operator, **+**, does have a behavior that is not supported on DB2 for *i*. If one of the character values being concatenated is **Null**, then DB2 and the SQL standard require the concatenated value to be **Null**. SQL Server has a proprietary behavior that causes the null character value to be replaced with an empty character string, "".

SQL Server also includes proprietary syntax that allows a function to be included on a **SELECT *** clause. This proprietary feature is not supported directly by DB2 and must be converted as follows:

SQL Server:

```
SELECT *,DATA_LENGTH(c3) FROM tbl1
```

DB2:

```
SELECT a.*, LENGTH(c3) FROM tbl1 a
```

Date and time processing

One of the big differences between DB2 and SQL Server is how the date values are displayed in the date and time processing. Many SQL Server applications use the **CONVERT** function to display the character format of a **datetime** value. The function supports many display formats that are specified with a style number parameter on the function. The DB2 **CHAR** function is the closest equivalent of the SQL Server **CONVERT** function, which allows date values to be displayed in different formats. The DB2 **CHAR** function allows the user to choose from a set of predefined formats (ISO, USA, EUR, JIS) that is used when displaying the character representation of a **datetime** value.

When a DB2 **datetime** value is retrieved without the **CHAR** function, the format of the resulting string depends on the date format (**DATFMT**), the date separator (**DATSEP**), the time format (**TIMFMT**) and the time separator (**TIMSEP**) parameters in effect when the statement is prepared. These parameters are assignable through various interfaces: precompiler parameter, ODBC data source option, SET OPTION statement and others.

Arithmetic

DB2 uses normal arithmetic operators (+, -) for date and time addition and subtraction. In contrast, SQL Server uses a **DATEADD** function for addition and subtraction. This means that **DATEADD(month, 1, ShipDate)** is converted to **ShipDate + 1 MONTH** on DB2. For computing the difference between two timestamps, SQL Server has a **DATEDIFF** function that is similar to the DB2 **TIMESTAMPDIFF** function. However, the parameters and output are different; therefore, those differences need to be examined closely when porting a **DATEDIFF** request to DB2.

Other functions

SQL Server applications use the **GET_DATE** function for retrieving the current timestamp. This can be converted to one of these DB2 special registers: **CURRENT TIMESTAMP**, **CURRENT DATE** or **CURRENT TIME**, depending on how it is being used.

Another common SQL Server function is the **DATEPART** function, which is used to extract specific units from a timestamp value. Instead of having one function that uses a parameter value to specify which unit to extract, DB2 supplies multiple functions that are each capable of just extracting one unit from a date or time value. For example, **DATEPART(YEAR, dtcol)** on SQL Server is converted to **YEAR(dtcol)** on DB2.

Stored procedures

With Microsoft SQL Server, stored procedures are usually written in Transact-SQL, (T-SQL) a proprietary language. In DB2 for i, SQL stored procedures can also be implemented with SQL. The language used for DB2 SQL procedures is based on the ANSI/ISO Persistent Stored Modules (PSM) standard and is similar to T-SQL and other stored procedure languages. This same PSM language is also available for the development of SQL triggers and SQL UDFs on DB2 servers.

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 supported by the IBM i platform (embedded SQL, CLI, JDBC and others). A DB2 stored procedure (regardless of the language) 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 is run.

When T-SQL stored procedures are migrated to DB2, the most appropriate approach is to use SQL procedures, because of their similarity to T-SQL. SQL procedures are automatically converted to C code and compiled when the **Create Procedure** statement is run. 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 T-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 was significantly improved with the IBM i V5R4 release. Tips and techniques for coding procedural SQL with good performance are documented in the *Improving SQL Procedural Performance* paper: ibm.com/partnerworld/wps/whitepaper/ibmi/sql_sql_v6r1/procedure

Table 5 shows the mapping of some of the Transact SQL constructs to the SQL procedural language.

T-SQL statement	SQL procedure language statement
@address VARCHAR(30);	DECLARE address VARCHAR(30);
SELECT variable=	SET variable =
EXECUTE	CALL
EXEC()	Run dynamic SQL statement and return a result set (PREPARE/EXECUTE)
RAISERROR	SIGNAL/RESIGNAL
BREAK	LEAVE
RETURN	RETURN
IF condition BEGIN statement1 statement2 END ELSE statement	IF condition THEN statement1; statement2; END IF ELSE statement;
WHILE condition BEGIN ...statements; END LOOP;	WHILE condition DO statements; END WHILE;

Table 5. Mapping of T-SQL and SQL procedural language

Other differences between DB2 and T-SQL stored procedures are that the individual statements within a T-SQL procedure do not end with a semicolon (;). In addition, SQL Server does not support **INOUT** parameters, but does support **OUTPUT** parameters, which are equivalent to the DB2 **OUT** parameters.

Performance consideration

It is fairly common to find a T-SQL stored procedure containing a single SQL statement, such as a **SELECT** statement. On DB2 for i, it is not recommended that you create an SQL procedure merely for the usage of a single SQL statement. Performance suffers in this scenario because there is so much overhead in resolving to the procedure's program object and copying in the input parameters. For a single SQL statement, it is usually faster to invoke the SQL statement directly.

Exception processing

Exception handling is an area where the SQL procedural language and T-SQL have significant differences. As mentioned, T-SQL uses the proprietary **@@error** global variable for error handling. The **@@error** global variable has the error status after each usage of an SQL statement. Here is an example of a T-SQL procedure that uses the **@@error** variable for error handling:

```
create proc x(@mydate datetime,@ok integer out)
begin
  insert into tabx values(@mydate)
  if(@@error = 0) select @ok=0;
  else
    select @ok=1;
end
```

DB2 has **SQLSTATE** and **SQLCODE** class codes for error handling. The error-handling model of the DB2 SQL stored procedure language is based on exceptions. An error that is not handled in the stored procedure causes the procedure to leave immediately before being able to check the **SQLSTATE** or the **SQLCODE** code in the stored procedure; the expectation is that the client handles it. To simulate the behavior of the **@@error** variable, the **CONTINUE** handler (which updates the error variable) must be defined to prevent the stored procedure from exiting prematurely. The following stored procedure behaves similar to the T-SQL procedure above. If the insert fails, the declared error handler is invoked and the variable for the **SQLCODE** class code is set. Then, control is returned to the statement that follows the **INSERT** statement that raised the exception.

```
CREATE PROCEDURE x(IN mydate TIMESTAMP, OUT ok INTEGER)
LANGUAGE SQL
BEGIN
  DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
  DECLARE var_state CHAR(5) DEFAULT '00000';
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION, SQLWARNING, NOT FOUND
    SET var_state= SQLSTATE;

  SET var_state = SQLSTATE;
  INSERT INTO tabx VALUES (mydate);
  IF (var_state <> '00000') THEN
    SET ok=0;
  ELSE
    SET ok=1;
  END IF;
END
```

The T-SQL **RAISERROR** command is not available in DB2, but the SQL procedural language does include the **SIGNAL** and **RESIGNAL** statements for emulation of raised errors.

Result set differences

When executing a **SELECT** statement, SQL Server returns a result set by default. On DB2 and in the SQL standard, a cursor has to be defined explicitly to return a result set.

WITH ENCRYPTION clause

DB2 does not support the SQL Server **WITH ENCRYPTION** clause, which is sometimes used to hide the SQL source for procedures, views and other objects from catalog (or metadata) viewers. The SQL source for DB2 objects is in the system catalog. Other SQL privileges are put on the system catalog to limit the number of viewers who can see the SQL source.



Sample conversion

The code below contains a before-and-after look of a Transact-SQL stored procedure that has been converted to DB2.

SQL Server:

```
CREATE proc GetTeam
    @InpTeam varchar(30),
    @InpPosition char(2)
AS
Declare
    @TblString varchar(255),
    @ColumnString varchar(255),
    @SearchString varchar(255)

SELECT @TblString = ' FROM PlayerData '
SELECT @SearchString = ' WHERE TeamName = ' + @InpTeam

If @InpPosition='QB'
    SELECT @ColumnString = 'SELECT Name, PassYds, PassTDs '
Else If @InpPosition='RB'
    SELECT @ColumnString = 'SELECT Name, RushYds, RushTDs '
Else If @InpPosition='WR'
    SELECT @ColumnString = 'SELECT Name, RecYds, RecTDs '

Exec(@ColumnString + ' ' +
    @TblString + ' ' +
    @SearchString)
GO
```

DB2:

```
CREATE PROCEDURE GetTeam
    (IN InpTeam CHAR(30), IN InpPosition CHAR(2))
RESULT SETS 1 LANGUAGE SQL
BEGIN
    DECLARE TblString,ColumnString,SearchString, QueryString
    VARCHAR(255);
    DECLARE ResultCursor CURSOR WITH RETURN FOR statement1;

    SET TblString = ' FROM PlayerData ';
    SET SearchString = ' WHERE TeamName = ' || InpTeam;

    IF InpPosition='QB' THEN
        SET ColumnString = 'SELECT Name, PassYds, PassTDs ';
    ELSEIF InpPosition='RB' THEN
        SET ColumnString = 'SELECT Name, RushYds, RushTDs ';
    ELSEIF InpPosition='WR' THEN
        SET ColumnString = 'SELECT Name, RecYds, RecTDs ';
    END IF;

    SET QueryString = ColumnString || TblString || SearchString;
    PREPARE statement1 FROM QueryString;
    OPEN ResultCursor;
END
```

Triggers

DB2 supports SQL and external triggers (implemented in high-level languages). Whenever porting T-SQL triggers, DB2 for i SQL triggers are the closest match. And because SQL Server's proprietary triggers currently only support statement-level triggers, it is hard to implement them as external triggers on the IBM i platform. The reason for this is that external triggers can be defined only at the row level. Thus, SQL triggers are the most likely conversion target on DB2.

DB2 SQL triggers support **BEFORE ROW**, **AFTER STATEMENT** and **AFTER ROW** triggers. Only **AFTER STATEMENT** triggers can be used in the conversion process because SQL Server triggers can only be activated one time per statement. SQL Server triggers supply the **INSERTED** and **DELETED** logical tables (that contain all of the rows affected by the activating statement) to help simulate row-level trigger capabilities. DB2 supplies similar logical tables with the **OLD_TABLE** and **NEW_TABLE** clauses. Row-level DB2 triggers typically access the triggering data with the **OLD** and **NEW** column clauses and typically offer better performance than statement-level triggers.

SQL Server allows triggers to be created for DDL statement events, this type of trigger is not supported by DB2.

As mentioned, PSM is available for implementing the business logic in the SQL trigger. In general, the SQL trigger body can contain any SQL statement. DB2 triggers can also invoke both stored procedures and functions. However, **BEFORE** triggers are allowed only to change or alter data with the **SET** statement; other SQL statements that insert, create or alter are prohibited. However, the **SQL_MODIFIES_SQL_DATA** QAQQINI parameter can be specified to disable this restriction. Refer to the **SQL Reference** for a full explanation of the IBM i SQL **CREATE TRIGGER** statement.

In SQL Server, the trigger body consists of a **BEGIN...END** block, (just as DB2 does) when there is more than one SQL statement. As with T-SQL procedures, T-SQL triggers do not have a delimiter that marks the end of a statement within the trigger body.

SQL Server triggers have an **UPDATE** function that allows an UPDATE trigger to be called only when a particular column value has actually changed. This same function can be accomplished on DB2 in two ways. First, the DB2 **UPDATE** clause can be used to activate an UPDATE trigger only for specific columns (for example, **BEFORE UPDATE OF c1, c3 ON mytbl**). The other option is do a comparison of the old and new values on the DB2 **WHEN** clause (for example, **WHEN oldval.c1 <> newval.c1**).

The DB2 **WHEN** clause can also be useful when converting T-SQL triggers that use an **IF** test to run the business logic conditionally within the trigger. DB2 triggers do a better job of handling recursive data changes that are performed within the trigger. A DB2 trigger does call itself in response to a second update to the same table within the trigger. For example, if an update trigger on a table results in another update to the same table, that update trigger is activated a second time. An SQL Server UPDATE trigger cannot be called a second time in the example shown in Table 6.

SQL Server:	DB2:
<pre>CREATE TRIGGER accounttrigger ON accounts FOR INSERT AS BEGIN DECLARE @chgcnt INT SELECT @chgcnt=count(*) FROM INSERTED INSERT INTO account_changes VALUES('I',USER_NAME(), @chgcnt) END</pre>	<pre>CREATE TRIGGER accounttrigger AFTER INSERT ON accounts REFERENCING NEW TABLE AS newtbl FOR EACH STATEMENT MODE DB2SQL BEGIN DECLARE chgcnt INT; SET chgcnt = (SELECT count(*) FROM newtbl); INSERT INTO account_changes VALUES('I', CURRENT USER, chgcnt); END</pre>

Table 6.

A single SQL Server trigger can also be associated with multiple events. See the following:

```
CREATE TRIGGER accounttrigger ON accounts FOR INSERT, UPDATE, DELETE AS...
```

With DB2, there are most likely three triggers (an INSERT, UPDATE and DELETE trigger) that need to be created where all of the triggers contain the same trigger logic. A single trigger design can be accomplished by having the SQL trigger call a common SQL procedure or by using an external trigger.

Constraints

DB2 and SQL Server support **primary key**, **foreign key** (referential integrity), **unique**, **NOT NULL** and **check** constraints, but there are some differences.

Referential integrity

Both DB2 and SQL Server 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. SQL Server does not allow referential constraints to reference a table in another database. DB2 does allow referential constraints to reference tables in another schema.

SQL Server referential constraints cannot cascade delete or update operations to related tables. SQL Server applications when use triggers for this purpose. DB2 supports cascading of deletes via the **CASCADE**, **SET NULL** and **SET DEFAULT** delete rules on the constraint definition.

SQL Server referential constraints do not require an index. DB2 creates an index if one does not exist to support its internal enforcement of the constraint.

Check constraints and rules

Although SQL Server does include the standard check constraint support, older applications might instead use rules (that is, **CREATE RULE**) to limit or restrict the data values for a column. These rules need to be converted into SQL standard and DB2 **check** constraint definitions.

Miscellaneous differences

Here are a couple of other SQL Server differences to consider porting.

Computed column

SQL Server allows a computed column to be defined as part of a table definition, as shown in the following example.

```
CREATE TABLE mytable
(
    low int,
    high int,
    myavg AS (low + high) / 2
)
```

DB2 for i does not support computed or generated columns. The circumvention is to create both a column and a trigger that is responsible for automatically computing the calculation in the same manner as the SQL Server computed column does. See the following example:

```
CREATE TRIGGER calculate_myavg
    BEFORE INSERT ON mytable
    REFERENCING NEW AS NEW
    FOR EACH ROW MODE DB2SQL
    set new.myavg = (new.low + new.high)/2
```

CREATE DEFAULT

Similar to rules, SQL Server includes support for an older **CREATE DEFAULT** statement that is a non-standard method for specifying the default value for a column. These default definitions have to be converted to the SQL standard default support in DB2.

TRUNCATE TABLE

In SQL Server, the application can remove all rows from a table by using the **TRUNCATE TABLE** statement, which also reclaims the storage used by the table. The **TRUNCATE TABLE** statement is a DDL command and cannot be rolled back.

DB2 does not support the **TRUNCATE TABLE** statement. There are two viable general solutions in DB2 for performing this function:

1. Use the **CLRPFM** CL command to delete all of the rows from the specified table. The **CLRPFM** command will not work if the specified table has **delete** triggers or is referenced by a **foreign key** constraint. Those objects have to be removed or disabled prior to the **clear** request. The **CLRPFM** command acquires an exclusive lock on the table when performing this operation. Therefore, after the **clear** operation, there will be no remaining open cursors on top of the table. This system CL command can be run easily from an SQL-based application using a system-provided stored procedure, the **QCMDExc** command. Here is an example of invoking that stored procedure to perform a **clear** operation. The first parameter is the CL command string and the second parameter is the number of characters in the CL command string:

CALL QSYS.QCMDXEC ('CLRPFM mylib/mytable', 20)

- The other solution is to delete all rows in the table with the **DELETE** statement (for example, **DELETE FROM t1 WITH NC**). The DB2 database manager internally tries to implement the DELETE statement with the **CLRPFM** command.

Unsupported features

Some SQL Server features or capabilities are extremely difficult to support or resolve with a DB2 for i circumvention.

- Indexed Views
- Optimizer hints (**Note:** The DB2 optimizer is cost-based and does not give the programmer any control. ISVs need to send someone to the **DB2 for i SQL and Query Performance** workshop. Refer to ibm.com/systems/i/db2/db2performance.html for details.

Application development

Although SQL Server does have precompilers for embedded SQL programming with C and COBOL, the majority of the ported applications use a programming interface that sends ODBC requests to the database.

Dynamic SQL

ODBC is the most popular dynamic SQL interface for SQL Server, but Microsoft SQL Server also has a proprietary interface, **DB-Library**. DB2 does not support the **DB-Library** APIs. However, this proprietary API maps most closely to the ODBC and CLI interfaces supported by DB2.

SQL Server applications using ODBC convert easily to the IBM i platform. Microsoft has added a few proprietary extensions into the SQL Server ODBC support that are not supported by DB2.

A key performance consideration when porting ODBC from SQL Server to DB2 is the usage of the **SQLExecDirect** function for submitting dynamic SQL requests to the database engine. For faster performance, Microsoft suggests using the **SQLExecDirect** function for SQL statements that do not return or require a result set (for example, the **INSERT** statement). On DB2, if an SQL statement runs multiple times within a connection, do not use the **SQLExecDirect** function, because it causes DB2 to prepare that statement many times. Instead, change the application to match the **Prepare once, run many** model suggested earlier in this paper.

Cursors

SQL Server supports two forms of syntax for declaring cursors: one form that is compliant with the SQL standard and another that employs a set of proprietary Transact-SQL extensions. Proprietary extensions that are not supported by DB2 include: **FORWARD ONLY**, **READ_ONLY**, **SCROLL_LOCKS** and **OPTIMISTIC**. This extension category includes SQL Server firehose cursors that are defined as **forward-only** and **read-only**; DB2 only allows a cursor to be declared as read-only (as defined by the SQL standard).

The standard style of cursors supported by SQL Server includes: **static**, **dynamic** and **keyset-driven**. There is also support for the **INSENSITIVE** and **SCROLL** cursor behavior and for all **fetch**

options. The DB2 cursor support is based also on the SQL standards. However, DB2 does not yet include support for **keyset-driven scrollable** cursors and the **fetch absolute** option. **Scrollable** cursors are available to any DB2 programming interface.

Blocked inserts, fetches and updates

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

Built-in stored procedures

SQL Server includes a set of built-in, proprietary stored procedures that are sometimes encountered when porting an application. DB2 does not include any built-in stored procedures for the same purposes; therefore, the available options when porting to DB2 are either to eliminate the usage of a procedure or to create a DB2 stored procedure that mirrors the function.

The SQL Server procedures most commonly used in an application are catalog and extended stored procedures. Catalog stored procedures are used by an SQL Server application to retrieve metadata from the system catalogs. For example, the **sp_column** procedure returns information on the columns in a specified table. The **sp_column**, **sp_table** and **sp_pkey** catalog procedures can be easily simulated with a DB2 stored procedure. SQL Server applications use extended stored procedures to interface with the operating system or other applications.

Security

SQL Server uses the concepts of authentication, roles and privileges to implement security. Two modes of authentication are supported:

- **Windows authentication:** The Windows operating system authenticates the login ID and password, and then passes only the login ID to the SQL Server (which matches it to the **sysxlogins** system table).
- **Mixed mode authentication:** SQL Server authenticates the login ID and password against the information it has in the **sysxlogins** system table.

After authentication, users cannot perform any operation against the SQL Server, unless they have been assigned a role or a given privilege. Roles are used to group users into a single unit to which permissions can be applied. Thus, rather than granting individual permissions to several users, the programmer can create a role with all of these permissions and then assign this role to the users.

In DB2, users do not exist within the database; rather, they are managed by the operating system. Thus, DB2 authentication is somewhat similar to the SQL Server Windows authentication; however, no database login information is kept in any database table. All users can potentially use DB2; however, unless they have been granted a given DB2 authority or privilege, there is not much they can do. Granting and revoking authorities and privileges can be handled easily through the IBM i system commands or the IBM i Navigator GUI, which is also part of the IBM i operating system.

Also, DB2 does not use the **roles** term; instead, it uses the **authorities** term, which is similar to the SQL Server **fixed server** and **database** roles. DB2 does not support the Microsoft SQL Server user-defined database roles; however, a group profile can be assigned authorities and privileges to

implement this. DB2 also uses the **privileges** term, which is equivalent to the SQL Server **permissions** term.

Transactions, concurrency and recovery

Transactions and concurrency controls behave differently between SQL Server and DB2 applications. This section covers these differences along with the recovery of transactions.

Microsoft SQL Server has an implicit and explicit transaction mode. The implicit transaction mode allows the SQL Server to behave in the same way as DB2 and the SQL standard. Some SQL Server applications activate this mode by using the **SET IMPLICIT_TRANSACTIONS ON** statement. ODBC applications activate the implicit mode by disabling the **AutoCommit** feature on the **SQLSetConnectOption** function.

When the implicit transaction mode is enabled, SQL Server implicitly invokes a **BEGIN TRANSACTION** statement before the following statements: **Delete**, **Insert**, **Open Fetch** and **Update**. If there is an open transaction, a new transaction is not started. The open transaction has to be committed explicitly by the user with the **COMMIT TRANSACTION** statement for the changes to take effect and for all locks to be released.

An explicit transaction is a grouping of proprietary SQL statements surrounded by the following transaction delimiters:

```
BEGIN TRANSACTION [transaction_name]
SAVE TRANSACTION savepoint_name
COMMIT TRANSACTION [transaction_name]
ROLLBACK TRANSACTION [transaction_name | savepoint_name]
```

DB2 does not require explicit transactions and always runs with an implicit transaction mode.

Concurrency and locks

Both SQL Server and DB2 use shared locks to control concurrent data access. There are some differences in the type of locks acquired and when the locks are acquired. SQL Server has three levels of locking: **row**, **page** and **table** locks. The default locking granularity in SQL Server is row-level (earlier versions of SQL Server defaulted to page-level). Lock escalation between these three levels does occur.

DB2 only supports row and table-level locks with row-level being the default locking granularity. DB2 typically only acquires row-level locks to ensure integrity in concurrent environments. However, some situations require DB2 to acquire an exclusive table-level lock instead of row locks. In general terms, there are **exclusive** (update) locks and **share** (read) locks. To update a row, the database server must first acquire an **exclusive** lock on that row. When a **share** lock is acquired for an object on behalf of an application, other applications can also acquire a **share** lock on the object, but requests for **exclusive** locks are denied. On the other hand, an **exclusive** lock blocks other applications from acquiring locks, even share locks, on the object.

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

Isolation Levels

Before looking at how to improve the concurrency of ported applications, it is useful to understand the differences between the DB2 and SQL Server implementation of concurrency control.

SQL Server cursors usually do not hold locks under the fetched row. Rather, they use an optimistic concurrency strategy to prevent updates from overwriting each other. If one user attempts to update or delete a row that has been changed since it was read into the cursor, SQL Server detects the problem and issues an error message. The application can trap this message and retry the update or delete. If an SQL Server application needs to ensure that a row cannot be changed after it is fetched, an **UPDLOCK** hint is used in the **SELECT** statement. This hint does not block other readers.

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 DB2 for i default isolation level is interface-dependent; therefore, it is best for the application to set the isolation level explicitly. DB2 provides several ways to do this:

- 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; for all levels except **No Commit**, the database manager places exclusive locks on every row that is inserted, updated or deleted. (**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 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 as long as the 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 would have read 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 through 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.

All of the DB2 isolation levels, except **No Commit**, are supported by SQL Server. The SQL Server default isolation level is **Read Committed (Cursor Stability)**. The **SET TRANSACTION** statement is the most commonly used method in SQL Server for changing the isolation level.

To increase the concurrency of the system, commit the 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; although switching the isolation levels is advised for some SQL Server applications. One suggested 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).

The SKIP LOCKED DATA and USE CURRENTLY COMMITTED clauses can also be used to improve concurrency. More details on this clause can be found in the DB2 for i SQL Reference.

AutoCommit

As with isolation levels, the DB2 **AutoCommit** default setting depends on the application interface.

Optimistic locking

DB2 does not support SQL Server's optimistic-cursor concurrency options. Optimistic locking can be manually implemented by the developer by creating a timestamp column with the hidden and row change-timestamp column attributes. This timestamp column can then be compared before updating a row to determine if the row has been changed since the last time the row was read by the application.

If the ported application requires minimal lock-wait time, then you might consider employing an optimistic-locking scheme. This locking technique requires the table to have a column (typically, a timestamp column) that can identify when the row's last change took place. The application reads the row and commits right away to release the lock, thus allowing other applications to have write access to the row. When it is time to update the row, the application retrieves the row again to see if it 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 should be a relatively rare event, the application must start the process again by rereading the row and presenting the latest version of its contents to the user.

See the following example:

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

... think time while 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. The implementations for DB2 and SQL Server are similar except for the terminology. SQL Server database changes are written to a transaction log file; DB2 changes are written to a journal.

As with SQL Server, DB2 implements write-ahead journaling (logging), in which the changed data is always written to the journal before the change is committed in the database. DB2 logs changes in its journal and journal receiver objects, including the old version of the data.

DB2 journals can be used to recover changes made to a specific table or schema, or to 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 must be done after the database has been moved to DB2 for i.

Functional testing

Functional testing must be performed to make sure that objects and requests on the source and target are returning equivalent results. Even when the SQL syntax is identical, the semantics and usage 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 that is returned to the application.

IBM does make IBM Power Systems hardware available for IBM i functional testing to developers and integrators through its Virtual Loaner Program. Refer to ibm.com/systems/vlp for more details. 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 SQL Server query optimizer might work great with a specific set of indexes on top of 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, the Microsoft Database Consistency Checker [DBCC] commands). The database manager automatically maintains and updates statistics (inside the table and index objects) as changes are made to the objects.

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 to run the same SQL request multiple times within a connection. DB2 typically does not perform well with either of these inefficient usage models.

DB2 includes several database performance tools, including IBM i Navigator SQL Performance Monitor and Visual Explain tools (see descriptions of these tools in the **Administration** section of this paper) of this paper. In general, however, to use these tools effectively and to be efficient at tuning DB2, you must attend the **DB2 for i SQL Performance** workshop (ibm.com/systems/i/db2/db2performance.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 to ensure that you use IBM i hardware 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 must know the minimum server configuration required to deliver acceptable performance.



The Power Systems and IBM i Benchmark Centers are 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/i/support/benchmarkcenters/index.html

For general IBM i performance information, go to: 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 Benchmark 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. Thus, the database administration tasks and database administration tools are quite different. The DBCC commands are the best example of this point. DBCC is used regularly by an SQL Server administrator to validate the structure and data within a database and to repair any database damage detected by the utility. These validation checks include: verification of index and data- page linkages, verification of index order and pointer-consistency checks. With DB2, an administrator rarely has to study or administer the internals of the databases manually.

IBM i Navigator

IBM i Navigator (formerly Operations Navigator) is the DB2 graphical interface for database administrators and application programmers. It is the rough equivalent to the SQL Server Enterprise Manager and Microsoft Management Console.

From the IBM i Navigator interface, almost all database administrative duties (ranging from performance tuning to journal management) can be performed. Here is a quick overview of some of the IBM 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:** Perform basic graphical modeling of existing database definitions.

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

Utilities

Here are a couple of SQL Server utilities and how they compare to the equivalent DB2 support.

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 using the **CPYFRMIMPF** utility to load a stream-based flat file where the column data is coma-delimited:

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

If the database parallelism feature (DB2 SMP) is installed and activated, the database manager can break the input file into logical partitions and load them into the target table in parallel.

Importing SQL Server **DATE** values can be a challenge because the SQL Server and DB2 default formats are not the same. It is possible to use the SQL Server **CONVERT** function to get the value into a DB2-supported format when writing data into the SQL Server export file. The **CPYFRMIMPF** utility does not support SQL Server image values and must be manually handled.

Data exports to delimited and fixed-format files are performed with the **CPYTOIMPF** command.

System stored procedures

SQL Server includes proprietary system stored procedures to perform administrative tasks. Some of these tasks are unnecessary with DB2; thus, an equivalent DB2 stored procedure is not needed. For those tasks that do need to be performed, DB2 provides APIs and system commands that can be issued from external DB2 stored procedures.

ALTER tasks

The majority of the SQL Server **ALTER TABLE** capabilities are supported by DB2. The biggest exception is the ability to rename an existing column. On DB2, the column must be dropped and then added. Although the DB2 **ALTER TABLE** statement does not allow constraints and triggers to be suspended (disabled), these suspensions can be done with the system **CHGPFCS**T and **CHGPFTRG** commands. The **ALTER VIEW** capability for SQL Server is not supported by DB2.



Summary

The purpose of this paper is to highlight potential issues in porting databases and applications from Microsoft SQL Server to DB2 for i. It 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.

NOTE: The DB2 for i 6.1 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 will take to convert a SQL Server application to DB2 for i. These estimates are provided AS IS without warranty of any kind. The amount of time to perform the conversion will vary widely, based on the experience level of the programmer and the complexity of the SQL application.

- **Join syntax changes:** Five minutes to one hour for query
- **TRUNCATE table:** A few hours to one day
- **Date/Time arithmetic changes:** Five minutes to two hours
- **Triggers and Stored Procedures:** Difficult to estimate; depends on the complexity of the procedure and trigger logic



Appendix B: Resources

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

- IBM Systems Information Center
<http://publib.boulder.ibm.com/eserver/ibmi.html>
- DB2 for i Online Manuals
ibm.com/systems/i/db2/books.html
- IBM Redbooks
ibm.com/redbooks
 - *Stored Procedures, Trigger, 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)
 - *DB2 for AS/400 Object Relational Support* (SG24-5409)
 - *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 SQL Query Engine (SQE)
ibm.com/systems/i/db2/sqe.html
- Education Resources (classroom and online)
ibm.com/systems/i/db2/db2educ_m.html
ibm.com/partnerworld/wps/whitepaper/i5os
ibm.com/partnerworld/wps/training/i5/courses
- **DB2** Performance Workshop
ibm.com/systems/i/db2/db2performance.html
- Improving SQL Procedural Performance white paper
ibm.com/partnerworld/wps/whitepaper/ibmi/sql_sql_v6r1/procedure
- What Does DB2 on IBM i Really Mean
ibm.com/partnerworld/wps/servlet/ContentHandler/SROY-6UZDN4
- IBM DB2 Migration Toolkit
ibm.com/partnerworld/i/db2porting
- Sample UDFs for Migration
ibm.com/developerworks/db2/library/samples/db2/0205udfs/index.html



- IBM Power Systems & IBM i Benchmark Centers
ibm.com/systems/i/support/benchmarkcenters/index.html
- Virtual Loaner Program
ibm.com/systems/vlp
- IBM Innovation Center
ibm.com/isv/iic,
- Performance Management for IBM i
ibm.com/systems/i/advantages/perfmgmt

Online Newsgroups and Forums

- IBM developerWorks
ibm.com/developerworks/forums/forum.jspa?forumID=292
- USENET
<http://comp.sys.ibm.as400.misc>, comp.databases.ibm-db2
ibm.com/servers/eserver/series/db2/support
- System i Network SQL and DB2 Forum
forums.systeminetwork.com/isnetforums

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

Third-party 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

© Copyright IBM Corporation 2010. All rights Reserved.

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.