# System versus SQL name, Part 2: Accessing database objects

Birgitta Hauser

August 14, 2012

When running SQL statements, you can run them either using system or SQL naming convention. The previous article was focused on the differences in ownership and access authorities when creating database objects with the SQL and system naming convention. This article examines the behavior differences with system and SQL naming when accessing tables and views as well as stored procedures and user-defined functions (UDFs), focusing mainly on the different behaviors when using unqualified references to those objects.

View more content in this series

This article examines the different behaviors when running SQL statements with the system and SQL naming conventions. The first part of this article focused on how the system and SQL naming conventions resulted in different object ownerships and access authorities when creating IBM® DB2® objects with SQL.

The naming convention also determines which character, either a slash (/) for System naming (*SYS) or a period (.) for SQL naming (*SQL), is used to separate the schema and the object name when DB2 object references are explicitly qualified with a schema.

IBM i applications, however, rarely access DB2 objects by explicitly specifying the schema name. Instead, these applications rely on the library list being searched to find the appropriate objects. The first object found within the library list with the specified name and the appropriate object type will be used. When testing applications, additional libraries containing new programs and data set are simply inserted at the top of the library list. In this way, it is easy to work with a mix of old and new programs as well as production data and test data. Applications with explicitly qualified schema references have to be manually changed to run in different environments.

With regard to the typical IBM i applications, let us analyze the different behaviors based on the naming convention for accessing unqualified database objects that are specified in the SQL statements.

## Accessing data

Persistent user data is only stored in tables. Data in a table can be accessed directly or indirectly through an alias or a view. Tables, views, or aliases can be accessed in SQL statements by

explicitly qualifying the object with the schema name or by having the schema name implicitly resolved based on the naming convention.

## Data access methods

Data in IBM DB2 for i objects can be accessed and maintained with record-level access interfaces which can be used in some high-level languages such as RPG or COBOL.

However, SQL is the most common interface used to access data in the new IBM i applications and programs. SQL statements can be run either as *static* or *dynamic* SQL. The main difference between static and dynamic SQL is based on how the SQL statement itself is generated.

- **Static SQL statements**

Static SQL is heavily used in SQL routines or application programs with embedded SQL. A static SQL statement is hard-coded within the source of the program or routine. For static SQL statements, the SQL precompiler checks the SQL syntax, evaluates the references to tables and columns, and declares data types of all the host variables. The SQL precompiler also determines the schema to be used at run time for resolving unqualified database objects based on the **naming convention used at compile time**. From a performance perspective, using the static SQL is the best option because several steps (for example, syntax checking) are already done at compile time.

Listing 1: Static SQL statement shows a static SQL statement embedded in a RPG program to determine the number of orders for a specific year. The year value is passed as the parameter value (ParYear) to the procedure.

## Listing 1: Static SQL statement

```
D GetNbrOfOrders... D PI 10I 0 D ParYear 4P 0 Const ... D NbrOfOrders S 10I 0 /Free ... Exec SQL Select
Count(*) Into :NbrOfOrders from Order_Header Where Year(OrderDate) = :ParYear;
```

- **Dynamic SQL statements**

Dynamic SQL statements are built during the run time execution of a program. After being constructed, the dynamic SQL statements are checked for syntax and then converted into an executable SQL statement that can be run.

Listing 2: Dynamic SQL statement in a SQL routine shows the SQL script to create the UDF COUNT_NUMBER_OF_ROWS. With this function, the number of rows in any table, view, or alias can be determined. The table (or view or alias) name, as well as the schema name is passed as the parameter value. When calling the UDF, the SQL statement to be executed is built as string including passed parameter values. This string is checked for syntax and then converted into an executable SQL statement by running the PREPARE statement and finally executed by using the EXECUTE statement. As neither the table nor the schema to be accessed at run time is known at compile time, dynamic SQL is needed.

### Listing 2: Dynamic SQL statement in a SQL routine

```
Create Function Count_Number_Of_Rows (ParTable VarChar(128), ParSchema VarChar(128)) Returns Integer
Language SQL Not Fenced Begin Declare RtnNbrRows Integer; Declare String VarChar(256); Set String =
'Values(Select Count(*) From '; If ParSchema &gt; ' ' Then Set String = String CONCAT ParSchema CONCAT '/';
End If; Set String = String CONCAT ParTable CONCAT ') into ?'; PREPARE DynSQL From String; EXECUTE DynSQL
Using RtnNbrRows; Return RtnNbrRows; End;
```

Dynamic SQL statements can be used in SQL routines or embedded in programs, but it is also most commonly used for running SQL statements from interfaces, such as ODBC or DB2 Web Query and from SQL command line processors, such as IBM System i® Navigator Run Script Interface or the RUNSQLSTM and RUNSQL command line commands. Listing 3: Dynamic SQL statement issued interactively shows a dynamic SQL statement run with the System i Navigator Run Script Interface.

### Listing 3: Dynamic SQL statement issued interactively

```
Select * from Order_Header;
```

## Determining the default schema

When an SQL statement contains an unqualified table, view, or alias reference, DB2 must determine the **default schema** and search for that schema. *Schema* is the SQL term analogous to an IBM i library. The initial value for the default schema depends on the naming convention used in the SQL environment and whether the SQL statement being run is *static* or *dynamic*.

### Default schema for static SQL statements

When using embedded SQL, the default schema for static SQL statements can be **explicitly set** in the compile command (CRTSQLxxxI) using the DFTRDBCOL (default collection) parameter. Alternatively, a SET OPTION statement with the DFTRDBCOL parameter can also be included in the source code.

In embedded SQL programs, only as single SET OPTION statement can be specified, even if the source code consists of several independent (exported) procedures. The SET OPTION statement must be placed in the source code as the first SQL statement.

Listing 4: SET OPTION Statement embedded in RPG shows the excerpt of an RPG source with a SET OPTION statement to set the naming convention to SQL naming and to set the default schema for static SQL statements to SALESDB01. The SET OPTION statement is included immediately after the global D specifications that means as the first statement in the C specifications.

### Listing 4: SET OPTION Statement embedded in RPG

```
D* Global D Specifications /Free EXEC SQL Set Option Naming = *SQL, DFTRDBCOL = SALESDB01; // RPG code and
 other embedded SQL Statements go here
```

With SQL routines, the default schema can also be explicitly set by including a SET OPTION statement with the DFTRDBCOL parameter. In Listing 5: SET OPTION statement in an SQL

routine , the default schema has been explicitly set to SALESDB01 for static SQL statements within the MyProcedure routine.

## Listing 5: SET OPTION statement in an SQL routine

```
Create Procedure MyProcedure () Language SQL Set Option DFTRDBCOL = SALESDB01 Begin -- SQL Routine Body –
 Source code End;
```

If the default schema is not explicitly set, it is **determined at the compile time** depending on the naming convention.

- For **system naming**, the default schema is the job **Library List** (*LIBL)

With system naming, the term *schema* can be misleading because the initial value is set to the special value *LIBL. This special value means that the library list is used and multiple schemas can be searched when trying to resolve unqualified object references. The first DB2 object found in the first library, which matches the unqualified specified database object name and object type, will be used. Database objects located in different schemas can be accessed within the same SQL statement without any schema specification.

- For **SQL naming,** the **default schema currently used** in the SQL environment where the SQL routine is created will be adopted.

Because application programs with embedded SQL are not created through an SQL interface, the default schema for the static SQL statements is set to the **runtime authorization ID.** On IBM i, the runtime authorization ID is the user profile of the job that is performing the compile. This means that the default behavior for SQL naming is for DB2 to try and find the unqualified object in the schema that has the same name as the creator's user profile.

SQL naming allows only a **single schema** to be searched when resolving unqualified DB2 object references.

## Default schema for dynamic SQL statements

For dynamic SQL statements, the default schema depends on whether a default schema value has been explicitly specified. If a default schema is not explicitly set, its **initial value** depends on the naming convention.

- For **System naming**, the default schema is the job **Library List** (*LIBL).
- For **SQL naming**, the default schema is the **runtime authorization ID** (current user profile). As stated earlier, the default behavior for SQL naming is for DB2 to try and find the unqualified object in a schema that has the same name as the current user profile.

### SET SCHEMA statement

The value of the default schema can be changed on all the interfaces by running the SET SCHEMA statement. The new default schema value supplied by the SET SCHEMA statement is

used **only** to resolve unqualified database objects by **dynamic SQL statements**. It has **no effect** when resolving unqualified object references for **static SQL statements** at run time.

In the SET SCHEMA statement, special registers, such as USER, SESSION_USER, or SYSTEM_USER can also be specified. The special value *LIBL is not allowed, not even in an environment where the System naming convention is used.

If the SET SCHEMA statement is run in an environment where the **system naming** convention is used, the **library list will no longer** be searched for dynamic SQL statements. Instead, the unqualified database objects are searched for in the **single schema** that was specified on the SET SCHEMA statement.

### Current schema

The default schema to be searched at runtime for unqualified data access in dynamic SQL statements is also referred to as the **current schema**. The CURRENT_SCHEMA special register returns the schema value currently being used for resolving unqualified data access in dynamic SQL statements.

It should be noted that *current schema* and *current library* are **not** identical terms. The *current library* is added to the current library list before the user portion of the list by running the CHGCURLIB (change current library) command. The library list is accessed only when System naming is used. Therefore, the *current library* can only be searched in composition with System naming.

The current schema (or the default schema) is either the current library list (system naming) or a single schema (SQL naming) which may or may not be part of the current library list.

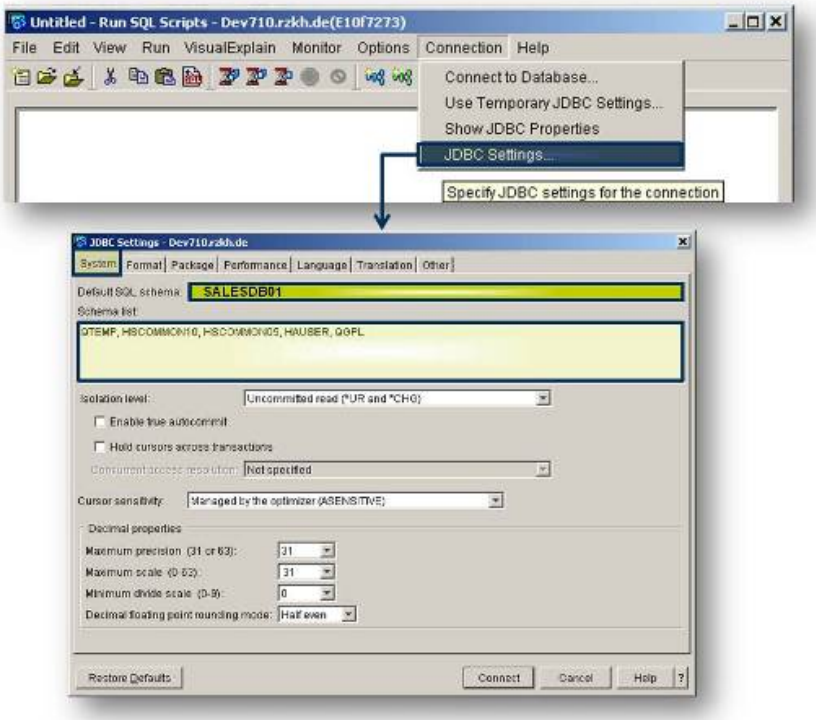
### SET SCHEMA statement and dynamic SQL interfaces

Many of the DB2 for i SQL interfaces can execute the SET SCHEMA statement automatically on your behalf through the ability to specify a default schema value for that dynamic SQL interface. The mechanism for specifying a default schema value depends on the interface.

- **IBM System i Navigator Run SQL Scripts tool**

The default schema can be preset by clicking **Connection  JDBC Settings**. The default schema or the library list can be specified on the System tab, as shown in the following figure.

**Figure 1: System i Navigator Run SQL Scripts – setting the default schema**




- **Command line commands: RUNSQLSTM and RUNSQL**

The default schema can be specified on these SQL commands with the DFTRDBCOL (default collection) parameter.

- **ODBC connections**

ODBC connections can be defined by using - ODBC Administration. When accessing your tables and views with ODBC, the Default Schema can be set from the IBM i Access for Windows ODBC Administration interface or programmatically by setting the DefaultLibraries connection keyword.

- **SQL call level interface (CLI)**

When using SQL CLI functions, the schema to be used can be explicitly specified by setting the SQL_ATTR_DEFAULT_LIB or SQL_ATTR_DBC_DEFAULT_LIB environment or connection variables.

- **Java™ Database Connectivity (JDBC) or Structured Query Language for Java (SQLJ)**

The default schema can be set though the libraries' property object.

- **OLE DB using the IBM i Access Family OLE DB Provider**

The default schema can be explicitly specified through DefaultCollection in Connection Object Properties.

- **ADO .NET using the IBM i Access Family ADO .NET Provider**

The default schema can be explicitly specified through DefaultCollection in Connection Object Properties.

**Notes:**

Some of these interfaces allow a default schema and a default library list to be set. If a default schema is specified and the System naming conventions is used, it is possible that DB2 uses only the specified default Schema while ignoring the library list when resolving unqualified DB2 object references. Based on this behavior, it is better to avoid specifying a value for the default schema when using the System naming convention.

## Test environment

To examine the different behaviors in accessing database objects (using either system or SQL naming), I created a test environment to represent a typical IBM i application. The test environment consists of four schemas:

- **Schema MASTERDB – master information**

Information such as the address of a particular customer, supplier, or an item is needed for multiple applications, such as Accounting, Purchase, Sales, ERP and so on.

Schema MASTERDB contains the following tables:

ADDRESS_MASTER, ITEM_MASTER, and ORDER_SUMMARY

- **Schema SALESDB01 and schema SALESDB02 – sales data**

The SALESDB01 and SALESDB02 schemas contain the necessary sales information for company 1 and company 2 respectively.

Both the schemas include an ORDER_HEADER and ORDER_DETAIL table.

- **Schema SALESPGM – program schema**

Schema SALESPGM does not contain any data, but it is used as a container for all of the (service) programs, stored procedures, and user defined functions.

## Executing dynamic SQL statements

In the following examples, let us examine the different behaviors when running dynamic SQL statements in composition with either System or SQL naming, using the System i Navigator Run Script tool as our dynamic SQL interface.

### Unqualified data access with system naming

When executing Dynamic SQL statements in an environment where System naming is used and the default schema is not explicitly set, the current **library list** is searched to find all unqualified tables and views.

The library list is initially set in any SQL interface based on the job description. However, the library list can be modified by running the commands such as CHGLIBL (change library list) or ADDLIBLE (add library list entry). When the default schema is changed by running a SET SCHEMA statement, the current library list will be ignored and the newly set (single) schema will be searched instead.

The following example demonstrates this behavior by executing several SQL statements.

Before running the first SELECT statement, the library list is explicitly set by executing the CHGLIBL (change library list) command. In the SELECT statement, the ORDER_HEADER table that is located either in the SALESDB01 or SALESDB02 schema, and the ADDRESS_MASTER table that is located in the MASTERDB schema are joined together.

The SALESDB01 and the MASTERDB schemas are both part of the current library list. Consequently, both the tables are found and the SELECT statement is executed successfully. As the ORDER_HEADER table is found in the SALESDB01 schema, the requested data for company 1 is returned.

## Listing 6: Accessing data in multiple schemas with system naming

```
CL: CHGLIBL LIBL(SALESDB01 MASTERDB QGPL); Select h.Company, h.OrderNo, AddressNo, a.Name1, a.Address, a.City
 From Order_Header h join Address_Master a Using(AddressNo);
```

| COMPANY | ORDERNO | ORDERDATE | ADDRESSNO | NAME1 | CITY |
|---|---|---|---|---|---|
| **1** | 100 | 1 | Fischer & Co | Wald- und Wiesenweg 16 | Dietzenbach |
| **1** | 110 | 3 | Bauer GmbH | Nordring 417 | Berlin |
| **1** | 120 | 4 | Rathaus Center | Hauptstr. 3 | Hamburg |
| **1** | 130 | 4 | Rathaus Center | Hauptstr. 3 | Hamburg |
| **1** | 140 | 4 | Rathaus Center | Hauptstr. 3 | Hamburg |

To retrieve the same information for company 2, the SALESDB02 schema must be added to the library list. The SALESDB01 schema can be removed or must be located after the SALESDB02 schema in the library list. The library list must be changed by running a command, such as CHGLIBL or ADDLIBLE.

If the SALESDB02 schema is set by running the SET SCHEMA statement, the default schema value is changed from *LIBL and is set to SALESDB02. When re-running the SELECT statement after this change, the execution fails with SQLSTATE 42704, because the ADDRESS_MASTER table that is located in the MASTERDB schema is not found in the SALESDB02 schema.

## Unqualified data access with SQL naming

In an environment where SQL naming is used, only a single schema is searched at run time to resolve the unqualified tables, views, and aliases. When executing the SELECT statement presented in Listing 6: Accessing data in multiple schemas with system naming with the SQL naming convention, the SELECT statement will always fail with an SQLSTATE value of 42704,

because the ADDRESS_MASTER and the ORDER_HEADER tables are located in different schemas.

When using the SQL naming convention, the database objects in different schemas either have to be qualified or must be accessed through aliases or views that are located in the default schema. The aliases or views can reference tables or views in different schemas.

## Executing dynamic SQL in SQL routines or programs

Dynamic SQL statements embedded in SQL routines or programs follow the same rules for resolving unqualified object references. However, dynamic SQL statements use the naming convention that was active when the SQL routine or program was **created** and **not** the naming convention that is specified when the SQL routine or program is run.

For example, if an SQL stored procedure is created with the SQL naming convention, the dynamic SQL statements within that procedure will use the SQL naming rules for resolving unqualified names, even when that SQL stored procedure is called from an SQL interface that is using the System naming convention.

The naming convention to be used for a program with embedded SQL can be defined at the compile time, through the OPTION parameter in the compile command or by embedding a SET OPTION statement within the source code. An SQL routine inherits the naming convention of the SQL interface that is being used to create the SQL routine. Even though a SET OPTION statement can be embedded in a SQL routine, specifying the NAMING option is not allowed.

The default schema to be used for the dynamic SQL statements at runtime can be **explicitly set** from within the routine or program by running either a command (for modifying the library list) or executing a SET SCHEMA statement embedded in the source code.

When modifying the library list within a SQL routine or application program, the modified library list is used by all the programs and procedures running within the same job. When running a SET SCHEMA statement within a SQL routine or an embedded SQL program, the SET SCHEMA setting will only be used by the dynamic SQL statements within this routine or program. The default schema value of the interface that called the SQL routine or program remains untouched.

The following SQL script creates the stored procedure ORDERADDRD in an environment where the System naming convention is used. The stored procedure accepts a single parameter (ParAddressNo) and returns all Order Header and Address information as a result set for this specific address. The ORDER_HEADER table is joined with the ADDRESS_MASTER table. None of these tables is qualified with a schema. The SQL SELECT statement is dynamically prepared and executed.

## Listing 7: Routine ORDERADDRD with dynamic SQL

```
Create Procedure SALESPGM/OrderAddrD (In ParAddressNo Integer) Dynamic Result Sets 1 Language SQL Begin
 Declare StringSQL01 VarChar(1024); Declare CsrC01 Cursor For DynSQLC02; Set StringSQL01 = 'Select Company,
 OrderNo, OrderDate, AddressNo, Name1, City From Order_Header Join Address_Master Using (AddressNo) Where
 AddressNo = ?'; Prepare DynSQLC01 From StringSQL01; Open CsrC01 Using ParAddressNo; End;
```

Listing 8 shows two successful calls of the ORDERADDR procedure as well as the content of the result sets returned by each procedure call. First, the library list is explicitly set with the CHGLIBL command. Because **System naming** was used to create the procedure, the **library list** is searched to find the unqualified references. The ORDER_HEADER table is found in the SALESDB01 schema while the ADDRESS_MASTER table is found in the MASTERDB schema. The result set contains the data from company 1 proving the information that the ORDER_HEADER table in SALESDB01 schema was used.

To get the order header data for company 2, the **library list is changed** with the CHGLIBL command, before calling the stored procedure.

### Listing 8: Dynamic SQL in SQL routines with System naming

```
CL: CHGLIBL LIBL(SALESDB01 MASTERDB SALESPGM QGPL); Call SalesPGM/OrderAddrD(4);
```

| COMPANY | ORDERNO | ORDERDATE | ADDRESSNO | NAME1 | CITY |
|---|---|---|---|---|---|
| **1** | 120 | 04/26/2012 | 4 | Rathaus Center | Hamburg |
| **1** | 130 | 04/27/2012 | 4 | Rathaus Center | Hamburg |
| **1** | 140 | 04/24/2012 | 4 | Rathaus Center | Hamburg |

```
CL: CHGLIBL LIBL(SALESDB02 MASTERDB SALESPGM QGPL); Call SalesPGM/OrderAddrD(4);
```

| COMPANY | ORDERNO | ORDERDATE | ADDRESSNO | NAME1 | CITY |
|---|---|---|---|---|---|
| **2** | 110 | 04/25/2012 | 4 | Rathaus Center | Hamburg |

If the ORDERADDRD routine had been created with the **SQL naming** convention, the stored procedure call in Listing 8 would fail. This is because, with the SQL naming convention, the library list is not searched and the ORDER_HEADER and ADDRESS_MASTER tables are located in different schemas.

## Running static SQL statements

Due to the fact that static SQL statements are hard-coded, they are analyzed by DB2 **at compile time.** Information about the SQL statement, such as the **default schema** is determined and stored in the newly created program or routine object. The naming convention used at compile time determines how the default schema will be computed for unqualified references on the static SQL statements.

The default schema for static SQL statements can also be manually controlled by explicitly specifying the DFTRDBCOL (default collection / schema) parameter on the precompile command or by specifying the DFTRDBCOL parameter on the SET OPTION statement or clause.

If an SQL routine is created from an SQL interface that had previously specified a default schema value using the SET SCHEMA statement or interface setting, DB2 for i will automatically add a SET OPTION clause with the DFTRDBCOL parameter to the SQL routine definition. In this situation, the resolution of unqualified references on static SQL statements is indirectly affected by the SET SCHEMA statement.

Even though the default schema is determined at compile time, the unqualified DB2 object references are not checked for existence. The SQL routine or program can be generated successfully even when tables or views that are referenced do not exist on the system.

## Running static SQL statements with System naming

To find unqualified database references for static SQL statements embedded in SQL routines or programs created with the System naming convention, DB2 searches the runtime definition of the **library list** even when the routine is called from an interface specifying SQL naming.

Listing 9 shows the SQL script for creating the ORDERADDR procedure in the SALESPGM schema using System naming. The ORDERADDR procedure runs the same SQL statements in the ORDERADDRD routine (as shown in Listing 7: Routine ORDERADDRD with dynamic SQL), but this time, static SQL statements are used instead of dynamic SQL.

## Listing 9: Routine ORDERADDR with static SQL statements created using system naming

```
Create Procedure SalesPGM/OrderAddr (In ParAddressNo Integer) Dynamic Result Sets 1 Language SQL Begin
 Declare CsrC01 Cursor For Select Company, OrderNo, OrderDate, AddressNo, Name1, City From Order_Header Join
 Address_Master using(AddressNo) Where AddressNo = ParAddressNo; Open CsrC01 ; End;
```

The OrderAddr stored procedure was created in an environment where **system naming** was used, so the library list at run time will be searched to resolve the unqualified object references on the SELECT statement.

The naming convention and other attributes about a SQL routine or embedded SQL program can be determined by using the PRTSQLINF (print SQL information) command or by accessing the SYSPROGRAMSTAT catalog view in QSYS2.

Listing 10: Static SQL in an SQL routine created with system naming shows two calls of the ORDERADDR procedure and the returned result sets from an environment where SQL naming is used. Remember that calling the stored procedure in an environment where System naming is used will result in the same results, because DB2 uses the naming convention specified at compile time for the SQL routine or program.

First, the library list is explicitly set with the CHGLIBL command. Additionally, the default schema is assigned to the value, SALESDB02. When calling the stored procedure, the library list is searched while the specified SALESDB02 schema is ignored as the procedure was created with the System naming convention. The ORDER_HEADER table is found in the SALESDB01 schema and the ADDRESS_MASTER table is found in the MASTERDB schema. The static Select statement is run successfully and the order header data for company 1 is returned.

To get the order header data for company 2, the library list is changed, replacing the SALESDB01 schema with the SALESDB02 schema. As you can see from the result set returned by the second call to the procedure, the order header data for company 2 is returned.

## Listing 10: Static SQL in an SQL routine created with system naming

```
CL: CHGLIBL LIBL(SALESDB01 MASTERDB SALESPGM QGPL); Set Schema SALESDB02; Call SalesPGM.OrderAddr(4);
```

| COMPANY | ORDERNO | ORDERDATE | ADDRESSNO | NAME1 | CITY |
|---|---|---|---|---|---|
| **1** | 120 | 04/26/2012 | 4 | Rathaus Center | Hamburg |
| **1** | 130 | 04/27/2012 | 4 | Rathaus Center | Hamburg |
| **1** | 140 | 04/24/2012 | 4 | Rathaus Center | Hamburg |

```
CL: CHGLIBL LIBL(SALESDB02 MASTERDB SALESPGM QGPL); Call SalesPGM.OrderAddr(4);
```

| COMPANY | ORDERNO | ORDERDATE | ADDRESSNO | NAME1 | CITY |
|---|---|---|---|---|---|
| **2** | 110 | 04/25/2012 | 4 | Rathaus Center | Hamburg |

## Running static SQL statements using SQL naming

When creating an SQL routine with the SQL naming convention, DB2 determines the default schema based on the SQL interface used to create the SQL routine.

Listing 11: Routine ORDERADDR1 with static SQL created using SQL naming contains the SQL script to create the ORDERADDR1 stored procedure in the SALESPGM schema with SQL naming. This procedure returns the same result as the ORDERADDR procedure (Listing 9: Routine ), but the source code is slightly modified. Instead of joining the ORDER_HEADER table and the ADDRESS_MASTER table located in different schemas, the ORDER_HEADER_JOIN_ADDRESS_MASTER view is used.

Before executing the CREATE PROCEDURE statement, the default schema is explicitly set to SALESDB01. In this case, the SALEDB01 is used for the DFTRDBCOL option on the SQL routine. The value of the DFTRDBCOL parameter can be checked by running the PRTSQLINF command or by accessing the SYSPROGRAMSTAT catalog view in QSYS2.

## Listing 11: Routine ORDERADDR1 with static SQL created using SQL naming

```
Set Schema SALESDB01; Create Procedure SALESPGM.OrderAddr1 (In ParAddressNo Integer) Dynamic Result Sets 1
 Language SQL Begin Declare CsrC01 Cursor For Select Company, OrderNo, OrderDate, AddressNo, Name1, City From
 Order_Header_Join_Address_Master Where AddressNo = ParAddressNo; Open CsrC01 ; End;
```

As a result of the DFTRDBCOL parameter being set, the **SALESDB01 schema** will be used to resolve any unqualified DB2 object references on the static SQL statements that are embedded within the ORDERADDR1 stored procedure.

When running the ORDERADDR1 stored procedure with either the System or SQL naming convention, the ORDER_HEADER_JOIN_ADDRESS_MASTER view is always retrieved from the SALESDB01 schema and the order header data for company 1 is returned.

Notice that in Listing 12: Static SQL in a stored procedure created with SQL naming, the default schema is explicitly set to SALESDB02 before the call of the ORDERADDR1 stored procedure, and the data for company 1 is returned in the stored procedure's result set signifying that the view

was found in the SALESDB01 schema. This behavior demonstrates that the DFTRDBCOL setting for the stored procedure is used, while the default schema value of SALESDB02 is ignored.

### Listing 12: Static SQL in a stored procedure created with SQL naming

```
Set Schema SalesDB02; Call OrderAddr1(4);
```

| COMPANY | ORDERNO | ORDERDATE | ADDRESSNO | NAME1 | CITY |
|---|---|---|---|---|---|
| **1** | 120 | 04/26/2012 | 4 | Rathaus Center | Hamburg |
| **1** | 130 | 04/27/2012 | 4 | Rathaus Center | Hamburg |
| **1** | 140 | 04/24/2012 | 4 | Rathaus Center | Hamburg |

As the default schema for the static SQL statements is determined at compile time and the default schema for the dynamic SQL statements is determined at runtime, the same SQL statement embedded in the same routine can return different results if that statement is executed as both, static and dynamic SQL requests.

By adding the DYNDFTCOL (dynamic default schema) parameter that is set to *YES to the SET OPTION statement, the dynamic SQL statements are forced to use the same default schema as that of the static SQL requests. In embedded SQL programs, the DYNDFTCOL parameter can also be specified in the compile command.

Listing 13: shows an excerpt of a CREATE PROCEDURE statement. The default schema for static SQL statements is set to SALESDB01 with the DFTRDBCOL option and the dynamic SQL statements are forced to use the same default schema as that of the static SQL statements by setting the DYNDFTCOL option to *YES.

### Listing 13: SET OPTION Statement with DYNDFTCOL

```
Create Procedure SALESPGM.OrderAddrX (In ParAddressNo Integer) Dynamic Result Sets 1 Language SQL Set Option
 DYNRDBCOL = SALESDB01, DYNDFTCOL = *YES Begin -- Routine Body – Source Code End;
```

## Unqualified data access in SQL triggers

SQL triggers are a special kind of SQL routines that are linked to a table, a physical file, or view. A trigger program is activated by DB2 as soon as a row in the associated table or view is inserted, updated, or deleted. When creating a trigger program, it is not mandatory to qualify the DB2 object references in the static SQL statements, but the schemas for all the unqualified DB2 objects are resolved when the trigger is being created. Thus, the trigger behavior cannot be changed at run time by using a different library list or the default schema setting.

Again, the naming convention determines how unqualified table, view, or alias references within the SQL trigger are resolved. Contrary to other SQL routines, a trigger is not created if any of the referenced DB2 tables, views, or aliases does not exist or is not found in the default schema.

As the schemas are resolved and included in the trigger program object, the trigger can be activated and executed in any environment correctly even though the library list or the default

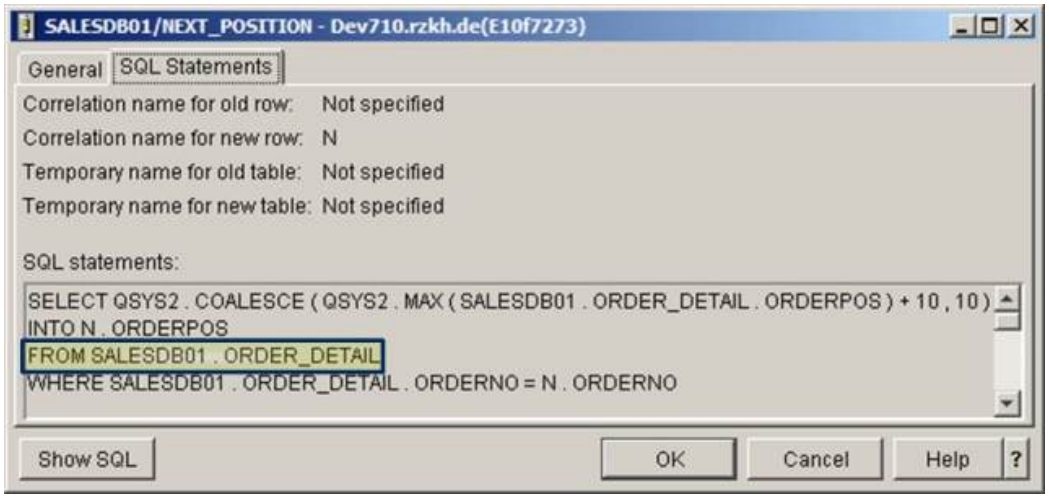
schema is not set as expected. Listing 14: Trigger created with system naming shows the SQL script for the NEXT_POSITION Before Insert trigger, to determine the next order position, by adding 10 to the maximum order position (OrderPos) of the current order in the ORDER_DETAIL table. If it is the first position row for the order, the order position number is set to 10.

### Listing 14: Trigger created with system naming

```
CL: CHGLIBL LIBL(SALESDB01 MASTERDB SALESPGM QGPL); Create Trigger SALESDB01/Next_Position Before INSERT on
 SALESDB01/ORDER_DETAIL Referencing NEW as N For Each Row Mode DB2ROW Select Coalesce(Max(OrderPos) + 10, 10)
 into N.OrderPos From Order_Detail where OrderNo = N.OrderNo;
```

The ORDER_DETAIL table exists in the SALESDB01 schema as well as in the SALESDB02 schema. Because the current library list contains the SALESDB01 schema, the ORDER_DETAIL table is found in this schema, the trigger program is created and the resolved SALESDB01 schema name is stored in the trigger program object. The following figure shows the SQL statements returned by the trigger definition task in System i Navigator for the NEXT_POSITION trigger. The originally unqualified table reference is stored in composition with the resolved schema.

### Figure 2: Trigger NEXT_POSITION – routine body




## Aliases and views

If your DB2 objects are spread over multiple schemas and you must use the SQL naming convention, then you may want to create views or aliases to support unqualified data access.

An alias is a permanent database object that points either to a table or view that the referenced object can be in either the same schema or a different one. Starting with the IBM i 7.1 release, an alias can reference objects on a remote server. Aliases can also reference individual partitions of a partitioned table or a member of a multi-member physical file. An alias is created by running the CREATE ALIAS statement. If the referenced object on the CREATE ALIAS statement is not qualified, the schema is resolved and stored in the alias object. The schema resolution is dependent on the naming convention that is active on the interface and running the CREATE ALIAS statement.

An SQL view is created by running the CREATE VIEW statement and is based on the SQL SELECT statement. Views are a very powerful instrument to simplify complex SQL requests and reduce source code. When creating a view based on a SELECT statement with unqualified object references, the schemas are resolved based on the naming convention and stored in the view object. The view is not generated if any of the unqualified DB2 objects is not found or does not exist.

In the following example, System naming is used to create the ORDER_HEADER_JOIN_ADDRESS_MASTER view. First, the library list is explicitly set by executing the CHGLIBL command. In the view definition, the ORDER_HEADER table (which is either located in the SALESDB01 schema or the SALESDB02 schema) is joined with the ADDRESS_MASTER table, which is located in the MASTERDB schema.

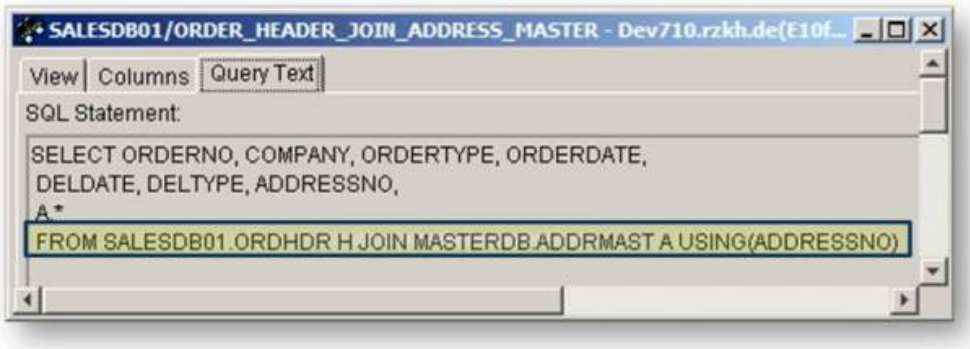
## Listing 16: Create a view with system naming

```
CL: CHGLIBL LIBL(SALESDB01 MASTERDB QGPL); Create View SALESDB01/Order_Header_Join_Address_Master as
 Select OrderNo, Company, OrderType, OrderDate, DelDate, DelType, AddressNo a.* from Order_Header h Join
 Address_Master a using(AddressNo);
```

The view is created successfully because the ORDER_HEADER table is found in the SALESDB01 schema while the ADDRESS_MASTER table is found in the MASTERDB schema, and both the schemas are included in the current library list.

The resolved schemas are added to the appropriate table references in the SELECT statement and stored in the view object. To prove this behavior, the following figure shows the Query Text, which is part of the System i Navigator View Definition output. Notice how the view definition now contains the schema names of SALESDB01 and MASTERDB on the table references.

## Figure 3: View created based on unqualified objects




If the CREATE VIEW statement in Listing 16: Create a view with system naming is executed in an environment where SQL naming is used, it will fail, because only a single schema can be searched at a time to resolve the unqualified specified database objects.

In Listing 17: Unqualified data access view with SQL naming the ORDER_HEADER_JOIN_ADDRESS_MASTER view created previously (in Listing 16: Create a view with system naming) is accessed with SQL naming. The default

schema is explicitly set to SALESDB01 to analyze the data for company 1. Because the ORDER_HEADER_JOIN_ADDRESS_MASTER view is found in this schema and the view object now explicitly references the order header data located in the SALESDB01 schema, the address information located in the MASTERDB schema can be successfully returned.

### Listing 17: Unqualified data access view with SQL naming

```
Set Schema SALESDB01; Select Company, OrderNo, OrderDate, AddressNo, Name1, City from
 Order_Header_Join_Address_Master;
```

| COMPANY | ORDERNO | ORDERDATE | ADDRESSNO | NAME1 | CITY |
|---|---|---|---|---|---|
| 1 | 100 | 04/28/2012 | 1 | Fischer & Co | Dietzenbach |
| 1 | 110 | 04/28/2012 | 3 | Bauer GmbH | Berlin |
| 1 | 120 | 04/26/2012 | 4 | Rathaus Center | Hamburg |
| 1 | 130 | 04/27/2012 | 4 | Rathaus Center | Hamburg |
| 1 | 140 | 04/24/2012 | 4 | Rathaus Center | Hamburg |

## Accessing other database objects

Until now, only unqualified data access has been discussed. However, there are also other objects such as stored procedures and user-defined functions (UDFs) that can be called in an SQL environment with or without explicitly specifying the schema. Similar to tables and views, these objects can be qualified by separating the schema and object, depending on the naming conventions with either a slash (/) for System naming or a period (.) for SQL naming.

When the invocation of procedures and functions do not explicitly specify the schema, DB2 uses the SQL path instead of the default schema to find the procedures and functions.

### SQL path

The SQL path is pretty similar to a library list, where multiple schemas can be listed and are searched in the same sequence in which they are specified.

The initial value of the SQL path depends on the naming convention that is used for the first SQL statement within an activation group. If the System naming convention was used for the first SQL statement, the initial value for the SQL path is set to the special value, *LIBL. If the SQL naming convention was used, the SQL path includes the schemas in the following sequence: QSYS, QSYS2, SYSPROC, SYSIBMADM, USER special register.

The SQL path can also be set or changed by executing the SET PATH statement. The SET PATH statement allows multiple schemas to be listed and separated by a comma. The schemas explicitly specified in the SET PATH statement may or may not be part of the current library list. The special value, *LIBL, can be used to set the SQL path to the current library list, even when the SQL naming convention is used.

The default schema setting has no effect on the SQL path, which means that the **default schema is not included in the SQL path**. This is not an issue because the SQL path is **not** searched to find unqualified table, view, or alias objects.

In the following example, the current path is first assigned to the current library list and then changed to a list of schemas.

### Listing 15: SET PATH

```
SET PATH = *LIBL; SET PATH = QSYS, QSYS2, SALESPGM, HAUSER, HSCOMMON10;
```

## Conclusion

You should now understand the different behaviors when accessing database objects with either the System or SQL naming convention, especially to identify how unqualified access is handled differently for static and dynamic SQL statements.

Because of these different behaviors, you should decide on a single naming convention method for accessing your database objects with SQL.

- If you are working with typical IBM i applications, where the data is spread over multiple schemas and a library list is always used to resolve unqualified objects, the usage of System naming is probably the best option.
- When working with the System naming convention and dynamic SQL statements, running the SET SCHEMA statement should be avoided. As soon as the SET SCHEMA statement is run, the library list is no longer searched to find unqualified tables, views, or aliases in dynamic SQL statements.
- If System naming is not an option for you, but your data is located in multiple schemas and you want to avoid qualifying database objects, you should create aliases or views located in your primary data schema that point to the tables or views located in other schemas.
- If your data is concentrated in a single schema or your application is developed to use different database systems, using SQL naming will be the best choice.

## Resources

- Database information finder
- SQL messages and codes
- IBM i - DB2 for i SQL Reference - 7.1
- IBM i – Database SQL programming - 7.1
- Stored Procedures, Triggers, and User-Defined Functions on DB2 Universal Database for iSeries
- A Sensible Approach to Multi-Step DB2 for i Query Solutions
- IBM developerWorks DB2 for i Forum
- IBM developerWorks - IBM i Technology Updates