IBM DB2 for i porting guide

MySQL to the IBM i platform

.

Database technology team

ISV Business Strategy and Enablement

July 2014
Version 8.0

@IBMSystemsISVs

Table of contents

Abstract **1**
Introduction **1**
- Target audience 1
- Assumptions 1

Preparing to port **2**
- Porting approaches and IBMDB2I storage engine 2
- Design tradeoffs 3
  - Adding an abstraction layer 3
  - Creating a separate application version 3
- Misused porting approach 4
- Porting tools 4

Sizing the port **5**
- Architecture 5
  - Brief history 5
  - Interfaces and packaging 6
  - Physical model 6
  - Storage model 7
- Metadata 7
- Data types 8
  - Unsigned attribute 8
  - Auto-increment attribute 8
    - Example: MySQL to DB2 table conversion 8
  - Timestamp considerations 9
  - VARCHAR performance considerations 9
  - CREATE TABLE statement 10
    - IF NOT EXISTS clause 10
    - TEMPORARY table 10
    - ENGINE_TYPE and other table options 10
- SQL language elements 11
  - Identifiers 11
    - Naming conventions and formats 12
  - Comments 13
  - SELECT syntax 13
    - STRAIGHT_JOIN 13
    - SQL_SMALL_RESULT and SQL_BIG_RESULT 13
    - Unsupported options 14
    - DISTINCTROW 14
    - INTO OUTFILE and DUMPFILE 14
    - FROM differences 14
    - LIMIT 14
    - PROCEDURE clause 15
  - SELECT list operators 15
  - JOIN syntax 15
  - GROUP BY syntax 15
  - String processing and functions 16

Wild-card processing and pattern matching 16
Date and time functions 17
Other functions and operators 18
INSERT statement 18
DELAYED clause 18
ON DUPLICATE KEY UPDATE clause 18
UPDATE statement 19
DELETE statement 19
Stored procedures, triggers, user-defined functions and views 19
Constraints 19
Unsupported features 20
Application development 20
mysql and mysqli function mapping 20
Performance 21
Concurrency and recovery 21
Locking 21
Transaction control and isolation levels 21
Database recovery and journaling 23
After the port 23
Functional testing 23
Performance tuning and sizing 24
Administration 25
System i Navigator 25
Utilities 25
Import 25
Show statements 26
Table-maintenance statements 26
Alter table 26
Indexes 27
Summary 27
Appendix A: User-defined function examples 28
UDF performance considerations 28
Reverse UDF example 28
Format UDF example 29
SUBSTRING_INDEX UDF example 30
BIT_COUNT UDF example 31
Appendix B: Resources 32
Trademarks and special notices 34

Abstract

This paper explains various processes and considerations regarding porting a PHP application using MySQL 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 paper – 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 paper only focuses on porting from MySQL to IBM® DB2® for i, it does **not** contain a list of the DB2 features that are missing from MySQL on the IBM i operating system.

Target audience

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

This paper concentrates primarily on the database differences between the MySQL and DB2 for i database servers. It does not focus on the differences between the underlying operating systems. Application-development issues are also only covered from a database perspective. Refer to Appendix B for a website listing that provides a more general perspective on porting applications to the IBM i platform (IBM System i®, IBM System i5®, IBM eServer™ i5, and IBM eServer iSeries).

Assumptions

The paper assumes you work with MySQL V5 and are porting to DB2 for i 7.2. The paper also assumes you are familiar with the concepts of RDBMS and SQL and have easy access to DB2 for i 7.2 documentation. Refer to that documentation for detailed information about the actual SQL statement syntax. Refer to Appendix B for DB2 for i documentation.

Preparing to port

Before you will see a detailed comparison of the differences between DB2 for i and MySQL, the first step in considering a port to DB2 is reviewing the source database. A firm understanding of the technologies and interfaces used by the application enables you to be more productive when using this paper to assess the extent of the porting effort to DB2 for i.

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

- Does the application use standard SQL or proprietary features?
- What programming interfaces does the application use for data access?
- Does the application rely on any platform-specific middleware or technology?
- Are there any known performance constraints or issues?
- Does the business logic reside in procedural database objects (that is, triggers and procedures) or within the application?

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

Porting approaches and IBMDB2I storage engine

In 2009, a new storage engine was delivered to enhance the integration between IBM i and MySQL. This new storage engine is officially known as the DB2 for i Storage Engine for MySQL with IBM i - the short name being IBMDB2I. This new storage engine makes it easier to run and integrate web applications on IBM i without users having to learn how to manage and backup MySQL databases.

The IBMDB2I storage engine delivers tight integration by enabling the data to reside in DB2 for i objects while the application continues run unchanged as it uses MySQL interfaces to process data. With the engine storing MySQL data in DB2 for i objects, IBM i users can go back to using familiar IBM i commands and interfaces to manage and backup the databases. In addition, IBM i developers can use their favorite programming languages and data-access interfaces (RPG, DB2 Web Query and others) to access the MySQL data and blend it with the business data stored in their DB2 databases.

The performance requirements of a MySQL-based application are a key determining factor on whether or not to use the IBMDB2I storage engine should be used. Thus, performance testing is highly recommended before deploying any mission-critical applications that use the IBMDB2I storage engine. The performance of the IBMDB2I storage engine is probably acceptable for those applications that primarily run simple lookup queries against the MySQL database. However, applications that feature complex SQL queries or generate heavy transactional workloads might not be able to achieve acceptable performance using the IBMDB2I storage engine for MySQL. In these cases, it is best to port the MySQL-based application to use DB2 directly. See the following presentation, *When and where to use the DB2 MySQL Storage Engine* (**ibm.com**/partnerworld/wps/servlet/ContentHandler/RBAA-7YF7YK), to learn more about when to use the IBMDB2I storage engine instead of porting the MySQL database to DB2.

If you use the IBMDB2I storage engine instead of porting the MySQL database, you can find details on using the DB2 for i storage engine in the IBM Redbooks® titled *Using IBM DB2 for i as a Storage Engine of MySQL* (**ibm.com/**redbooks/abstracts/sg247705.html).

If you need to port the MySQL database to DB2 for i for performance reasons, consider the approach that you want to take when porting your application to DB2 for i. It is possible to port any solution, given enough time and money. However, the questions to ask are:

- What is the goal for the application being ported?
- Does it need to be a portable solution that can easily support multiple DBMS products or a solution that is tightly integrated and tuned for DB2 for i?”

This porting issue needs to be discussed before the database-porting project starts to make sure that all parties agree on the priority.

If the solution already supports multiple DBMS products, this is a relatively easy question to answer. With its support for industry-standard interfaces, DB2 for i can accommodate this application design with minor changes. If the solution currently does not support multiple DBMS products, you need to consider design issues, which are covered in the following section.

Design tradeoffs

However, if your application only supports MySQL, you must investigate some application-design issues.

Adding an abstraction layer

The first alternative to consider is redesigning your application with an abstraction layer to support various database server types more easily with a single code base. 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 if faster to deploy these enhancements to all of your supported platforms.

If maintaining and porting the source code is a top requirement, a good application design is to isolate all of the database runtime calls from the application. You usually accomplish this isolation by having the application create its own set of database methods or calls that are, then, passed to a single procedure or module. That procedure or module then turns the application-database request into the specific format or API (PHP, JDBC or other database interface) that the target database supports.

Creating a separate application version

Another alternative is to create a separate version of your application specifically for DB2 for i. Although this can require less upfront investment than the previous approach, the long-term expenses will be greater because of having a second code base to maintain, enhance and test. The benefit of this approach is that the best performance is usually obtainable by changing the application and database to exploit the target server. However, the more changes are made to exploit the target, the harder it is for a single set of application source code to run on multiple database servers. If the goal is to have a common set of single-source code, then you must avoid the use of platform-specific features. The tradeoff of this approach is usually reduced application performance because you must code to the lowest common denominator. For example, MySQL has a scalar function called **X** and DB2 for i has an equivalent function called **Y**. In this case, you have two choices: change the application to call function **Y** (which yields the best performance) or code a DB2 for i user-defined function (UDF) called **X** whose only purpose is to call function **Y**. The latter option allows the application code to remain unchanged, but results in slower performance.

This tradeoff is especially critical with DB2 for i because of some nuances in its implementation of how dynamic SQL is prepared and run. Often, 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 once. Or, the application uses the db2_exec function or JDBC Statement class to run the same SQL request multiple times within a connection. DB2 for i is not well suited for either of these inefficient program models, although it does use caching and other techniques to enhance the performance of this inefficient model. If the application logic cannot be changed to a model where an SQL statement is prepared once and run many times, then the ported application will probably not perform well on the IBM i platform.

Note: On the topic of performance, it is strongly recommended that database administrators and SQL developers who are new to the IBM i platform attend the *DB2 for i SQL Performance Workshop*. This course teaches the proper way to design and implement a high-performing DB2 solution. You can find more information about this course at:
ibm.com/systems/power/software/i/db2/education/performance.html

Misused porting approach

Because DB2 is also available for workstations that use the Linux® and Microsoft® Windows® operating systems, many developers wonder if they can perform their porting and associated testing tasks on workstations running these operating systems, and then move the ported application and application data to the IBM i platform to use DB2 for i. Does that work? The short answer is, “Probably not.” Although DB2 for i does belong to the DB2 family, there are differences in the SQL syntax and features supported by each DB2 product. There is no master DB2 version that works across all IBM hardware platforms; each product has features that one of the other database products does not support.

Starting with DB2 for Linux or Windows, this will only work if, prior to using any SQL statement on the workstation version, you first verify that DB2 for i supports the SQL statement and syntax. You can reference an online white paper for a high-level examination of DB2 for i and the IBM DB2 product suite (**ibm.com/**partnerworld/wps/servlet/ContentHandler/SROY-6UZDN4).

Porting tools

IBM and other software providers provide tools to automate some of the porting processes.

The IBM DB2 Migration Toolkit for MySQL automates many of the steps involved in porting a database from MySQL to DB2 for i. This no-charge software download automates the migration of MySQL objects such as tables, indexes and primary-key definitions. You can access this toolkit at **ibm.com**/partnerworld/i/db2porting.

Although the DB2 Migration Toolkit can help greatly reduce time it takes to convert from MySQL to DB2, it does not completely automate the database-conversion process. You have to perform some manual work. The toolkit flags items that you must convert manually. In addition, the DB2 Migration Toolkit generates SQL code that might not result in the best-performing SQL statements. This means that you must also do some performance tuning after using the toolkit.

The SQLWays and SwisSQL migration tools are two tools, among others that are available from other vendors to help automate the conversion of MySQL databases. IBM does not endorse these or any other vendors' conversion tools. Also, be aware that these conversion tools might not generate SQL that is specifically supported by DB2 for i.

Sizing the port

Although the design approach and usage of porting tools definitely affects the porting costs, the effort is also dependent on the features used in the source database. This section highlights and compares some of the key differences frequently encountered when porting from MySQL to DB2 for i. This information will help you better assess the difficulty and complexity of your porting project.

Architecture

This section summarizes the DB2 for i architecture and compares that architecture with the MySQL database. In addition, the interfaces used with MySQL and DB2 for i databases are also compared.

Brief history

An integrated, fully relational database has shipped with every IBM i model 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 IBM DB2/400. Then, in 1999, the IBM DB2 Universal Database™ (DB2 UDB) branding was added. DB2 for i still has some roots from the original database engine. However, a substantial amount of new technology and features were added to DB2 for i. In fact, the database is integrated so well that some IBM i users do not realize that they are using DB2.

Because IBM developed the AS/400 family of systems 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 specifications (DDS) is the language used for creating DB2 objects. The language extensions (or APIs), known as the *record-level I/O interfaces* are available in most of the languages supported by the IBM i platform with the most usage in RPG and COBOL programs. These extensions and DDS are also known as the *native database interface* and are quite similar to the indexed sequential access method (ISAM).

Because of this native, non-SQL interface, some IBM i developers 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, both the native and SQL interfaces are almost completely interchangeable. You can access objects created with DDS through the use of SQL statements. And, you can access objects created with SQL through the native record-level-access APIs. The DB2 SQL interface is compliant with the core level of the SQL 2008 standard.

Interfaces and packaging

Because DB2 for i is integrated, most SQL-based applications run on any server running IBM i without requiring additional products to be purchased for the client or server. The IBM i operating system includes the following DB2 capabilities, 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
- Open Group DRDA application requester and server
- SQL statement processors (the IBM i RUNSQLSTM and RUNSQL CL commands)
- DB2 Qshell utility
- Performance-tuning tools
- System i Navigator

For PHP applications, DB2 for i offers both a generic ODBC and a native DB2 interface for data access. The DB2 PHP extension provides a generic ODBC-based interface. The ibm_db2 extension provides the native interface for DB2 for i and is the recommended interface in terms of function and performance.

These PHP middleware interfaces are not listed in this section because they are optional components. The data access extensions are included in the Zend Server Community Edition (CE) for IBM i, which is a no-charge licensed product available to IBM i developers.

- You can find the product and documentation for the Zend Server CE product at: www.zend.com/en/products/server/zend-server-ibm-i
- You can find documentation for the ibm_db2 extension functions at: http://us3.php.net/manual/en/book.ibm-db2.php

Only if you must embed SQL in a high-level language (C, C++, RPG or COBOL), do you need to purchase and install the DB2 for i SQL Development Kit and Query Manager product (5722-ST1). This product is not necessary for most other ports of MySQL applications.

Physical model

MySQL provides the capability to define multiple databases on a single server — as a way to group related database objects. DB2 for i also supports this notion of independent, isolated databases. However, the default configuration for DB2 for i is as a single system-wide database. Schemas are the logical containers for related database objects within DB2 for i. Any user can access any database object in the schema as long as that user has the proper authorization.

Storage model

The storage model used by MySQL tables depends on the storage engine used for the table. You must create tablespaces for MySQL tables using the InnoDB and NDB cluster-storage engines; other engine types do not require tablespaces for storage allocation. With DB2 for i, the database manager and operating system handle storage allocation and management; there are no tablespaces to create or manage on the IBM i platform. In addition, data for a DB2 for i object is written across the available disk devices automatically to reduce contention and maximize performance.

MySQL supports two types of tables: *transaction-safe* and *non-transaction-safe* tables. In DB2, all tables are designed to support transaction processing and are, thus, transaction-safe by default. A regular DB2 table might have slower data-retrieval and insertion operations when compared to a non-transaction-safe MySQL table. However, the transactional capabilities of DB2 tables provide better concurrency and safer database recovery.

DB2 for i does offer two optional table types, materialized query tables (MQTs) and partitioned tables, but they are designed only for use in specific situations. Although in some limited circumstances, you can improve query performance, the primary uses of table partitioning within DB2 for i are to overcome the size limitations of an individual SQL table and some bulk-data operations such as add, drop, save and restore. For details on the DB2 for i partitioned table implementation, see the white papers listed on the IBM i Education Curriculum web site at **ibm.com**/partnerworld/wps/whitepaper/i5os.

MySQL also supports a Merge and Heap table type for nonpersistent storage. On DB2 for i, you can simulate a Merge table by creating an SQL view that performs a union of all the tables referenced in the Merge collection on MySQL. This type of SQL view on DB2 for i cannot be updated. You can migrate a MySQL HEAP table to a DB2 temporary table, MQT or index — depending on the requirements of the application.

Metadata

Metadata provides the roadmap to interpret the information stored in the database. The metadata for MySQL database objects is contained in the INFORMATION_SCHEMA database. The MySQL views contained in this database approximate the views defined in the SQL standard. DB2 for i also has an INFORMATION_SCHEMA schema that contains similar views and complies with the SQL standard. INFORMATION_SCHEMA is the standard schema name that contains catalog views. It is a synonym for the QSYS2 schema on IBM i. You can find detailed descriptions of the catalog views in an appendix of the *DB2 for i SQL Reference Guide*. (See Appendix B for a link to this Guide.)

You can also find a similar set of catalog views in DB2 for i in the SYSIBM schema. These views generally offer better performance and are compatible with catalog views supported by DB2 for Linux, UNIX and Windows operating systems.

Data types

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

MySQL type	DB2 type	Notes
TINYINT	SMALLINT	
BIT/BOOLEAN	SMALLINT	DB2 check constraint needed to limit Boolean values
MEDIUMINT	INTEGER	
FIXED	DECMIAL	
DATE	DATE	The range of the MySQL date is year 1000-9999, whereas DB2 supports dates from 0001-9999.
DATETIME	TIMESTAMP	MySQL value is displayed as: YYYY-MM-DD HH:MM:SS
TIMESTAMP	TIMESTAMP	MySQL Timestamp value range limited to: 1970-01-01 00:00:00 to the year 2037.
TIME	TIME	DB2 time only supports a 24-hour clock, MySQL range is: -838:59:59 to 838:59:59.
YEAR	SMALLINT	
CHAR, VARCHAR	CHAR, VARCHAR	
TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB	BLOB(255), BLOB(65K), BLOB(16M) BLOB(2G)	DB2 BLOB maximum half the size of a MySQL LONGBLOB
TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT	CLOB(255), CLOB(65K), CLOB(16M) CLOB(2G)	DB2 CLOB maximum half the size of a MySQL LONGTEXT
LONGTEXTRAW	CLOB(2G)	DB2 CLOB maximum half the size of a MySQL LONGTEXTRAW
ENUM VARCHAR2, SET VARCHAR2	VARCHAR(n)	DB2 check constraints needed to enforce list of values defined for ENUM or SET columns

Table 2. Mapping of MySQL data types to DB2

Unsigned attribute

All DB2 numeric data types are signed by default, so that the MySQL column attribute must be ignored when generating the equivalent DB2 column definition.

Auto-increment attribute

You can implement the MySQL auto_increment column attribute on DB2 with the GENERATED AS IDENTITY clause. The following example highlights this difference in implementation by comparing the equivalent MySQL and DB2 table definitions.

Example: MySQL to DB2 table conversion

```
CREATE TABLE mysql_test (
   wk_id int(11)unsigned NOT NULL auto_increment,
   user_id int(11)unsigned default NULL,
   cnt int(10)unsigned default NULL,
   cat_id int(12)unsigned default NULL,
   status varchar(10)default NULL,
      PRIMARY KEY (wk_id))   type=MyISAM;
```

```
CREATE TABLE db2_test (
   wk_id INT NOT NULL GENERATED BY DEFAULT AS IDENTITY,
   user_id INT default NULL,
   cnt INT default NULL,
   cat_id INT default NULL,
   status VARCHAR(10) default NULL,
   PRIMARY KEY (wk_id));
```

The MySQL mysql_index_id() function provides the last-generated value for an auto_increment column. With DB2, the Identity_Val_Local function retrieves the last generated value for an identity column within a connection.

Timestamp considerations

With MySQL, the first `TIMESTAMP` column in a table is automatically set to the date and time of the most recent operation, if you do not assign it a value yourself. That proprietary behavior is not supported by a DB2 timestamp column. However, this behavior can be supported on DB2 with the hidden column and row change-timestamp column attributes. Here is an example of a DB2 timestamp column 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)
```

DB2 updates the ticket_ts column each time a row is inserted into the tickets table or an existing row is updated. The updated value of the timestamp column is not guaranteed to be unique within the table.

A MySQL TIMESTAMP column can be set to the current date and time by assigning it a NULL value. The CURRENT TIMESTAMP special register must be used on DB2 to get this behavior.

VARCHAR performance considerations

Many MySQL applications use VARCHAR types for almost all character-string values. In these cases, it is better to port the values to the fixed-length DB2 data type CHAR(*n*); 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 deciding on the column type, based on performance, carefully read the DB2 for i paging and I/O behaviors for variable-length and LOB columns described here.

In general, the DB2 variable-length data types (VARCHAR, VARGRAPHIC, BLOB, CLOB and DBCLOB) are only used for long-memo or text-description columns that are not referenced very 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 character column to a fixed-length column, you can use the ALLOCATE keyword to improve performance. ALLOCATE(0) is the default and causes variable-length column values 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; it also stores only the column in the overflow area when the column value exceeds *n*. Setting n so that almost all 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.

Consider the following 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 shows that this address column is retrieved frequently, so you want to move most of the address values back into the fixed portion of the row. If most of the address values are 40 bytes or fewer 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 fewer are stored in the fixed portion of the row, thus, eliminating the extra I/O operation on the auxiliary overflow area.

The potential performance problem is further magnified when both LOB and VARCHAR values are stored in the auxiliary overflow area. LOB storage is almost identical to VARCHAR column storage; therefore, it is possible for both large LOB values and smaller VARCHAR columns to share the same overflow area. The sharing is not a problem, but when a VARCHAR column that sits in the overflow area is referenced, the entire overflow area for that row is paged into main memory, including all BLOB and VARCHAR values. The application might only retrieve a VARCHAR(50) column, but if that column is in the same overflow storage area as three 2 MB 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, you must set the VARCHAR column’s ALLOCATE value to the maximum to ensure that the value is always stored in the fixed portion of the row.

Another option is to move the LOB column to a different table and join to the data only when needed.

CREATE TABLE statement

Some issues are related to using the CREATE TABLE statement.

IF NOT EXISTS clause

The IF NOT EXISTS keyword causes MySQL to skip creating a specified table if the specified table name already exists (instead of signaling an error). There is no equivalent DB2 clause, but you can specify a Continue exception handler to simulate the MySQL behavior.

TEMPORARY table

The DB2 CREATE TABLE statement does not support the proprietary MYSQL TEMPORARY clause. Use the DECLARE GLOBAL TEMPORARY TABLE statement in DB2 to create a temporary table.

ENGINE_TYPE and other table options

The MySQL CREATE TABLE statement supports many proprietary options that enable control of administrative settings such as the storage engine or block. On DB2 for i, the VOLATILE keyword is the only administrative setting available. The code page (or character set) is specified at a column level, and the remaining options do not apply because IBM i takes care of low-level administrative settings and maintenance.

SQL language elements

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

Identifiers

IBM i supports two SQL identifiers: *ordinary* and *delimited* identifiers. Ordinary identifiers begin with an uppercase letter and convert to all uppercase. A delimited identifier (for example, “My Column Name” is a sequence of one or more characters enclosed in SQL escape characters [“…”]. With delimited identifiers, leading blanks are significant and letters in the identifier do not convert to uppercase.

By default, MySQL uses a different character, the backtick character (`…`), for delimited identifiers — also known as the *identifier-quote character*. When you wrap a MySQL table or column name with this delimiter, the application might not use the delimited version of the name to access the MySQL object. The following SQL statements shows MySQL’s support for quoted and unquoted delimited-identifier names in both SELECT statements to work against the users table created with delimited names:

MySQL:

```
CREATE TABLE `users` (`user_key` INTEGER, `user_name` CHAR(30))
SELECT user_name FROM users
SELECT `user_name` FROM `users`
```

These MySQL identifiers have no blanks and do not try to preserve case. when using a MySQL identifier to embed blanks or preserve the characters’ case, the application must use the delimited identifier name.

After an object is created with delimited identifiers on DB2, the application must always use the delimited name when referencing a delimited identifier. Thus, you must use delimited identifiers on DB2 only when delimited identifiers are used on MySQL to include blanks and to preserve the case of a character. You can change the MySQL application to match the standard SQL behavior of DB2 by specifying a value of ‘ANSI_QUOTES’ for the sql_mode option.

DB2 for i does not support double-byte table names. It is best not use variant characters (such as **$**, @, #, **^** and **~**) in SQL identifiers because the underlying code points of these characters can vary depending on the coded character set identifier (CCSID) or language that is active on the system. If variant characters are in your identifiers, unpredictable results can occur, especially on systems that use non-English CCSID values. The underscore (_) character is not treated as a variant character.

Generally, DB2 for i SQL identifier-length maximums are not an issue when porting databases. DB2 supports identifiers up to 128 characters for columns, tables, indexes, procedures and constraints. Authorization names are the only identifier type with a lowest maximum length (10 characters) that can cause an issue. The *DB2 for i SQL Reference* explains all SQL limits (see **Appendix B: Resources**).

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

System-generated short names have a downside. 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 (CUSTO00001). More importantly, there is no guarantee that these short names are consistent across systems or repeated creations of the same SQL object, because of dependencies on creation order and other identifiers that share the same first five characters. For consistency, special SQL syntax was added to DB2 for i that allows the the developer to control the short name.

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

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

To overwrite the customer_master system-generated name (*CUSTO0001*), the FOR SYSTEM NAME clause assigns a short name (*cusmst*). Thus, customer_master is the SQL table name; cusmst is the system table name. Applications can refer to either name on other SQL statements.

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

Naming conventions and formats

Almost all of the DB2 for i SQL interfaces allow you to specify an SQL naming mode or format. This option controls the syntax for qualifying an object with a schema (or library) name as well as for defining the default search path when referencing unqualified SQL object names.

***SYS** (System naming) mode is based on the traditional, native database-access method on the IBM i platform. MySQL does not support this naming mode. System naming mode dictates the use of the traditional IBM i slash (/) to explicitly qualify SQL objects with a schema name (for example, mylib/mytablename). Furthermore, when a schema is not specified, then the system-naming method searches the libraries defined in the job's library list to find the unqualified object.

***SQL** (SQL naming) is based on the SQL standard, requiring a period (**.**) to separate schema and object names (for example, mylib.mytablename). The only library (or schema) searched for unqualified objects is the current one; the current schema value usually searches the library that matches the name of the current SQL authorization name (or user profile name). If JOHNDOE is the authorization name and an unqualified object is referenced, DB2 for i expects to find it in a schema named JOHNDOE. The IBM PHP data-access interfaces use the SQL naming option by default. Other settings impact the default-search path, but this section describes the naming conventions' default behavior. See the *SQL Reference Guide* for details. (See Appendix B.)

Comments

MySQL supports a couple of proprietary options for including comments within SQL scripts and statements. One method is to use the pound (#) sign as the starting character, as demonstrated here:

```
# Table definition for users
 CREATE TABLE users (c1 INT, c2 VARCHAR(20))
```

You must replace the # comment character on DB2 for i with the standard SQL simple comments, which you add after two consecutive hyphens (--). MySQL also supports the same two-hyphen syntax, but requires a blank space after the hyphens.

SELECT syntax

The MySQL Select statement supports a number of extensions to the SQL standard. Those extensions are highlighted in bold and are covered in this section:

```
SELECT [STRAIGHT_JOIN ]
[SQL_SMALL_RESULT ] [SQL_BIG_RESULT ] [SQL_BUFFER_RESULT ]
[SQL_CACHE |SQL_NO_CACHE ] [SQL_CALC_FOUND_ROWS ] [HIGH_PRIORITY ]
[DISTINCT |DISTINCTROW |ALL ]
select_expression,...
[INTO {OUTFILE |DUMPFILE }'file_name'export_options ]
[FROM table_references
[WHERE where_definition ]
[GROUP BY {unsigned_integer |col_name |formula}[ASC |DESC ],...
[WITH ROLLUP ]]
[HAVING where_definition ]
[ORDER BY {unsigned_integer |col_name |formula}[ASC |DESC ] ,...]
[LIMIT [offset,] row__count |row_count OFFSET offset ]
[PROCEDURE procedure_name(argument_list)]
[FOR UPDATE |LOCK IN SHARE MODE ]
```

STRAIGHT_JOIN

The STRAIGHT_JOIN keyword forces the MySQL query optimizer to join the tables in the order listed on the SELECT statement. DB2 for i does not support any SQL syntax for forcing the join order, instead the query optimizer determines the optimal join order.

SQL_SMALL_RESULT and SQL_BIG_RESULT

The SQL_SMALL_RESULT and SQL_BIG_RESULT keywords are used to give the optimizer information on the size of the result set that is returned by the SELECT statement. This input must then influence the plan picked by the MySQL optimizer.

The OPTIMIZE FOR *n* ROWS clause is the closest equivalent on DB2 for i. The OPTIMIZE FOR *n* ROWS clause also influences the DB2 query optimizer's choice of access plan, but the optimizer behavior usage of this input is different.

The number-of-rows value tells the optimizer how many rows the application fetches on each access of the result set. For example, a value of 10 tells the query optimizer that the application fetches the first 10 rows of the result set and then does some processing before returning to DB2 for the next 10 rows. Thus, in this example, the DB2 for i query optimizer builds an access plan that returns the first 10 rows of the result set as fast as possible. The number of rows specified on the OPTIMIZE clause does not have to match the number of rows fetched or the size of the entire result set. Here is an example of converting the MySQL syntax to DB2 for i if you decide to use the DB2 OPTIMIZE FOR *n* ROWS clause.

```
MySQL: SELECT SQL_SMALL_RESULT c1, c2 FROM t1
DB2:   SELECT c1, c2 FROM t1 OPTIMIZE FOR 40 ROWS
```

Unsupported options

There are no DB2 equivalents to the MySQL keywords SQL_BUFFER_RESULT, SQL_CACHE, SQL_CALC_FOUND_ROWS and HIGH_PRIORITY.

DISTINCTROW

The DISTINCTROW keyword is a synonym for DISTINCT and is supported by DB2.

INTO OUTFILE and DUMPFILE

The OUTFILE and DUMPFILE keywords cause the result of the SELECT statement to be written to a stream file. DB2 for i has no equivalent on the SELECT statement. The result of the SELECT statement can be written to another DB2 table using the following CREATE TABLE syntax:

```
DB2:   CREATE TABLE out1 AS (SELECT c1, c2 FROM t1) WITH DATA
```

You must use the IBM i CPYTOIMPF system command to export DB2 data into a stream file.

FROM differences

Another proprietary behavior that MySQL supports is the use of the SELECT statement to return computations with no reference to a table (for example, SELECT 1+1). DB2 supports table-less queries with the VALUES statement (for example, VALUES 1+1).Often a MySQL request such as this refers to the DUAL dummy table – SELECT 1+1 FROM DUAL. DB2 for i provides a dummy table (SYSDUMMY1) in schema SYSIBM. You must change MySQL statements that reference the DUAL table or have no table reference so that they reference the SYSIBM.SYSDUMMY1

LIMIT

If the MySQL application uses the LIMIT keyword with a single argument, the DB2 for i FETCH FIRST n ROWS ONLY clause provides the same function to limit the size of the result set.

```
MySQL: SELECT c1, c2 FROM t1 WHERE c3>100 LIMIT 10
DB2:    SELECT c1, c2 FROM t1 WHERE c3>100 FETCH FIRST 10 ROWS ONLY
```

If the LIMIT clause contains two arguments, the first argument specifies the offset of the first row to return; the second specifies the maximum number of rows to return. The offset of the initial row is 0 (not 1). The following SQL statement on MySQL returns rows 6 through 15 from the result set.

```
MySQL: SELECT lastname, salary FROM t1
       ORDER BY workdept,lastname LIMIT 5,10
```

For this usage of LIMIT, the DB2 for i ROW_NUMBER expression to simulate this MySQL behavior is shown here:

```
DB2:   WITH numbering AS (
          SELECT ROW_NUMBER() OVER (ORDER BY workdept, lastname) AS rowno,
              lastname, salary, workdept FROM emp)
       SELECT lastname, salary FROM numbering WHERE rowno BETWEEN 6 AND 15
        ORDER BY workdept, salary
```

PROCEDURE clause

The procedure clause names a stored procedure that is called *MySQL* to process the data in the result set. DB2 for i has no direct equivalent for this proprietary extension; however, it might be possible to use a UDF call on the SELECT statement to simulate the behavior.

SELECT list operators

MySQL supports several proprietary functions on the SELECT list, such as Logical OR, Logical AND, Logical OR, Comparison operators and the BETWEEN operation. In almost every case, you can map these functions to the DB2 for i CASE expression. Here is a simple example:

MySQL: `SELECT (col1=100) FROM t1`

DB2: `SELECT CASE WHEN col1=100 THEN 1 ELSE 0 END FROM t1`

JOIN syntax

Here is a listing of the join syntax that MySQL supports; the large majority of these join types are available on DB2 for i. The join types not supported are highlighted in bold, as follows:

```
table_reference,table_reference
table_reference [INNER | CROSS ] JOIN table_reference [join_condition ]
table_reference STRAIGHT_JOIN table_reference
table_reference LEFT [OUTER ]JOIN table_reference [join_condition ]
table_reference NATURAL [LEFT [OUTER ] ] JOIN table_reference
{OJ table_reference LEFT OUTER JOIN table_reference ON conditional_expr }
```

As documented earlier, the MySQL Straight_Join support is not supported to control the DB2 query optimizer. The Natural join syntax results in MySQL automatically picking the join columns as those with the same name in both tables and ensuring the result set contains only one copy of the join columns. If both tables in the following example contain a column named *custid*, then the natural join syntax results in MySQL using the custid in both tables on the join column.

MySQL:
```
SELECT LastName, FirstName, Orderdate
       FROM customers NATURAL JOIN orders
```

On DB2 for i, you must convert natural joins to use the JOIN USING syntax. Here is an example of converting the previous join to this DB2 syntax.

DB2:
```
SELECT LastName, FirstName, Orderdate
       FROM customers INNER JOIN orders USING (custid)
```

DB2 for i supports the MySQL CROSS JOIN and LEFT OUTER JOIN.

GROUP BY syntax

The GROUP BY syntax between MySQL and DB2 is quite similar. The biggest difference is that MySQL supports proprietary extensions that allow the application to reference grouping columns with an alias or position. DB2 for i does not support these extensions. In both cases, you must change the MySQL grouping clause to refer to the column name.

MySQL:

```
SELECT col4 FROM t1                         SELECT col4 AS c4 FROM t1
  GROUP BY 1                                  GROUP BY c4
```

DB2: SELECT col4 FROM t1 GROUP BY col4

There are also some differences with the column functions available for grouping operations. The MySQL COUNT column function supports multiple expressions or columns; in contrast, DB2 for i only supports a single expression or column. Simulating this type of support on DB2 for i requires concatenating the columns or expressions as shown:

MySQL:
```
SELECT c1, COUNT(DISTINCT c1, c2) FROM t1
         GROUP BY c1
```
DB2:
```
SELECT c1, COUNT(DISTINCT CONCAT(c1, c2)) FROM t1
         GROUP BY c1.
```

String processing and functions

In default mode (non-ANSI), MySQL supports both single or double quotes as a valid string identifier. Because DB2 is designed according to the ANSI standard, there is support only for single quotes as string delimiters. In the ANSI standard, double quotes are reserved for delimiting SQL identifiers.

The MySQL concatenation functions do not require arguments to be of the same data type; DB2 for i requires the arguments to be compatible strings. In addition, MySQL concatenation also has the proprietary behavior of **not** returning the null value when a string argument is concatenated with a null value. In this case, DB2 returns the null value, as required by the standard (see Table 3).

MySQL	DB2	Notes
+ (concatenation)	|| or CONCAT	MySQL function returns ASCII-code point value for specified input; the DB2 for i default encoding is EBCDIC
CONCAT('a','b','c')	('a' || 'b' || 'c')	MySQL function returns ASCII-code point value for specified input; the DB2 for i default encoding is EBCDIC
ELT(n,string1,string2)	CASE	Returns the *nth* string or null
FORMAT	DB2 UDF	Refer to Appendix A
MID	SUBSTR	
SUBSTRING ('abcd' 2 FROM 2)	SUBSTR ('abcd',2,2)	
SUBSTRING_INDEX	DB2 UDF	Refer to Appendix A
TRIM LEADING TRIM TRAILING TRIM BOTH TRIM	LTRIM RTRIM TRIM TRIM	
REVERSE	DB2 UDF	Refer to Appendix A

Table 3.

Wild-card processing and pattern matching

DB2 and MySQL both support the LIKE comparison operator for wild-card searches and pattern matching, including the percent sign (%) and the underscore (_) wild-card characters. MySQL supports additional proprietary matching capabilities with the REGEXP (or RLIKE) operator. DB2 for i has no direct equivalent for these proprietary operators. The article, *Bringing the Power of Regular Expression Matching to SQL*, explains some workarounds for these advanced pattern-matching capabilities (**ibm.com**/developerworks/db2/library/techarticle/0301stolze/0301stolze.html).

Date and time functions

The following table is a list of MySQL functions commonly used for processing with date and time values and how they map to functions on DB2 for I (see Table 4).

MySQL	DB2	Notes
EXTRACT(MINUTE FROM TIMESTAMP '2000-02-23 18:43:12')	MINUTE('2000-02-23 18:43:12')	DB2 also supports extraction of different units with MONTH, DAY, YEAR and HOUR functions
CURDATE()	CURRENT DATE	MySQL function returns the date value in YYYY-MM-DD or YYYYMMDD format
DayOfMonth('2004-02-23')	DAY('2004-02-23')	
WEEK	WEEK or WEEK_ISO	MySQL WEEK number starts at **0**; DB2 starts with **1**
WEEKDAY	DAYOFWEEK_ISO - 1	MySQL day range is from 0(Mon) to 6(Sun), DB2 range is **1** (Mon) to **7** (Sun)
FROM_DAYS(n)	DATE(n-365)	MySQL and DB2 have different base years
FROM_UNIXTIME	DB2 UDF	
TO_DAYS('1997-01-14')	DAYS('1997-01-14') + 365	DB2 DAYS function does not support numeric date value
DATE_FORMAT	DAYNAME, MONTHNAME and other date-related functions	DB2 function usage depends on the format parameter for the MySQL function
LOCAL TIME, LOCAL TIMESTAMP, NOW()	CURRENT TIMESTAMP	MySQL functions return timestamp string in `YYYY-MM-DD HH:MM:SS` or `YYYYMMDDHHMMSS` format
HOUR(TIME '11:14:23')	HOUR(TIME('11:14:23'))	Small syntax difference
DATEDIFF('2007-10-11', '2007-01-09)	TIMESTAMPDIFF(16, (TIMESTAMP_ISO(DATE('2007-10-11')) – TIMESTAMP_ISO(DATE('2007-01-09'))))	The DB2 function returns the estimated number of days, which can be different than the output of the MySQL function.

Table 4: List of MySQL functions

DATE and TIME arithmetic

MySQL supports adding and subtracting values from date and time values with arithmetic operators or functions (DATE_ADD, DATE_SUB, PERIOD_ADD, PERIOD_DIFF). DB2 for i only supports the usage of arithmetic operators. Here are some examples of converting the MySQL date arithmetic to the DB2 equivalent. You must convert both forms of MySQL date arithmetic because DB2 for i does not require the INTERVAL keyword.

MySQL: `SELECT DATE_ADD(datecol, INTERVAL 7 DAY),  datecol + INTERVAL 7 DAY FROM t1`

DB2: `SELECT datecol+ 7 DAYS,  datecol + 7 DAYS FROM t1`

Other functions and operators

Here is a list of other MySQL functions and their suggested equivalent on DB2 for I (see Table 5):

MySQL	**DB2**	**Notes**
ROUND(col1)	INTEGER(ROUND(col1,0))	MySQL function always rounds to an integer value
Bitwise Shifts (<< & >>) X >> Y X << Y	(X / POWER(2,Y)) (X * POWER(2,Y))	
BIT_AND	BITAND	
BIT_COUNT	DB2 UDF	Refer to Appendix A
1 AND 1 , 1 && 1	INTEGER(LAND(1,1))	Logical AND
1 OR 0, 1 || 0	INTEGER(LOR(1,0))	Logical OR
XOR	BITXOR	
POW	POWER	
LOG(m,n)	LOG(m) / LOG(n)	
LEAST	MIN scalar function	
GREATEST	MAX scalar function	

Table 5: Other MySQL functions

INSERT statement

The MySQL INSERT statement supports some proprietary extensions to consider when porting to DB2 for i.

DELAYED clause

The DELAYED keyword provides support for asynchronous insert operations. If an insert statement from a client contains the DELAYED keyword, the MySQL server queues the insert request and returns back to the client immediately. If the MySQL server terminates unexpectedly, then the rows queued for inserting might be lost. DB2 does not support this proprietary behavior.

ON DUPLICATE KEY UPDATE clause

The DUPLICATE key clause contains an UPDATE operation that must be performed if the INSERT request causes a duplicate key violation. If a duplicate key error is signaled for a primary key or unique index, MySQL performs the specified update operation instead. Here is an example that shows this behavior:

```
INSERT INTO t1 (cola,colb,colc)  VALUES(1,’AA’,10)
     ON DUPLICATE KEY UPDATE colc = colc+10
```

If `cola` is defined as the primary key and already contains a value of **1**, MySQL performs the following update statement instead of the insert:

```
UPDATE t1 SET colc=colc+10 WHERE t1=1
```

On DB2, you have to code a MERGE statement. Another possibility is coding the INSERT and UPDATE statements separately, with the application first checking for a duplicate-key condition on the INSERT and then conditionally performing the specified update function. Depending on the logic, it might be possible to embed the conditional UPDATE statement into a Before Insert trigger on DB2.

UPDATE statement

The UPDATE statement support in DB2 for i and MySQL is similar. However, there is a significant syntax difference when the UPDATE statement references multiple tables. Here is an example of how to convert the proprietary MySQL syntax to DB2:

MySQL:
```
UPDATE items,month SET items.price=month.price
          WHERE items.id=month.id
```

DB2:
```
UPDATE items SET items.price = (SELECT m.price FROM items i, month m
              WHERE i.id = m.id)
                  WHERE EXISTS (SELECT *  FROM items i, month m
                                                WHERE i.id = m.id)
```

DELETE statement

The MySQL DELETE statement can delete rows from multiple tables on a single DELETE statement (for example, DELETE t1, t2 FROM t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id). However, the DB2 for i DELETE statement can only delete rows from a single table. Depending on the application requirement, it might be possible to simulate the desired behavior on DB2 by creating a DELETE trigger on the first table or using a Cascade Delete rule on a foreign key constraint.

Stored procedures, triggers, user-defined functions and views

MySQL did not add support for SQL procedure, triggers and user-defined functions (UDFs) until recently with MySQL v5.1. Thus, this paper does not discuss porting these object types containing business logic because very few existing MySQL databases and applications use these objects.

The entire IBM DB2 family of database products supports the SQL stored-procedure language. This support is based on the ANSI/ISO Persistent Stored Module (PSM) standard specification in the 2003 SQL standard. The MySQL support for these objects is similar to the standard, but contains some differences. One major difference is how stored procedure result sets are returned. When running a **SELECT** statement, MySQL returns a result set by default to the procedure invoker. On DB2 and in the SQL standard, a cursor has to be defined explicitly to return a result set.

For more details, refer to IBM Redbooks entitled, *Developing PHP Applications for IBM Data Servers*, which you can find at www.redbooks.ibm.com/abstracts/sg247218.html?Open.

MySQL was also missing support for views until the most recent release. You can consult the same Redbooks document for more details on the MySQL proprietary extensions for views.

Constraints

The constraint support for DB2 for i and MySQL is similar. However, there are differences:

- For foreign-key constraints, DB2 for i only supports the No Action and Restrict Update rules.
- For foreign-key constraints, MySQL supports the No Action and Restrict Update rules and also supports the Cascade and Set Null Update rules.
- MySQL constraints are only supported by those tables that use the InnoDB storage engine. For other storage engines, the foreign key constraints are ignored.
- Check constraints that are defined with the Check keyword are always ignored.

- DB2 for i automatically creates an index for use by its database manager in enforcing the constraint when the following types of constraints are created:
  - Primary key
  - Foreign key
  - Unique key

Unsupported features

DB2 for i does not yet support some MySQL features or capabilities, or it is extremely difficult to support or devise a DB2 for i circumvention. Two features (spatial data types and spatial indexes) might require rewriting or redesigning portions of the application.

Application development

The majority of ported MySQL applications are programmed in PHP. As mentioned earlier, the Zend Server Community Edition for IBM i distribution contains extensions that allow PHP applications to run on IBM i and access data stored in DB2 for i. Instead, MySQL Java™ applications are obviously converted to use one of the DB2 for i JDBC interfaces discussed earlier in this paper.

With MySQL, several different interfaces can be used for data access from a PHP application. The following table documents these MySQL interfaces, along with the recommended DB2 for i interface when porting the MySQL application to DB2 for i.

MySQL interface	**Recommended DB2 interface**	**Comments**
mysql functions	ibm_db2	
mysqli functions	ibm_db2	
Unified ODBC	Unified ODBC	Not recommended for either database

You can use the following two main PHP extensions for PHP applications to access DB2 for i applications:
- ibm_db2
- Unified ODBC

For the best performance and function, IBM recommends using the ibm_db2 extension. The Redbooks document, *PHP: Zend for i5/OS*, has information on installing and using this extension with DB2 for i. This Redbooks document can be accessed online at: www.redbooks.ibm.com/abstracts/sg247327.html.

Documentation for the ibm_db2 extension is available at: http://us3.php.net/manual/en/book.ibm-db2.php.

mysql and mysqli function mapping

The following table shows a mapping of select `mysql` and `mysqli` functions with the `ibm_db2` functions.

mysql functions	**mysqli functions**	**ibm_db2 function**
mysql_connect	Mysqli_connect	db2_connect
mysql_pconnect	Mysqli_pconnect	db2_cponnect
mysql_close	mysqli_close	db2_close
mysql_query	mysqli_query	db2_exec
		db2_prepare

		db2_execute
mysql_fetch_*	mysqli_fetch_*	db2_fetch_*
mysql_errno	mysqli_connect_errno	db2_conn_error
mysql_error	mysqli_connect_error	db2_conn_errormsg
mysql_error	mysqli_stmt_errno	db2_stmt_error
mysql_error	mysqli_stmt_error	db2_stmt_errormsg

Although the mapping of these MySQL functions to ibm_db2 functions are rather straightforward, there are differences in the implementation. The Redbooks document *Developing PHP Applications for IBM Data Servers* document some of these key differences. You can access this document online at: www.redbooks.ibm.com/abstracts/sg247218.html

Performance

The Redbooks document *PHP: Zend for i5/OS* has several performance recommendations in the Database Access chapter. Review these before converting a PHP MySQL application to DB2 for i.

Concurrency and recovery

There are differences between MySQL and DB2 for i when in the areas of concurrency and locking that you need to consider when porting an application.

Locking

The level of locking (for example, row- or table-level locks) used on MySQL depends on the storage engine used for the table involved in the SQL request. On DB2 for i, it is typical to use row-level locks, with the DB2 for i engine automatically escalating locking to a table level on rare occasions.

The following table compares the locking capabilities of DB2 tables with the MySQL table types.

DB2 table	MySQL ISAM table	MySQL InnoDB table	MySQL BDB table
Row and table level	None or table level	Row and table level	Page and table level

Both MySQL and DB2 for i support a LOCK TABLE statement for explicit table-level locks, but the syntax differs.

Transaction control and isolation levels

Some MySQL interfaces require that you start a transaction with a BEGIN or START TRANSACTION statement. No such statements exist on DB2 for i; the database manager automatically starts a new transaction at the beginning of a connection or after a COMMIT and ROLLBACK statement.

On MySQL, only table types of InnoDB and BDB can participate in an atomic transaction. By default, MySQL runs with AutoCommit enabled. AutoCommit is also **on** by default for the ibm_db2 interface for DB2 for i. Other DB2 for i programming interfaces have different default values for the AutoCommit setting, refer to the DB2 documentation listed in Appendix B to find the default AutoCommit setting.

The default isolation-level setting for DB2 for i is also interface-dependent. No Commit is the default isolation level for the ibm_db2 interface to DB2 for i. Repeatable Read is the default isolation level for

MySQL. MySQL supports three other isolation levels: Read Uncommitted, Read Committed and Serializable. In addition to No Commit, DB2 for i supports the same four isolation levels as MySQL.

Implementing these isolation levels is different; thus, the application can experience different behavior in concurrent environments. One difference is MySQL multiversioning support (known as *consistent read*) that enables a query to read a snapshot of the database at that point in time. The query can see changes made by those transactions that committed before that point of time, but cannot access changes made by later or uncommitted transactions. On DB2 for i, the default behavior is for the query to wait for the release of the lock on the changed row. If the changes are not committed, the operation fails with a lock-timeout error. Applications dependent on this multiversioning behavior can use the USE CURRENTLY COMMITTED concurrent access resolution support in DB2 to mimic this behavior.

Transactions using the CS isolation level can specify a USE CURRENTLY COMMITTED behavior at a system, connection or statement level. When a reader encounters a row locked for update or delete by another transaction, the CURRENTLY COMMITTED concurrent access resolution behavior causes DB2 to scan the journal receiver for a previously committed version of the row to use instead of waiting on the row-level lock. If a previously committed version of the row is not found in the journal receiver, the reader waits for the update lock to be released. Rows in the process of being inserted by other transactions are skipped when employing the CURRENTLY COMMITTED resolution.

DB2 also supports SKIP LOCKED DATA concurrent-access resolution behavior as another method to increase application concurrency. When specifying the SKIP LOCKED DATA clause, DB2 skips the candidate row (does not wait for the share-row lock to be acquired). The skipped row is not included in the final result set. SKIP LOCKED DATA support is available for isolation levels except for RR. In the following list, the DB2 isolation levels are highlighted in bold and the ANSI and ISO term for that level is in italics. The list is ordered from highest to lowest isolation level:

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

For best performance, use the lowest isolation-level settings possible; but, do not change the isolation level frequently. You can 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 DB2 for i default isolation level can be interface-dependent; therefore, it is best for the application to set the isolation level explicitly. There are several ways to specify the isolation level:

- Use the options parameter on the db2_connect extension when accessing from PHP.
- Use the SET TRANSACTION statement to temporarily override the default isolation level within a unit of work. When the unit of work ends, the isolation level returns to the value it had at the beginning of the unit of work.
- Use the isolation clause on the SELECT, SELECT INTO, INSERT, UPDATE, DELETE, PREPARE and DECLARE CURSOR statements to override the default isolation level for a specific statement or cursor temporarily.
- Use the connection attributes or data-source settings for ODBC, JDBC and CLI applications.
- Use the SET OPTION statement to specify the default isolation level within the source of a procedure, function, module or program that contains embedded SQL.
- Use the COMMIT parameter on the CRTSQLxxx and RUNSQLSTM commands to specify the default isolation level.

Database recovery and journaling

MySQL and DB2 for i both implement logging (journaling) to aid in database recovery. DB2 for i logs all changes in its journal and journal receiver objects, including the old version of the data.

You can use DB2 for i journals to recover changes made to a specific table or schema or all objects involved in the logged transaction. Also, an IBM i application or administrator can actually view and use the data stored in the journal. However, database objects involved in a transaction can be logged to a different journal. To facilitate easy recovery, log all objects in a transaction to the same journal.

You can use the IBM i ENDJRNPF CL command to turn off journaling for database objects that do not need recovery capabilities. This improves performance — at the expense of transaction recovery.

You can find more information on DB2 for i journaling and recovery in the *SQL Programming Concepts and Backup and Recovery Guides*. (See Appendix B for a link to these guides.)

After the port

Just as with any software project, database porting is not complete at the point that all the code is converted and compiled successfully. You must perform thorough testing and performance tuning after moving the database to DB2 for i.

Functional testing

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

This functional testing phase needs to include error-recovery and concurrent user 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 makes IBM Power Systems hardware available for IBM i functional testing to developers and integrators through its Power Development Platform. Refer to **ibm.com**/partnerworld/pdp for 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 details.

Performance tuning and sizing

Often, the biggest hurdle to overcome in database porting projects is the tuning of the application with the new database. Each database engine has its strengths and weaknesses from a performance point of view, especially when comparing query optimizers. The MySQL query optimizer might work great with a specific set of indexes over the database, although the DB2 for i optimizer might require additional indexes to be present. The opposite holds true, as well. The DB2 for i cost-based query optimizer does not require the manual collection of database statistics. The database manager automatically maintains and updates statistics (inside the table and index objects) as changes are made to the objects.

Because each DBMS query optimizer tries to exploit its target system and hardware fully, each database engine has a unique performance personality. Therefore, it might be necessary to tune applications as you move them to a different database server.

As mentioned, you do not need to perform low-level performance-tuning tasks on the IBM i platform; and there is no need to place tables on specific drives or configure buffer pools and caches.

A common problem in the design of applications ported from other databases is that the 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 PHP db2_exec function or the JDBC Statement class to run the same SQL request multiple times within a connection. DB2 for i typically does not perform well with either of these inefficient program models.

DB2 for i includes several performance tools, including the System i Navigator SQL Performance Monitor and Visual Explain tools. (The **Administration** section of this paper describes these tools). In general, though, to use these tools effectively and to efficiently tune DB2 for i, you must attend the DB2 for i SQL Performance Workshop (**ibm.com**/systems/power/software/i/db2/education/performance.html). Some performance-tuning information is also available in the *Database Performance and Query Optimization* book. Refer to Appendix B for a link to this book.

In addition to the IBM performance tools, more advanced DB2 for i performance-tuning and management tools are available from Centerfield Technology. (Appendix B provides a link to this site.)

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

The IBM i Performance and Scalability Center is a great resource for helping performance test and size the hardware required by your application. Visit the following web site for details: **ibm.com**/systems/ services/labservices/psscontact.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 them are specifically designed for DB2 for i. An engagement with the Benchmark Center is the safest and most accurate approach.

Administration

As discussed in the Architecture section, administrative controls and tasks are quite different for DB2 for i. Several administrative functions are unnecessary because the DB2 database manager and the IBM i operating system automatically handle the tasks. For instance, DB2 for i does not provide a statistics-collection utility for optimizer statistics, because its database manager always keeps these statistics current. Similarly, there also is no concept of table spaces or low-level storage management (for example, manually spreading data across drives) in DB2 for i.

System i Navigator

System i Navigator (formerly *iSeries Navigator*) is the graphical interface into DB2 for i for database administrators and application programmers. From the System i Navigator interface, you can perform almost all database-related tasks, ranging from tuning performance to journal management. Here is a quick overview of some of the System i Navigator capabilities:

- **SQL Script Center:** Develop and run SQL scripts.
- **SQL Performance Monitors:** Collect and analyze database Performance Monitor data for query optimization issues and performance of specific SQL requests.
- **Visual Explain:** Analyze a graphical representation of the access plan generated by the 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 a basic graphical modeling of existing database definitions.

MySQL CLP provides command-line access to the database server. Although you can use CLP with DB2 for i, the interactive IBM i SQL utility (STRSQL) and System i Navigator SQL Script Center are the recommended choices.

The System i Navigator SQL Script Center has no scheduling capabilities (as does the Script Center that is part of DB2 Control Center). You can use the IBM i job scheduler for scheduling activities.

Utilities

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

Import

The IBM i platform has an import command, just as MySQL does. DB2 for i imports are normally done with the IBM i CPYFRMIMPF CL command. This command can import files containing delimited data or fixed-format data. You must use the MySQL mysqldump tool to get the data into one of these formats. The CPYFRMIMPF command assumes that the specified input file does not contain any column-definition information.

The IBM DB2 MTK does generate data-transfer scripts using the CPYFRMIMPF command; therefore, it is a good tool when the MySQL port to DB2 for i also involves moving data.

You must manually perform the migration of binary data stored in MySQL BLOB columns.

In addition, you can use the IBMDB2I storage engine to help migrate the MySQL data to DB2 for i objects. The IBMDB2I storage engine, discussed earlier, creates and uses DB2 for i tables to store the data passed in through the MySQL interfaces. Thus, the migration of MySQL table definitions and data can be done by creating a MySQL database with the IBMDB2I storage engine. After the MySQL database tables are created and populated, applications can just use the DB2 for i tables created by the storage engine. The MySQL database instance that is associated with the IBMDB2I storage engine can be removed by deleting the corresponding IFS directory that MYSQL uses. However, this deletion step is not required.

Show statements

MySQL provides many types of SHOW statements to list information about databases, tables, columns or status information about the server. DB2 for i does not support any SQL statements that provide this metadata information. Instead, you must write queries against the system catalogs, as shown in Table 6.

MySQL	**DB2 equivalent**
SHOW DATABASES	Not applicable

SHOW TABLE	SELECT table_name,table_schema FROM QSYS2.SYSTABLES
	or
	SELECT table_name,table_schem FROM SYSIBM.SQLTABLES

SHOW INDEX FROM *mytable*	SELECT index_name FROM QSYS2.SYSINDEXES
	WHERE table_name='MYTABLE'

Table 6: DB2 equivalent of MySQL statements

The ibm_db2 extension does provide several functions such as db2_server_info and db2_foreign_keys that you can use to return metadata information to the application.

Table-maintenance statements

MySQL has a number of SQL statements (ANALYZE, CHECK, OPTIMIZE, REPAIR) that you can use to perform lower-level table maintenance tasks (such as: identify and repair table corruption or collect information for the query optimizer). These types of statements do not exist on DB2 for i because the DB2 database manager and the IBM i operating system automate many of the lower-level administrative and maintenance tasks.

Alter table

Although MySQL and DB2 for i both have an ALTER TABLE statement, there are differences in what you can change. You can use the ALTER TABLE statement on both databases to change the attributes (for example, type or length) for an existing column, drop a column and add a new column. MySQL offers additional capabilities on the ALTER statement to rename a column within an existing table.

It is also possible to use the MySQL ALTER TABLE statement to add and drop indexes from a table. With the DB2 for i ALTER TABLE support, you can only add or drop constraints.

Indexes

Both MySQL and DB2 support the creation of indexes to provide the query optimizer with methods to speed query processing tasks such as row selection and joins. The MySQL CREATE INDEX statement supports two types on indexes, BTREE or HASH. DB2 for i also supports two index types. The default, binary radix, is quite similar to the MySQL BTREE type and the Encoded Vector Index (EVI) structure on DB2 is a good match to the MySQL HASH index type.

MySQL allows indexes to be created on the leading part of character-key columns, as demonstrated with the following SQL statement:

```
CREATE INDEX ix1 ON customer (last_name(10))
```

On DB2 for i, you can create a similar index by using the substring function in the key definition, as shown in the following index example:

```
CREATE INDEX ix1 ON customer (SUBSTRING(last_name,1,10))
```

Summary

The purpose of this paper is to highlight potential issues in porting databases and applications from MySQL to DB2 for i. It is a living paper. As there is more experience with different porting situations, the paper will grow accordingly. If the reader encounters an issue that is not covered in this paper, contact the owner, Kent Milligan, at rchudb@us.ibm.com. Future readers can benefit from the new information.

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

Appendix A: User-defined function examples

This appendix includes examples of UDFs that you can use to simulate MySQL functions where an equivalent DB2 for i function does not exist. DB2 for i supports both user-defined table functions and user-defined scalar functions. Here is a summary of the DB2 for i UDF support.

- **Sourced UDF:** This is a reuse of the implementation of an existing function, usually with some attributes changed, such as input data types. A sourced UDF can also be used as a form of alias. If there is a function in MySQL (for example, *MYSFUNC*), that has an equivalent called *DB2FUNC* in DB2 for i, you can create a sourced UDF (called *MYSFUNC***)** in DB2 for i, invoking the DB2FUNC function; you do not need to change MySQL statements that call MYSFUNC.
- **SQL-bodied UDF:** You can fully implement this UDF type with SQL procedural language (PSM).
- **External UDF:** You can use this type of UDF when it is easier to implement the function with a high-level programming language. You can write an external UDF in any high-level programming language supported on the IBM i operating system, including Java.

UDF performance considerations

If performance is more important than maintaining a single code base, then you might want to avoid using UDFs in a porting effort, especially when the same function can be implemented inline with a series of calls to IBM i system-supplied SQL functions. Specifying the NOT FENCED and DETERMINISTIC attributes when creating functions can dramatically improve the performance of DB2 for i UDFs.

Reverse UDF example

This sample DB2 UDF simulates the MySQL Reverse function that returns a string with the order of characters reversed.

```
CREATE FUNCTION REVERSE(INSTR VARCHAR(4000))
RETURNS VARCHAR(4000)
LANGUAGE SQL
DETERMINISTIC
NO EXTERNAL ACTION
NOT FENCED
BEGIN ATOMIC
DECLARE REVSTR,RESTSTR VARCHAR(4000) DEFAULT '' ;
DECLARE LEN INT;

IF INSTR IS NULL THEN
   RETURN NULL;
END IF;

SET (RESTSTR,LEN)=(INSTR,LENGTH(INSTR));

WHILE LEN >0 DO
  SET (REVSTR,RESTSTR,LEN)=
   (SUBSTR(RESTSTR,1,1)CONCAT REVSTR,SUBSTR(RESTSTR,2,LEN -1),LEN -1);
END WHILE;
RETURN REVSTR;
END;
```

Format UDF example

This sample DB2 UDF simulates the MySQL Format function that formats the input number to a format such as **`'#,###.##'`**, rounded to *D* decimal places, and returns the result as a string.

```
CREATE FUNCTION FORMAT (X DECIMAL(31,10), D Integer)
RETURNS VARCHAR(50)
LANGUAGE SQL
NO EXTERNAL ACTION
DETERMINISTIC
NOT FENCED
BEGIN ATOMIC
DECLARE XN DECIMAL(21,0);
DECLARE RetVal VARCHAR(50);

SET RetVal =SUBSTR(CHAR(MOD(ABS(X),1)),22,D+1);
SET XN =ABS(X);

Main_Loop:
WHILE XN >0 DO
   SET RetVal =SUBSTR(CHAR(MOD(XN,1000)),19,3) || RetVal;
   SET XN =XN/1000;
   IF XN >0 THEN
      SET RetVal =','||RetVal;
    ELSE
       LEAVE Main_Loop;
   END IF;
END WHILE;

RETURN
   CASE WHEN X <0 THEN '-'ELSE '' END ||
                TRANSLATE(LTRIM(TRANSLATE(RetVal,'','0')),'0','');
END;
```

SUBSTRING_INDEX UDF example

This sample DB2 UDF simulates the MySQL Substring_Index function that returns the substring from input string before counting occurrences of the delimiter.

```
CREATE FUNCTION Substring_Index(i1 VARCHAR(2000), delimit VARCHAR(200), n INT)
RETURNS VARCHAR(2000)
LANGUAGE SQL
DETERMINISTIC
NOT FENCED
NO EXTERNAL ACTION
BEGIN ATOMIC
DECLARE dem VARCHAR(2000);
DECLARE num INT;
DECLARE pos INT;
DECLARE temp VARCHAR(2000);
SET dem=delimit;
SET temp=i1;
SET num=n;
SET pos=1;

IF (num < 0) THEN
  WHILE (LOCATE(delimit,temp) != 0) DO
    SET temp = SUBSTR(temp,LOCATE(delimit,temp)+1);
    SET num = num+1;
  END WHILE;

  SET num=num+1;
  SET temp=i1;
END IF;

WHILE (num > 0) DO
  SET pos = pos + LOCATE(delimit,temp)-1;
  SET temp = SUBSTR(temp, LOCATE(delimit,temp)+1);
  SET num = num-1;
END WHILE;

IF (n > 0) THEN
 RETURN SUBSTR(i1,1,pos);
ELSE
 RETURN SUBSTR(i1,pos+1);
END IF;

END;
```

BIT_COUNT UDF example

This sample DB2 UDF simulates the MySQL BIT_COUNT function that returns the number of set bits in the input parameter, assuming that the parameter is a 32-bit integer value.

```
CREATE FUNCTION BIT_COUNT(n1 INTEGER)
RETURNS INTEGER
LANGUAGE SQL
NO EXTERNAL ACTION
DETERMINISTIC
NOT FENCED
RETURN
  WITH RepeatExp(s,m1,ans) AS (
     (SELECT 0, n1,0 FROM sysibm.sysdummy1
        UNION ALL
      SELECT s+1, m1/2, ans+MOD(m1,2)
      FROM RepeatExp
      WHERE m1<>0 AND s<32) )
  SELECT CASE WHEN (ans>0) THEN ans ELSE ans+32 END
  FROM RepeatExp
  WHERE s =(SELECT MAX(s) FROM RepeatExp);
```

Appendix B: Resources

These websites provide useful references to supplement the information contained in this paper:

- IBM i Knowledge Center
  ibm.com/support/knowledgecenter/ssw_ibm_i_72
- DB2 for i online manuals
  ibm.com/systems/power/software/i/db2/docs/books.html
- Performance Management on IBM i
  ibm.com/systems/power/software/i/management/performance/index.html
- IBM Redbooks
  ibm.com/redbooks
  - *Stored Procedures, Triggers and user-defined functions on DB2 for i*, SG24-6503
  - *Advanced Database Functions and Administration on DB2 for i*, SG24-4249-03
  - *Diagnosing SQL Performance on DB2 for i*, SG24-6654
  - *Preparing for and Tuning the SQL Query Engine on DB2 for i*, SG24-6598-01
  - *The Ins and Outs of XML and DB2 for i (SG24-7258)*
  - *OnDemand SQL Performance Analysis … in V5R4* (SG24-7326)
  - *DB2 for AS/400 Object Relational Support*, SG24-5409
- DB2 for i home page
  ibm.com/systems/power/software/i/db2/index.html
- DB2 for i Performance Workshop
  ibm.com/systems/power/software/i/db2/education/performance.html
- IBMDB2I MySQL Storage Engine Resources
  ibm.com/partnerworld/wps/servlet/ContentHandler/RBAA-7YF7YK
  ibm.com/redbooks/abstracts/sg247705.html
- IBM education resources and white papers
  ibm.com/systems/power/software/i/db2/education/index.html
  ibm.com/partnerworld/wps/training/i5os/courses
  ibm.com/partnerworld/wps/whitepaper/i5os
- *What does DB2 on IBM i really mean?*
  ibm.com/partnerworld/wps/servlet/ContentHandler/SROY-6UZDN4
- IBM i Performance and Scalability Center
  ibm.com/systems/services/labservices/psscontact.html
- IBM Power Development Platform
  ibm.com/partnerworld/pdp
- Performance Management on IBM i
  ibm.com/systems/i/advantages/perfmgmt
- IBM Innovation Center
  ibm.com/isv/iic

Conversion services

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

Online forum

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

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)

Vendor performance tools

- Centerfield Technology
  www.insuresql.com

@IBMSystemsISVs

Trademarks and special notices