DB2 for i porting guide

Informix to the IBM i platform

Version 8.0

The database technology team
ISV Business Strategy and Enablement
July 2014

IBM

Table of contents

Abstract

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

The audience for this paper

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

This paper concentrates primarily on the database differences between the Informix and DB2 for i database servers. It does not focus on the differences between the underlying operating systems. Application development issues are covered only from a database perspective. Refer to the **Resources** section of this paper for a website listing that provides a more general perspective on porting applications to the IBM Power Systems and the IBM i (System i™ System i5™, eServer™ i5 and iSeries™) platform.

Assumptions

The paper assumes readers are working with Informix IDS 12 and are porting to DB2 for i 7.2 release. For simplicity, this paper uses DB2 to represent DB2 for i.

The paper also assumes readers are familiar with the concepts of RDBMSs and with Informix SQL. It is also assumed that readers have easy access to DB2 for i 7.2 documentation. Refer to that documentation for detailed information about the actual SQL statement syntax. More information regarding DB2 for i can be found online at the following website: **ibm.com**/systems/power/software/i/db2/docs/books.html

Preparing to port

Before diving into a detailed comparison of DB2 and Informix, 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 Informix application enables you to be more productive as you use this paper to assess the port to DB2. Here are some questions that you need to be able to answer about your application and database:

- Does the application use standard or proprietary SQL such as the Informix DataBlade modules?
- 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 logic reside in procedural database objects (triggers and procedures) or the application?

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

Porting approaches

After you have a thorough understanding of the source database, it is time to consider the approach that you want to take when porting your application to DB2. Any solution can be ported, given enough time and money. However, the 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, then this is a relatively easy question to answer.

With its support for industry-standard interfaces, DB2 can accommodate this design with minor changes.

Design trade-offs

However, if your application currently supports just a single DBMS product, you have some application design issues to investigate further. The first alternative to consider is redesigning your application with an abstraction layer to support database servers more easily with a single code base. This design approach requires more effort and investment, but 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 the source code to be maintained must also remain portable, it is best to isolate all database runtime calls from the application. Usually, this isolation is done by having the application create its own set of database methods or calls that are passed to a single procedure or module. That procedure or module then turns the application database request into the specific format or API (ODBC, JDBC, or other interface) supported by the target database.

Another alternative is to create a separate version of your application for DB2 for i. This can require less up-front investment than the previous approach, but the long-term expenses are greater because of maintaining, enhancing and testing two code sets. The benefit is that the best performance is obtained by changing the application and database to exploit the target server. However, the more changes made, 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 the tradeoff when you code to the lowest common denominator. For

example, Informix has a scalar function called **X** and DB2 has an equivalent function called **Y**. The developer who converts to DB2 has a choice: change the application to call function **Y** (to yield the best performance) or code a DB2 user-defined function (UDF), called **X** (to merely call function **Y)**. This allows the application code to remain unchanged, but results in slower performance.

These porting issues need to be discussed before the database porting project starts, to ensure that all of the parties on both sides agree on the priority (best performance or easiest code maintenance).

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

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

Misused porting approach

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

This approach works if, you first verify that DB2 for i supports the SQL statement and syntax (see **ibm.com/**partnerworld/wps/servlet/ContentHandler/SROY-6UZDN4 for a listing that provides a high-level examination of DB2 for i and the IBM DB2 Family.

Porting tools

IBM offers an Informix Migration Toolkit to assist in porting Informix solutions. This toolkit automates many steps in migrating a database from Informix to DB2 for i (tables, views, indexes, triggers and stored procedures, see **ibm.com**/partnerworld/i/db2porting for this toolkit). Although the DB2 Migration Toolkit greatly reduces the time it takes to convert from Informix to DB2, but does not completely automate the process. For example, the toolkit does not convert Informix catalog references to the corresponding DB2 catalog references; you must perform some manual work. However, the toolkit flags items that must be converted manually. The converted procedural objects (triggers, functions and stored procedures) need to be reviewed to ensure their semantic equivalence. In addition, the generated SQL code might not result in the best-performing SQL statements. Thus, you must do some performance tuning after using the toolkit. If your application has dependencies on proprietary features that existing migration tools do not support, you might consider building your own migration tool. This approach involves a significant upfront time investment and requires an indepth knowledge of the source and target databases.

Sizing the port

Although the design approach and usage of porting tools definitely affects the costs required to complete the port, the effort is also dependent on the features used in the source database. This section highlights and compares some key differences encountered between Informix 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 Informix database product. In addition, the interfaces used with Informix and DB2 for i databases are also compared.

Brief history

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

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

Because of this native, non-SQL interface, some IBM i users and consultants might use terminology that is not familiar to those coming from an SQL background. Table 1, below, provides a mapping of that terminology:

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

Table 1: Mapping of SQL and IBM i terms

Because of the integrated nature of the IBM i database, both the native and SQL interfaces are almost completely interchangeable. Objects created with DDS can be accessed with SQL statements. Also, objects created with SQL can be accessed with the native record level access APIs. The DB2 SQL interface is compliant with the core level of the SQL 2008 standard.

Interfaces and packaging

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

- SQL parser and run-time support
- SQL interfaces (server side)
  - CLI (call level interface)
  - Native JDBC driver (Version 4.0)
  - SQLJ
- IBM i Access for Windows (formerly IBM Client Access Express and IBM iSeries Access)
  - ODBC (Version 3.5)
  - JDBC (Version 4.0)
  - OLE DB provider
  - .NET provider (Version 2.0)
- PHP ibm_db2 extension (Zend Server for IBM i)
- Open Group™ DRDA application requester and server
- SQL Statement Processors (**RUNSQLSTM** & **RUNSQL** CL commands)
- 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 needs to be installed only on the development system.

If your application already supports DB2 access through the middleware provided by DB2 Connect, another option is DB2 Connect Unlimited Edition for System i. This product is not included with the base operating system and must be purchased separately. For more information, see the **Resources** section of this paper.

Many vendors prefer using a native database interface instead of an industry-standard interface, such as ODBC or CLI, because of performance concerns. DB2 does not have a native SQL-based interface; instead the ODBC and CLI interfaces are treated as the native SQL interface for DB2 for i and provide performance similar to that of 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 via the **QSQPRCED** API or **XDA** API set. However, this interface is proprietary and can only be used effectively with a thorough understanding of the IBM i SQL engine. That knowledge can be obtained from the **DB2 for i SQL Performance** workshop (referenced in the **Resources** section of this paper).

Physical model

Frequently, Informix database design is based on creating multiple databases within one instance where all the databases are shared by the same applications. Often, the tables from the separate databases are joined and even have the same names. Although this design is suitable for Informix, in most cases this design is not optimal for DB2 for i. In fact, DB2 for i does not even support the **CREATE DATABASE** statement.

Although DB2 can support multiple databases on a single server with independent auxiliary storage pools, this design is not commonly used. The default configuration for DB2 for i is a single, system-wide database. Therefore, it is recommended that all Informix instances and associated databases be consolidated into a one system-wide database. For this reason, when consolidating multiple Informix databases that use the same table name across several databases, DB2 requires the application to use unique, two-part table names. DB2 tables can be made unique by adding schema names that correspond to the Informix database from which they are sourced. The application code must also be modified to reference the new table names. Alternatively, aliases can be created to point to the new names.

Storage model

With DB2 for i, all of the storage allocation and management is handled by the database manager and operating system. Thus, there are no table spaces to create or manage on the IBM i platform. In addition, the data for DB2 for i non-partitioned tables is automatically partitioned (or striped) across the disk devices (or chunks) to eliminate contention.

Thus, Informix fragments, chunks, dbspaces, buffers, configuration files and the Sysmaster database (which contains resource and storage information) have no equivalent on DB2 for i. A DBA does not need to allocate or manage these object types because the operating system and database engine automatically take care of this processing. In addition, the page size on DB2 is fixed and cannot be changed.

DB2 for i supports partitioned tables with partitioning by range and hash value. However, partitioned tables must only be used when the row or size limit for a single table is about to be exceeded. For details on the DB2 for i partitioned table implementation, consult the DB2 white papers at the website listed in the **Resources** section of this paper.

Metadata

Metadata provides the roadmap to interpret the information stored in the database. DB2 catalog views have a different structure from Informix system tables. Applications that are designed to access Informix system tables require rewriting when migrating to DB2.

Detailed descriptions of the catalog views can be found in an appendix of the **DB2 for i SQL Reference Guide**. The use of a CLI and ODBC programming interface can help in making structural differences (among the system catalogs of various database vendors) transparent to an application. DB2 for i as well as DB2 for Linux™, UNIX® and Windows both support a common set of catalog views for ODBC and JDBC clients in the **SYSIBM** schema.

Data types

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

IDS data type	Notes	DB2 data type	Notes
CHAR(n)	n < 32,767	CHAR(n) VARCHAR(n)	n <= 32766, **CHAR** n <= 32740, **VARCHAR**
VARCHAR	n <= 255	VARCHAR(n)	n <= 32740
LVARCHAR	n <= 2000	VARCHAR(n)	
NCHAR		NCHAR	
TEXT		CLOB VARCHAR	**CLOB** limit is 2 gigabytes.
INT8	10 bytes on disk	BIGINT	8-byte value
DECIMAL(p,s)	p <= 32 digits	DECIMAL(p,s)	p <= 63 digits
DECIMAL(p)		FLOAT, DECFLOAT	
SMALLFLOAT		REAL	
MONEY		Use DECIMAL	
ROWID		ROWID	
BYTE		BINARY VARBINARY BLOB VARCHAR for bit data	
BOOLEAN		Use CHAR(1)	
DATETIME		DATE TIME TIMESTAMP	
INTERVAL		Use DECIMAL	
SERIAL SERIAL8		INTEGER BIGINT	Use the **Identity** clause.

Table 2: Mapping of IDS data types to DB2

DECIMAL

In an Informix non-ANSI database, a **DECIMAL** data type without a scale, such as **DECIMAL(9)**, is represented internally as a **FLOAT** data type. DB2 does not support this variation on the **DECIMAL** data type. With DB2, a **DECIMAL** data type that is specified with a precision but without a scale, defaults to **DECIMAL(9,0)**. The DB2 DECFLOAT data type can also be used.

With Informix, when a value of **123.456** is inserted into a **DECIMAL(3,2)** column, Informix rounds the second decimal place value and stores **123.46**. With DB2, when a value of **123.456** is inserted into a **DEC(3,2)** column, DB2 truncates the second decimal place value and stores **123.45**.

NCHAR

NCHAR data type maps to the DB2 **NCHAR or GRAPHIC** data type, but there are differences.

The **NCHAR** data type can be used for single- or double-byte characters. The Informix server uses the codepage to determine whether the **NCHAR** string is required to store single- or double-byte data. With the DB2 **NCHAR** data type, DB2 always allocates double the length specified for the **NCHAR** data type and always stores the data in the UTF-16 double-byte Unicode format.

MONEY

DB2 supports the **DECIMAL** data type instead of a **MONEY** data type. If monetary formatting is needed, a DB2 user-defined function can be written to handle this requirement. Note that **MONEY(16)** maps to the DB2 **DECIMAL(16,2)**. The scale on the **MONEY** data type defaults to **2** if it is omitted.

ROWID

The DB2 **ROWID** type stores a 40-byte unique value that does not contain any information about the block or row. Although DB2 allows direct access to a row via its **ROWID** value, the access of this row is no faster than a **WHERE** clause that is implemented with an index scan. DB2 does automatically create a unique constraint on top of **ROWID** columns. Some DB2 for i solutions have also successfully used the **RRN** scalar function to simulate the **ROWID** data type.

DATETIME

The IDS **DATETIME** data type supports variable-length precision; the output formatting is controlled by setting the **DBDATE** value. Examine the following **DATETIME** formats:

yyyy-mm
yyyy-mm-dd hh
hh:mm:ss

Of these, only hh:mm:ss can be mapped directly to DB2 as a **Time** value. The first format requires a **DAY** value to complete a **DATE** function. The second format requires minutes and seconds to complete a **TIMESTAMP** function.

DB2 **TIMESTAMP** data type does not require that a value is specified for microseconds (defaults to 0. However, a value for **year**, **month**, **date**, **hour**, **minute** and **second** must always be provided. The nondesignated DB2 **timestamp** literal formats are:

'yyyy-mm-dd-hh.mm.ss.nnnnnn' , 'yyyymmddhhmmss' or .'yyyy-mm-dd hh:mm:ss'

INTERVAL

With Informix, **INTERVAL** values are specified as string literals and are stored internally as decimals. In DB2, the month and second intervals must be converted to the total number of months and seconds and stored in a decimal column. An example of an interval format is:

INTERVAL(123456 13:07:56) DAY(6) TO SECOND

This value is stored in Informix as the decimal number **123456130756**.

The Informix Migration Toolkit normalizes this value to seconds and represents the value as the number of seconds in an interval. Thus, the DB2 value in seconds for this literal is:

123456*86400 + 13*3600 +7*60 +56

To complete the example, compare the Informix **CREATE TABLE** statement with its DB2 translation:

Informix:

```
CREATE TABLE t1 (col1 INTERVAL DAY (6) TO SECOND
        DEFAULT INTERVAL (2  0:0:56) DAY (6) TO SECOND NOT NULL)
```

DB2:

```
CREATE TABLE t1 (col1 DECIMAL (20,5) DEFAULT 172856 NOT NULL)
```

SERIAL

The Informix **SERIAL** data types are mapped to DB2 using the **IDENTITY** attribute on a column with any number data type with a scale of zero. If **GENERATED ALWAYS** clause is used with the **IDENTITY parameter**, DB2 always supplies a value for the column.

If the **GENERATED BY DEFAULT** clause is used with the **IDENTITY** attribute, DB2 generates a value for the column, as long as no value is provided on the **INSERT** statement. However, this can have a different behavior than when using the **Serial** column on subsequent **Insert** statements. Consider the following comparison:

Use **ALTER TABLE** to reset the internal counter to the next number in sequence. This updates the internal counter so that the next value DB2 generates is in sequence with the last value inserted or loaded. See the example below:

DB2:

```
CREATE TABLE tab
    (c1 INTEGER,
     c2 INTEGER GENERATED BY DEFAULT AS IDENTITY);

INSERT INTO tab(c1) VALUES(5);              -- 1 generated for c2
INSERT INTO tab(c1,c2) VALUES (5,10);       -- No value generated, 10 assigned
INSERT INTO tab(c1) VALUES(5);              -- 2 generated for c2
```

IDS:

```
CREATE TABLE tab
  (c1 INTEGER, c2 SERIAL);

INSERT INTO tab(c1) VALUES(5);              --1 generated for c2
INSERT INTO tab(c1,c2) VALUES(5,10);        --internal counter = 10

INSERT INTO tab(c1) VALUES(5);              --11 generated for c2
```

To ensure that the DB2 **IDENTITY** column has the same behavior as the **Serial** column, run the following **ALTER TABLE** statement after the second **INSERT** statement to reset the **IDENTITY** column's generated value.

```
ALTER TABLE tab ALTER c2 RESTART WITH 11
```

When inserting values into a table with the **IDENTITY** column, use **DEFAULT** as a placeholder for the value of the **IDENTITY** column. This eliminates the requirement to list every column of the table in the **INSERT** statement. To retrieve the last value inserted into an **IDENTITY** column, use the **IDENTITY_VAL_LOCAL** built-in function.

Character types (including variable-length types)

DB2 supports the **CHAR** and **VARCHAR** data types for single-byte character strings and the NCHAR, NVARCHAR, NCLOB, **GRAPHIC** and **VARGRAPHIC** data types for double-byte character strings. Unicode® data is supported with the **GRAPHIC** and **VARGRAPHIC** data types with a CCSID value of 13488 for the UCS-2 encoding or a CCSID value of 1200 for the UTF-16 encoding. The national-character data types (NCHAR, NVARCHAR and NCLOB) can also be used to create a column with the UTF-16 encoding. The UTF-8 Unicode encoding is also available with the **CHAR** and **VARCHAR** data types with a CCSID value of 1208.

It is usually better to port variable-length columns to the fixed-length DB2 **CHAR(n)** data type, which is more efficient and uses less storage than **VARCHAR**. A general guideline is to use the **CHAR** data type for columns of 40 bytes or less. Before deciding on the column type based on performance, read the DB2 for i paging and I/O behaviors for variable-length and LOB columns described in this section.

In general, the DB2 variable-length data types (**VARCHAR** and **VARGRAPHIC**) should only be used for long-memo or text-description columns that are referenced infrequently. DB2 variable-length types usually cause additional disk I/O processing 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 magnified further 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 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 must be set to the maximum to ensure that the value is always stored in the fixed portion of the row.

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

Quoted character strings

Informix supports both single and double-quoted character strings. For example, each of the following search predicates is legal with Informix:

- WHERE color ='RED'
- WHERE color = "RED"

DB2 does not support double-quoted character strings. The Informix Migration Toolkit does translate this difference automatically.

Collection data types

DB2 for i does not support any of the Informix Collection Data Types (**SET**, **MULTISET** or **LIST).** Any table using these data types have to be redefined into multiple tables.

SQL language elements

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

Identifiers

DB2 for i supports two types of SQL identifiers: ordinary 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, 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. Table 3 provides a summary of the Informix object types that support a longer identifier name. All of the SQL limits are documented in the **DB2 for i SQL Reference**. A listing for this online manual is provided in the **Resources** section of this paper.

Object type	DB2 for i	IDS V9	XPS V8.5
User name	10	32	32
Cursor name	128	128	
Host identifier	64 or 128, depending on the language	128	128
Schema name	128	32	
Statement name	128	128	
Column name	128	128	128

Table 3: Informix object types that support a longer identifier name

Although DB2 supports long SQL identifiers, the commands and interfaces native to IBM i, there is a hard maximum of 10 characters for object names. For example, the **Save** commands that are used to back up database object, 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 a 10-character maximum are also known as system names.

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

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

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

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

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

Naming conventions and formats

Most 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 IBM i 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.

There are other settings that can impact the default search path, but this section describes the default behavior of each of the naming conventions. The **SQL Reference Guide** can be consulted for a full explanation. (See the **Resources** section of this paper.)

Synonyms

Informix supports private and public synonyms to access tables on a local or remote database.

With DB2, an alias is comparable to the IDS synonym. The user of the alias must have privileges to access the base table where the alias is created. The DB2 aliases can be created by using the **CREATE ALIAS** statement. Although not preferred, the **CREATE SYNONYM** statement is also recognized.

When creating an alias, the **Private** or **Public** keyword is not used in the syntax, as it is with IDS. All DB2 aliases are essentially **private and** can only be used by the owner or another user, as long as the owner of the alias is explicitly specified. The owner of an alias grants privileges to other users to use the same alias. A **public** synonym (supported by IDS) is one in which any user can use any synonym without specifying an owner's name. DB2 does not support this.

A DB2 alias can be created for a table or view that resides on a non-local database by using three-part naming as the following example shows:

```
CREATE ALIAS mylib.tab1  FOR rdb1a.mylib.tab1
```

String truncation with inserts

A non-ANSI Informix database supports truncation of strings that are longer than the columns into which they are inserted; no error is returned. An ANSI Informix database supports truncation of strings that are longer than the columns into which they are inserted; however, an error is returned. DB2 does not truncate strings that are longer than the columns into which they are inserted. Instead, an error is returned to the application.

Query syntax considerations

There are some syntax points to be aware of when migrating queries.

SELECT FIRST

Both IDS and DB2 enable you to specify that only **n** number of rows are to be returned by a SELECT statement. They use different placement and words to specify this however:

IDS: SELECT FIRST n ...
DB2: SELECT ...FETCH FIRST n ROWS

SELECT stored procedure calls

IDS supports the calling of a stored procedure from within a SELECT list:

```
SELECT marks_stored_procedure(col1) FROM t1
```

DB2 does not support the calling of a stored procedure from within a **SELECT** list. However, DB2 does support the calling of user-defined functions.

GROUP BY

Informix supports a **GROUP BY** clause with either the column name or an ordinal number that corresponds to the column in the **SELECT** list. Currently, DB2 only supports the column name and does not support an ordinal number (because use of the ordinal number in a **GROUP BY** clause is not part of the ANSI standard).

ORDER BY

Informix supports the ordering of a column by a substring notation. This is not supported by DB2. For example, the following clause instructs Informix to sort by the first two characters in the company name, instead of sorting the entire value.

```
ORDER BY company_name(1,2).
```

To accomplish the same operation with DB2, the **Substring** function must be coded on the **Order By** clause, as follows:

```
ORDER BY SUBSTR(company_name,1,2)
```

OUTER JOINS

As of IDS V9.2, Informix supports the same ANSI **Outer Join** syntax that is supported by DB2. Informix also supports a proprietary **Outer Join** syntax, as well. Here is an example of converting the proprietary **Outer Join** syntax to DB2.

IDS:

```
Select c.customer_num, c.lname, c.company, c.phone, u.call_dtime, u.call_descr,
        s.support_level
    From customer c, OUTER cust_calls u, OUTER support_packages s
        Where c.customer_num = u.customer_num AND c.customer_num = s.support_id;
```

IDS and DB2:

```
Select c.customer_num, c.lname, c.company, c.phone, u.call_dtime, u.call_descr,
        s.support_level
    From ((customer AS c LEFT OUTER JOIN cust_calls u
        ON c.customer_num = u.customer_num)
          LEFT OUTER JOIN support_packages s ON c.customer_num = s.support_id;
```

UPDATE WITH JOIN CONDITION (XPS)

XPS supports updating a column to a value of a column from another table, based on a join condition. For example:

```
UPDATE tab a
  SET a.col1 = b.col2
    FROM tab1 a, tab2 b
      WHERE b.coljoinb = a.coljoina
```

DB2 does not support this exact syntax. The following are two examples of how to code the same type of update with DB2:

```
UPDATE tab a SET a.col1 =
    CASE
      WHEN (SELECT b.col1 FROM tab2 b
                  WHERE b.coljoinb = a.coljoina) IS NOT NULL
                  THEN(SELECT b.col1 FROM tab2 b WHERE b.coljoinb = a.coljoina)
      ELSE a.col1
     END

UPDATE tab a SET a.col1 =
      (SELECT b.col1 FROM tab2 b WHERE b.coljoinb = a.coljoina)
  WHERE EXISTS (SELECT 'x' FROM tab2 bb WHERE bb.coljoinb = a.coljoina)
```

MINUS operator

The DB2 equivalent to the **MINUS** set operator is the **EXCEPT** operator.

MATCHES predicate

Informix supports both the **MATCHES** and **LIKE** predicates. DB2 only supports the SQL Standard **LIKE** predicate. (See Table 4.) Without escape characters, the Informix LIKE clause behaves differently than DB2 when processing a fixed-length character column. The search predicate, FixedCharCol LIKE 'A', returns a match on Informix when FixedCharCol equals 'A'. This is not the case with the DB2 LIKE clause, a match is only returned for this predicate if FixedCharCol is defined as a variable-length column.

	One of any character	None or many of any character
MATCHES	?	*
LIKE	_	%

*Table 4: The **MATCHES** and **LIKE** predicate*

Functions

Many of the functions supplied with Informix are also supplied with 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 Informix, but serve the same purposes and these cases can also be easily ported as sourced UDFs (see next). The remaining functions used in an Informix application are either: functions supplied with IDS that have no equivalent in DB2 or user-written functions. Both of these must be converted as UDFs to run 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 was a function in Informix, such as **IDSFUNC**, that had an equivalent called **DB2FUNC** in DB2, then a sourced UDF, called **IDSFUNC**, can be created in DB2, invoking the **DB2FUNC** function and Informix statements calling **IDSFUNC** do not need to be changed.
- **SQL-bodied UDF:** This 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. Creating the UDF as **NOT FENCED** and **DETERMINISTIC** can dramatically improve performance in some situations.

Table 5 provides a comparison of those Informix functions that are named differently from the equivalent DB2 function or that have no direct equivalent in DB2.

IDS function	DB2 equivalent	Comments
CARDINALITY	--	Need to create a UDF
CURRENT	CURRENT DATE or CURRENT TIMESTAMP	
DBSERVERNAME	CURRENT SERVER	
DECODE	CASE statement	
EXTEND	--	MTK supported
INITCAP	--	
LENGTH	LENGTH	IDS length does not include trailing spaces for fixed-length columns
MDY	--	
NVL	COALESCE or VALUE	Same function; however, all arguments must be the same data type on DB2
TODAY	CURRENT DATE	
TO_CHAR	CHAR or VARCHAR_FORMAT	These functions are a partial replacement. (See the **DB2 SQL Reference** for details.)
TO_DATE	DATE or TIMESTAMP_FORMAT	These functions are a partial replacement. (See the **DB2 SQL Reference** for details.)
WEEKDAY	DAYOFWEEK(date) - 1	**Sunday** on the DB2 function returns a value of **1** instead of the **0** value returned by **WEEKDAY.**

Table 5: Naming of Informix and DB2 functions

Concatenation

Informix supports implicit casting; therefore, the concatenation **||** operator can be used with numeric values. DB2 does not support implicit casting; only character values can be concatenated.

When moving IDS SQL to DB2, be aware of operators and the data types on which they operate. A casting function might be necessary before the expression is valid on DB2.

Substring notation

Informix and DB2 support the **SUBSTR(col, start, length)** function. Informix also supports a column substring notation, that is, **col[start,end]**. DB2 does not support this syntax. Notice the following mapping from an IDS substring notation to the DB2 **SUBSTR** function:

```
Col[s,e] = SUBSTR(col,s,e-s+1);
```

Length function

The Informix Length function does not include trailing spaces in the length for fixed-length character fields. Thus, if a 30-byte character column contains 10 nonblank characters, a value of

10 is returned by the Length function. The DB2 Length function always returns the maximum length of a fixed-length character field (30, in this case). The following combination of functions can be used to simulate the IDS behavior: LENGTH(RTRIM(charcol)).

Unique function

Informix supports both the **UNIQUE** and **DISTINCT** functions. DB2 only supports the **DISTINCT** function.

Implicit casting

Informix supports implicit casting of data types; DB2 requires explicit casting on some interfaces. When using different data types (such as **CHAR** and **INTEGER** or **CHAR** and **DATE**) in field assignments, joins or equality operators, be sure to use a casting function on one of the data types to match the other before the operation.

For example, to do a substring on DB2 numeric data where **x = an integer**, do the following:

```
INTEGER(SUBSTR(CHAR(x),1,1))
```

DB2 performs implicit promotion of data types. For example, joining a **SMALLINT** data type to a **BIGINT** data type implicitly promotes the **SMALLINT** to a **BIGINT** before the join.

Stored procedures

Both Informix and DB2 for i support SQL stored procedures. However, the DB2 for i SQL procedural language is based on a subset of the ANSI/ISO Persistent Stored Modules (PSM) specification and is similar to the Informix Stored Procedure Language (SPL). The DB2 PSM language is also available for the development of SQL triggers and functions on DB2.

DB2 also allows stored procedures to be written in any language that is supported by the DB2 precompilers, including: Java™, C, C++, RPG and COBOL. Stored procedures can use any database interface that is 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 program object. SQL procedures are converted automatically to C code and are compiled when the **Create Procedure** statement is run.

Table 6 shows the mapping of some of the Informix procedural constructs to the DB2 SQL procedural language.

INFORMIX SPL	DB2 SQL PSM
DBINFO (row count)	Get Diagnostics row_count
Raise Exception	Signal
FOREACH	FOR loopname AS c1 CURSOR FOR SELECT or Cursor Processing
Exception Handling	Condition Handlers
Return with Resume	WITH RETURN FOR

Table 6: Mapping Informix procedural constructs to DB2 procedural language

As mentioned earlier, the IBM DB2 Migration Toolkit does translate Informix procedures into DB2 SQL procedures.

Declarations

DB2 requires that all declarations be made at the start of each **BEGIN/END** block in the procedure. This includes the **DECLARE CURSOR** statement. **BEGIN/END** blocks can be nested.

File and screen I/O

Informix SPL stored procedures can use the **DEBUG** statement to write errors to a stream file. The DB2 procedure language does not directly support file or screen I/O. However, debug data can be written to a DB2 table or an external stored procedure can be written to write data to stream files.

Operating system commands

Informix procedures support the use of the **SYSTEM** statement for calling operating system commands. DB2 for i includes an external stored procedure, **QCMDEXC**, that can be used to call most IBM i commands. Here is an example of a call to this stored procedure to run the IBM i Delete File (**DLTF**) command. The second parameter is the number of characters in the command (the first parameter).

```
CALL qsys2.qcmdexc ('DLTF MYFILE')
```

An external stored procedure can also be written to issue system commands.

Returning results sets

With Informix, the **RETURN WITH RESUME** clause is used to return single or multiple result sets from an SPL procedure. The **RETURN WITH RESUME** clause is used within a looping construct, such as a **FOREACH** loop. The variables specified in the **RETURN** clause must match those in the **RETURN** clause of the routine. Here is an example of such an Informix stored procedure that is followed by the equivalent DB2 version.

Informix SPL version:

```
CREATE PROCEDURE getcusts RETURNING int, char(15), char(15);
DEFINE rcity, rname char (15);
DEFINE i int;

  FOREACH
    SELECT id, name, city INTO i, rname, rcity
      FROM custinfo
      WHERE id < 200;
   RETURN i, rname, rcity WITH RESUME;
  END FOREACH

END PROCEDURE;
```

DB2 version:

```
CREATE PROCEDURE getcusts ()
DYNAMIC RESULT SETS 1
LANGUAGE SQL
BEGIN
  DECLARE cust1 CURSOR WITH RETURN TO CLIENT FOR
            SELECT id, name, city
              FROM custinfo WHERE id < 200;

  OPEN cust1;
END;
```

The preceding example demonstrates several key points:

- Specifying **DYNAMIC RESULT SETS** in the **CREATE PROCEDURE** statement to indicate that result sets are being returned on the procedure call and the number of result sets being returned (In this example, one result set is returned.)
- Declaring the cursor using the **WITH RETURN** clause
- Keeping the cursor open so the result set can be returned to the invoking application

Triggers

IDS and DB2 SQL triggers are very close in function and syntax. The IDS **EXECUTE PROCEDURE** statement needs to be mapped to a stored procedure call on DB2.

One difference with IDS triggers is support for **SELECT** triggers. DB2 for i does allow the creation of external (non-SQL) triggers for a **Read** (or **Select**) event. However, these external **Read** triggers can have a negative impact on query optimization and performance. An alternative that possibly offers better performance is the use of the **QIBM_QDB_OPEN** exit point.

One other item to consider about DB2 triggers is that

Before statement-level triggers are not supported.

Constraints

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

- Primary key
- Foreign key
- Unique key

Primary key

IDS allows a primary key value to be null, this proprietary behavior is not supported on DB2.

RI constraints

It is a good practice to use referential integrity (**RI**) constraints instead of creating database triggers to enforce **RI**. Although there are minor differences in the syntax of **RI** constraints between Informix and DB2, conversion effort is decreased when using referential constraints

instead of triggers. One minor difference is that Informix generates a constraint name if it is not included in the definition. The constraint name is needed with DB2.

Examples of IDS and DB2 **RI** syntax:

IDS:

```
CREATE TABLE accounts (
  acc_num INTEGER,
  acc_type INTEGER,
  acc_descr CHAR(20),
 PRIMARY KEY (acc_num, acc_type))

 CREATE TABLE sub_accounts (
  sub_acc INTEGER PRIMARY KEY,
  ref_num INTEGER NOT NULL,
  ref_type INTEGER NOT NULL,
  sub_descr CHAR(20),
 FOREIGN KEY (ref_num, ref_type) REFERENCES accounts (acc_num, acc_type))
```

DB2:

```
 CREATE TABLE accounts (
  acc_num INTEGER,
  acc_type INTEGER,
  acc_descr CHAR(20),
 CONSTRAINT generated_pkname PRIMARY KEY (acc_num, acc_type))

 CREATE TABLE sub_accounts (
  sub_acc INTEGER PRIMARY KEY,
  ref_num INTEGER NOT NULL,
  ref_type INTEGER NOT NULL,
  sub_descr CHAR(20),
 CONSTRAINT generated_fkname FOREIGN KEY (ref_num, ref_type)

REFERENCES accounts (acc_num, acc_type))
```

Deferred constraint checking

Deferred constraint checking supports the changing of keys during a transaction that might otherwise violate **RI** constraints. With IDS, referential integrity checking can be deferred to the end of the transaction (at the time of commit when all changes have been completed). DB2 does not support deferred-constraint checking. With DB2, enforcing **Check** and **Referential** constraint checking, you can temporarily suspend and reactivate it by using the **Change Physical File Constraint** (**CHGPFCST**) command. When you disable and re-enable constraint checking for a table, DB2 verifies that all of the rows in the table meet the constraint criteria - only limited access to the table is allowed during this verification process.

Temporary tables

Though Informix temporary tables can be created implicitly when data is selected, it is a good practice to create the temporary table explicitly in a separate step because this is required for DB2.

Also, because DB2 temporary tables are qualified with the **session** schema name and because no other qualifier is allowed, do not use **session** as a user ID or schema name in an Informix application.

With DB2, all global temporary tables must be declared before using the table, by using a **SELECT** statement with the **WITH NO DATA** clause. After the temporary table is explicitly declared, data can be inserted into it. Here is an example of converting the Informix implicit temporary table creation to an explicitly created temporary table that DB2 can support.

In IDS, rather than doing the following:

```
1) SELECT col1, col2, col3, col4
     FROM tab1
      WHERE col1 = 'Hello'
        INTO TEMP temptab1 WITH NO LOG;
```

Do this:

```
1) CREATE TEMP TABLE temptab1(col1 CHAR(5), col2 INT, col3 INT, col4 INT))
   WITH NO LOG;

2) INSERT INTO temptab1
   SELECT col1, col2, col3, col4
     FROM tab1
       WHERE col1 = 'Hello';
```

The following is the equivalent DB2 example of using a temporary table by first declaring it and then inserting data into the temporary table.

DB2:

```
1) DECLARE GLOBAL TEMPORARY TABLE session.temptab1 AS
     (SELECT col1, col2, col3, col4 FROM tab1)
   WITH NO DATA
   ON COMMIT PRESERVE ROWS
   NOT LOGGED;

2) INSERT INTO session.temptab1
     SELECT col1, col2, col3, col4
       FROM tab1
         WHERE col1 = 'Hello';
```

With DB2, the default action with a **commit** operation is to delete rows from the temporary table. With Informix, the default action with a **commit** operation is to preserve rows in the temporary table.

Unsupported options

Some Informix features or capabilities are extremely difficult to support or come up with a DB2 circumvention. The features listed here can require portions of the application to be rewritten or redesigned.

- Objected-oriented and object-relational support
- Compound SQL (DB2 supports only compound statements in SQL procedures, triggers and functions.)
- IDS **DataBlade** modules (DB2 does support XML and Text Extenders.)
- Optimizer directives (The DB2 optimizer is cost-based and does not give the programmer any control. Software vendors need to have a person on their staff attend the **DB2 for i SQL Performance** workshop.)

Application development

When porting IDS applications to DB2 for i, JDBC and CLI/ODBC are the preferred programming interfaces. If the IDS application uses ESQL/C, the most likely target for that migration is a C or C++ program that invokes embedded SQL.

Some SQL applications require a manual conversion to DB2. The migration toolkit does not convert ESQL/C or Informix 4GL code. To minimize these conversion efforts, you should favor coding new IDS applications with the use of embedded ANSI SQL or CLI/ODBC, whenever possible.

Try to design SQL applications by limiting SQL statements to a few modules that are called by other modules rather than distributing SQL throughout all modules. This helps to contain the scope of the SQL-conversion effort.

DBANSIWARN environment variable

It is a good practice to set the Informix **DBANSIWARN** environment variable to check for Informix extensions to ANSI-standard SQL syntax. At run time, the **DBANSIWARN** environment variable causes the sixth character of the **sqlwarn** array in the **SQLCA** to be set to **W** when running a statement that is recognized as including any Informix extension to the ANSI/ISO standard.

Informix 4GL

For Informix 4GL applications, IBM has an Informix 4GL-to-EGL Conversion Utility with the Rational Developer V6.0 products. The latest version of this utility offers these key features and benefits:

- **Uniform conversion from 4GL-to-EGL:** Retains the look-and-feel of your 4GL program with the equivalent EGL program after conversion
- **Graphical conversion wizard:** Steps you through each step of the conversion process
- **Command line conversion:** Supports scripted or automated usage

More information on the Conversion Utility and EGL can be found at **ibm.com**/developerworks/rational/products/egl/.

ESQL/C

The ESQL/C precompiler enables access of Informix data via embedded SQL in C programs.

Precompiler tags

The ESQL/C precompiler recognizes the tokens **$** or **EXEC** SQL as a tag for embedded SQL statements within C code. Because the DB2 precompiler only recognizes **EXEC** SQL, the **$** token tag must be converted. Similarly, Informix allows **$** as a prefix for host variables; this needs to be converted to a colon (**:**) as supported by DB2 and the SQL standard.

Indicator variables

ESQL/C supports using **$** or a colon (**:**) to denote a null indicator variable. Always use a colon (**:**) rather than **$** to stay consistent with ISO SQL and DB2 syntax.

Declare cursor considerations

The IDS **DECLARE CURSOR** statement provides support for **INSERT** statements. A **PUT** statement is used to perform the insert operation with the cursor. These Insert cursors must be converted to DB2 **Insert** statements. If possible, these cursors are converted to a **Blocked Insert** request. Informix also supports a cursored **SELECT** statement, where multiple rows are retrieved using a **SELECT… INTO** clause or a **FETCH INTO** clause. DB2 only supports the **FETCH INTO** clause. The **DB2 SELECT… INTO** statement is used only for single **SELECT** statements. Here is an example of the IDS support and the equivalent DB2 statement.

IDS:

```
EXEC SQL DECLARE cur1 CURSOR FOR
                    b  FROM t1;
EXEC SQL OPEN cur1;

  EXEC SQL FETCH cur1;
```

DB2 and IDS:

```
  EXEC SQL DECLARE cur1 CURSOR FOR
              SELECT col1, col2 FROM t1;
EXEC SQL OPEN cur1;
EXEC SQL FETCH cur1 INTO :var1, :var2;
```

Array host variables

Unlike IDS, DB2 does not support array variables in embedded **SELECT** statements; it supports the values selected into host variables or data structures. For embedded DB2 SQL statements, use **SELECT** with the cursor and the **FETCH** clause to retrieve multiple values into host variables.

IDS:

```
SELECT col1, col2, col3 INTO :arr1[b], :arr2[c], :arr3[d]
```

Declaring date and time variables

For DB2 **date** and **time** values in embedded SQL programs within the **EXEC SQL DECLARE BEGIN** section, declare the **DATE**, **TIME** and **TIMESTAMP** values as character strings:

```
DATE    CHAR(10)
TIME    CHAR(8)
TIMESTAMP  CHAR(26)
```

By contrast, Informix stores a **DATE** as an integer and **DATETIME** and **INTERVAL** as decimals.

The ESQL/C declaration for the **DATETIME** value is:

```
Typedef struct dtime{
    short dt_qual;
    dec_t dt_dec;
} dtime_t;
```

Here, the **dt_qual** field shows the level of granularity for the **DATETIME** value (**Year to Day**, **Hour to Second**). The **dt_dec** field contains the digits of the **DATETIME** value in decimal format.

Cursor and statement-named variables

Informix supports cursor name variables and dynamic SQL statement variables in ESQL/C. DB2 does not support these constructs in embedded SQL, but does support these statements in CLI. ESQL/C for DB2 also supports translation of ESQL/C to DB2 CLI/ODBC, thus supporting these constructs. Here is an Informix example that uses the **Cursor** and **Statement Name** variables:

```
EXEC SQL PREPARE :stmt1 FROM  :stmtstrg
EXEC SQL OPEN CURSOR :getrows
```

Here is a DB2 example without the **Cursor** and **Statement Name** variables:

```
EXEC SQL PREPARE stmt1 FROM :stmtstrg
EXEC SQL OPEN CURSOR getrows
```

C library functions for date and time

ESQL/C supports manipulating **date** and **time** values through C library functions, in addition to SQL built-in or UDFs. With DB2, only SQL built-in or UDFs are supported for C programs, which should be favored to manipulate date and time values when coding applications in ESQL/C.

Null terminator truncation

When assigning values to host variables in a C program, if the variable is not declared long enough to hold the string and null terminator, IDS truncates characters but not the null terminator. DB2 supports the same behavior if you specify the ***CNULRQD** option on the **CRTSQLCI** or **CRTSQLCPPI** precompiler command. ***CNULRQD** is not the default option on these commands.

Comments

DB2 does not support curly braces **{}** for comments. DB2 supports double dashes (**--**) and the forward slash with asterisk **/* */** for commenting in SQL procedures, triggers and UDFs.

Error handling

SQLCODE values and the **SQLCA** structure differ somewhat between Informix and DB2. To resolve these inconsistencies, use the **SQLSTATE** variable, which consists of two parts:

- A two-character error class that identifies the general classification of the error
- A three-character error subclass that identifies a specific error type in a general error class

IDS and DB2 share many **SQLSTATE** codes; yet some values are specific to IDS. In IDS logged nonmode ANSI databases and unlogged databases, the **SQLCODE** value is set to **100** (rows found) only for **SELECT** statements that return no rows. For **DELETE**, **UPDATE** and **INSERT** statements that run but find no rows, **SQLCODE** is set to **0**. IDS supports the **WHENEVER ERROR STOP** argument; DB2 does not. But this is easy to simulate in DB2 with the **WHENEVER SQLERROR GO TO** argument. Another ANSI standard is the **GET DIAGNOSTICS** statement

that allows retrieval of information for the last SQL statement run. The **GET DIAGNOSTICS** statement is supported for ESQL/C and DB2 host-language programs and procedural objects.

System catalog access

To ease your transition, avoid designing applications that depend on system and catalog table access, because DB2 accesses these through views defined in the **QSYS2** or **SYSIBM** schema.

Concurrency and recovery

A few differences in the locking and concurrent access behavior of IDS and DB2 need to be considered.

Lock modes

With Informix, the lock mode is specified in the **CREATE TABLE** statement; if unspecified, the default lock mode is **PAGE**. DB2 implicitly uses row-level locking and does not support an interface for altering this locking mode. DB2 does support a table-level lock with the **LOCK TABLE** statement.

Isolation levels

Table 7 compares the isolation levels supported Informix and DB2.

IDS	DB2
Repeatable Read (RR)	Repeatable Read (RR)
Cursor Stability (CS)	Cursor Stability (CS)
N/A	Read Stability (RS)
Committed Read (CR)	N/A
Dirty Read (DR)	Uncommitted Read
N/A	No Commit

Table 7: Supported isolation levels for IDS and DB2

The default isolation level for IDS logged databases (nonlog mode ANSI) is **Committed Read**. For IDS log-mode ANSI, the default isolation level is **Repeatable Read**; unlogged Informix databases default to **Dirty Read**. Most applications use this standard, because the need for lock resources is small with relatively high transaction encapsulation. DB2 has no exact equivalent to **Committed Read**, but you might consider **Read Stability**. The DB2 for i default isolation level is interface-dependent; thus, it is best that the application set the isolation level. DB2 has 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 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.

Changing the isolation level

The **Set Isolation To** statement is used with Informix to change isolation levels. DB2 has a similar **Set Transaction Isolation Level** statement.

With hold cursors

In IDS, declaring a **WITH HOLD** cursor causes a cursor to be held open through the end of any transaction (for both **COMMIT** and **ROLLBACK** transactions). In DB2, the **WITH HOLD** declaration holds a cursor open only through a **COMMIT** transaction.

Transaction mode

With an IDS nonANSI database, each SQL statement run is implicitly committed unless a transaction is explicitly stated. A transaction is designated by using a **BEGIN WORK** or **BEGIN TRANSACTION** statement and is ended with a **COMMIT WORK** statement. With DB2, all SQL is automatically part of a transaction. The transaction begins implicitly with the first executable SQL statement and is ended with a **COMMIT** statement. If a commit is required after each SQL statement with embedded SQL, the COMMIT must be explicitly performed or the isolation level set to No Commit (***NONE**). Other SQL interfaces such as CLI and JDBC provide programming interfaces to enable autocommit behavior.

Journaling and recovery

As seen in the terminology chart earlier, logging on DB2 for i is known as *journaling*. You can use the system **ENDJRNPF** CL command to turn off journaling for database objects that need no recovery capabilities. Turning off journaling does improve performance, but it is at the expense of transaction recovery. DB2 journaling is most similar to Informix unbuffered logging, but it also supports a variation of the buffered-logging mode, where the logical log buffer is flushed when full or when a transaction is completed. This mode offers an optimal combination of high performance and transactional integrity.

Recovery

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

After the port

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

Functional testing

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

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

IBM does make 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. Testing engagements can also be performed remotely or onsite at one of the IBM Innovation Centers for those companies that are members of IBM Partnerword. Refer to **ibm.com**/isv/iic for more details.

Performance tuning and sizing

Many times, the biggest hurdle to overcome in database projects is the tuning of the application with the new database. Each database engine has its strength and weaknesses from a performance point of view, especially when comparing query optimizers. The Informix query optimizer might work great with a specific set of indexes over the database; at the same time, the DB2 optimizer might require additional indexes to be added to that base set of indexes. The opposite holds true, as well. The DB2 cost-based query optimizer does not support hints or require the collection of database statistics. The database manager automatically maintains and updates statistics (inside the table and index objects) as changes are made to the objects. Starting with OS/400 V5R2, the optimizer also automatically collects stand-alone statistics for some SQL requests. Refer to **ibm.com**/systems/power/software/i/db2/support/sqe/index.htmlfor more information on this recent change regarding the DB2 SQL Query Engine.

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

DB2 includes several different database performance tools including the IBM i Navigator SQL Performance Monitor and Visual Explain tools (see description in Administration section). However, to use these tools effectively and to tune DB2 efficiently, in general, you need to attend the **DB2 for i SQL Performance** workshop (**ibm.com/** systems/power/software/i/db2/education/performance.html). Some performance- tuning information can be found in the **Database Performance and Query Optimization** book.

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

The IBM i Performance and Scalability center is a great resource for procuring help with performance tests and sizings on the hardware required by your application. Visit the following website for more details: **ibm.com**/ systems/services/labservices/psscontact.html

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

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

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

Administration

As discussed in the **Architecture** section 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.

System i Navigator

System i Navigator (formerly iSeries Navigator) is the DB2 graphical interface for database administrators and developers. It is the rough equivalent to the Enterprise Manager in Oracle.

From the System 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 System i Navigator capabilities:

- **SQL Script Center:** Develop and run SQL scripts.
- **SQL Performance Monitors:** Collect and analyze database performance monitor data for query optimization issues and performance of specific SQL requests.
- **Visual Explain:** Analyze a graphical representation of the access plan generated by query optimizer.
- **SQL Plan Cache:** Provide a live analysis of the access plans for the most current and frequently run SQL requests.
- **Journal Management:** Start and end logging and swapping of journal receivers.
- **Database Navigator:** Perform basic graphical modeling of existing database definitions. The IBM InfoSphere Data Architect product also provides full modeling support for DB2 for i databases.

Note: this collection of tools is sometimes referred to as the OnDemand Performance Center.

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

DB-Access provides Informix command line access to the database server. DB2 does not have an analogous tool. Instead, it is recommended that you use the System i Navigator SQL Script Center to perform some of the scripting and use the IBM i job scheduler for any scheduling activities. Note that the SQL Script Center does not have any support for variables in the script. The DB2 QSHELL

command does provide some variable support. Refer to **ibm.com**/systems/i/db2/qshellperl.html for more details on this interface.

Utilities

Several utilities are available to help with your migration efforts.

Load and import

DB-Access supports the **INSERT INTO tab1 LOAD FROM filename** and **UNLOAD TO filename SELECT …FROM tab1** pseudo-SQL statements. DB2 load and unload support is provided with system commands instead of SQL statements.

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

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

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

The System i Navigator toolset also provides a graphical interface to this command.

ALTER TABLE statements

DB2 supports the same operations as the IDS **ALTER** statement.

Summary

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

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

Resources

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

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

- IBM i Knowledge Center
  ibm.com/support/knowledgecenter/ssw_ibm_i_72
- IBM i on IBM PartnerWorld®
  ibm.com/partnerworld/i
- DB2 for i online manuals
  ibm.com/systems/power/software/i/db2/docs/books.html
- IBM Redbooks®
  ibm.com/redbooks
  - *Stored Procedures, Triggers and User-defined Functions on DB2 for i* (SG24-6503)
  - *Advanced Database Functions and Administration on DB2 for i* (SG24-4249-03)
  - *Diagnosing SQL Performance on DB2 for i* (SG24-6654)
  - *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
- Education Resources and white papers
  ibm.com/systems/power/software/i/db2/education/index.html**ibm.com**/partnerworld/wps/whitepaper/i5os
  ibm.com/partnerworld/wps/training/i5os/courses
- DB2 for i Performance Workshop
  ibm.com/systems/power/software/i/db2/education/performance.html
- IBM DB2 Migration Toolkit
  ibm.com/partnerword/i/db2porting
- IBM Power Development Platform
  ibm.com/partnerworld/pdp
- IBM Innovation Center
  ibm.com/isv/iic
- IBM Performance and Scalability Center
  ibm.com/systems/services/labservices/psscontact.html

- Performance Management for IBM i
  ibm.com/systems/i/advantages/perfmgmt

Online forum

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

Conversion services and tools

- IBM Systems and Technology Group Lab Services
  ibm.com/systems/services/labservices
- DB2 Informix Migration Toolkit
  ibm.com/software/data/db2/migration/mtk

Other publications and websites

- *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

- Centerfield Technology
  www.insuresql.com

Trademarks and special notices

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

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

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

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

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.

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.