

# Lab BL01- Part I Piloting DB2 UDB for iSeries with iSeries Navigator V5R2



**ITSO iSeries Technical Forum 2003**



---

# Contents

---

<b>Part 1. Basic database functions</b> . . . . .	1
<b>Lab 1. iSeries Navigator setup and basic operations</b> . . . . .	3
Task 1: Configuring a PC connection to the iSeries server . . . . .	3
Task 2: Creating an alternate connection to the same iSeries server . . . . .	5
Task 3: Changing the Refresh option . . . . .	6
Task 4: Maintaining your active library list for iSeries Navigator . . . . .	7
Task 5: Creating a library/schema . . . . .	7
Task 6: Using the Find and Positioner features (optional) . . . . .	9
Task 7: Creating Database function shortcut on Windows desktop (optional) . . . . .	10
Task 8: Deleting a library/schema (optional) . . . . .	10
<b>Lab 2. SQL Script Center</b> . . . . .	11
Task 1: Creating a sample DB schema and setting up a JDBC connection . . . . .	11
Task 2: Using SQL Script Center . . . . .	14
Task 3: Running SQL scripts in debug mode (optional) . . . . .	17
Task 4: Using basic Visual Explain . . . . .	18
Task 5: Using SQL Assist to build your basic SQL statement . . . . .	23
Task 6: Running OS/400 CL commands from SQL Script Center (optional) . . . . .	27
<b>Lab 3. Working with tables using iSeries Navigator</b> . . . . .	29
Task 1: Creating a database table with a primary key constraint . . . . .	30
Task 2: Adding a Check constraint to a table . . . . .	32
Task 3: Adding rows to the table and testing the check constraint . . . . .	33
Task 4: Testing the primary key constraint . . . . .	35
Task 5: Generating SQL from existing database objects . . . . .	36
Task 6: Altering a table . . . . .	37
Task 7: Displaying properties and descriptions of DB objects . . . . .	38
Task 8: Using the Hot Link feature in the Quick View function (optional) . . . . .	40
<b>Lab 4. Other database tasks using iSeries Navigator</b> . . . . .	43
Task 1: Creating a view . . . . .	43
Task 2: Applying permissions to a database object . . . . .	47
Task 3: Creating a view with SQL statement (optional) . . . . .	49
Task 4: Displaying the latest SQL statement in other jobs (optional) . . . . .	50
<b>Lab 5. Journal management (optional)</b> . . . . .	53
Task 1: Viewing journals . . . . .	53
Task 2: Swapping journal receivers . . . . .	56
Task 3: Creating journals/receivers . . . . .	57
Task 4: Dropping journals/receivers . . . . .	58
<b>Part 2. Advanced database functions</b> . . . . .	59
<b>Lab 6. Database referential constraint</b> . . . . .	61
Task 1: Creating a primary key . . . . .	61
Task 2: Creating a referential constraint . . . . .	62
Task 3: Testing referential constraints . . . . .	65
<b>Lab 7. Database trigger</b> . . . . .	67
Task 1: Creating an SQL trigger for AFTER INSERT . . . . .	68

Task 2: Creating a column-level SQL trigger for AFTER UPDATE (optional) .	70
Task 3: Testing the SQL trigger for AFTER INSERT . . . . .	72
Task 4: Testing the column-level SQL trigger for AFTER UPDATE (optional) .	73
<b>Lab 8. Stored procedure . . . . .</b>	<b>75</b>
Task 1: SQL stored procedure that returns result sets . . . . .	76
Task 2: SQL stored procedure that returns the value of output parameter . . .	79
Task 3: Perform SQL source-level debugging (new in V5R2 - optional) . . . . .	81
<b>Lab 9. User-defined function . . . . .</b>	<b>87</b>
Task 1: Creating and using a scalar SQL UDF . . . . .	87
Task 2: Creating and using an SQL UDTF . . . . .	91

---

## Part 1. Basic database functions



---

## Lab 1. iSeries Navigator setup and basic operations

This lab explains the tasks that a database administrator performs during an initial iSeries Navigator setup.

The *XX* notation that appears in library names, profile names, and so on refers to your *team number* (for example, DBNAVXX, SAMPLEDBXX and LIBXX). Refer to your lab worksheet for details.

### Objectives

This lab teaches you how to:

- Configure a connection to the iSeries server from iSeries Navigator
- Maintain your working library list in iSeries Navigator
- Create and delete an OS/400 library/SQL schema
- Use Find and Positioner features and create database function shortcut

### Lab prerequisites

Before you begin this lab, be sure the following prerequisites are available:

- An IBM @server iSeries or AS/400e server with OS/400 V5R2 (or higher) with:
  - 5722-SS1 Option: Host Servers
  - 5722-SS1 Option: System Openness Includes
  - 5722-TC1 TCP/IP Connectivity Utilities
- A PC with Client Access Express V5R2M0 with the *latest* Service Pack applied.
- A Windows HOSTS file containing I400WS and I400WS2 entries that point to your iSeries server's IP address.
- User profiles DBNAVXX and DBNAVXX\_A created in the iSeries server.
- Use WRKRDBDIRE command to create a named database entry of I400WS as your \*LOCAL database. You can also use the existing \*LOCAL entry of your machine throughout the lab exercises, but be sure you recognize it when we refer to our assumed named database of I400WS.

### Time required

The time required to efficiently complete this lab is 20 minutes.

### Naming convention for a 'schema'

Starting at V5R1, the term "schema" is used in the same sense as the term "collection". This is an OS/400 library created with automatic DB journaling enabled and local DB catalog views.

---

## Task 1: Configuring a PC connection to the iSeries server

### Before you begin

Be sure that you have met the prerequisites that are outlined for this lab. You must also have completed the lab setup instructions.

Your first task is to create a connection definition from the iSeries Navigator on your PC to the iSeries server:

- \_\_\_ 1. Double-click the **iSeries Navigator** icon on your Windows Desktop.

If this is your first time running iSeries Navigator on your PC, a message window appears indicating that there are no connections to the server yet. Click **Yes**. The Add Connection - Welcome window appears.

If this is not your first time running iSeries Navigator and you need to define a connection, follow these steps.

  - a. Right-click the **My Connections** icon in the left panel.
  - b. Select **Connections to Servers**.
  - c. Click **Add connection...** The Add Connection - Welcome window appears.
- \_\_\_ 2. In the Server input field, type I400WS and click **Next**. The Signon Information window appears. Select the **Use default User ID, prompt as needed** radio button and type your team profile (DBNAVXX) in the input field shown in Figure 1 (XX is your team number).

**Note**

For the purposes of this lab, the server names I400WS and I400WS2 are used for the primary and alternate connections. If you decide to use different server names, you must make appropriate changes throughout this and subsequent lab documents.

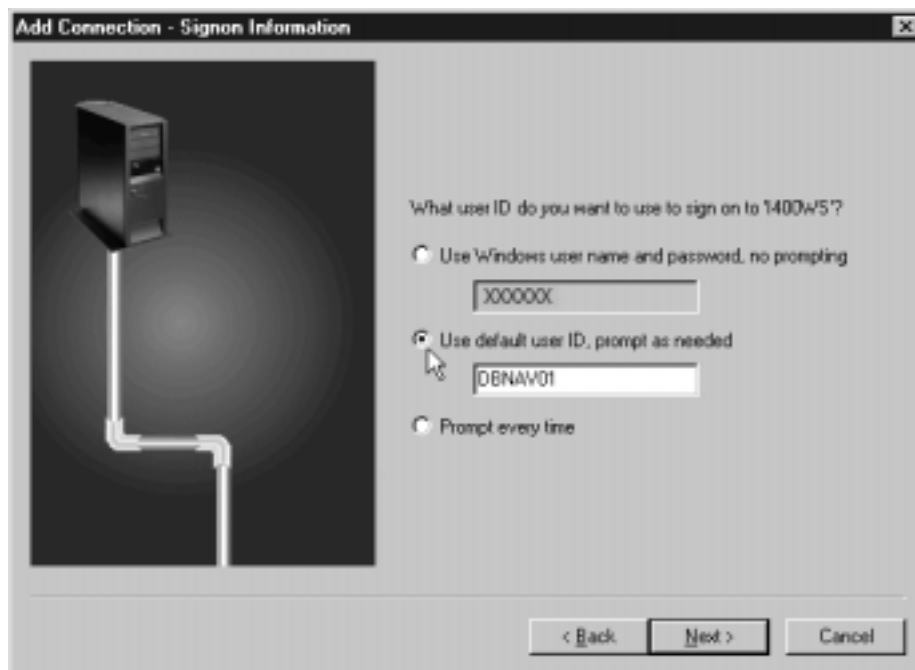


Figure 1. Signon Information window

- \_\_\_ 3. Click **Next**. The Verify Connection window appears. Click **Verify Connection** to verify that the Host Servers functions on the server are up and running. If you encounter an error message in this step, notify your lab



supervisor. When the verification is successful, click **OK** and then click **Finish**.

---

## Task 2: Creating an alternate connection to the same iSeries server

In this task, you create an *alternate* connection to the *same* iSeries server:

- \_\_\_ 1. From the main iSeries Navigator window, right-click the **My Connections** icon in the left panel. Select **Connection to servers** and **Add Connection....** The Add Connection - Welcome window appears.
- \_\_\_ 2. In the Server input field, type `I400WS2` and click **Next**. The Signon Information window appears. Select the **Use default User ID, prompt as needed** radio button, and type `DBNAVXX_A` in the input field below the radio button. Make sure you use the *alternate* user profile (`DBNAVXX_A`) for this step.
- \_\_\_ 3. Click **Next**. The Verify Connection window appears. Click the **Verify Connection** button. When the verification finishes, click **OK** and then click **Finish**.
- \_\_\_ 4. From the iSeries Navigator window, click the plus sign (+) in front of the **I400WS** connection. The Signon window (Figure 2) appears, on which you must enter your user name and password.

Refer to your lab worksheet for your user name and password. When you are finished, click **OK**.

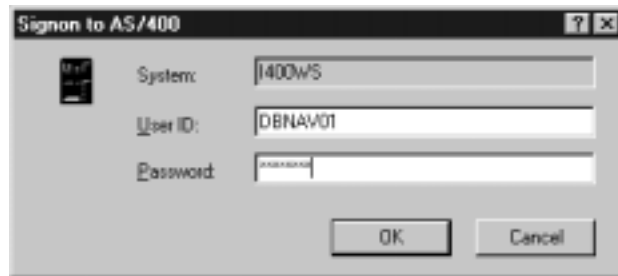


Figure 2. Signon to the iSeries server

Because this is your first time signing on to the iSeries server, the client checks the server for the available functions to be displayed in the left panel.

- \_\_\_ 5. Click the plus sign (+) in front of the **Databases** icon to expand it. Then click the plus sign (+) in front of the **Named Database** icon (*I400ws* in this case). The Libraries, Database Navigator, SQL Performance monitors, and Transactions options appear in the expanded list (Figure 3).

### Note

You see a different name for the Named Database icon if you let the system create it automatically.

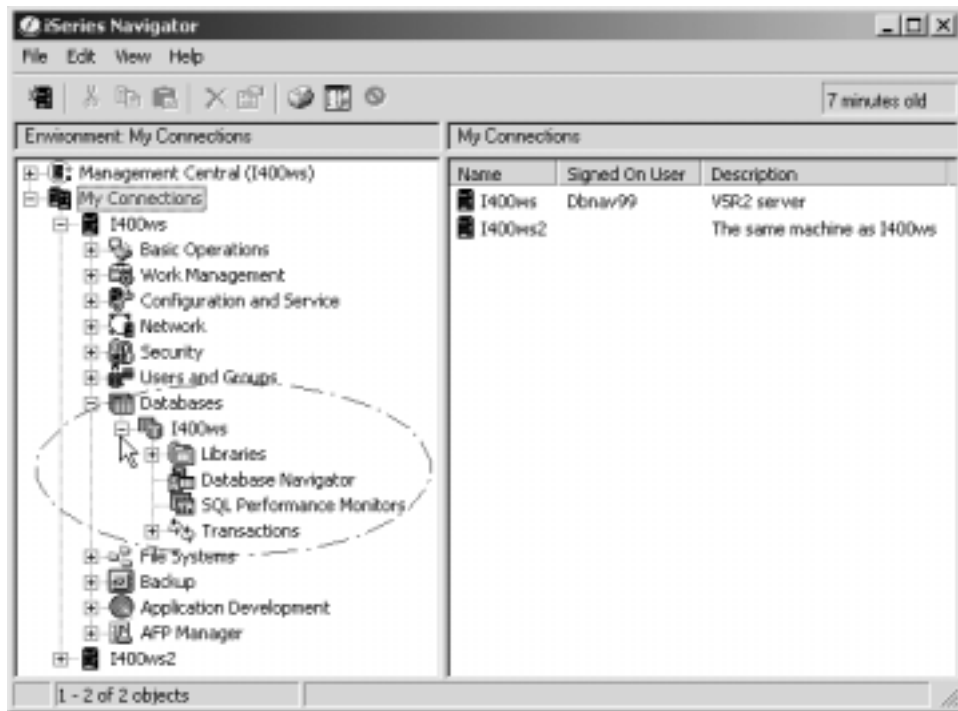


Figure 3. Main iSeries Navigator window

#### New in V5R2: Databases icon

With V5R2 support of OS/400 Independent Disk Pool (IASP) for DB2 storage space, you can now create multiple Named Databases using the WRKRDBDIRE command. This serves the same purpose of failover clustering that was introduced in V5R1 for OS/400 User-defined File System (UDFS).

Each Named Database resides in separated auxiliary storage pool. The one created in System ASP (ASP 1) is called SYSDATABASE. The rest are called "User Database".

You work with SYSDATABASE in this entire lab.

### Task 3: Changing the Refresh option

In this task, you change the Auto Refresh option for the database work area on the right panel of iSeries Navigator window. If you do not change the Refresh option, the screen is only refreshed when you click the Refresh icon on the toolbar, press F5, or change an object.

- \_\_\_ 1. Right-click the **Databases** icon to display the pop-up context menu. Then select **Customize this View** and then **Auto Refresh...** option. The Database Auto Refresh window appears.
- \_\_\_ 2. Deselect **Use automatic refresh options from parent folder** if it is selected.
- \_\_\_ 3. Select the **Refresh contents every time list is displayed** check box and click **OK**. The screen now refreshes each time a list is displayed.

---

## Task 4: Maintaining your active library list for iSeries Navigator

This task explains how to maintain the list of libraries that you want to work with in iSeries Navigator.

### The initial active library list of iSeries Navigator

When you use iSeries Navigator for the first time, the active library list (under the Libraries icon) contains all the libraries in the *user part* of your user profile library list. If you make any changes to the list (which you do in this task), the user-part library list is no longer in effect.

- \_\_\_ 1. Click the plus sign (+) in front of the **Libraries** icon to display the current active list. There may be no library listed at all, depending on the user part of your profile's library list.
- \_\_\_ 2. Right-click the **Libraries** icon and select **Select Libraries to Display** from the pop-up context menu. The Select Libraries to Display window appears.
- \_\_\_ 3. In the Enter libraries input field, type `QSYS2 QGPL QTEMP` (each separated by a space), and click **Add**. If `QGPL` and `QTEMP` are already in your active library list, just type `QSYS2` and click **OK**.

The `QGPL`, `QSYS2`, and `QTEMP` libraries are added (in alphabetical order) to the list in the right panel of the window.

### Note

The client always passes the library name to the server to check for its existence before adding it to the list. Therefore, if you incorrectly type the library name, you may see the error message "Library xxxxxx does not exist". It also checks your access authorization to the library. If you do not have proper access right, you will see the error message.

- \_\_\_ 4. Click **OK**. The three libraries are added to the active library list (Figure 4).



Figure 4. iSeries Navigator: Active library list

- \_\_\_ 5. Try removing a library from the active list by right-clicking **QTEMP** and selecting **Remove from List**. Wait a little while. The screen refreshes and the `QTEMP` library disappears from the active library list.

---

## Task 5: Creating a library/schema

This task explains creating a library/schema, the available options at the time of creation, and viewing the properties of a library:

- \_\_\_ 1. In the left-hand panel of iSeries Navigator main window, right-click the **Libraries** icon and select **New Library**.

\_\_\_ 2. Name the new library **LIBXX** and enter a description for the library as:

SQL schema for team DBNAVXX

Here, XX is your team number.

\_\_\_ 3. Select the **Add to list of libraries displayed** (should be selected by default) and **Create as an SQL schema** options.

Do *not* select *Create a data dictionary*. This is *not* an SQL data dictionary. It is an OS/400 IDDU dictionary. See Figure 5.

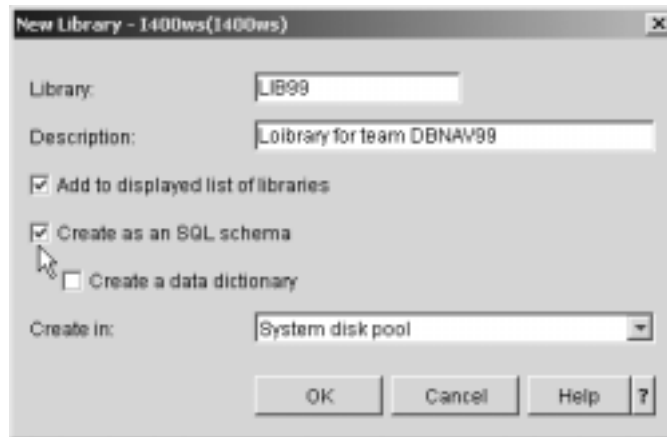


Figure 5. New Library window

#### A note on the OS/400 SQL ‘schema’

An SQL schema (called “collection” before V5R1) is an OS/400 library created with a journal, a journal receiver, local catalog views, and, optionally, IDDU data dictionaries (used for S/36 backward compatibility).

All tables created in an SQL schema are *automatically* associated to the database journal object in that corresponding schema.

OS/400 database journal is generally equivalent to what is called a “database transaction log” or “database redo log” on other database products.

The *local* catalog views in a schema contain information of all tables, views, indexes, packages, and other database-related entities created in that corresponding schema. These are different from the *system-wide* catalog views (in the QSYS2 library), which cover all database-related entities in the server.

OS/400 catalog views are generally equivalent to what is called “data dictionary” in other database products.

\_\_\_ 4. Click **OK** to create the library. The new library is automatically added into the active library list under the Libraries icon as a result of the option you selected in the previous step.

\_\_\_ 5. Under Libraries, click the newly-created **LIBXX** schema icon. In the right panel, you can see that catalog views, a journal, and a journal receiver were automatically created in your schema.

- \_\_\_ 6. Right-click the **LIBXX** icon and select **Properties**. A new window showing all the properties of your library in four different tabs appears (General, Storage, Save, Creation). Explore these properties and click **OK** to return to the main iSeries Navigator window.

## Task 6: Using the Find and Positioner features (optional)

A large number of objects may occasionally exist in a library. iSeries Navigator provides two methods to help you locate an entry in a list of objects:

- **The Find function:** This allows you to find the next occurrence of a text string in a list. This is a *modeless window*, so you can keep using it to find the next instance. It supports searching up or down, case-sensitive/insensitive, and partial/whole word searches.

### A modeless window

A modeless window can be opened without locking the parent window from which it is invoked. This means that you can open a modeless window and switch back to work in its parent window.

- **The Positioner function:** This allows you to position quickly to an item by pressing a single keyboard character. This is similar to the Find function that works only on the *first character* of the object names in a list.

- \_\_\_ 1. Under **Libraries** in the left panel, click the **QGPL** library. The list of all objects in this library appears in the right panel.
- \_\_\_ 2. Click the **Name** field bar to sort the list before you search for a particular object. The objects in the QGPL library are now sorted by name (Figure 6). Clicking once more on the Name field bar results in sorting in reverse order (ascending <--> descending toggle).

You can also sort by object type or by description by clicking the corresponding field bars.

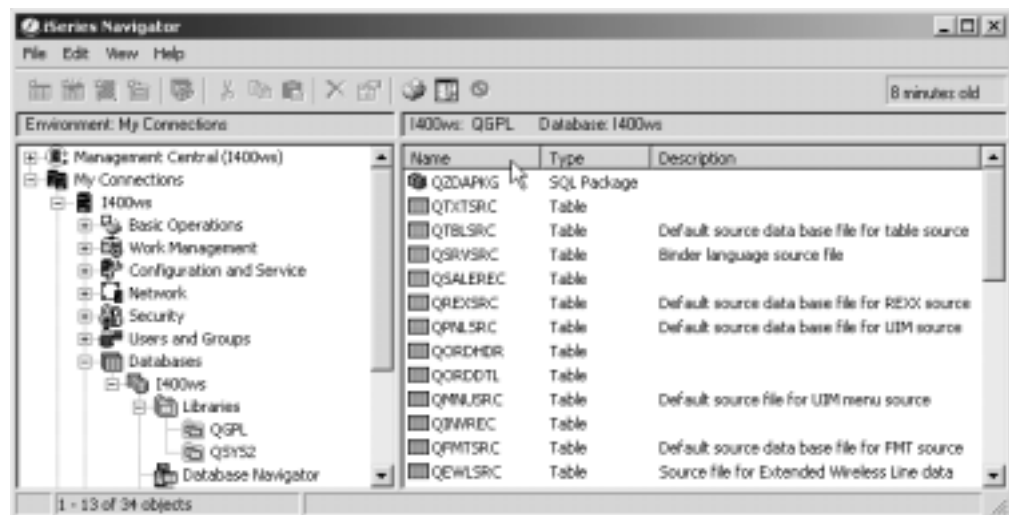


Figure 6. Sorting objects by name

- \_\_\_ 3. Select **Edit->Find** from the menu bar (or press Ctrl+F). The QGPL - I400WS - Find window appears. In the Search for field, type `src` and click **Find**. iSeries Navigator locates the first occurrence of the “src” text string in the list by highlighting the object name.

**A note on the Find function**

The Find function searches text strings in the Name and Description fields of each object in the list.

- \_\_\_ 4. Click **Find** repeatedly to progress down the list. Click **Close** when you are finished.
- \_\_\_ 5. To use the positioner function, you must place the cursor on one of the objects in the list. Go back and click the name of the *first* object in the list. The name should be highlighted.
- \_\_\_ 6. Press the letter Q on the keyboard. iSeries Navigator finds the first occurrence of an object that starts with the letter Q.
- \_\_\_ 7. Press Q repeatedly to progress down the list.

---

## Task 7: Creating Database function shortcut on Windows desktop (optional)

iSeries Navigator allows you to create a shortcut icon on a Windows desktop for all its main functions, including Database, Named Database, and Libraries. This helps you directly launch the function of your interest without having to start the main iSeries Navigator first.

On the right-hand panel of iSeries Navigator window, right-click the **Databases** icon and select **Create Shortcut**. You now create a shortcut icon on Windows desktop for Databases function.

Shortcuts for each Named Database (I400ws in this case), Libraries functions, and many other iSeries Navigator functions can also be created this way. Look for **Create Shortcut** in the pop-up context menu.

---

## Task 8: Deleting a library/schema (optional)

You can delete a library/schema using iSeries Navigator if you have proper authority to do so. We do not delete the library since we will use it in later.

- \_\_\_ 1. On the right-hand panel of iSeries Navigator window, right-click the library **LIBXX** and select **Delete....** The Confirm Object Deletion window appears containing a single entry of LIBXX.
- \_\_\_ 2. Click **Delete** and you see a message: “This object has dependent tables, views, indexes, or constraints. Are you sure you want to delete this object and all dependent objects?”.
- \_\_\_ 3. Click **Cancel** since we will use the library later. If you click **Yes**, LIBXX disappears from under the Libraries icon.

You have now completed this lab!

---

## Lab 2. SQL Script Center

This lab teaches you how to use the SQL Script Center, which is an integrated component of the iSeries Navigator - Database function.

The notation *XX* that appears in library names, profile names, and so on refers to your *team number*.

### Objectives

This lab teaches you how to:

- Create, run, and save SQL statements and OS/400 CL commands
- Run SQL Script Center in debug mode and view job log output
- Use SQL Assist feature (new in V5R2)
- Use the basic Visual Explain function (enhanced in V5R2)

### Lab prerequisites

You must complete *Lab 1, "iSeries Navigator setup and basic operations"* on page 3, before you proceed with this lab.

### Time required

The time required to efficiently complete this lab project is 25 minutes.

### Naming convention for a 'schema'

Starting in V5R1, the term "schema" is used in the same sense as the term "collection". This is an OS/400 library created with automatic DB journaling enabled along with local DB catalog views.

---

## Task 1: Creating a sample DB schema and setting up a JDBC connection

In this task, you run a system-provided stored procedure to create a sample schema with which you will work. You also set up the JDBC connection parameters used by SQL Script Center.

- \_\_\_ 1. Under the **Databases** icon, right-click the database **I400WS** icon and select **Run SQL Scripts...** to launch the SQL Script Center. A new Run SQL Scripts window appears.
- \_\_\_ 2. Click the **Options** menu item and select the following options (one at a time) as shown in Figure 7:
  - Stop on Error
  - Smart Statement Selection
  - Run Statement on Double-Click

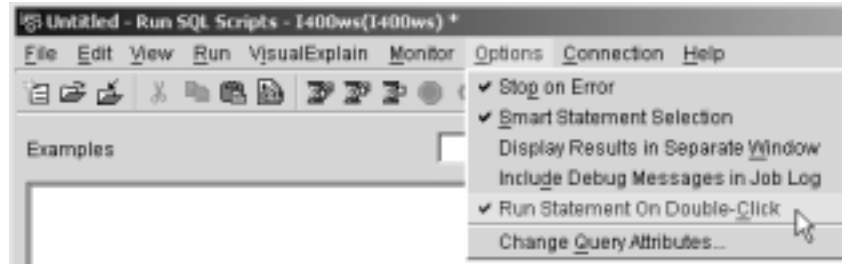


Figure 7. Run SQL Scripts: Options menu items

**Note**

When the Smart Statement Selection option is selected, all highlighted SQL statements run in sequence. If this option is not selected, the highlighted SQL statements are executed as single statements. This option also ensures that complete statements are run, even if one or more statements are only partially highlighted.

3. Invoke the DB2-supplied stored procedure that creates a sample schema. You use the schema in subsequent tasks and labs. In the SQL statement working area, enter the following statement:

```
CALL QSYS.CREATE_SQL_SAMPLE('SAMPLEDBXX');
```

Be sure to replace *XX* with your team number.

**Case sensitivity in SQL statements**

Note that SQL commands are *not* case-sensitive. However, SQL-related names, such as table and column names and character strings data *are* case-sensitive.

Be sure to include a *semicolon (;)* at the *end* of every SQL statement to ensure that the Smart Statement Selection feature knows where each statement ends. The SQL Script Center recognizes a semicolon as the SQL statement separator.

**A note on CREATE\_SQL\_SAMPLE**

As of V5R1, the CREATE\_SQL\_SAMPLE stored procedure is included in the QSYS library. You call it to create a sample schema that contains many database objects, such as tables (with rows of data), views, indexes, aliases, constraints, journal, and journal receiver. This sample schema is about 10 megabytes in size.

This procedure takes one parameter for the name of the library/SQL schema to be created for the database. We suggest that you use a name (of up to ten characters) typed in single quotation marks (for example 'SAMPLEDB01').

You can find details of this sample schema in Appendix A of the V5 version of the manual *DB2 UDB for iSeries SQL Programming Concepts*.



- \_\_\_ 4. Move the text cursor anywhere into the statement and double-click to run it. Monitor the Messages tab for a completion notification, which indicates that you now have a sample schema to work with.

If there is any error message, notify your lab supervisor.

**Note**

For convenience, *double-click* each SQL statement to run it in all exercises, unless you are instructed otherwise.

Since V5R1, the SQL Script Center uses JDBC to connect to the server. To maintain compatibility for those who are familiar with an ODBC connection, JDBC connection parameters are generally equivalent to those previously used by ODBC.

- \_\_\_ 5. In the menu bar, click **Connection-> JDBC Setup**. The JDBC Setup window appears.
- \_\_\_ 6. Click the **Server** tab and type `SAMPLEDBXX` in the SQL default library input field.

Type your *team number* in place of `XX`.

From this point on, `SAMPLEDBXX` is your SQL default library. You do not have to specify a fully-qualified object name in SQL statements if that object is in the `SAMPLEDBXX` library. For example, you can specify `EMPLOYEE`, rather than `SAMPLEDBXX.EMPLOYEE`. However, if the object is *not* in the library list, you need to specify a fully-qualified name.

Set Commit immediate to `*NONE` under Commit mode. In a real-life application setting, you would set this parameter to `*CS`, `*CHG`, `*ALL`, or `*RR` rather than `*NONE` to ensure data integrity within each application transaction.

- \_\_\_ 7. Click the **Format** tab, select **SQL(\*SQL)** under Naming convention, and click **OK**.

This requires you to use a dot (.) as a separator symbol in a fully-qualified DB object name (for example, `schema.table`).

A naming convention of `*SYS` requires that you use a slash (/) symbol as the separator (for example, `library/object`).

**Note**

With `*SQL` naming convention, both the SQL Default Library and Library List parameters (in the Server tab) are active for use. But with `*SYS` naming, only Library List is active.

- \_\_\_ 8. You should now be in the Run SQL Scripts window. In the SQL statement working area, enter and run the following statement:

```
SELECT * FROM VEMPDP1 ORDER BY EMPNO;
```

The selected rows from the view appear in the lower panel of the window (in a *separate* result tab), as shown in Figure 8.

SELECT \* FROM VEMPDP1 ORDER BY EMPNO;

DEPTNO	DEPTNAME	EMPNO	FRSTINIT	MDINIT	LASTNAME	WORKDEPT
A00	SPIFFY COMPUTER SERVICE DM.	000010	C	I	HAAS	A00
B01	PLANNING	000020	M	L	THOMPSON	B01
C01	INFORMATION CENTER	000030	S	A	KWAN	C01
E01	SUPPORT SERVICES	000050	J	B	GEYER	E01
D11	MANUFACTURING SYSTEMS	000060	I	F	STERN	D11
D21	ADMINISTRATION SYSTEMS	000070	E	D	PULASKI	D21
E11	OPERATIONS	000090	E	W	HENDERSON	E11
E21	SOFTWARE SUPPORT	000100	T	Q	SPENSER	E21
A00	SPIFFY COMPUTER SERVICE DM.	000110	V	G	LUCCHESI	A00
A00	SPIFFY COMPUTER SERVICE DM.	000120	S		O'CONNELL	A00
C01	INFORMATION CENTER	000130	D	M	QUINTANA	C01
C01	INFORMATION CENTER	000140	H	A	NICHOLLS	C01
D11	MANUFACTURING SYSTEMS	000150	B		ADAMSON	D11
D11	MANUFACTURING SYSTEMS	000160	E	R	PIANKA	D11
D11	MANUFACTURING SYSTEMS	000170	M	I	VOCUMBERG	D11

Messages: SELECT \* FROM VEMPDP1 ORDER BY EMPNO

Figure 8. Results of the SELECT \* FROM VEMPDP1 statement

#### A note on the Script Center result view

When you want to close the result tab, click **Edit-> Clear Results** from the menu bar.

If you want to display the results in a separate window (like what it was before V5R1), click **Options-> Display Results in Separate Window** before you run the statement.

- \_\_\_ 9. Add the following line to the end of the original SQL statement:

```
FETCH FIRST 5 ROWS ONLY;
```

The statement looks like this:

```
SELECT * FROM VEMPDP1 ORDER BY EMPNO FETCH FIRST 5 ROWS ONLY;
```

Run the SQL statement. Only the first five rows appear in the result panel.

#### FETCH FIRST n ROWS ONLY

*FETCH FIRST n ROWS ONLY* is a new SQL syntax added to V5R1. It is most suitable for filtering out the “Top n” rows from the result set. Normally, you would use it in conjunction with the ORDER BY statement for a meaningful view of the results.

You are now ready to explore other features of the V5R2 SQL Script Center.

## Task 2: Using SQL Script Center

In this task, you create, run, and save SQL scripts using the Script Center. You also learn many Script Center functions.

- \_\_\_ 1. Switch to the Run SQL Scripts window and remove any existing SQL statements from the working area.

- \_\_\_ 2. The SQL statement examples drop-down list displays supported SQL statements and their proper syntax (Figure 9).



Figure 9. SQL Script Center: Selecting an SQL statement example

The list can be used as a template for your SQL statements. Scroll down the list in the Data Manipulation Statements section and click the second SELECT statement:

```
SELECT * FROM QSYS2.SYSTABLES WHERE TABLE_NAME LIKE 'FILE%';
```

- \_\_\_ 3. Click **Insert** to place the selected SQL statement into the working area.
- \_\_\_ 4. Modify the statement and run it. The edited statement should look like this example:

```
SELECT * FROM QSYS2.SYSVIEWS WHERE TABLE_NAME LIKE 'VE%';
```

Here you should see all the table names that start with VE.

If the syntax is correct, you should see the returned rows. If the syntax is incorrect, a message indicates why it does not execute properly.

- \_\_\_ 5. If the Messages tab does not provide sufficient information regarding the operation of your SQL statements, you can view the OS/400 job log for additional information, if there is any.

To do this, click **View-> Job Log**. A new XXXXXX/Qzdasoinit/Quser Job Log window appears.

- \_\_\_ 6. To view a second level message of any message in the job log, double-click the message ID of the line you want to view. A new Message Details window appears showing all the detailed information about that message. You can also see the Advanced button that can display further details of that message.

Close the Message Details window and the Job Log window.

At this point, you know how to display a job log of your activities in the SQL Script Center.

- \_\_\_ 7. Click **File->Save As**. The Save As window appears. In the Look in drop-down list, open the C:\temp directory (or any temporary directory on your PC) and type `MySQL` in the File name input field (Figure 10).

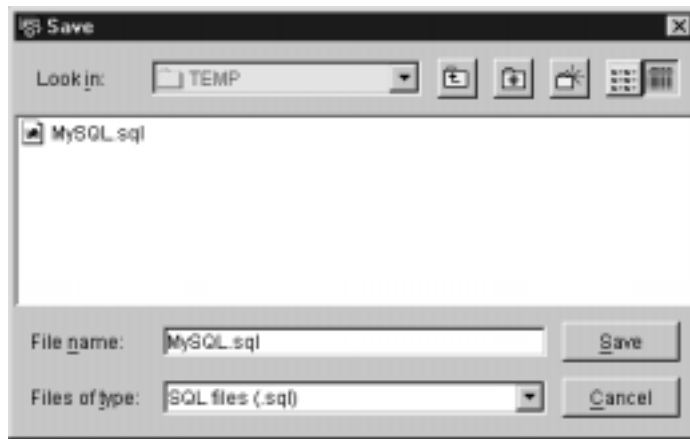


Figure 10. SQL Script Center: Saving an SQL script

- \_\_\_ 8. Click **Save** to return to the Run SQL Scripts window. Then, close the Run SQL Scripts window.
- \_\_\_ 9. Create a shortcut icon of MySQL.sql on your Windows desktop. To do this, using Windows Explorer, right-click the file **MySQL**, select **Send To**, and then select **Desktop (Create Shortcut)**.
- \_\_\_ 10. Minimize all windows and double-click the **MySQL.sql** shortcut (Figure 11).  
If you do not know how to create a shortcut, go directly to the **MySQL.sql** file and double-click it.



Figure 11. SQL Script Center: MySQL.sql shortcut icon

- \_\_\_ 11. The Run SQL Scripts window appears. From the toolbar, click the **Run All** icon (Figure 12).



Figure 12. SQL Script Center: Run All icon

- \_\_\_ 12. A small Connect To Server window appears and prompts you for the server name you want to connect. Select or type **I400WS** and click **OK**.
- \_\_\_ 13. A Signon to the Server window may appear prompting you to log on to the target server. Type your team user profile and password and click **OK**.  
The result should appear in a window. Close the result tab by clicking **Edit->Clear Results**.

### SQL Script Center file association

When you install the Database function of the iSeries Navigator, the “.sql” file name extension is automatically associated with the SQL Script Center. Therefore, you can run it without having to invoke iSeries Navigator.

## Task 3: Running SQL scripts in debug mode (optional)

This exercise explains how to run an SQL script in debug mode and view the messages in the job log. Debug mode messages can help you determine any possible problems or figure out the access plan when executing SQL statements.

1. Click the **New** icon on the toolbar (or click **File-> New** from the menu bar). If you are prompted to save the existing script, select **No**.

Then, type and run the following SQL statement:

```
SELECT * FROM EMPLOYEE WHERE EMPNO IN ( SELECT EMPNO FROM EMPPROJECT  
WHERE PROJNO = 'MA2113' ) ORDER BY EMPNO;
```

This query identifies all employees who are engaged in project number MA2113. The resulting rows should appear.

2. From the menu bar, click **View-> Job Log** to display the job log window. You should *not* see any message about database operations. Do *not* close the job log window.
3. Switch back to the Run SQL Scripts window by pressing Alt+Tab and clicking **Options-> Include Debug Messages in Job Log** from its menu bar.
4. Run the same SQL statement *again* by double-clicking it. You should see the same result.
5. Switch to the job log window by pressing Alt+Tab and pressing F5 to *refresh* the contents in Job Log window. You should now see a few messages about database operations of the query engine (Figure 13).

Message ID	Message Text	Sent	Message Type	Severity
CPI434B	**** Ending debug message for query .	3/15/02 9:33:53 AM	Information	0
CPI4326	File EMPPROJECT processed in join position 2.	3/15/02 9:33:53 AM	Information	0
CPI4326	File EMPLOYEE processed in join position 1.	3/15/02 9:33:53 AM	Information	0
CPI4322	Access path built from keyed file EMPPROJECT.	3/15/02 9:33:53 AM	Information	0
CPI432C	All access paths were considered for file EMPPROJECT.	3/15/02 9:33:53 AM	Information	0
CPI432C	All access paths were considered for file EMPLOYEE.	3/15/02 9:33:53 AM	Information	0
CPI432B	Subselects processed as join query.	3/15/02 9:33:53 AM	Information	0
CPI433A	Unable to retrieve query options file.	3/15/02 9:33:53 AM	Information	0
CPI433A	Unable to retrieve query options file.	3/15/02 9:33:53 AM	Information	0
CPI4323	The QS/400 Query access plan has been rebuilt.	3/15/02 9:33:53 AM	Information	0
CPI434A	**** Starting optimizer debug message for query .	3/15/02 9:33:53 AM	Information	0
CPI432A	Unable to retrieve query options file.	3/15/02 9:33:53 AM	Information	0

Figure 13. Query Optimizer messages in Job Log window

If you do not see the messages shown in Figure 13, review what you have done so far.

### Browsing the job log window

The messages in the job log window are sorted so that the *most recent* message appears as the *top* line. You can also observe the “Time Sent” column to identify the time sequence of all the messages.

You can see where the debug message starts and ends in the job log by locating the following messages (Figure 13):

```
CPI434A *** Starting optimizer debug message for query
CPI434B *** Ending optimizer debug message for query
```

- \_\_\_ 6. Double-click any message ID to see a new Message Details window for that particular message. Close the window when you finish viewing the details.

Close the job log window when you are finished, and you are back to SQL Script Center window.

#### Note

To use debug mode, you must have \*USE authority to OS/400 command object STRDBG. This authority can be arranged for you by a security officer.

An alternative to tracing the query engine debug messages in the job log (and much easier to use) is the Visual Explain. Task 4 introduces you to this tool.

---

## Task 4: Using basic Visual Explain

Introduced in OS/400 V4R5, the Visual Explain helps you more easily analyze messages from the OS/400 query optimizer. Rather than browsing for database-related messages in the job log (as you did in the preceding task), you use Visual Explain to filter them and display a graphical representation that contains detailed useful information and is easier to understand.

- \_\_\_ 1. Go back to Run SQL Scripts window and place the text cursor anywhere in the statement:

```
SELECT * FROM EMPLOYEE WHERE EMPNO IN ( SELECT EMPNO FROM EMPPROJECT
WHERE PROJNO = 'MA2113' ) ORDER BY EMPNO;
```

This is the same SQL statement as in Task 3. You may need to retype the statement if it is not already in the working area.

- \_\_\_ 2. From the menu bar, click **Visual Explain-> Explain**. A new Visual Explain window appears that shows a graphical representation of the resulting *access plan* of the SQL statement along with other information as shown in Figure 14.

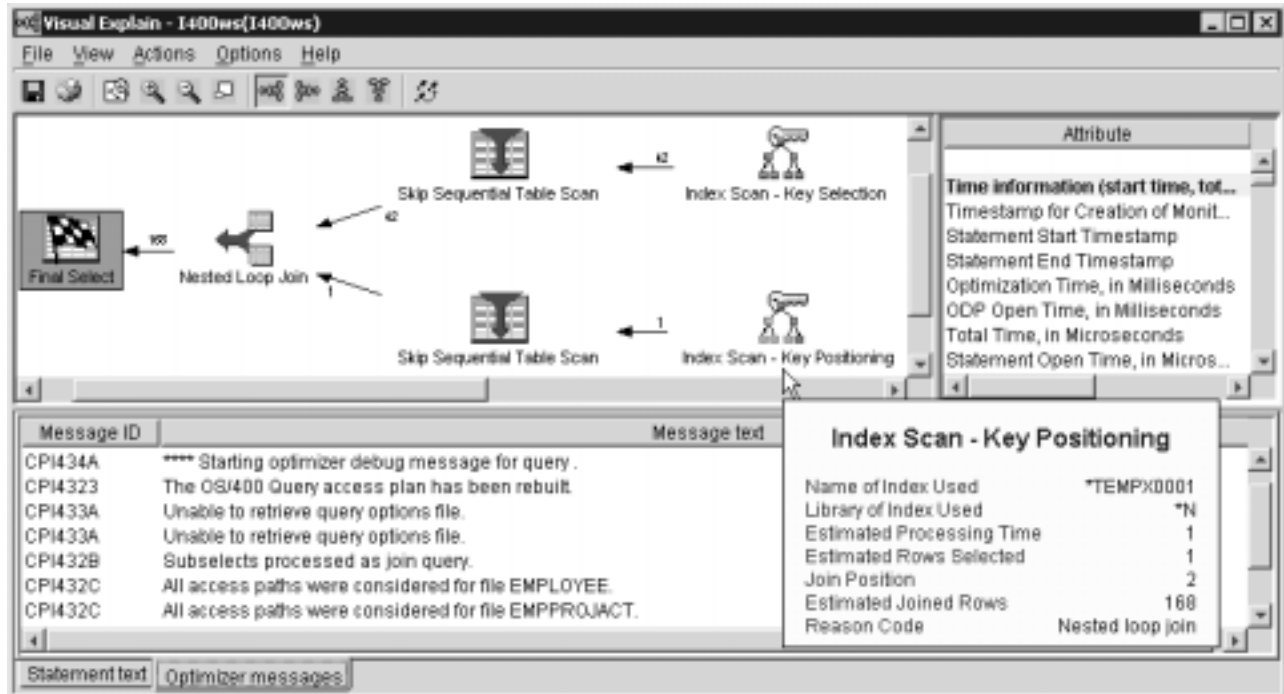


Figure 14. A Visual Explain window showing the access plan of a query

- \_\_\_ 3. In the top-left panel, you can scroll left to right to see the entire graphical representation of the resulting access plan.
- \_\_\_ 4. Go to the bottom panel of the window and click the **Optimizer Messages** tab.

If you do not see any messages, click **Options** and select **Show Optimizer Messages**. Messages similar to those shown in the Job Log window (Figure 13 on page 17) appear. However, the messages are sorted in a reverse order (the *most recent* message appears as the *bottom-most* line).

Visual Explain filters only messages that relate to the DB2 optimizer engine for you in the Optimizer message tab (Figure 14). This saves you time from manually browsing in the job log, which can sometimes contain too many other messages.

Double-click a message of your interest to *expand* the detailed information of that message. Double-click the detailed information again to *collapse* it.

- \_\_\_ 5. Browse the detailed query attributes information in the top-right panel of the Visual Explain window.
- \_\_\_ 6. Go to the top-left panel of the window, move the pointer over an icon in the graph, and leave it there for a short while. A fly-over panel appears showing information about the operation that the icon represents.



Figure 15. Visual Explain: Fly-over panel example

- \_\_\_ 7. In V5R2, we make it easier than in V5R1 for you to identify any possible query engine advisor information by providing a new toolbar icon that brings up such information in details. On the toolbar, click the **Statistics and Index Advisor** icon (Figure 16).

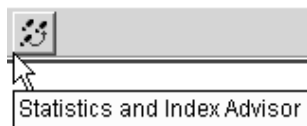


Figure 16. Visual Explain: Statistics and Index Advisor icon (new in V5R2 client)

A new Statistics and Index Advisor window appears. It contains two tabs: Statistics Advisor and Index Advisor.

Click the **Index Advisor** tab. You should see one entry for the table EMPPROJACT.

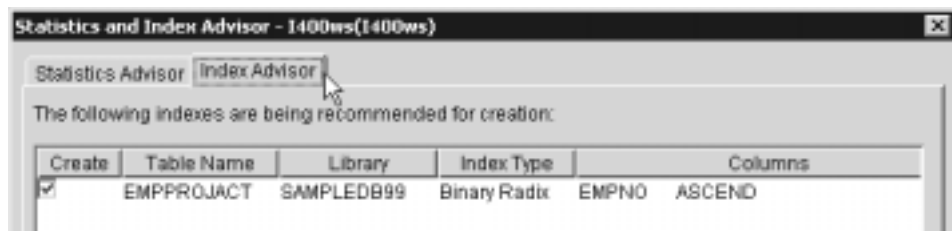


Figure 17. Visual Explain: Statistics and Index Advisor window

The Columns bar indicates that an index should be created with the column EMPNO in ASCENDING order. Notice a Create... button at the bottom-right corner of the window. This helps you with an immediate index creation. You do not need to click it.

- \_\_\_ 8. Click **OK** to close the Statistics and Index Advisor window and return to Visual Explain window.
- \_\_\_ 9. Change the orientation of the graph to a vertically aligned format by clicking the **Orient Bottom** icon in the toolbar (Figure 18).



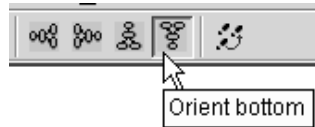


Figure 18. Visual Explain: Orient Bottom icon

You should see the alignment change accordingly.

- \_\_\_ 10. In V5R1, a new print capability is added to Visual Explain to help you produce hardcopy outputs. From the menu bar, click **File-> Print Preview....** The preview window appears with a preview of the graph displayed (Figure 19).

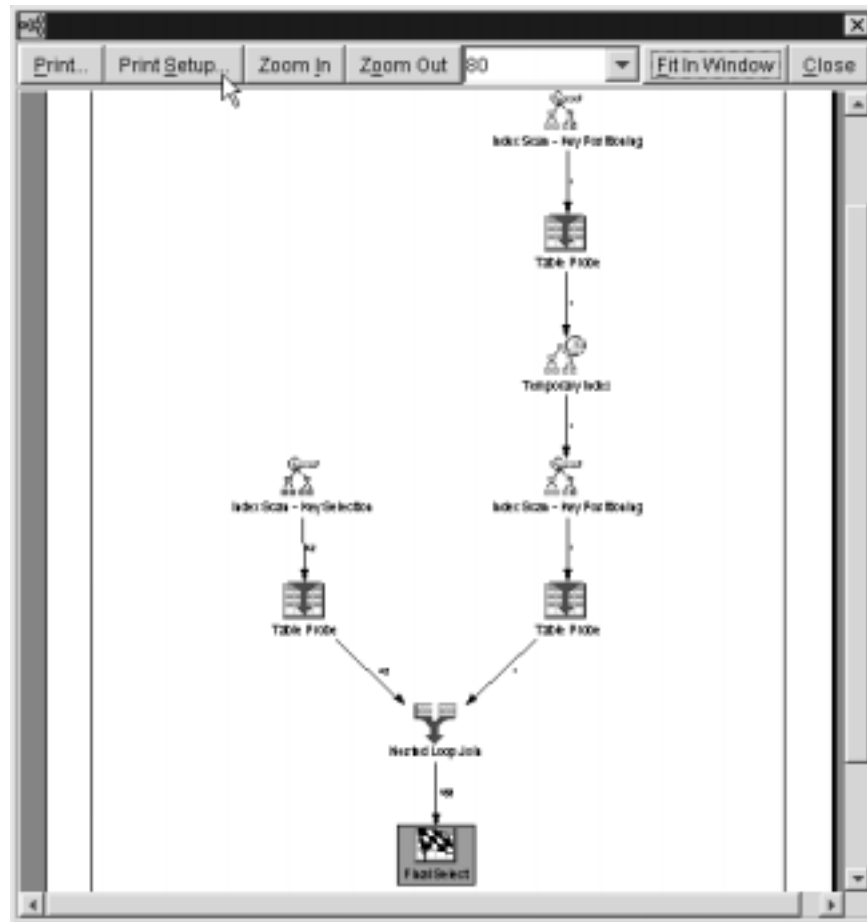


Figure 19. Visual Explain: Print Preview window

- \_\_\_ 11. Click the **Print Setup** button to bring up the window that contains many print setup parameters that you can tailor to your hardcopy requirements. Explore the parameters as you choose.
- \_\_\_ 12. Click **Cancel** to close the Print Setup window. Then, click **Close** in the Print Preview window to return to Visual Explain window.

Sometimes, you may want to save the resulting access plan for later use.

- \_\_\_ 13. From the menu bar, click **File-> Save As SQL Performance Monitor**. Then specify a name of `QUERYXX` (`XX` is your team number) for the saved data and specify the save library as `SAMPLEDBXX` (`XX` is your team number). Click **OK** to save the data and return to Visual Explain window. Then, close the Visual Explain window.
- \_\_\_ 14. You can then open the saved data later for analysis from SQL Script Center. Go to SQL Script Center window and click **Monitor-> List Explainable Statements** from its menu bar. A new Explainable Statements window appears.
- \_\_\_ 15. Select `QUERYXX` (`XX` is your team number) from the drop-down list Recent performance monitors. A few entries appear under SQL statements monitored list box.
- \_\_\_ 16. Click *each* entry once until you see the following statement in SQL statement selected box ( ):

```
SELECT * FROM EMPLOYEE WHERE EMPNO IN ( SELECT EMPNO FROM EMPPROJECT
WHERE PROJNO = ? ) ORDER BY EMPNO
```

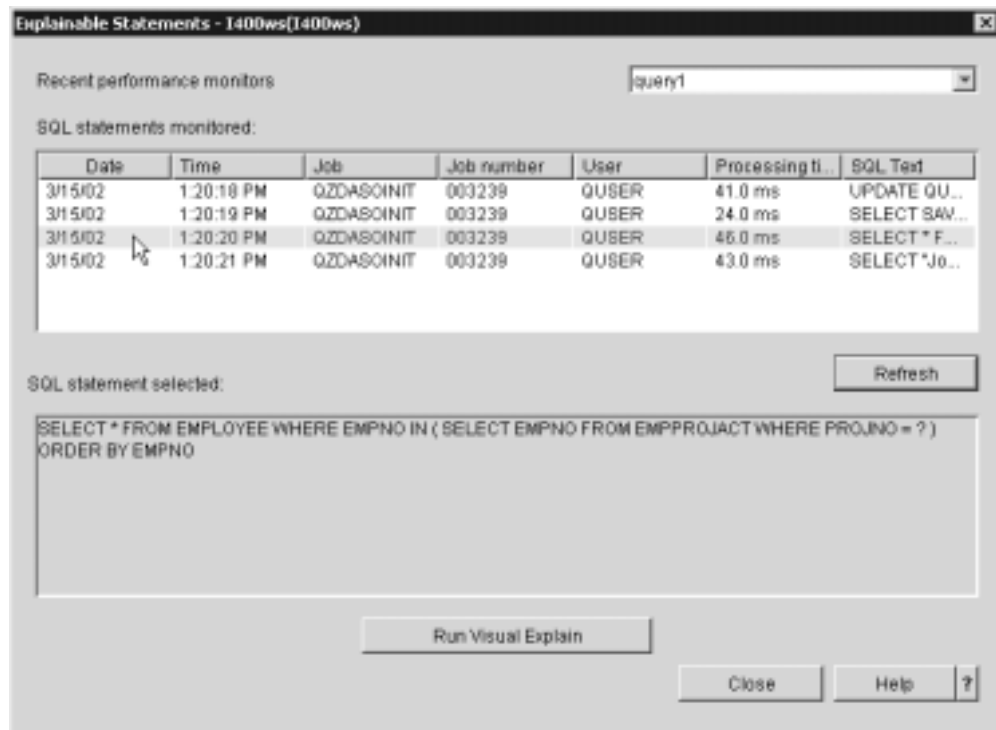


Figure 20. Visual Explain: Explainable Statements window

Once you find the desired statement, you can click the **Run Visual Explain** button to launch it. You do not need to do it.

- \_\_\_ 17. Close the Explainable Statements window and then close the SQL Script Center window.

Now you have a basic idea on what Visual Explain is about.

If you are not quite fluent in writing SQL code, there's a new V5R2 feature in SQL Script Center that helps you build some statements using a prompting method. The following tasks explains this new SQL Assist tool.

## Task 5: Using SQL Assist to build your basic SQL statement

SQL Assist is a new feature in V5R2 that helps you build some basic statements (SELECT, INSERT, UPDATE, and DELETE) by going through step by step prompts.

Let's try building the following statement that joins the STAFF and ORGanization tables and calculates each staff's earning by adding SALARY and COMMISSION. The resulting statement should look similar to this:

```
SELECT STAFF.ID, STAFF.NAME,  
  
( COALESCE(STAFF.SALARY, 0) + COALESCE(STAFF.COMM, 0) ),  
ORG.DEPTNAME, ORG.MANAGER, ORG.LOCATION  
  
FROM ORG, STAFF WHERE (STAFF.DEPT = ORG.DEPTNUMB) ORDER BY ID ASC;
```

The function COALESCE converts a *NULL* value, if found in SALARY or COMM column, to a value of zero (0) so that they can be added together.

Complete the following steps to learn how to build the statement:

1. From the menu bar of SQL Script Center, click **Edit-> Insert Built SQL**. The SQL Assist window appears (Figure 21).

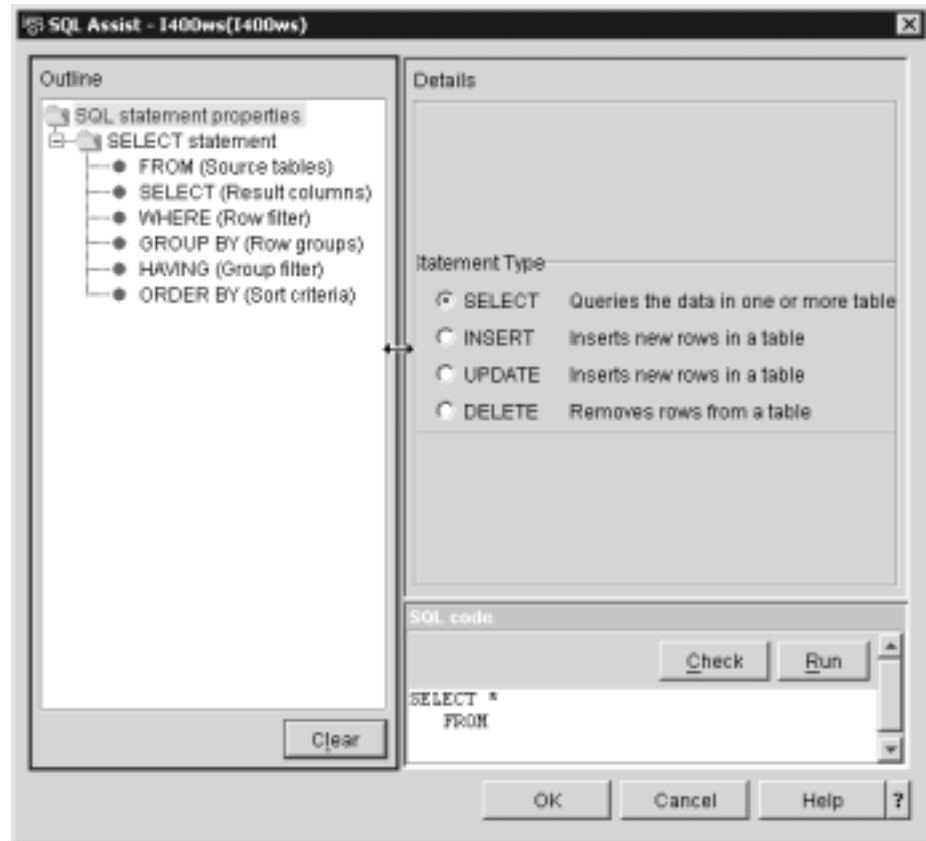


Figure 21. SQL Assist window

There are three panels within this window: Outline, Details, and SQL Code. The SQL Code panel displays the statement being built, but you can also directly edit the code in this panel if necessary.

This window and all three panels are resizable. Resize them to your preference.

- \_\_\_ 2. In the Outline panel, click **FROM (Source tables)** and then move your pointer to the Details panel and expand your **SAMPLEDBXX** schema.
- \_\_\_ 3. Scroll down to the **ORG** table, click it, and then click the Add (>) button to select the table. Also select the **STAFF** table in the same way (Figure 22).

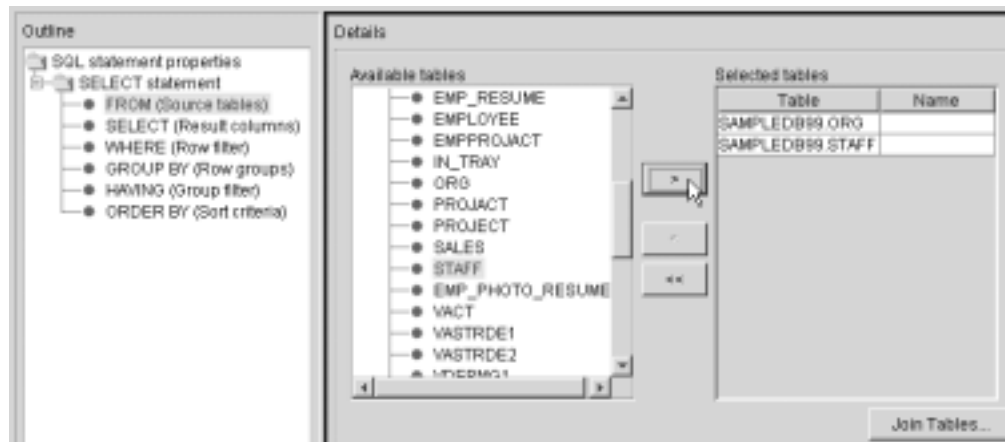


Figure 22. SQL Assist: Selecting tables or views

You can now see that the statement in the SQL Code panel is updated accordingly.

- \_\_\_ 4. In the Outline panel, click **SELECT (Result columns)**. Then move your pointer to the Details panel and expand the **SAMPLEDBXX.STAFF** and **SAMPLEDBXX.ORG** tables to see their column names.
- \_\_\_ 5. Select, one-by-one, the columns **ID**, **NAME**, **SALARY**, **DEPTNAME**, **MANAGER**, and **LOCATION**, by clicking each column name and then the Add (>) button.

Since we want to calculate the staff's total earnings by adding their salaries and commissions, we make changes to the result column SALARY.

- \_\_\_ 6. Move your pointer into the Result Columns list box and click the (...) button on the *right* of SALARY column. This launches the Expression Builder window for that particular result column (Figure 23).

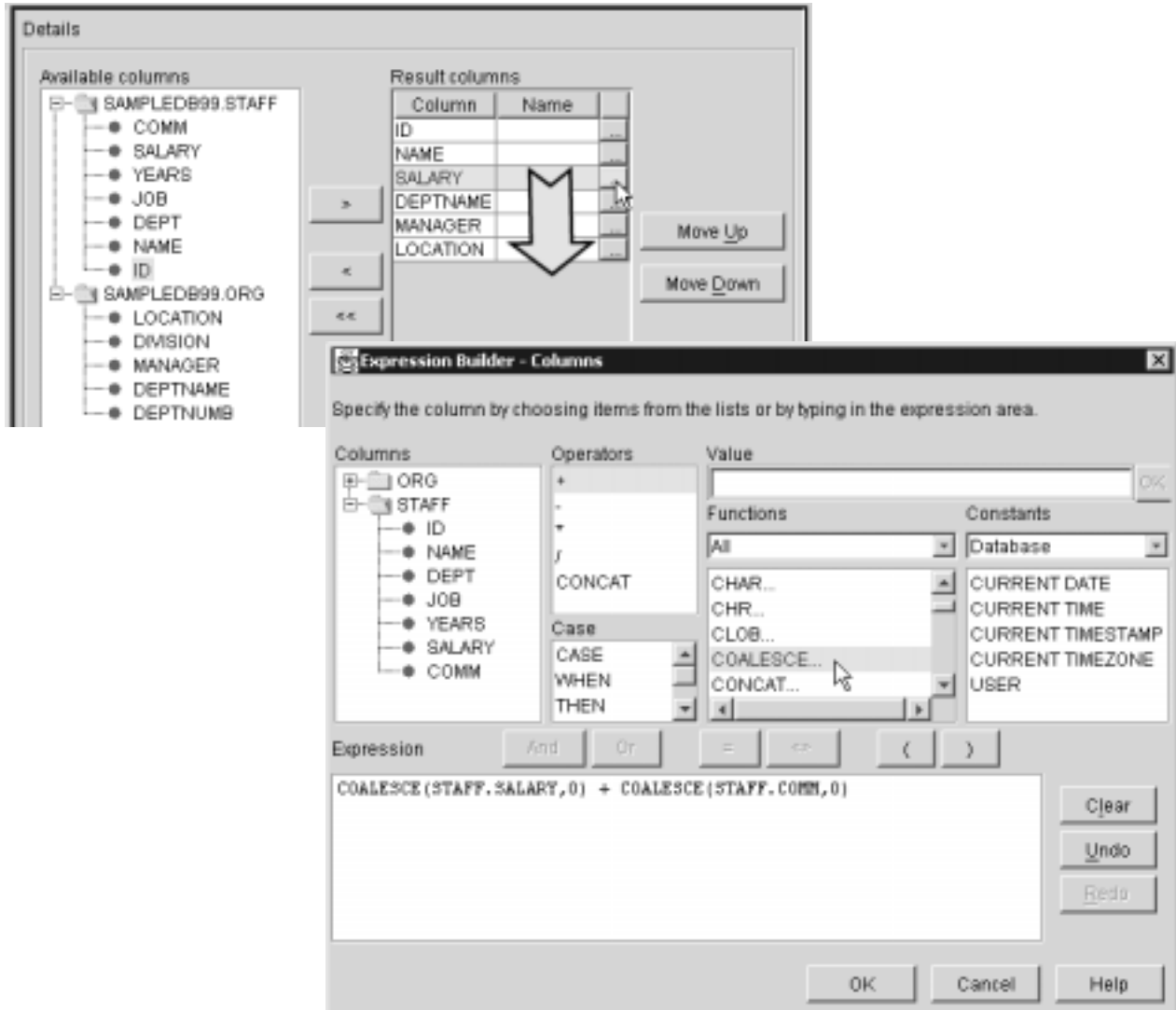


Figure 23. SQL Assist: Selecting columns and creating a result column with Expression Builder

- \_\_\_ 7. Click the **Clear** button (right of the Expression text box) to delete the existing text in the box. Then move your pointer to the Functions box, scroll down to locate the **COALESCE** function, and double-click it. A new Function Parameters window opens for COALESCE (Figure 24).

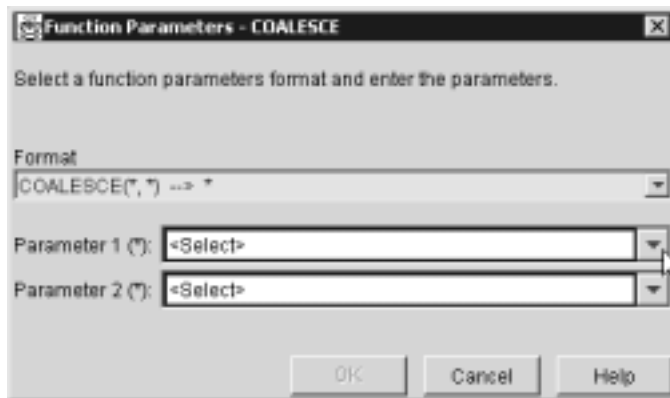


Figure 24. SQL Assist: Function Parameter window

- \_\_\_ 8. Click the down arrow button on the right of Parameter1 (\*) to expand it. Then locate and click **STAFF.SALARY** to select it as the first parameter.
- \_\_\_ 9. Type 0 for Parameter2 (\*), and then click **OK**.  
You now see that COALESCE(STAFF.SALARY,0) is added to the Expression text box. Make sure you put the cursor at the end of the added expression.
- \_\_\_ 10. Double-click the plus sign (+) in the Operators list box to add it to the expression.
- \_\_\_ 11. Double-click **COALESCE** in the Functions list box again. Select **STAFF.COMM** as its first parameter and type 0 as its second parameter. Click **OK** when finished.  
You can see now that COALESCE(STAFF.SALARY,0) + COALESCE(STAFF.COMM,0) is built in the Expression text box of the Expression Builder window.
- \_\_\_ 12. In the Expression Builder window, click **OK**. You are now back to the SQL Assist window.
- \_\_\_ 13. In the Outline panel, click **WHERE (Row filter)** and then move your pointer to the Predicate box and expand the Column field. Then select the **DEPT** column under SAMPLEDBXX.STAFF.
- \_\_\_ 14. Select '=' from the Operator field.
- \_\_\_ 15. Expand the Value field and select **Expression**. The Expression Builder window appears.
- \_\_\_ 16. In the Columns box, expand the **ORG** table and double-click the **DEPTNUMB** field. The text string ORG.DEPTNUMB now appears in the Expression text box.  
Click **OK**. Then you return to SQL Assist window with ORG.DEPTNUMB appearing in the Value field.
- \_\_\_ 17. Click the Add (>) button to add this search condition. The SQL code panel is now updated with the WHERE condition. (You may need to scroll down the SQL Code panel to see this).
- \_\_\_ 18. In the Outline panel, click **ORDER BY (Sort criteria)**. Then move your pointer to the Available Columns box to expand your **SAMPLEDBXX.STAFF** table.

\_\_\_ 19. Click the **ID** column and then click the Add (>) button.

You can now see the complete statement in the SQL Code panel.

\_\_\_ 20. In the SQL Code panel, click **Run** to execute the statement. You should see the new Query Result window. Review your results. Click **OK** to close it and go back to SQL Assist window.

If your SQL statement does not run in this step, you should review the steps that you have completed so far for any possible errors.

\_\_\_ 21. In the SQL Assist window, click **OK** to finish building the statement.

You return to SQL Script Center with the built statement inserted in its working area.

\_\_\_ 22. Double-click anywhere in the statement to run it. You should see the same result as the one you tested two steps back.

Clear the result when finished (click **Edit-> Clear results**).

You have now learned how to use SQL Assist.

---

## Task 6: Running OS/400 CL commands from SQL Script Center (optional)

Using the SQL Script Center, you can also run OS/400 CL commands. The following steps show an example on how to display all the column information of the STAFF table of the sample database by using the CL command Display File Field Description (DSPFFD).

\_\_\_ 1. In the SQL Script Center window, click the **New** icon on the toolbar (or select **File->New** from the menu bar). If you are prompted to save the existing script, select **No**.

\_\_\_ 2. Insert the following statements:

```
CL: DSPFFD FILE(STAFF) OUTPUT(*OUTFILE) OUTFILE(QTEMP/DSPFFD);  
SELECT * FROM QTEMP.DSPFFD;
```

\_\_\_ 3. From the toolbar, click the **Run All** icon (Figure 12). The column information for the STAFF table should appear in a new tab or window.

\_\_\_ 4. From the menu bar, click **Edit-> Clear Results** when finished.

### Note

As of Client Access V5R1, the Navigator provides a new support for running a CL command with prompts for its parameters. You can use this feature to create a full CL command line so that you can copy it to the Run SQL Scripts window to ensure a correct syntax.

You can use this feature by right-clicking the server name icon and selecting **Run Command** as shown in Figure 25.

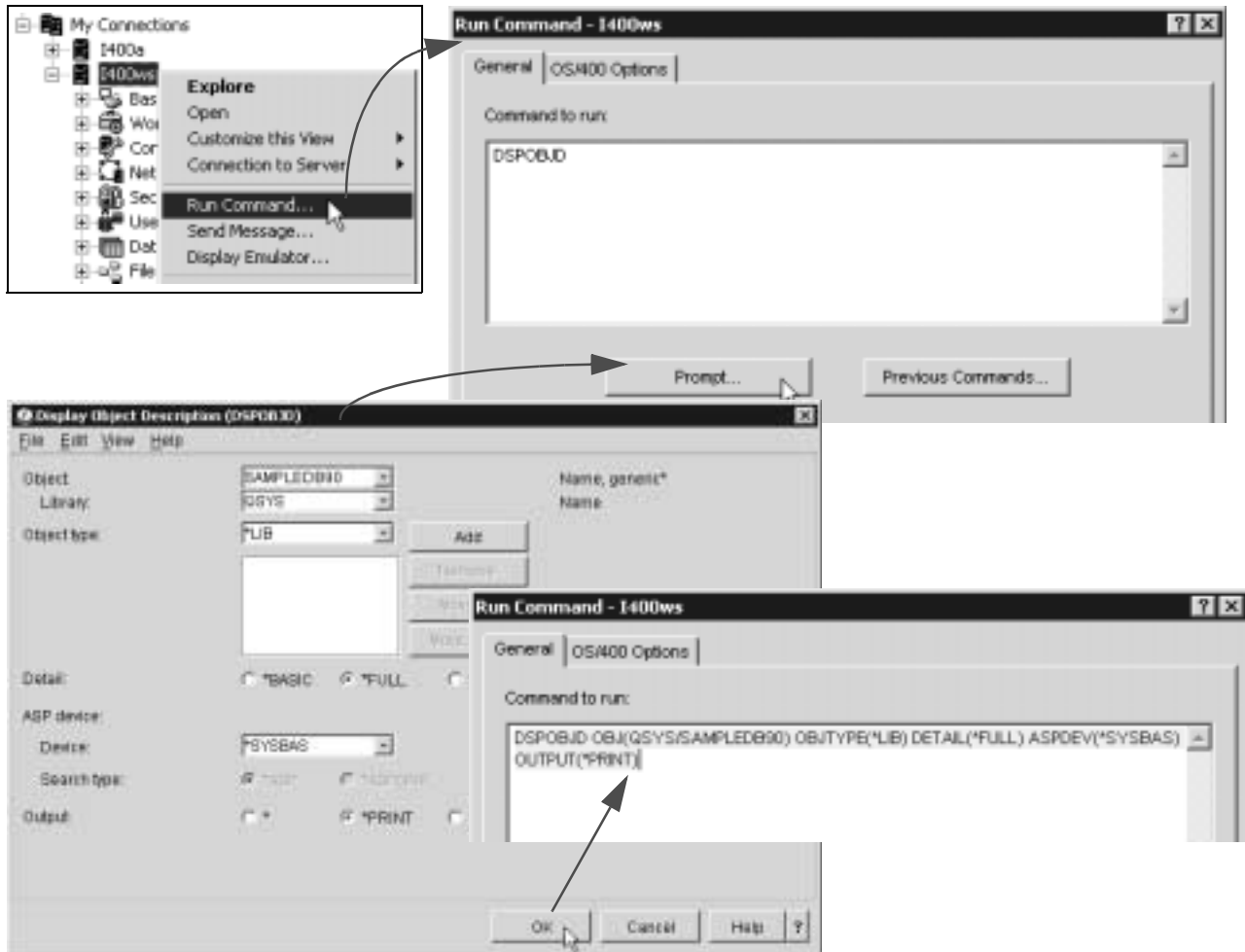


Figure 25. CL command prompting as of Client Access V5R1

You have now completed this lab!



---

## Lab 3. Working with tables using iSeries Navigator

This lab explains how to work with tables. It also explains how to perform such tasks as creating a table, generating an SQL statement from an existing table, and creating constraints for the table.

The notation *XX* that appears in library names, profile names and so on refers to your team number.

### Objectives

This lab teaches you how to:

- Create a database table and insert rows of data
- Create an identity column for automatic value maintenance
- Create a primary key and check constraints for the database table
- Generate an SQL DDL statement from the existing database table
- Display a database table description
- Use the hot link feature in quick view window

### Lab prerequisites

Before you begin this lab, be sure you meet the following prerequisites:

- You must have completed Lab 1, “iSeries Navigator setup and basic operations” on page 3.
- You must have completed Task 1 in Lab 2, “SQL Script Center” on page 11, because the *SAMPLEDBXX* schema and SQL Script Center are used in this lab.

### Time required

The time required to efficiently complete this lab is 20 minutes.

### Introduction

An *SQL table* is the equivalent of a DDS defined physical file. Similarly, *table rows* equate to physical file records for DB2 UDB for iSeries, and *SQL columns* are a synonym for record fields.

An SQL index provides a keyed access path for physical data exactly the same way as a keyed logical file.

An SQL view is similar to a logical file on a physical file. It provides a different view of the data to allow columns, subsetting, record selection, and joining of multiple database files.

### Naming convention for a ‘schema’

In V5R1, the term “schema” is used in the same sense as the term “collection”. This is an OS/400 library created with automatic DB journaling enabled and local DB catalog views.

---

## Task 1: Creating a database table with a primary key constraint

This task explains how to create a table named CUSTOMER with iSeries Navigator.

### Before your begin

Do *not* click the OK button on the Create Table window before you finish entering *all* the required column definitions in a table. By clicking OK, you actually submit a CREATE TABLE statement to the server with the columns currently defined in the dialog. All subsequent column-related actions are then submitted as ALTER TABLE statements rather than CREATE TABLE. This may cause unexpected results and generate SQL errors.

Let's start:

- \_\_\_ 1. Start iSeries Navigator and expand **I400WS-> Database-> Libraries**.
- \_\_\_ 2. Right-click the **LIBXX** library icon and click **New-> Table**.
- \_\_\_ 3. In the Table input field, type `Customer`. In the Description input field, type `Customer Master`. Click **OK**. The New Table - Customer window appears.
- \_\_\_ 4. To add a new column, click the **Insert** button.
- \_\_\_ 5. Type `Customer_number` under Column Name.
- \_\_\_ 6. Select **BIGINT** from the drop-down list of the column Type.
- \_\_\_ 7. In the Column tab, change the Short column name input field to **CUSNUM**.

### A note on short column name

If you do not specify a short column name, OS/400 generates a default name for you.

If your column name is longer than 10 characters, a 10-character short column name is automatically generated for you. It uses the first five characters of the original column name followed by a five-digit unique number (starting from 00001).

If the column name is *not* longer than 10 characters, the short column name is the same as the column name.

- \_\_\_ 8. Select the **Set as identity column** check box. Additional parameters appear as: Step value, Starting value, Minimum value, and Maximum value.
- \_\_\_ 9. Specify 999999 as the Maximum value. Leave the rest at their default values.
- \_\_\_ 10. Select the **Cycle values when the limit is reached** check box.

### Identity Column support

New in V5R2, the identity column feature lets you declare that a column (with a data type of INTEGER, SMALLINT, BIGINT, NUMERIC or DECIMAL) be maintained by the DB2 database engine with automatic increment (or can be decreased) applied to its value every time a new row is inserted. This feature allows you to specify the starting value, minimum and maximum values, incremental step value, and whether the value can be restarted when the specified maximum is reached. You can also specify whether the identity column can receive its value from an external source.

The ALTER TABLE statement is enhanced in V5R2 to support identity column. See V5R2 *SQL Reference* manual for more details.

- \_\_\_ 11. Click **Insert** to add another column definition. Do *not* click OK yet because it creates the table even though you are not finished adding the columns.

Add the remaining columns using the information provided in Table 1 (remember to click **Insert** between each column). The completed window is shown in Figure 26.

*Table 1. Customer table properties*

Column name	Column type	Column length	Short column name	Must contain a value	Default value
Customer_Number	BIGINT	-	CUSNUM	-	-
Customer_Name	Character	20	CUSNAM	Yes	no
Customer_Telephone	Character	15	CUSTEL	Yes	no
Customer_Address	Character	20	CUSADR	Yes	no
Customer_Cred_Lim	Decimal	11, 2	CUSCRD	No	1000
Customer_Tot_Amt	Decimal	11, 2	CUSTOT	No	0 (zero)



Figure 26. Creating the Customer Master table

- \_\_\_ 12. When you finish adding all the columns, click the **Key Constraints** tab. Click **New**. A New Key Constraint window appears.
- \_\_\_ 13. At the Constraint input field, type the key constraint name as CUSTELKEY. Move down to click the **Customer\_Telephone** column name. The number 1 appears in front, which means that this is the first field of the key constraint you want to create. In the Constraint type field, select the **Primary** radio button. Click **OK** to return to the New Table - Customer window.
- \_\_\_ 14. Click **OK** to create the table. The CREATE TABLE statement is submitted to the server with the specified primary key constraint.
- \_\_\_ 15. In the left panel, click **LIBXX**. In the right panel, to right-click the **CUSTOMER** table and select **Properties**. From the Table Properties window, you can view all the column definitions and the primary key constraint. Check that you entered the columns correctly as shown in Table 1.

## Task 2: Adding a Check constraint to a table

This task explains how to add check constraints to the table and test it by adding rows that violate the constraint.

- \_\_\_ 1. Click the **Check Constraints** tab, and click **New**. A new Check Constraint Search Condition window appears.
- \_\_\_ 2. In the Constraint input field, type `CREDITLIMIT`. This serves as the name of the check constraint that you are about to create.  
 In the Columns list box, double-click the **CUSTOMER\_TOT\_AMT** column. The column name should appear underneath in the Clause text box.  
 In the Operators list box, double-click the less-than-or-equal-to symbol (`<=`). The `<=` operator should appear to the right of the `CUSTOMER_TOT_AMT` in the Clause text box.  
 In the Columns list box, double-click the **CUSTOMER\_CRED\_LIM** column. The complete clause should be:  

```
CUSTOMER_TOT_AMT <= CUSTOMER_CRED_LIM
```
- \_\_\_ 3. Click **OK** to return to the Check Constraint tab. Click **OK** again to finish creating the check constraint. The `ALTER TABLE` with `ADD CONSTRAINT` statement is now submitted to the server.

The check constraints you created ensure that a customer's accumulated credit (`CUSTOMER_TOT_AMT`) does not exceed the specified credit limit (`CUSTOMER_CRED_LIM`).

---

### Task 3: Adding rows to the table and testing the check constraint

You now add a few rows to test the Check and Primary Key constraints that you created. You start by testing the check constraint.

**Note**

During the row insertion operation, if you see a message window stating that "the table you are going to change is not being journaled", simply click **Yes** to proceed.

- \_\_\_ 1. Right-click the **CUSTOMER** table and select **Open** (or double-click it). A new window appears that contains column headings but has no data rows. This is the empty `CUSTOMER` table you created.  
 Resize the window to your preference.
- \_\_\_ 2. At the menu bar, click **Rows-> Insert**. A new row appears containing empty cells. Add the first row with the data shown in Table 2 (use the left and right arrow keypad to move through the columns). Press Enter when you are finished.

Table 2. Inserting a Customer row

Customer_Number	- Do not type any value -
Customer_Name	John Doe
Customer_Telephone	777-555-2222
Customer_Address	New York, NY
Customer_Cred_Lim	99999
Customer_Tot_Amt	55555

\_\_\_ 3. Test the check constraint that you created in the previous task.

From the menu bar, click **Rows->Insert** again to add the second row using the information provided in Table 3. You may see a warning message stating that the table is not being journaled. Simply click **Yes** to proceed.

Table 3. Inserting a Customer row

Customer_Number	- Do not type any value -
Customer_Name	Dave Jones
Customer_Telephone	666-555-4444
Customer_Address	Camel, CA
Customer_Cred_Lim	44444
Customer_Tot_Amt	55555

Note that CUSTOMER\_TOT\_AMT exceeds CUSTOMER\_CRED\_LIM.

Press Enter when you are finished. You should see the SQL0545 error message “INSERT or UPDATE not allowed by CHECK constraint” shown in Figure 27.

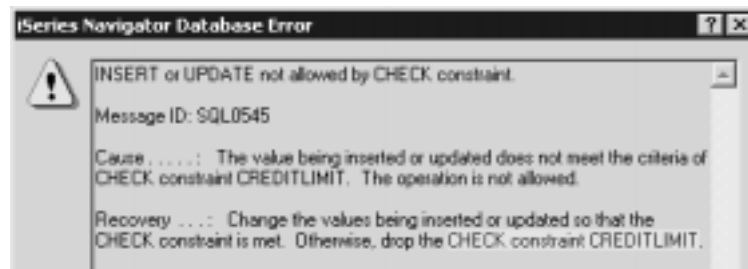


Figure 27. SQL0545: Check Constraint violation message

Note that the CREDITLIMIT constraint that you created is mentioned. If you do not see the constraint mentioned, you should review what you have done so far.

\_\_\_ 4. Click **OK**. A message should appear stating “The insert failed. Do you want to remove the row?”.

Click **No** and go back and change the Customer\_Cred\_Lim to 77777. Then press Enter. The error message should not appear this time because the check constraint is not violated.

\_\_\_ 5. From the menu bar of the table content window, click **View-> Refresh**. The message “You have made changes to LIBXX.CUSTOMER. Do you want to save the changes?” appears. Click **Yes**.

CUSTOMER_NUMBER	CUSTOMER_NAME	CUSTOMER TELEPHONE	CUSTOMER ADDRESS	CUSTOMER_CRED_LIM	CUSTOMER_TOT_AMT
1	John Doe	777-555-2222	New York, NY	99999.00	55555.00
3	Dave Jones	666-555-4444	Camel, CA	77777.00	55555.00

Figure 28. Customer table content with auto-incremented CUSTOMER\_NUMBER value

Notice that the identity column CUSTOMER\_NUMBER contains auto-incremented values as specified when you created it in Task 1.

Also notice that the CUSTOMER\_NUMBER of this second row is “3” rather than a “2”. This is because the value “2” is generated when the row with constraint violation was inserted. When you changed CUSTOMER\_CRED\_LIM to the new value that did not violate the constraint, it became the THIRD row to the database engine.

## Task 4: Testing the primary key constraint

In this task, you test the primary key constraint you created in the previous task.

- \_\_\_ 1. Insert a third row using the information provided in Table 4.

Table 4. Inserting a Customer row

Customer_Number	- Do not type any value -
Customer_Name	Brian Smith
Customer_Telephone	666-555-4444
Customer_Address	Rochester, MN
Customer_Cred_Lim	90000
Customer_Tot_Amt	55555

Note that CUSTOMER\_TELEPHONE *intentionally* contains a duplicated value with the preceding row.

Press Enter when you are finished. The SQL0803 message “Duplicated Key value specified” shown in Figure 29 appears. This indicates that your primary key constraint works. If you do not see this error message, review your work so far.

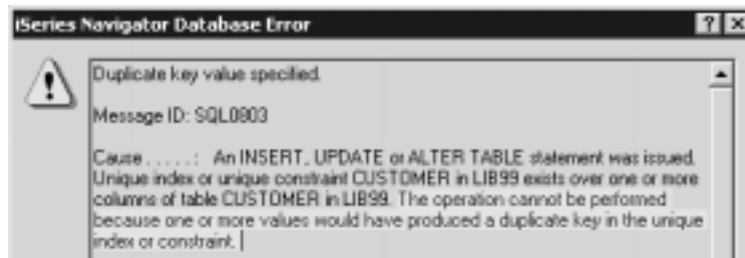


Figure 29. SQL0803: Duplicate key violation message

- \_\_\_ 2. Click **OK**. The message “The insert failed. Do you want to remove the row?” should appear.

Click **No** and return to change the Customer\_Telephone to 111-222-3333. Then press Enter. The error message should not appear this time because the key constraint is not violated.

- \_\_\_ 3. From the menu bar of the table content window, click **View-> Refresh**. The message “You have made changes to LIBXX.CUSTOMER. Do you want to save the changes?” appears.

Click **Yes**. You should see the new row in the table with a CUSTOMER\_NUMBER of “5”. This is because of the same reason as explained in the previous task.

- \_\_\_ 4. From the menu bar, click **File-> Save** and then close the window.

## Task 5: Generating SQL from existing database objects

In this task, you learn about an V5R1 enhancement that generates SQL data definition language (DDL) statements from the existing database objects in your system. There are separate workshop materials and a redpaper that describe this new function in detail. The redpaper is *DB2 UDB Database Navigator and Reverse Engineering in V5R1 on the IBM @server iSeries Server*, REDP0515.

- \_\_\_ 1. Right-click your **CUSTOMER** table icon and select **Generate SQL**. A new Generate SQL window appears with the CUSTOMER table displayed in the list box.
- \_\_\_ 2. Click the **Output** tab, and select the **Open in Run SQL Scripts** option. This ensures that the SQL Script Center is invoked to store the generated statements.
- \_\_\_ 3. Click the **Options** tab and make sure the following options are selected:
  - DB2 UDB Family
  - Extensions
  - Generate labels
  - Format statements for readability
  - Include informational messages
- \_\_\_ 4. Click the **Format** tab and choose a naming convention of your choice. Click **Generate**. The SQL Script Center appears with the generated SQL DDL statements posted in its working area as shown in Figure 30.

```
-- Generate SQL
-- Version:          V5R1M0 010525
-- Generated on:     02/05/01 13:23:04
-- Relational Database: RCHASM27
-- Standards Option: DB2 UDB AS/400

CREATE TABLE LIB01.CUSTOMER (
    CUSTOMER_NUMBER FOR COLUMN CUSNUM  CHAR(5) CCSID 37 NOT NULL ,
    CUSTOMER_NAME FOR COLUMN CUSNAM  CHAR(20) CCSID 37 NOT NULL ,
    CUSTOMER_TELEPHONE FOR COLUMN CUSTEL  CHAR(15) CCSID 37 NOT NULL ,
    CUSTOMER_ADDRESS FOR COLUMN CUSADR  CHAR(20) CCSID 37 NOT NULL ,
    CUSTOMER_CRED_LIM FOR COLUMN CUSCRD  DECIMAL(11, 2) DEFAULT 1000 ,
    CUSTOMER_TOT_AMT FOR COLUMN CUSTOT  DECIMAL(11, 2) DEFAULT 0 ,
    CONSTRAINT LIB01.CUSTNUMKEY PRIMARY KEY( CUSTOMER_NUMBER ) ;

ALTER TABLE LIB01.CUSTOMER
    ADD CONSTRAINT LIB01.CREDITLIMIT
    CHECK( CUSTOMER_TOT_AMT <= CUSTOMER_CRED_LIM ) ;

LABEL ON TABLE LIB01.CUSTOMER
    IS 'Customer Master Table' ;
```

Figure 30. SQL scripts generated from the CUSTOMER database table

Three statements appear:

- **CREATE TABLE**: This corresponds to your tasks for creating the CUSTOMER table and its primary key constraint.
  - **ALTER TABLE**: This corresponds to your tasks for creating check constraints.
  - **LABEL ON TABLE**: This adds a label to the CUSTOMER table.
- \_\_\_ 5. Close the SQL Script Center without saving the statements.



### Two choices for creating tables

You now have experienced two aspects for creating a database object in V5R1:

- Creating a database object using a GUI provided by iSeries Navigator. This method generates SQL DDL statements for you under the covers so that you do not need to remember all the SQL syntax involved in the process. The statements are passed to the server for execution but are not stored for future use.
- Once the object is created, if you need to repeat this process in a different system or just want to keep a documented reference of the database, the Generate SQL function helps you extract all SQL DDL statements, store them, and run them on a different system.

Now let's move on to the sample schema SAMPLEDBxx that you created in Lab 2 Task 1 and explore it with iSeries Navigator.

---

## Task 6: Altering a table

This task explains how to alter an existing table using iSeries Navigator. You modify a column of the SALES table in your SAMPLEDBXX library. This modified SALES table is used in subsequent lab exercises on referential constraints and triggers.

### A note on the table properties GUI

With the GUI provided by iSeries Navigator, you can drop or add columns in a table. You cannot change the data type of a column definition (use ALTER TABLE statement to do this instead). But you can alter the NOT NULL and DEFAULT VALUE attributes of a column from the GUI.

1. In the left panel of the main iSeries Navigator window, click **I400WS-> Database-> Libraries**.

You should see your SAMPLEDBXX library in the active library list under the Libraries icon. If not, add it by right-clicking **Libraries** and selecting **Select Libraries to Display**. Refer to Task 4, "Maintaining your active library list for iSeries Navigator" on page 7, in Lab 1, for instructions on how to do this.

2. Click your **SAMPLEDBXX** library. All the objects in this library appear in the right panel.
3. Locate and right-click the **SALES** table. Select **Properties**. The Table Properties window appears.

A column named SALES\_PERSON appears with a VARCHAR data type. This column contains the names of the sales persons who make the sales transaction. We change the data type to SMALLINT and insert new data in this column so that it represents staff ID numbers instead. The staff ID column exists in the STAFF table that you work with later.

4. Click the **SALES\_PERSON** column and click the **Delete** button. The column immediately disappears.

- \_\_\_ 5. Click **New** to add a column. A `New_Column` entry appears and is immediately highlighted. Change the entry to `SALES_STAFF` and press the Tab key to move right. Select **SMALLINT** data type from the drop-down list box.

**Note**

You cannot specify the same name of `SALES_PERSON` to the newly-added column. If you do so, the operation fails when you click OK. This is a current limitation of the GUI.

- \_\_\_ 6. Click the **Short** column name input field and type `SALESSTAFF`. Select the **(not null)** check box and specify a Default value of 99.  
Click **OK** to make changes to the table and the window disappears.
- \_\_\_ 7. Right-click the **SALES** table and select **Properties**. Click the **SALES\_STAFF** column to verify your work. Click **OK** to close.
- \_\_\_ 8. Right-click the **SALES** table and select **Quick View**. Make sure that the `SALES_STAFF` column appears with a value of 99 in all the rows.

You work with this `SALES` table again in Lab 6, "Database referential constraint" on page 61.

You now know how to alter a table with iSeries Navigator. Remember that you can also use the SQL Script Center (from *Lab 2, "SQL Script Center" on page 11*) to execute SQL statements for a more complex table alteration.

You are ready to complete the next task.

---

## Task 7: Displaying properties and descriptions of DB objects

This task explains how to use other database functions in iSeries Navigator to access various kinds of useful information for database administration tasks. The Database Description feature used in this task was added in V4R5.

- \_\_\_ 1. In the left panel of the main iSeries Navigator window, click **I400WS-> Database-> Libraries**.  
You should see your `SAMPLEDBXX` library in the active library list under the Libraries icon. If not, add it by right-clicking **Libraries** and select **Select Libraries to Display**. Refer to Task 4, "Maintaining your active library list for iSeries Navigator" on page 7, in Lab 1, for instructions on how to do this.
- \_\_\_ 2. Right-click your **SAMPLEDBXX** library and select **Properties**. A new `SAMPLEDBXX` Properties window appears. There are four information tabs in this window as shown in Figure 31.

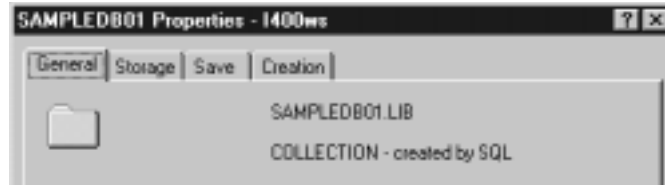


Figure 31. Information tabs in your library properties window

\_\_\_ 3. Explore the information in this window and answer the following questions:

- How large is this library? \_\_\_\_\_ megabytes

**Hint:** General tab->Total allocated size

- How many objects are there in this library? \_\_\_\_\_ objects

**Hint:** Storage tab->Contents

- When was this library last-saved? \_\_\_\_\_

**Hint:** Save tab->Last saved

- When was this library created? By whom? \_\_\_\_\_

**Hint:** Creation tab->Created

Close the window when you are finished.

\_\_\_ 4. Click your SAMPLEDBXX library to see all of the objects in this library displayed in the right panel.

\_\_\_ 5. Locate and right-click the **EMPLOYEE** table. Select **Description**. A new Description window appears with six different information tabs as shown in Figure 32.

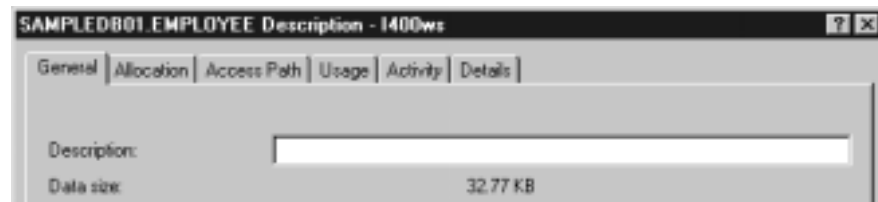


Figure 32. Information tabs in your table description window

This option is used in the same manner as the CL commands Display File Description (DSPFD), Change Physical File (CHGPF), and Change Logical File (CHGLF).

You can click **Help** here to see detailed information about your table description attributes in the corresponding tab that you are in.

\_\_\_ 6. Explore the information in this window and answer the following questions:

- How large is this table? \_\_\_\_\_ kilobytes

**Hint:** Select General tab->Data size

- How many rows are in this table? \_\_\_\_\_ rows

- Is Reuse deleted rows feature active? \_\_\_\_\_

- How large is the access path of this table? \_\_\_\_\_ bytes

- How many *distinct valid indexes* does this table have? \_\_\_\_\_

**Hint:** Click the Activity tab

- How many bytes is the longest row in this table? \_\_\_\_\_ bytes

**Hint:** Select Details tab->Maximum row length

Close the window when you are finished.

- \_\_\_ 7. In the previous step, you discovered that there are two indexes built for the EMPLOYEE table. How do you identify the names of its indexes?

**Hint:** right-click the **EMPLOYEE** table and select **Properties**. Find the proper information tab that shows you the required information.

If you think the **Indexes** tab is correct, click it. If you are correct, two index names appear: *XEMP1* and *XEMP2*. Click **OK** to close the window.

#### Displaying database object relationship

In V5R1, a new function named *Database Navigator* was added to iSeries Navigator. It generates a graphical representation of many database objects in which you are interested. For example, you can use it to display indexes of a specific table instead of using the Properties window.

If you are interested in learning more about Database Navigator, consult the “DB2 UDB for iSeries Database Navigator and Reverse Engineering” hands-on lab.

- \_\_\_ 8. In the iSeries Navigator main window, locate and right-click the **VEMPDPT1** view and select **Description**.
- \_\_\_ 9. Click the **Details** tab and look at the Allowed activities attribute. Why is it read-only? Notice the check mark in front of Read, while there is none for Update, Write, or Delete.

**Answer:** Normally, a joined view is a read-only object. You can prove that VEMPDPT1 is a joined view by right-clicking **VEMPDPT1** and selecting **Properties**. Notice the SQL DDL statement that joins DEPARTMENT and EMPLOYEE tables.

Close the window when you are finished.

## Task 8: Using the Hot Link feature in the Quick View function (optional)

In V5R1, the Quick View function is enhanced with a Hot Link feature. This task explains how to use the Hot Link feature.

#### Before you begin

This task uses an HTTP server on your iSeries machine. If you already have one configured, start it now. If not, you can still proceed without encountering serious problems.

- \_\_\_ 1. Locate and right-click the **EMP\_PHOTO\_RESUME** view. Select **Properties**.

This view joins EMP\_PHOTO and EMP\_RESUME tables (A.EMPNO = B.EMPNO). These two tables contain columns that have a Data Link data type. Click **OK** to close the window.

**Data Link data type**

The Data Link column type was introduced in OS/400 V4R4. It stores a URL-format name that provides links to data sources that are outside of the database. Normally, these external data sources are binary-large-object types, such as images, digitized voice, and so on.

- \_\_\_ 2. Right-click the **EMP\_PHOTO\_RESUME** view again and select **Quick View**.
- \_\_\_ 3. Select the **DL\_PICTURE** and **DL\_RESUME** columns to see URL names that are automatically enabled as hot links as shown in Figure 33.



	EMPNO	EMP_ROWID	DL_PICTURE	DL_RESUME
1	000130		HTTP://AS27.ITSOBROCH.IBM.COM/QIBM/ProdData	HTTP://AS27.ITSOBROCH.IBM.COM/QIBM/ProdData
2	000140		HTTP://AS27.ITSOBROCH.IBM.COM/QIBM/ProdData	HTTP://AS27.ITSOBROCH.IBM.COM/QIBM/ProdData
3	000150		HTTP://AS27.ITSOBROCH.IBM.COM/QIBM/ProdData	HTTP://AS27.ITSOBROCH.IBM.COM/QIBM/ProdData
4	000150		HTTP://AS27.ITSOBROCH.IBM.COM/QIBM/ProdData	HTTP://AS27.ITSOBROCH.IBM.COM/QIBM/ProdData

Figure 33. Hot links in the EMP\_PHOTO\_RESUME view

If an HTTP server is active on your iSeries server, notice that, when you click any of the URL hot links, the default Web browser of your PC is invoked to display the destination file to which the URL points (which really exists on a V5R1 server).

If an HTTP server is not active in your server, the Web browser displays a "File not found" error message.

Close the browser and the quick-view windows when you are finished.

You have now completed this lab!



---

## Lab 4. Other database tasks using iSeries Navigator

This lab teaches you how to perform various database tasks, such as creating a view and displaying the current SQL for a job.

The notation *XX* that appears in library names, profile names, and so on refers to your team number.

### Objectives

This lab teaches you how to:

- Create a view that joins two base tables
- Grant permissions on libraries, tables, and views
- Display the current SQL statement in other jobs

### Lab prerequisites

Before you begin this lab, be sure you meet the following prerequisites:

- You must have completed Lab 1, “iSeries Navigator setup and basic operations” on page 3.
- You must have completed Task 1, “Creating a sample DB schema and setting up a JDBC connection” on page 11, in Lab 2, because the SAMPLEDBXX schema and SQL Script center are used in this lab.
- You must have completed Lab 3, “Working with tables using iSeries Navigator” on page 29.

### Time required

The time required to efficiently complete this lab is 20 minutes.

### Introduction

An SQL table is equivalent to a DDS-defined physical file. Similarly, table rows equate to physical file records for DB2 UDB for iSeries. SQL columns are synonymous with record fields.

An SQL index provides a keyed access path for the physical data just like a keyed logical file.

SQL views are similar to logical files on a physical file. They provide a different view of the data and allow columns, subsetting, record selection, and joining multiple database files.

### Naming convention for a ‘schema’

Starting in V5R1, the term “schema” is used in the same sense as the term “collection”. This is an OS/400 library created with automatic DB journaling enabled and local DB catalog views.

---

### Task 1: Creating a view

In your SAMPLEDBXX library, you create a view that joins the STAFF and ORG tables with a new result column that calculates the total earnings of each staff.

- \_\_\_ 1. Locate and right-click the **STAFF** table (in your SAMPLEDBXX library). Select **Properties** from its pop-up menu to see the column definitions.  
Click the **SALARY** column. Notice that this column can contain NULL as the default value. The same is true with the **COMM** column. You must deal with this fact when you create the view.  
Click **OK** to close the window
- \_\_\_ 2. Locate and right-click the **ORG** table. Select **Properties** to see the column definitions.  
Notice that the **MANAGER** column (the manager who is in charge of the department) has a **SMALLINT** data type. This column is, in fact, the staff ID like the **ID** column in the **STAFF** table.  
You also deal with this fact when you want to enhance the view.  
Click **OK** to close the window
- \_\_\_ 3. In the left panel of the iSeries Navigator window, right-click your **SAMPLEDBXX** library and click **New->View**. A New View in **SAMPLEDBXX** window appears.
- \_\_\_ 4. Name the view **VSTAFFD**. Type a Staff Details View Description, and click **OK**. A New View - **VSTAFFD** window appears.
- \_\_\_ 5. Click the **Select Tables** button. The Browse Tables window appears.
- \_\_\_ 6. Expand your **SAMPLEDBXX** library. Click the **ORG** table and then click **Add** button. Scroll down the list to locate and click the **STAFF** table. Then click **Add**.

**Note**

Another way to perform this step is to drag the selected tables from the Browse Tables window and drop them (one-by-one) into the New View - **VSTAFFD** window.

Close the Browse Tables window when you are finished. Then you return to the New View - **VSTAFFD** window.

- \_\_\_ 7. Reposition the **ORG** and **STAFF** tables in the working area so that the **STAFF** table is on the left side of the working area and the **ORG** table is on the right side (Figure 34).
- \_\_\_ 8. Specify the join condition by *dragging* the **DEPT** column from the **STAFF** table and *dropping* it precisely *on* the **DEPTNUMB** column of the **ORG** table. A Join Properties window appears.
- \_\_\_ 9. Select the first (**Inner Join**) radio button (it should already be selected by default) and then click **OK**. This specifies the join condition as:  
  
`SAMPLEDBXX.STAFF.DEPT = SAMPLEDBXX.ORG.DEPTNUMB`
- \_\_\_ 10. Drag the **ID** column from the **STAFF** table and drop it into the bottom panel. This is how you select which columns are included into the view.
- \_\_\_ 11. Click the word **ID** under the Column Name column and change it to *Staff ID*. This is how you rename the column heading when the view is displayed.



**Attention**

Do *not* press Enter key each time you finish renaming the column name of the view. Instead, simply click the *next column name* to continue.

\_\_\_ 12. Select and rename the remaining columns using the information shown in Table 5.

Table 5. Columns for view

Table	Column	Rename to:
STAFF	NAME SALARY	Staff Name Total Earnings
ORG	DEPTNAME MANAGER LOCATION	Dept Name Report To Office City

The result should resemble the windows shown in Figure 34.

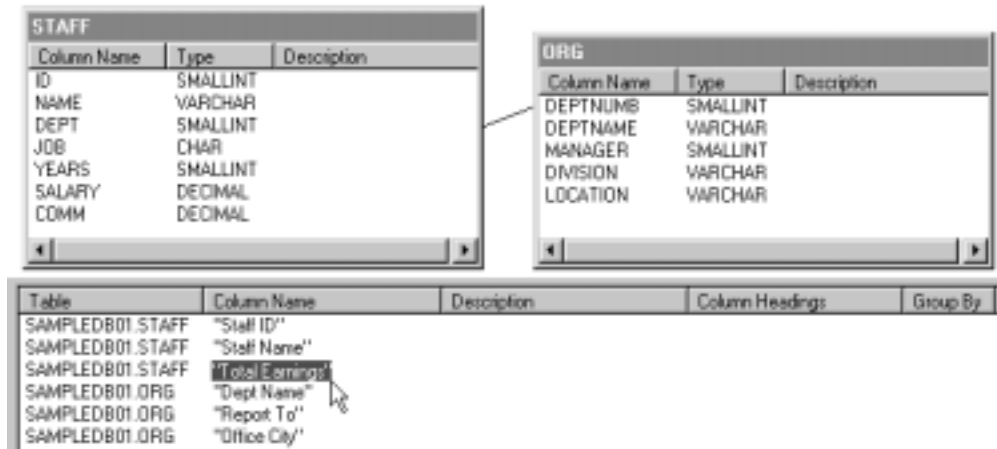


Figure 34. Creating the VSTAFFD view

**Note**

The original SALARY column is renamed to Total Earnings because you calculate a total of *salary+sales commission* instead.

\_\_\_ 13. Click the **Total Earnings** column and click the **Formula** button. A new window appears with SAMPLEDBXX.STAFF.SALARY already shown in the Clause text box.

As you noticed earlier, the SALARY and COMM columns can contain a NULL value that is *incompatible* with the arithmetic addition operation. A NULL added to a decimal value generates a NULL that does not convey a meaningful result.

Fortunately, there is a COALESCE SQL function that converts a NULL value into another specific value instead. Here, you use this function to return a 0 if NULL is detected. To do this, you follow these steps:

\_\_\_ 1. Scroll down through the Functions list box. Locate and double-click **COALESCE** to add COALESCE( ) into the Clause text box.

Move SAMPLEDBXX.STAFF.SALARY inside the parentheses (using cut (Ctrl+X) and paste (Ctrl+V)) and add a comma and a 0. The result is:

```
COALESCE(SAMPLEDBXX.STAFF.SALARY,0)
```

Now, place the text cursor at the end of this clause.

- \_\_\_ 2. Add the additional operator by double-clicking the plus symbol (+) in the Operators list box.

You now want to add the COMM (sales commission) column to the SALARY column.

- \_\_\_ 3. Double-click the **COALESCE** function and put the text cursor inside the parentheses. Double-click **SAMPLEDBXX.STAFF.COMM** in the Columns list box and add a comma and a 0. The result is:

```
COALESCE(SAMPLEDBXX.STAFF.SALARY,0) + COALESCE(SAMPLEDBXX.STAFF.COMM,0)
```

Click **OK**.

- \_\_\_ 4. Click the **Show SQL** button to view the SQL statements that are prepared for view creation (Figure 35).

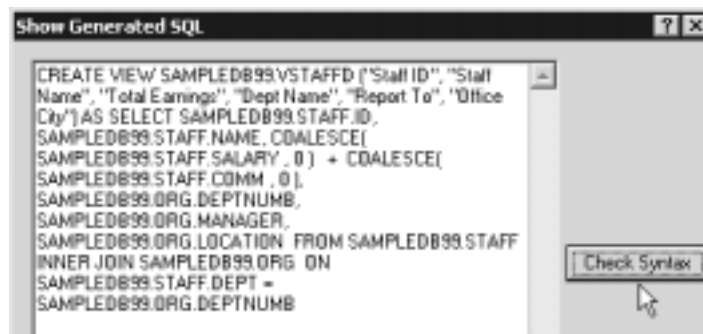


Figure 35. Show Generated SQL result

Click **OK** to close the window and return to the New View - VSTAFFD window.

**Note**

To create more complex statements, you can click the Edit SQL button that invokes SQL Script Center for you to make changes to the statements before submitting them to the server.

- \_\_\_ 5. Click **OK** to create the view. The CREATE VIEW is submitted to the server. When the view is created, you are back to iSeries Navigator window.
- \_\_\_ 6. Locate and right-click the **VSTAFFD** view in the right panel of iSeries Navigator window to invoke the pop-up menu. Select the **Properties** option and you see the SQL statement that represents this view. It is the portion after CREATE VIEW ... AS of the statement that you saw in Figure 35. Then, click **Close**.
- \_\_\_ 7. To display the view, double-click it. This opens a quick-view window that displays the content of the view.

### Notes on double-clicking

Double-clicking a table and a view object produces different results:

- Double-clicking a table object opens it for update. A record locking mechanism is active on the table. To quick-view the table, right-click it and select **Quick View**.
- Double-clicking a view object allows you to quick-view its contents.

Notice that the Report To column contains manager's staff ID rather than their names (Figure 36).

If you want to enhance this view so that it displays the manager's name instead, you need to use the SQL Script Center to create more sophisticated SQL statements. If you want, you can do this in Task 3, "Creating a view with SQL statement (optional)" on page 49.



	Staff ID	Staff Name	Total Earnings	Dept Name	Report To	Office City
1	10	Sanders	18357.50	Mid Atlantic	10	Washington
2	20	Pernal	18783.70	Mid Atlantic	10	Washington
3	30	Marenghi	17506.75	South Atlantic	30	Atlanta
4	40	O'Brien	18852.55	South Atlantic	30	Atlanta
5	50	Hanes	20659.80	New England	50	Boston
6	60	Quigley	17458.55	South Atlantic	30	Atlanta
7	70	Rothman	17654.83	New England	50	Boston
8	80	James	13632.80	Mid Atlantic	10	Washington

Figure 36. The VSTAFFD view

8. Close the quick-view window when you are finished.

### Note

You cannot alter a view once it is created. To alter a view, you must drop the view and recreate it with the appropriate changes.

## Task 2: Applying permissions to a database object

This exercise shows you how to apply permissions to a library/schema, table, and view using iSeries Navigator. To allow your alternate user DBNAVXX\_A to access the view created in the previous task, you must authorize the user to the schema (SAMPLEDBXX), the view (VSTAFFD), and the tables that are being joined in the view (STAFF and ORG).

1. Right-click your **SAMPLEDBXX** library icon and select **Permissions**. The SampledbXX.lib Permissions window appears.  
You can see from the list that the public permission is Exclude and DBNAVXX's permission is All. You now add a permission for DBNAVXX\_A.
2. Click the **Add** button. The Add Users or Groups window appears.

- \_\_\_ 3. In the User or group name field, type `DBNAVXX_A`. and click **OK**. You now see `DBNAVXX_A` in the permission list.
- \_\_\_ 4. Click the **Details** button and apply the permissions shown in Table 6 for the `DBNAVXX_A` user profile.

Table 6. Library permissions

Type	Permission
<i>Operational</i>	Yes
Management	No
Existence	No
Alter	No
<i>Reference</i>	Yes
<i>Read</i>	Yes
Add	No
Update	No
Delete	No
<i>Execute</i>	Yes

Click **OK** when finished.

- \_\_\_ 5. Repeat steps 1 through 4 for the `STAFF` and `ORG` tables and the `VSTAFFD` view.
- \_\_\_ 6. Log on to the iSeries server using the alternate connection name by clicking the plus sign (+) in front of **I400WS2**. Make sure you log on with the `DBNAVXX_A` user profile.  
  
This is not another iSeries server. Rather, it is the *same* server known in the host table by another name.
- \_\_\_ 7. Expand the **Libraries** icon and add your `SAMPLEDBXX` library into the `I400WS2` active library list.
- \_\_\_ 8. Click the newly-added **SAMPLEDBXX** schema in the left panel. Then move to the right panel and double-click the **VSTAFFD** view. A new window shows you the content of the view.  
  
Close the results window.
- \_\_\_ 9. Try deleting the view by right-clicking the **VSTAFFD** view and selecting the **Delete** option. Then click the **Delete confirmation** button.

Although you attempt to delete the view, a window appears with an error message stating that you are not authorized to `VSTAFFD` (Figure 37).

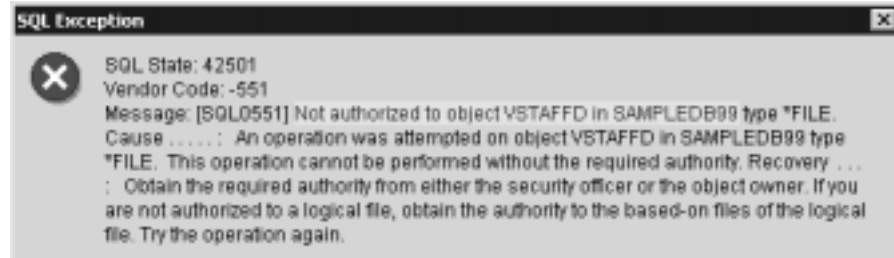


Figure 37. SQL0551 authority violation message

- \_\_\_ 10. Click **OK** to return to the main iSeries Navigator window.

**Note**

Authority to delete an object in a library must be obtained from a security officer or from the library owner.

---

### Task 3: Creating a view with SQL statement (optional)

You now create an enhanced view to VSTAFFD that displays manager names rather than their staff IDs. You must write SQL statements to achieve this goal.

- \_\_\_ 1. Make sure the **I40WS** icon under **Databases** is expanded. You must sign on with your original user ID (DBNAVXX).

Then expand **Libraries** and click **SAMPLEDBXX**.

- \_\_\_ 2. Move to the right panel. Then, locate and right-click the **VSTAFFD** view and select **Generate SQL**. (VSTAFFD should be the *bottommost* entry in the list).

Click the **Generate** button in the subsequent window. The Run SQL Scripts window appears with the CREATE VIEW statement of VSTAFFD view generated.

- \_\_\_ 3. In the SQL codes, change the original view name to VSTAFFDX :

```
CREATE VIEW SAMPLEDBXX.VSTAFFDX  
LABEL ON TABLE SAMPLEDBXX.VSTAFFDX)
```

- \_\_\_ 4. Change the statements so that it looks like the example shown in Figure 38. The changed portion is highlighted.

Make sure you type your team number in all occurrences of SAMPLEDBXX.

```

CREATE VIEW SAMPLEDB01.VSTAFFDX (
    ""Staff ID"" FOR COLUMN STAFF00001 ,
    ""Staff Name"" FOR COLUMN STAFF00002 ,
    ""Total Earnings"" FOR COLUMN TOTAL00001 ,
    ""Dept Name"" FOR COLUMN DEPT_00001 ,
    ""Report To"" FOR COLUMN REPOR00001 ,
    ""Office City"" FOR COLUMN OFFIC00001 )
AS
SELECT A.ID, A.NAME, COALESCE(A.SALARY, 0) + COALESCE(A.COMM, 0),
C.DEPTNAME, B.NAME, C.LOCATION
FROM SAMPLEDB01.STAFF A, SAMPLEDB01.STAFF B, SAMPLEDB01.ORG C
WHERE A.DEPT = C.DEPTNUMB AND B.ID = C.MANAGER ;

LABEL ON TABLE SAMPLEDB01.VSTAFFDX
IS "Enhanced Staff details View";

```

Figure 38. SQL for VSTAFFDX view

- \_\_\_ 5. From the menu bar, click **Run-> All** to create the new VSTAFFDX view. Close the SQL Script Center. You may want to save the statements for future use.
- \_\_\_ 6. To display the view, go to the right panel of the iSeries Navigator window and press F5 key to refresh the content in the panel.
- \_\_\_ 7. Locate and double-click the **VSTAFFDX** view to open a quick-view window for the result. It should be the last entry in the right panel.

Notice now that the Report To column contains manager names rather than their IDs (Figure 39).

	"Staff ID"	"Staff Name"	"Total Earnings"	"Dept Name"	"Report To"	"Office City"
1	160	Molinare	22959.20	Head Office	Molinare	New York
2	210	Lu	20010.00	Head Office	Molinare	New York
3	240	Daniels	19260.25	Head Office	Molinare	New York
4	260	Jones	21234.00	Head Office	Molinare	New York
5	50	Hanes	20659.80	New England	Hanes	Boston
6	70	Rothman	17654.83	New England	Hanes	Boston
7	110	Ngan	12714.80	New England	Hanes	Boston
8	170	Kermisch	12368.60	New England	Hanes	Boston
9	10	George	18357.50	Mid Atlantic	George	Washington
10	20	Jones	18783.70	Mid Atlantic	George	Washington
11	80	James	13632.80	Mid Atlantic	George	Washington
12	190	Sneider	14379.25	Mid Atlantic	George	Washington

Figure 39. The VSTAFFDX view

If you want to proceed on to the next task, *minimize* the quick-view window when you are finished. Otherwise, close the window.

#### Task 4: Displaying the latest SQL statement in other jobs (optional)

If your work requires supporting other SQL users, the function you learn in this task helps you acquire the SQL statements of other user jobs when you need to analyze them. This function has been available since V4R5.

**Note**

Your profile needs special *Job Control* authority to execute this task.

- \_\_\_ 1. In the iSeries Navigator main window, expand the **I400WS2** alternate server icon and log on to it using the DBNAVXX\_A profile.
- \_\_\_ 2. Right-click the **Databases** icon and select **Current SQL for a job....** The Current SQL - I400WS2 window that displays all current active jobs in your server appears as shown in Figure 40.

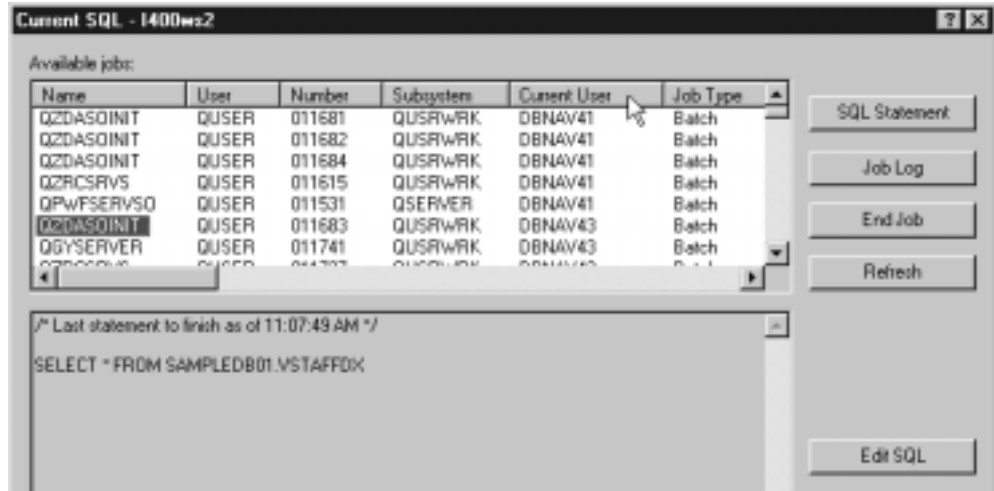


Figure 40. The Current SQL window

- \_\_\_ 3. Click the **Current User** column heading to sort the jobs by user IDs who are being served by the server jobs. You may need to slightly narrow the width of the first four columns. You can click the column heading twice for descending sort order.
- \_\_\_ 4. Browse through the job list to locate a job named QZDASOINIT. This serves your DBNAVXX profile as its Current User. Click the name **QZDASOINIT** to highlight it (Figure 40).
- \_\_\_ 5. Click the **SQL Statement** button to see the most current SQL statements that you executed in the previous task. They appear in the lower panel:

```
/* Last statement to finish as of hh:mm:ss */  
SELECT * FROM SAMPLEDB01.VSTAFFDX
```

This is an example of the statement issued by iSeries Navigator to your QZDASOINIT server job when you quick-view the VSTAFFDX view in the last step of Task 3.

- \_\_\_ 6. Click the **Edit SQL** button to see the SQL Script Center invoked with the statements entered in its working area.

You now learn what you can do if you want to edit, save, or analyze SQL statements of other jobs from the Script Center.

Close the SQL Script Center when you are finished. You do not have to save anything.

- \_\_\_ 7. You may experiment with other jobs, but do not spend too much time on this step.

\_\_\_ 8. Close all windows, except the iSeries Navigator main window, when you are finished.

This is not necessarily an exhaustive lab exercise, but you have learned many interesting database-related functions here. You also have a better understanding of how iSeries Navigator can help you do your job better.

You have now finished Lab 4, “Other database tasks using iSeries Navigator”!



---

## Lab 5. Journal management (optional)

This lab explains how to manage database journals and journal receivers.

### Objectives

This lab teaches you how to:

- View journals
- Swap receivers
- Create journals and journal receivers
- Drop journals and journal receivers

### Lab prerequisites

Before you begin this lab, be sure you meet the following prerequisites:

- You must have completed Lab 1, “iSeries Navigator setup and basic operations” on page 3.
- You must have completed Task 1 of Lab 2, “SQL Script Center” on page 11, because the SAMPLEDBXX schema is used in this lab.

### Time required

The time required to efficiently complete this lab project is 10 minutes.

### Naming convention for a ‘schema’

Starting in V5R1, the term “schema” is used in the same sense as the term “collection”. This is an OS/400 library created with automatic DB journaling enabled and local DB catalog views.

---

## Task 1: Viewing journals

Since database journaling is already activated for the SAMPLEDBXX sample schema, you initially learn how to view the entries in the journal using the SQL Script Center:

- \_\_\_ 1. Start iSeries Navigator and expand **I400WS-> Database-> Libraries**.  
You should see SAMPLEDBXX in the active library list under the Libraries icon. If you do not, add it by right-clicking **Libraries** and selecting **Select Libraries to Display**. Refer to Task 4, “Maintaining your active library list for iSeries Navigator” on page 7, of Lab 1, for instructions on how to do this.
- \_\_\_ 2. Click the **SAMPLEDBXX** library to display its current content in the right panel of the window.
- \_\_\_ 3. In the right panel, locate and right-click the **QSQJRN** journal. Select **Properties**. The Journal Properties window appears (Figure 41).



Figure 41. The Journal Properties window

Notice that the Activate journal check box is selected. This means that journaling is active. Leave it active.

**Note:** You can stop journaling by deselecting this check box.

#### Managing journal receivers

If you need journaling only to ensure database transaction integrity, you can save disk space by allowing OS/400 to automatically manage the journal receivers.

To do this, refer to the “Receivers managed by” section in Figure 41. Select **System** and **Delete receivers when no longer needed**.

- **System:** This means, when a receiver reaches its size threshold, OS/400 detaches it from the journal and automatically attaches a new receiver (swapping receivers).
- **Delete receivers when no longer needed:** This means the detached receiver is automatically deleted.

If you need the receivers for other purposes, such as auditing or data recovery, you should select **User** or **System**. Do *not* select *Delete receivers when no longer needed* to maintain the receivers in your system.

See the help message in the Journal Properties window for more details.

- \_\_\_ 4. To display the tables that are being journaled, click the **Tables** button. The Start/End Journaling window appears. The current journaled tables appear in the list box under the Tables already journaled.

**Note**

You can add or remove journaled tables using this window.

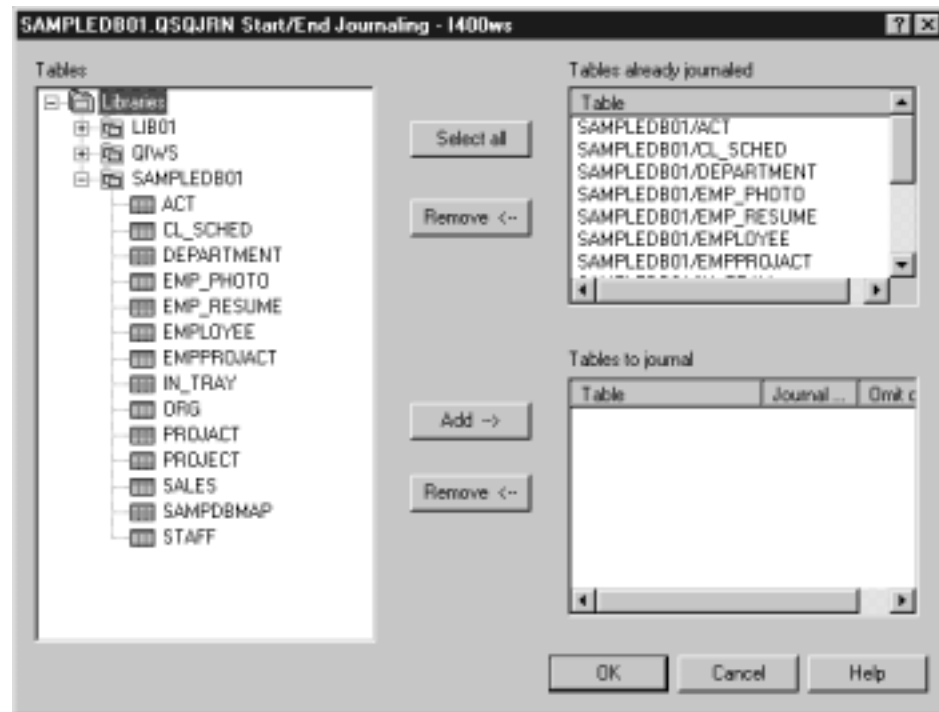


Figure 42. The Start/End Journal window

- \_\_\_ 5. Click **Cancel** to close the current window. Click **Cancel** again to go back to the iSeries Navigator main window.
- \_\_\_ 6. Start the SQL Script Center by right-clicking the **Database** icon and selecting **Run SQL Scripts**. Remove any existing SQL statements from the working area.
- \_\_\_ 7. Scroll to the bottom-most part of the SQL Statement Examples drop-down list. The `/* Display Journal Entries */` section appears.
- Click **Insert** to insert both statements of this section into the work area.
- \_\_\_ 8. Change library1 to `SAMPLEDBXX` as shown in the following code sample:
- ```
CL: DSPJRN JRN(SAMPLEDBXX/QSQJRN) OUTPUT(*OUTFILE) OUTFILEMT(*TYPE4)
OUTFILE(QTEMP/DSPJRN);
SELECT * FROM QTEMP.DSPJRN;
```
- \_\_\_ 9. From the menu bar, click **Run-> All**. The result window appears with all the entries in the journal.
- Close the result and the Run SQL Scripts windows when finished.

**Note**

You must have appropriate permission to view the journal.

---

## Task 2: Swapping journal receivers

This exercise explains the two methods for swapping receivers for journaling:

- \_\_\_ 1. Right-click the **QSQJRN** journal icon and select **Swap receivers**. The right panel is refreshed and you see a new QSQJRN000X receiver added to the panel. With this method, the system generates a new name when it creates the receiver.

**Note**

The name of the new receiver is similar to the old one with a different running number in the last four digits. Using this method, you cannot change the attributes of the new receiver.

- \_\_\_ 2. To attempt a different method for creating a new receiver and swapping it with the current one, double-click the **QSQJRN** journal icon.
- \_\_\_ 3. Click the **Receivers** button. A new window appears displaying all of the receivers that are associated with the journal.
- \_\_\_ 4. To add another new receiver, click **New**. A New Journal Receiver window appears. All of the fields are filled with the default values for the new receiver. You may make any appropriate changes.
- \_\_\_ 5. Change the storage space threshold to 20 MB and click **OK**.  
The new receiver goes into a pending state (Figure 43).

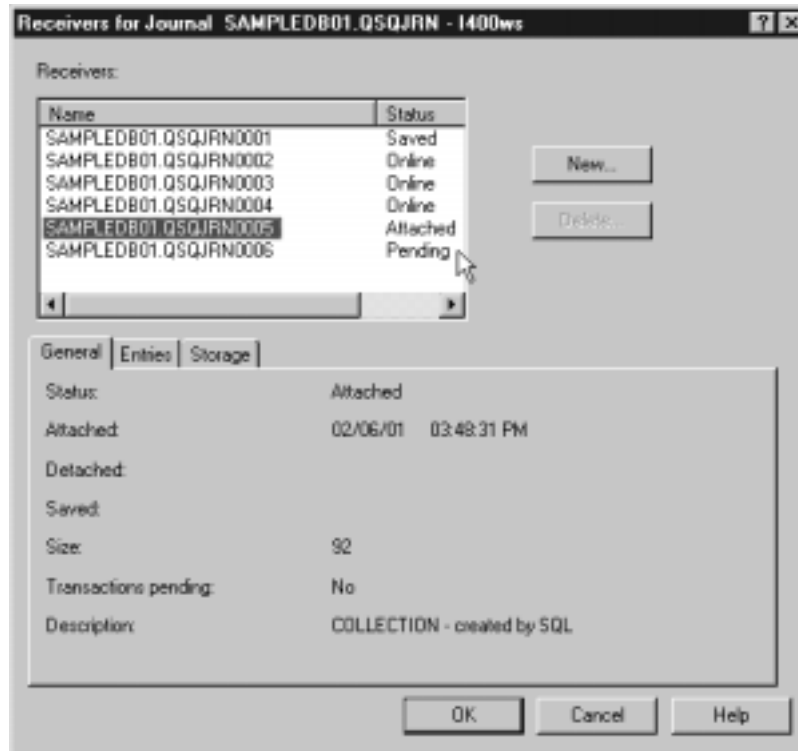


Figure 43. Journal receivers status window

- \_\_\_ 6. Click **OK** to close the Journal Receivers window. Click **OK** again. The new journal receive window changes its status to attached.

**Note**

When the system is managing the receivers, a new receiver is created during an IPL.

### Task 3: Creating journals/receivers

This task explains how to create a journal and a receiver. However, you learn to do this at the end of this task, because there is already a journal and receiver.

- \_\_\_ 1. Right-click your **SAMPLEDBXX** library and click **New-> Journal**. A New Journal window appears.
- \_\_\_ 2. In the Name input field, type **DBJRNXX**. Here, **XX** is your team number.
- \_\_\_ 3. For the Description, type: **Journal for SAMPLEDBXX library**.
- \_\_\_ 4. Type **SAMPLEDBXX** in both the Library and the Library to hold receivers boxes.

**Note**

Normally, the receivers are not kept in the same library as the tables that are to be journaled. For the sake of illustrating how to create a journal and receivers, they are kept in the same library.

- \_\_\_ 5. Click **Advanced**. An Advanced Journal Attributes window appears.
- \_\_\_ 6. Click **New Receiver** to open the New Journal Receiver window.  
Type `DBRCVXX` as the Receiver name, and change the receiver's Storage space threshold to 10 MB.
- \_\_\_ 7. Click **OK** to complete the new receiver properties.
- \_\_\_ 8. Click **OK** to complete the advanced options.
- \_\_\_ 9. Click **OK** to create the journal and receiver. The screen refreshes and the newly created `DBJRNXX` journal and `DBRCVXX` receiver appear in the right panel.
- \_\_\_ 10. Double-click **DBJRNXX** to bring up the Journal Properties window.
- \_\_\_ 11. Click the **Tables** button. This window allows you to select the tables on which you want to activate DB journaling. This option is also available by right-clicking the journal icon menu item **Starts and ends table journaling**.  
You do not add a table here, so click **Cancel** to go back.
- \_\_\_ 12. Click **Cancel** again to finish this task.

---

#### Task 4: Dropping journals/receivers

The purpose of this task is to drop a journal and a receiver. Delete the journal and then the receiver.

**Note**

Journal receivers are not automatically deleted when you delete their associated journal. They must be manually deleted.

- \_\_\_ 1. Right-click your **DBJRNXX** journal icon and select **Delete**. A new window appears to confirm your delete action. Check that you specified the correct `DBJRNXX` to be deleted, and then click **Delete**.
- \_\_\_ 2. Take the same action with your `DBRCVXX` receiver. A warning message appears indicating that the receiver is not yet saved. Click **Yes** to delete it.

You have now completed this lab!

---

## Part 2. Advanced database functions





---

## Lab 6. Database referential constraint

Referential Integrity is a set of mechanisms by which a DB engine enforces some common integrity rules among related tables. Without the Referential Integrity function in the DB engine, the only way to ensure that integrity rules are enforced is to write application codes to take care of them.

With Referential Integrity, these rules can be implemented directly into the database. Once the rules are defined, DB2 UDB for OS/400 automatically enforces them for you.

### Objectives

This lab teaches you how to:

- Create primary key and referential constraints
- Test the referential constraint

### Lab prerequisites

Before you begin this lab, be sure you meet the following prerequisites:

- You must have completed *Lab 1, "iSeries Navigator setup and basic operations" on page 3.*
- You must have completed Task 1 of *Lab 2, "SQL Script Center" on page 11,* because the SAMPLEDBXX schema and SQL Script Center are used here.
- You must have completed *Task 6, "Altering a table" on page 37,* in Lab 3, because the modified SALES table is used here.

### Time required

The time required to efficiently complete this lab project is 15 minutes.

### Naming convention for a 'schema'

Starting in V5R1, the term "schema" is used in the same sense as the term "collection". This is an OS/400 library created with automatic DB journaling enabled and local DB catalog views.

---

## Task 1: Creating a primary key

This exercise explains how to create a primary key and view the effects on the database when data is inserted:

- \_\_\_ 1. In the left panel of the main iSeries Navigator window, expand **I400WS-> Database-> Libraries** and click your **SAMPLEDBXX** library.

Your SAMPLEDBXX library should appear in the active library list under the Libraries icon. If it does not, add it by right-clicking **Libraries** and selecting **Select Libraries to Display**. Refer to Task 4, "Maintaining your active library list for iSeries Navigator" on page 7, in Lab 1, for information on how to do this.

- \_\_\_ 2. In the right panel, right-click the **STAFF** table and select **Quick View** from its pop-up menu. Scan through the rows of this table to see what kind of data it contains.

Close the quick-view window when you are finished.

- \_\_\_ 3. Right-click the **STAFF** table and select **Properties**.
- \_\_\_ 4. In the new Table Properties window, click the **Key Constraints** tab and then click **New** to bring up a New Key Constraint window.
- \_\_\_ 5. In the Constraint field, type its name as `STAFFID_PK`. Then, click the **ID** column. The number 1 appears in front of the column.  
Select the **Primary** Constraint type. Click **OK**.
- \_\_\_ 6. Click **OK** again to create the primary key constraint (Figure 44).

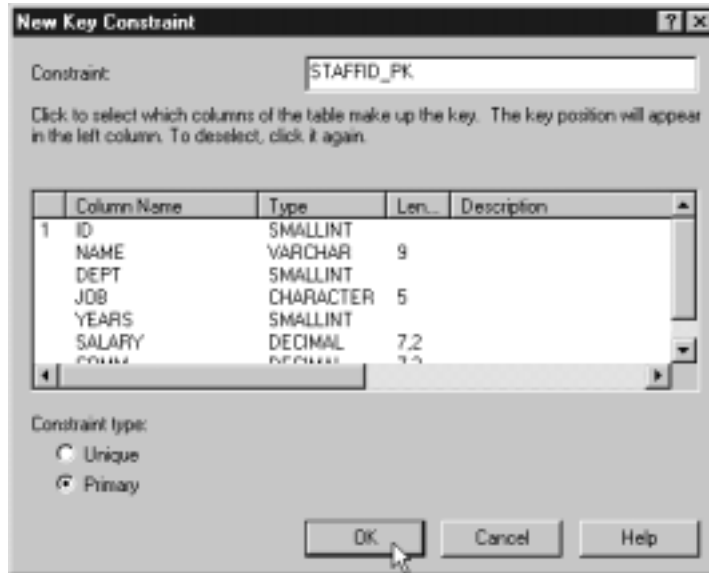


Figure 44. `STAFFID_PK` primary key constraint

At this time, the `STAFF` table is ready to be used as a parent table for a referential constraint. You now define a referential constraint for the `SALES` table, which is called a *dependent table* to the `STAFF` table.

---

## Task 2: Creating a referential constraint

Create and test a referential constraint:

- \_\_\_ 1. In the right panel, right-click the **SALES** table and select **Quick View**. Scan through the rows of this table to see what kind of data it contains. Notice that the `SALES_STAFF` column contains only a value of 99 for all rows.  
Close the quick-view window when you are finished.
- \_\_\_ 2. Start the SQL Script Center by right-clicking the **Database** icon on the left panel and selecting **Run SQL Scripts**.
- \_\_\_ 3. Clear the Script Center working area, and enter the following statement:  

```
INSERT INTO STAFF VALUES (99, 'UNKNOWN', NULL, NULL, NULL, NULL, NULL);
```

Do *not* run it now. Switch back to the iSeries Navigator main window to continue (press Ctrl+Tab or use the Windows task bar).
- \_\_\_ 4. Right-click the **SALES** table and select **Properties**.

- \_\_\_ 5. In the new Table Properties window, click the **Referential Constraints** tab and then click **New** to bring up a New Referential Constraint window.
- \_\_\_ 6. In the Constraint field, type its name as SALESID\_FK. Then, move down to click the **SALES\_STAFF** column in the list box beneath. The number 1 appears in front of the column.

Specify your SAMPLEDBXX library in the Parent table library field (XX is your team number).

Move down to the Parent table: list box. Scroll down to locate and click the **STAFF** table. All the columns of the STAFF table appear in the list box on the right side with a number 1 in front of the ID column.

Move down to the Delete action and Update action fields. Select **Restrict** for both (Figure 45).

Click **OK** to return to the Table Properties window.

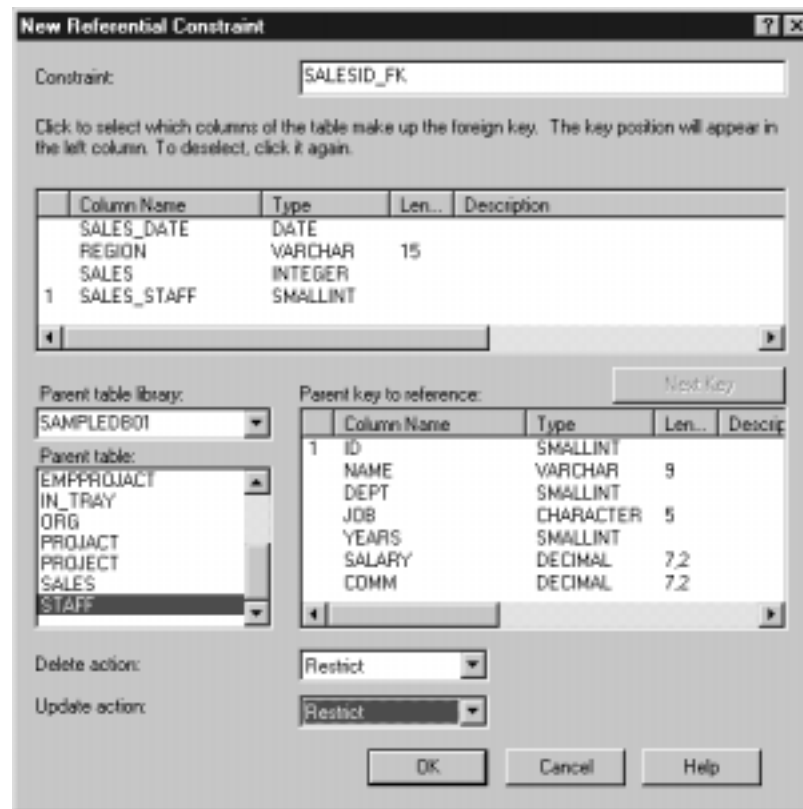


Figure 45. SALESID\_FK referential constraint

- \_\_\_ 7. Click **OK** again to create the constraint. An SQL0667 error message window appears (Figure 46).

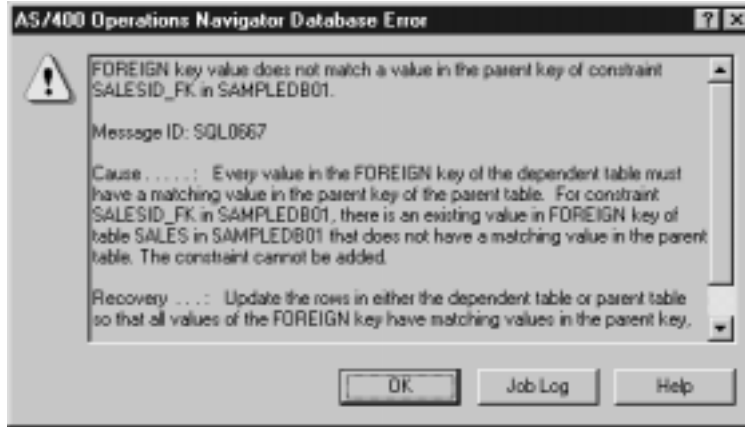


Figure 46. SQL0667 error message

Click **OK** to close the error message window.

This error occurs because the SALES\_STAFF column of every row in the SALES table refers to the staff ID number 99, which does not yet exist in the STAFF table. This is a violation to the referential constraint you are adding. Therefore, DB2 stops adding the constraint and issues an error.

- \_\_\_ 8. Add a new row with staff ID 99 in the parent table (STAFF) by switching to the SQL Script Center window and running the INSERT INTO STAFF statement that you created previously. Close the Script Center when you are finished (there is no need to save anything).

You should now be able to create the referential constraint without a violation.

- \_\_\_ 9. Switch back to Table Properties window and click **OK** to create the referential constraint.

The STAFF and SALES tables are now related with the referential constraint you just created (Figure 47).

| ID | NAME | DF      |
|----|------|---------|
| 33 | 330  | Burke   |
| 34 | 340  | Edwards |
| 35 | 350  | Gafney  |
| 36 | 99   | UNKNOWN |

| SALES_DATE | REGION     | SALES         | SALES_STAFF |
|------------|------------|---------------|-------------|
| 2          | 1995-12-31 | Ontario-South | 3           |
| 3          | 1995-12-31 | Quebec        | 1           |
| 4          | 1995-12-31 | Manitoba      | 2           |
| 5          | 1995-12-31 | Quebec        | 1           |
| 6          | 1996-03-29 | Ontario-South | 3           |
| 7          | 1996-03-29 | Quebec        | 1           |
| 8          | 1996-03-29 | Ontario-South | 2           |
| 9          | 1996-03-29 | Ontario-North | 2           |
| 10         | 1996-03-29 | Quebec        | 3           |
| 11         | 1996-03-29 | Manitoba      | 5           |
| 12         | 1996-03-29 | Ontario-South | 3           |
| 13         | 1996-03-29 | Quebec        | 1           |
| 14         | 1996-03-29 | Manitoba      | 7           |

Figure 47. Data relationship

### Task 3: Testing referential constraints

Referential constraints are active when:

- You delete a row from the parent file.
- You insert a row into the dependent file.
- You update a row in the parent or dependent file.

However, referential constraints are checked but not enforced during an apply/remove DB journal change operation. Therefore, a check pending status may result in which you have to resolve the violation.

You now test the constraint you added in the preceding task:

1. Double-click the **STAFF** table to open it (for update). Keep the window open in a corner of your PC screen for future use.
2. Double-click the **SALES** table to open it (for update).
3. Click the **SALES\_STAFF** column of the first row and change the value from 99 to 88. Press Enter.
4. The SQL0530 error message should appear:

Operation not allowed by referential constraint SALESID\_FK in SAMPLEDBXX

Read through the message details to obtain more information. Click **OK**.

You see another message stating:

The update failed. Do you want the original values reset into the row?

Click **Yes** to return to the SALES table window.

#### Explanation

You cannot update the SALES row to refer to a SALES\_STAFF value of 88 because a STAFF row with such an ID does not exist. Browse through the STAFF table. Notice that there is no row with an ID of 88. This operation violates the basic referential constraint.

5. From the menu bar, click **Rows-> Insert** and insert a row with the information provided in Table 7. Press Enter when you are finished. The row should be inserted without any error.

Table 7. Inserting a SALES row

|             |            |
|-------------|------------|
| SALES_DATE  | 2000-11-11 |
| REGION      | Quebec     |
| SALES       | 3          |
| SALES_STAFF | 20         |

6. Switch to the STAFF window and locate a row with an ID of 20. This may be located in the second row.
7. Click anywhere in the row and click **Row-> Delete** from the menu bar.

The following SQL0532 error message should appear:

Delete prevented by referential constraint SALESID\_FK in SAMPLEDBXX

Read through the details of the message to obtain more information. Click **OK** to close the message window.

**Explanation**

You cannot delete a STAFF row with an ID of 20 because you just added a SALES row with a SALES\_STAFF value of 20, which refers to the STAFF row you just tried to delete. The operation failed because you specified a Restrict delete action.

- \_\_\_ 8. Try changing the ID of this STAFF row from 20 to 77. Another error message should appear because you also specified a Restrict update action. You cannot change the ID of this row while it is still referred to by a row from the SALES table.

This means that the referential constraint is working correctly.

You have now completed this lab!

---

## Lab 7. Database trigger

Triggers are user-written programs that are activated by the database manager when a data change is performed in the database. Triggers are mainly intended for monitoring database changes and taking appropriate actions. The main advantage of using triggers, instead of calling the program from within an application, is that triggers are activated automatically. It does this regardless of the application (local or remote, or whichever programming language it is developed in) that generates the data change.

In addition, once a trigger is put in place, application programmers and end users cannot circumvent it. When a trigger is activated, the control shifts from the application program to the database manager. The operating system then executes the specifications you coded in the trigger program to perform the actions you designed. The application waits until the trigger ends and then regains control.

In OS/400 V5R1, new support for an SQL trigger is added. This means that you can now use procedural SQL to create a trigger program. To do this, you need the DB2 SQL Development Kit (Licensed Program 5722-ST1) on your system to support compile-time generation of the trigger program. It is *not* required for run-time support and you do *not* need a C language compiler to create an SQL trigger.

There are now two types of trigger program implementations:

- **External trigger:** This is a trigger program developed in OS/400-supported high-level language compilers, such as CL, RPG, COBOL, and C (but not Java) and compiled into an OS/400 program object. With the exception of CL, these languages can include embedded SQL. External triggers have been available since V3R1. They have simply been given this new name in V5R1.
- **SQL trigger:** This is an internal trigger program created purely in procedural SQL statements. A program object is created for an SQL trigger that is equivalent to an ILE C program with embedded SQL (\*PGM type with CLE attribute).

The SQL trigger is available in V5R1. You are given the opportunity to use it in this lab exercise.

Two new features added in V5R1 include a column-level SQL trigger (for update event only) and Read After event support for an external trigger. This makes the DB2 UDB for OS/400 trigger more flexible than ever to use.

### Objectives

This lab teaches you how to:

- Add SQL triggers to a table
- Code procedural SQL for the triggers

## Lab prerequisites

Before you begin this lab, be sure you meet the following prerequisites:

- You must have completed *Lab 1, "iSeries Navigator setup and basic operations" on page 3.*
- You must have completed Task 1 of *Lab 2, "SQL Script Center" on page 11,* because the SAMPLEDBXX schema and SQL Script Center are used here.
- You must have completed *Task 6, "Altering a table" on page 37,* in Lab 3, because the modified SALES table is used here.
- You need to install an OS/400 optional component named "System Openness Includes" in your machine. It contains include files necessary for creating SQL triggers.

## Time required

The time required to efficiently complete this lab project is 30 minutes.

## Naming convention for a 'schema'

Starting in V5R1, the term "schema" is used in the same sense as the term "collection". This is an OS/400 library created with automatic DB journaling enabled and local DB catalog views.

---

## Task 1: Creating an SQL trigger for AFTER INSERT

You now move on to explore triggers. In this task, you create an SQL trigger, named COMMCALC, that maintains an accumulated sales commission for a sales staff when someone makes a sales transaction. This is done when a new row is added into the SALES table. A value of 100 (dollars) is multiplied to the accomplished number of transactions. The result is added and updated into the COMM column of the STAFF table.

The trigger is therefore associated with the SALES table.

- \_\_\_ 1. In the left panel of the main iSeries Navigator window, expand **I400WS-> Database-> Libraries** and click your **SAMPLEDBXX** library.

Your SAMPLEDBXX library should appear in the active library list under the Libraries icon (XX is your team number). If it does not, add it by right-clicking **Libraries** and selecting **Select Libraries to Display**. Refer to Task 4, "Maintaining your active library list for iSeries Navigator" on page 7, in Lab 1, for instructions on how to do this.

- \_\_\_ 2. In the right panel, right-click the **STAFF** table and select **Quick View**. Scan through the rows of this table to see what kind of data it contains. Note the COMM column, which keeps the current total sales commission income of each staff member.

Close the quick-view window when you are finished.

- \_\_\_ 3. Right-click the **SALES** table and select **Properties**.

Considering the information in the Table Properties window, you can make the following assumptions to prepare for the use of the triggers that you create:



- Each row in this table represents a sales record that a staff member accomplishes on a specific date.
- The SALES\_STAFF column represents the staff member who makes the sales record.
- The SALES column represents the number sales transactions a staff member makes for that sales record. Each transaction means that a \$100 sales commission is added to that staff member's current sales commission (this is the value in the COMM column of the corresponding row in the STAFF table).

You now create the first SQL trigger for an insert-after event that accumulates sales commissions for the staff member who accomplishes sales transactions.

- \_\_\_ 4. Click the **Triggers** tab, and then click **Add SQL Trigger** to bring up the Add SQL Trigger for Table... window.
- \_\_\_ 5. Select the **General** tab and perform the following steps:
  - a. In the Trigger field, type `COMMCALC` as the name.
  - b. Make sure the Library field contains your `SAMPLEDBXX` library.
  - c. Type the Description as `Sales Commission Calculation`.
  - d. Select **Insert** for Event

- \_\_\_ 6. Click the **Timing** tab and perform the following steps:

- a. Select **After event** for When to run.
- b. Select **For each row** for Run Trigger.
- c. Specify `NewSalesRow` for Correlation name for new row.

`NewSalesRow` is used in the procedural SQL code of the trigger to refer to the record being inserted.

- d. Select **DB2ROW** for Mode.

- \_\_\_ 7. Click the **SQL Statements** tab. In the SQL statement working area, delete the first line `WHEN` (search condition).

Then, enter the following statements:

```
BEGIN
```

```
DECLARE NEWCOMM DECIMAL(7,2);
```

```
SET NEWCOMM = 0;
```

```
SELECT COALESCE(COMM,0) INTO NEWCOMM FROM SAMPLEDBXX.STAFF WHERE ID =  
NEWSALESROW.SALES_STAFF;
```

```
SET NEWCOMM = NEWCOMM + (100 * NEWSALESROW.SALES);
```

```
UPDATE SAMPLEDBXX.STAFF SET COMM = NEWCOMM WHERE ID =  
NEWSALESROW.SALES_STAFF;
```

```
END
```

- \_\_\_ 8. Click **Check Syntax** to ensure that you entered the code correctly. If you receive an error message, verify the syntax of the SQL statements and make the proper correction.

Click **OK** when you are finished.

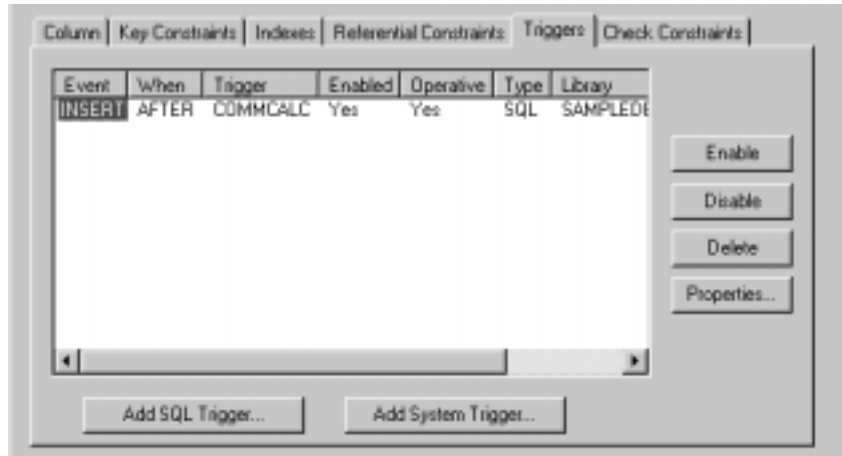


Figure 48. Trigger tab of the Table Properties window

- \_\_\_ 9. If you do *not* want to add another trigger (in the next optional task), click **OK** to create the COMMCALC SQL trigger. Then, go to Lab 3, “Testing the SQL trigger for AFTER INSERT” on page 72.

If you *do* want to add another trigger (in the next task), do *not* click **OK**.

**Note**

After adding a trigger, you can disable/enable it by using the **Disable** and **Enable** buttons in the **Triggers** tab as shown in Figure 48. The initial status of a newly-added trigger is Enable=Yes.

You have now successfully prepared an SQL trigger for the insert-after event to the SALES table. You must now add another SQL trigger.

## Task 2: Creating a column-level SQL trigger for AFTER UPDATE (optional)

Assume that an existing row in the SALES table can be changed. One possibility is that the number of the sales transactions (in the SALES column) can be adjusted. Therefore, the sales commission of that sales staff member must also be adjusted accordingly. The adjustment can be an addition or a subtraction of a proper multiplication of 100 (dollars) to the current value.

- \_\_\_ 1. In the Table Properties window of the SALES table, click the **Triggers** tab and then click **Add SQL Trigger** to bring up an Add SQL Trigger for Table... window.
- \_\_\_ 2. Select the **General** tab and perform the following steps:
- In the Trigger input field, type the name `COMMADJ`.
  - Make sure the Library field contains your `SAMPLEDBXX` library.
  - Type the Description as `Sales Commission Adjustment`.
  - Select **Update for selected columns** for Event.

Under the Available columns: list box, select **SALES\_STAFF** and click **Add->**. Then, select **SALES** and click **Add->**

- \_\_\_ 3. Click the **Timing** tab and perform the following steps:

- a. Select **After event** for When to run.
- b. Select **For each row** for Run Trigger.
- c. Specify:
  - OldValue for Correlation name for old row
  - NewValue for Correlation name for new row

OldValue and NewValue are used in the procedural SQL code of the trigger (shown in Step 4) to refer to the before-image and the after-image of the record being updated.

- d. Select **DB2ROW** for Mode.

- \_\_\_ 4. Click the **SQL Statements** tab and type the following statements (use copy/paste, or Ctrl+X/Ctrl+V, to speed up your typing and reduce mistakes):

```

WHEN(NEWVALUE.SALES <> OLDVALUE.SALES OR NEWVALUE.SALES_STAFF <>
OLDVALUE.SALES_STAFF)
BEGIN

DECLARE NEWCOMM DECIMAL(7,2);
SET NEWCOMM = 0;

SELECT COALESCE(COMM, 0) INTO NEWCOMM FROM SAMPLEDBXX.STAFF WHERE ID
= OLDVALUE.SALES_STAFF;

SET NEWCOMM = NEWCOMM - (100 * OLDVALUE.SALES);

UPDATE SAMPLEDBXX.STAFF SET COMM = NEWCOMM WHERE ID =
OLDVALUE.SALES_STAFF;

SELECT COALESCE(COMM, 0) INTO NEWCOMM FROM SAMPLEDBXX.STAFF WHERE ID
= NEWVALUE.SALES_STAFF;

SET NEWCOMM = NEWCOMM + (100 * NEWVALUE.SALES);

UPDATE SAMPLEDBXX.STAFF SET COMM = NEWCOMM WHERE ID =
NEWVALUE.SALES_STAFF;

END

```

- \_\_\_ 5. Click **Check Syntax** to ensure that you entered the code correctly. If you receive an error message, verify the syntax of the SQL statements and make the proper correction.

Click **OK** when you are finished.

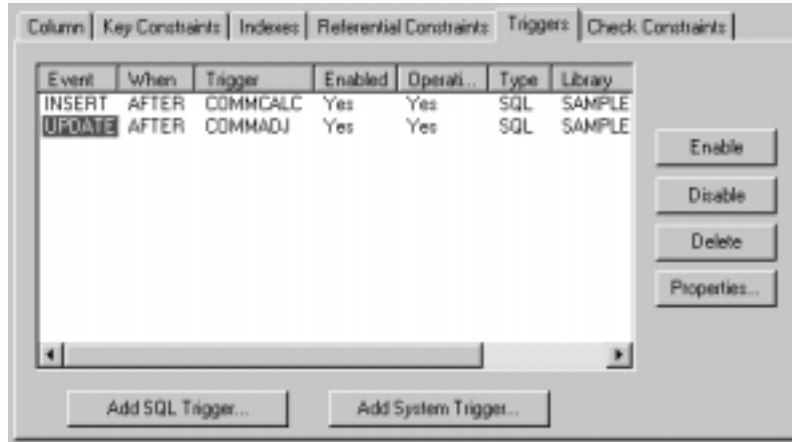


Figure 49. Trigger tab of Table Properties window

\_\_\_ 6. In the Table Properties window (Figure 49), click **OK** to create the two SQL triggers.

You must now test the AFTER INSERT trigger.

### Task 3: Testing the SQL trigger for AFTER INSERT

In this task, you test the trigger by inserting three rows (representing sales records of three staff members) into the SALES table and looking into the STAFF table for trigger updates of their commission earnings (in the COMM column).

\_\_\_ 1. Right-click the **STAFF** table and select **Quick View** from the pop-up menu to quick-view its contents. Write down the current values in the COMM column of the following staff IDs:

- Staff ID = 10: COMM = \_\_\_\_\_ dollars
- Staff ID = 20: COMM = \_\_\_\_\_ dollars
- Staff ID = 40: COMM = \_\_\_\_\_ dollars

Close the quick-view window when you are finished.

\_\_\_ 2. Double-click the **SALES** table to open it for update.

From the menu bar of the SALES table window, click **Rows-> Insert** and enter the information shown in Table 8 for the three new rows. Press Enter for each row that you add.

#### Note

If you see the message "The table is not being journaled", simply click **Yes** to proceed.

When you finish entering the three rows, click **File-> Save** from the menu bar and close the window.

Table 8. Inserting rows to the SALES table

| SALES_DATE | REGION     | SALES | SALES_STAFF |
|------------|------------|-------|-------------|
| 2001-01-01 | North West | 3     | 10          |
| 2001-01-01 | North West | 5     | 20          |
| 2001-01-01 | North West | 2     | 40          |

\_\_\_ 3. Quick-view the **STAFF** table again and write down (in the COMM column) the current values of the following staff IDs. Compare them to the values that you wrote down previously:

- Staff ID = 10: COMM = \_\_\_\_\_ dollars (should be 300 *more*)
- Staff ID = 20: COMM = \_\_\_\_\_ dollars (should be 500 *more*)
- Staff ID = 40: COMM = \_\_\_\_\_ dollars (should be 200 *more*)

If you do not see the result suggested above, go back to the preceding task and verify your SQL trigger codes.

Close the quick-view window when you are finished.

If you ran Task 2, you are ready to go to the next task to test it. Otherwise, skip the next task and go to Lab 8, “Stored procedure” on page 75.

#### Task 4: Testing the column-level SQL trigger for AFTER UPDATE (optional)

In this task, you test the trigger by updating an existing SALES row and looking into the STAFF table for the trigger updates of their commission adjustment:

\_\_\_ 1. Quick-view the **STAFF** table and write down (in the COMM column) the current values of the following staff IDs:

- Staff ID = 30: COMM = \_\_\_\_\_ dollars
- Staff ID = 40: COMM = \_\_\_\_\_ dollars

\_\_\_ 2. Double-click the **SALES** table to open it for update. Locate the last row that you inserted in the previous task and change it as indicated in Table 9.

**Note**

If you see the message “The table is not being journaled”, simply click **Yes** to proceed. When you finish entering these rows, click **File-> Save** to finish updating the row and close the window.

Table 9. Updating a row in the SALES table

| SALES_DATE | REGION     | SALES          | SALES_STAFF      |
|------------|------------|----------------|------------------|
| 2001-01-01 | North West | 2 changed to 5 | 40 changed to 30 |

\_\_\_ 3. Quick-view the **STAFF** table again and write down (in the COMM column) the current values of the following staff IDs. Compare them to the values that you wrote down previously:

- Staff ID = 30: COMM = \_\_\_\_\_ dollars (should be 500 *more*)
- Staff ID = 40: COMM = \_\_\_\_\_ dollars (should be 200 *less*)

If you do not see the result suggested above, go back to the preceding task and verify your SQL trigger codes.

Try changing a few other rows of the SALES table and see that your changes are reflected in the STAFF table by the trigger.

Close the quick-view window when you are finished.

You have now completed this lab!

---

## Lab 8. Stored procedure

Stored procedures provide a standard method for calling an external program module from within an SQL application by the CALL statement.

The implementation of SQL stored procedures is based on the SQL standard. It supports constructs that are common to most programming languages. It supports the declaration of local variables, statements to control the flow of the procedure, assignment of expression results to variables, receiving and returning of parameters, and error handling.

In V4R5, Java stored procedure support is added to DB2 UDB to provide more choices for implementation.

In V5R1, like the SQL trigger, the C language compiler is no longer required to create an SQL stored procedure. Only an SQL Development Kit is needed. As for the run-time environment of stored procedures, only OS/400 is required.

In V5R2 a new support is added where the SQL Script Center can receive and display the results from the “output parameters” of the procedure.

### Objectives

This lab teaches you how to create an SQL stored procedure that:

- Returns result sets to SQL Script Center
- Returns the value of output parameter to SQL Script Center

You can optionally learn the V5R2 support for SQL source-level debugging.

### Lab prerequisites

Before beginning this lab, be sure you meet the following prerequisites:

- You must have completed *Lab 1, “iSeries Navigator setup and basic operations” on page 3.*
- You must have completed Task 1 of *Lab 2, “SQL Script Center” on page 11,* because the SAMPLEDBXX schema and SQL Script Center are used in this lab.
- You need to install an OS/400 optional component named “System Openness Includes” in your machine. It contains include files necessary for creating SQL procedure.

### Time required

The time required to efficiently complete this lab project is 15 minutes.

### Naming convention for a ‘schema’

Starting in V5R1, the term “schema” is used in the same sense as the term “collection”. This is an OS/400 library created with automatic DB journaling enabled and local DB catalog views.

---

## Task 1: SQL stored procedure that returns result sets

In this lab, you create a stored procedure named EARNLIST that generates up to two lists of top “n” staff earnings from the STAFF table in your SAMPLEDBXX schema. The lists are:

- Staff members who earn top “n” salaries
- Staff members who earn top “n” sales commissions

This procedure requires four input parameters, as indicated in the following order:

- **SAL CHAR(6)**: Type ‘salary’ here if you want to see the salary earning list.
- **TOPSAL CHAR(2)**: Type up to two digits for the top ‘N’ result.
- **COMM CHAR(6)**: Type ‘comm’ here if you want to see a commission earning list.
- **TOPCOMM CHAR(2)**: Type up to two digits for the top ‘N’ result.

Any of the four parameters listed above can be NULL.

Start by creating EARNLIST (sal, topsal, comm, topcomm):

- \_\_\_ 1. In the left panel of the main iSeries Navigator window, expand **I400WS-> Database-> Libraries** and click your **SAMPLEDBXX** library.  
Your SAMPLEDBXX library should appear in the active library list under the Libraries icon (XX is your team number). If it does not, add it by right-clicking **Libraries** and selecting **Select Libraries to Display**. Refer to Task 4, “Maintaining your active library list for iSeries Navigator” on page 7, in Lab 1, for instructions on how to do this.
- \_\_\_ 2. In the right panel, right-click the **STAFF** table and select **Quick View**. Scan through the rows of this table to see what kind of data it contains. Note the COMM column, which keeps the current total sales commission earnings of each staff member, and the SALARY column that indicates the staff member’s monthly fixed earning.  
Close the quick-view window when you are finished.
- \_\_\_ 3. In the left panel, right-click your **SAMPLEDBXX** library and click **New->Procedure->SQL**. A New SQL Procedure in ... window appears.
- \_\_\_ 4. Click the **General** tab and perform the following steps:
  - a. In the Procedure field, type the name **EARNLIST**.
  - b. Type the Description as **Staff Earnings List**.
  - c. Specify the Maximum number of result sets as **2**.
  - d. Select **Reads SQL data** for Data access.
- \_\_\_ 5. Click the **Parameters** tab and click the **Insert** button to enter each of the four parameters shown in Figure 50.



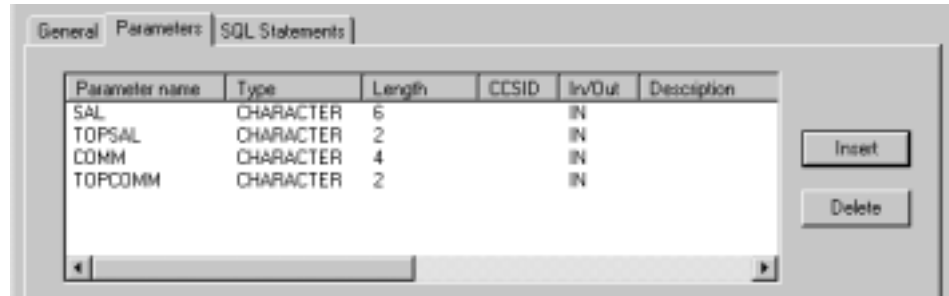


Figure 50. Parameters for EARNLIST stored procedure

- \_\_\_ 6. Click the **SQL Statements** tab. In the SQL statement working area, enter the following statements:

**Note**

The text indentation format shown here is only for readability. You do not have to follow this exact format. Please use care in typing the semicolon (;) as indicated only. Also type your team number for SAMPLEDBXX.

```
-- EARNLIST (sal , topsal, comm, topcomm)
BEGIN
    DECLARE SQLS1 CHAR(256);
    DECLARE SQLS2 CHAR(256);
    DECLARE C1 CURSOR WITH RETURN FOR S1;
    DECLARE C2 CURSOR WITH RETURN FOR S2;

    IF SAL IS NOT NULL THEN
        SET SQLS1 = 'SELECT ID, NAME, DEPT, JOB, SALARY FROM
SAMPLEDBXX.STAFF WHERE SALARY IS NOT NULL ORDER BY SALARY DESC
FETCH FIRST ' CONCAT TOPSAL CONCAT ' ROWS ONLY';
```

**Note**

Type a space after FIRST and before ROWS. XX is your team number.

```
        PREPARE S1 FROM SQLS1; OPEN C1;
    END IF;
    IF COMM IS NOT NULL THEN
        SET SQLS2 = 'SELECT ID, NAME, DEPT, JOB, COMM FROM SAMPLEDBXX.STAFF
WHERE COMM IS NOT NULL ORDER BY COMM DESC FETCH FIRST ' CONCAT
TOPCOMM CONCAT ' ROWS ONLY';
```

**Note**

Type a space after FIRST and before ROWS. XX is your team number.

```
        PREPARE S2 FROM SQLS2; OPEN C2;
    END IF;
    SET RESULT SETS CURSOR C1, CURSOR C2;

END
```



Figure 51. SQL codes for EARNLIST stored procedure

After you verify the codes, click **OK** to create the EARNLIST stored procedure. Notice the message “Procedure SAMPLEDBXX.EARNLIST created successfully” in the bottom-most status bar of the iSeries Navigator window.

You must now test the stored procedure.

- \_\_\_ 7. Open the SQL Script Center by right-clicking the **I400WS** icon (under Databases) in the left panel. Select **Run SQL Scripts**.
- \_\_\_ 8. Clear the working area and type the following statement:

```
CALL SAMPLEDBXX.EARNLIST('salary', '10', 'comm', '10');
```

Replace **XX** with your team number and then double-click the statement to run it. This specifies that you want to see the top ten salary earners and the top ten commission earners. You should see the result sets of the two separate tabs shown in Figure 52 (or in two new separate windows).

CALL SAMPLEDB01.EARNLIST (SALARY, 10, COMM, 10).

| ID  | NAME     | DEPT | JOB   | SALARY   |
|-----|----------|------|-------|----------|
| 160 | Martinez | 10   | Mgr   | 22950.20 |
| 200 | Jones    | 10   | Mgr   | 21234.00 |
| 140 | Faye     | 51   | Mgr   | 21150.00 |
| 310 | Graham   | 66   | Sales | 21000.00 |
| 50  | Hanes    | 15   | Mgr   | 20055.60 |
| 210 | Lee      | 10   | Mgr   | 20010.00 |
| 290 | Quill    | 84   | Mgr   | 19818.00 |
| 150 | Williams | 51   | Sales | 19456.50 |
| 240 | Daniels  | 10   | Mgr   | 16260.25 |
| 260 | Wilson   | 66   | Sales | 18674.50 |

Messages: CALL SAMPLEDB01.EARNLIST (SALARY, 10, COMM, 10) CALL SAMPLEDB01.EARNLIST (SALARY, 10, COMM, 10)

Figure 52. Two result sets of top 'n' list from EARNLIST stored procedure

- \_\_\_ 9. Close the result window and the SQL Script Center when you are finished. You do not need to save anything.

**Note**

The stored procedure used in this exercise is a good example of how you can test a functional SQL module from a PC client before you incorporate it into the productive application of your business.

Notice how convenient this can be done using the tools provided by iSeries Navigator. You should now have an idea of how to test a stored procedure using the SQL Script Center and iSeries Navigator.

**Task 2: SQL stored procedure that returns the value of output parameter**

In this lab, you create a stored procedure named TOPCOMPAY(N, TOTAL) that calculates a summation of the total commission payment to the top 'N' commission makers. This procedure takes one input parameter named N of data type INTEGER. It returns the result to an output parameter named TOTAL of data type DECIMAL(10,2).

You call this procedure from SQL Script Center. You can see the value returned to the output parameter displayed in the Script Center. This is a new feature in V5R2.

Follow these steps to create the procedure (the steps are generally similar to the previous task):

- \_\_\_ 1. In the left panel of the main iSeries Navigator window, expand **I400WS-> Database-> Libraries** and click your **SAMPLEDBXX** library.

Your SAMPLEDBXX library should appear in the active library list under the Libraries icon (XX is your team number). If it does not, add it by right-clicking **Libraries** and selecting **Select Libraries to Display**. Refer to

Task 4, "Maintaining your active library list for iSeries Navigator" on page 7, in Lab 1, for instructions on how to do this.

- \_\_\_ 2. In the left panel, right-click your **SAMPLEDBXX** library and click **New-> Procedure-> SQL**. A New SQL Procedure in ... window appears.
- \_\_\_ 3. Click the **General** tab and perform the following steps:
  - a. In the Procedure input field, type the name `TOPCOMPAY`.
  - b. Type the Description as `Top N commission payment summation`.
  - c. Select **Reads SQL data** for Data access.
- \_\_\_ 4. Click the **Parameters** tab and click the **Insert** button to enter one output parameter as shown in Table 10.

Table 10. Parameter declaration for `TOP10COMPAY` procedure

| Parameter name | Type    | Length | in/Out |
|----------------|---------|--------|--------|
| N              | INTEGER | -----  | IN     |
| TOTAL          | DECIMAL | 10,2   | OUT    |

- \_\_\_ 5. Click the **SQL Statements** tab. In the SQL statement working area, enter the following statements:

**Note**

The text indentation format shown here is only for readability. You do not have to follow this exact format. Use care in typing the semicolon (;) as indicated only. Also type your team number for `SAMPLEDBXX`.

```
-- TOPCOMPAY (N, TOTAL)
BEGIN
DECLARE SQLS1 CHAR(256);
DECLARE C1 CURSOR FOR S1;
SET TOTAL = 0;

IF N IS NOT NULL THEN
  SET SQLS1 = 'SELECT SUM(COMM) FROM (
  SELECT COMM FROM SAMPLEDBXX.STAFF
  WHERE COMM IS NOT NULL ORDER BY COMM DESC
  FETCH FIRST ' CONCAT CHAR(N) CONCAT
  ' ROWS ONLY) AS T1';

  PREPARE S1 FROM SQLS1;
  OPEN C1;
  FETCH C1 INTO TOTAL;
  CLOSE C1;
END IF;
END
```

After verifying the codes, click **OK** to create the `TOPCOMPAY` SQL procedure. Notice the message "Procedure `SAMPLEDBXX.TOPCOMPAY` created successfully" in the bottom-most status bar of the iSeries Navigator window.

Now test the stored procedure.

\_\_\_ 6. Open the SQL Script Center by right-clicking the **I400WS** icon (under Databases) in the left panel. Select **Run SQL Scripts**.

\_\_\_ 7. Clear the working area and type the following statement:

```
CALL SAMPLEDBXX.TOPCOMPAY(10, ?);
```

Replace **XX** with your team number and then double-click the statement to run it.

You should see the result as shown in Figure 53 where the calculated value of the output parameter **TOTAL** is displayed in the lower panel of the Script Center (Messages tab area). The figure shows a sample of the top 10, 15, and 20 commission payment summation.

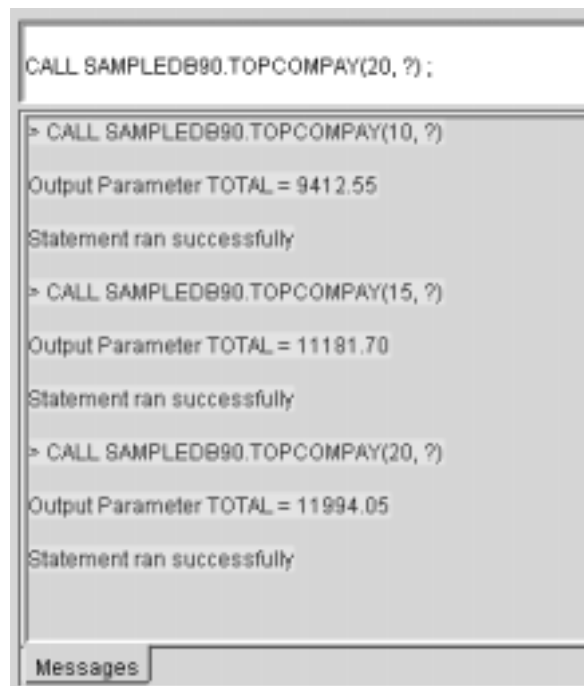


Figure 53. The total top N commission payment from TOPCOMPAY procedure

\_\_\_ 8. Close the result window and the SQL Script Center when you are finished. You do not need to save anything.

Now you have learned about an SQL stored procedure that returns the value of the output parameter.

---

### Task 3: Perform SQL source-level debugging (new in V5R2 - optional)

If you are an SQL programmer, you may find that, in many occasions, you want to perform program source-level debugging on the SQL routines that you create. Since V4R2, you have been able to do this by using the syntax `SET OPTION DBGVIEW = *STMT` or `*LIST` with the `CREATE FUNCTION`, `PROCEDURE`, or `TRIGGER` syntaxes. But the source-level listing, you can see in the debugger view is displayed in C-generated codes only. This tends to be quite long and, therefore, difficult to identify the SQL entities of your interest. In V5R2, an enhancement helps you see only SQL source codes listing in the debugger view.

Before you create an SQL routine, add the new syntax: `SET OPTION DBGVIEW =*SOURCE` before the program body of the SQL routine. After the routine is created, you can use the following OS/400 command to invoke the source debugger:

```
STRDBG lib/<routine name>
```

You can see SQL code debug view by pressing F15 and selecting the **SQL Output View** option.

This support is, currently at V5R2, *not* available through iSeries Navigator. You must use 5250 session for this support.

It is also important that you use OS/400 command RUNSQLSTM to execute the SQL scripts that create the routines (with `SET OPTION DBGVIEW = *SOURCE`) and then run STRDBG command in that *same* 5250 session. This is because when the SQL routine is created, its source codes are added as a member (with the same name as the routine's name) into a source file named QSQLSRC, which is always created in QTEMP library. The debugger then uses this member to display the source codes in the debugger view. QTEMP library is always destroyed when you log off.

Let's start SQL source-level debugging:

- \_\_\_ 1. In the left panel of the main iSeries Navigator window, expand **I400WS-> Database-> Libraries** and click your **SAMPLEDBXX** library.  
Your SAMPLEDBXX library should appear in the active library list under the Libraries icon (XX is your team number). If it does not, add it by right-clicking **Libraries** and selecting **Select Libraries to Display**. Refer to Task 4, "Maintaining your active library list for iSeries Navigator" on page 7, in Lab 1, for instructions on how to do this.
- \_\_\_ 2. In the right panel, locate and right-click the **TOPCOMPAY(INT, DEC() )** procedure that you created in the previous task. Then select **Generate SQL** from the pop-up menu. The Generate SQL -I400WS window appears.
- \_\_\_ 3. In the **Output** tab, select the **Write to file** radio button and specify the following parameters:
  - **File type:** Database source file (default)
  - **Library:** SAMPLEDBXX (XX = your team number)
  - **File name:** QSQLSRC
  - **Member:** TOPCOMPAID (we use a different name for debugging)
  - Deselect the **Append** option.Then click **Generate** button to create the SQL source of CREATE PROCEDURE.
- \_\_\_ 4. In the right panel, refresh the list by pressing F5 and then make sure you can locate the QSQLSRC table (an OS/400 source physical file is identified as a table).
- \_\_\_ 5. Open a 5250 session and sign on to the server. Then use OS/400 PDM - SEU (STRSEU or WRKOBJPDM commands) to open the member TOPCOMPAID of the source file QSQLSRC.
- \_\_\_ 6. Locate the line with the syntax CALLED ON NULL INPUT and insert a *new* line *underneath* with the following syntax :

```
SET OPTION DBGVIEW = *SOURCE
```

This line is shown in bold in the example in Figure 54.

```

Columns . . . : 1 71          Edit          SAMPLEDB90/QSQLSRC
SEU==>
FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
***** Beginning of data *****
0001.00 -- Generate SQL
0002.00 -- Version:          V5R2M0 020719
0003.00 -- Generated on:     04/30/02 08:37:36
0004.00 -- Relational Database: I400WS
0005.00 -- Standards Option:  DB2 UDB AS/400
0006.00
0007.00 CREATE PROCEDURE SAMPLEDB90.TOPCOMPAID (
0008.00   IN TOP_N INTEGER ,
0009.00   IN TOTAL DECIMAL(10, 2) )
0010.00 LANGUAGE SQL
0011.00 SPECIFIC SAMPLEDB90.TOPCOMPAID
0012.00 NOT DETERMINISTIC
0013.00 READS SQL DATA
0014.00 CALLED ON NULL INPUT
0015.00 SET OPTION DBGVIEW = *SOURCE
0016.00   PROC: BEGIN

F3=Exit  F4=Prompt  F5=Refresh  F9=Retrieve  F10=Cursor  F11=Toggle
F16=Repeat find  F17=Repeat change  F24=More keys
(C) COPYRIGHT IBM CORP. 1981, 2002.

```

Figure 54. Inserting the line: SET DBGVIEW = \*SOURCE

- \_\_\_ 7. Change all occurrences in the source codes of the name **TOPCOMPAY** to TOPCOMPAID. There are three occurrences in the lines that begin with CREATE PROCEDURE, SPECIFIC, and COMMENT ON SPECIFIC.
- \_\_\_ 8. Change TOTAL, which is an OUT parameter, to be an IN parameter.
- \_\_\_ 9. Qualify the BEGIN clause with PROC:. This is because to evaluate (eval) a variable, it has to be qualified.
- \_\_\_ 10. Save and exit the Source Entry Utility. Go back to the OS/400 command line in your 5250 session.
- \_\_\_ 11. Use the Run SQL Statement (RUNSQLSTM) command to create a new procedure TOPCOMPAID:
 

```
RUNSQLSTM SRCFILE(SAMPLEDBXX/QSQLSRC) SRCMBR(TOPCOMPAID) NAMING(*SQL)
```
- \_\_\_ 12. Once the procedure is created, use the DSPJOBLOG command (with F10 for detailed messages). Make sure you see the following messages in your job log (Figure 55):
 

```

File QSQLSRC created in library QTEMP.
File QSQLSRC in library QTEMP changed.
Member TOPCOMPAID added to file QSQLSRC in QTEMP.
File QSQDSRC created in library QTEMP.
File QSQDSRC in library QTEMP changed.
Member TOPCOMPAID added to file QSQDSRC in QTEMP.

```

This indicates that the SQL source-level debug information is ready for the debugger to use.

```

                                Command Entry                                I400WS
  Request level:  1

All previous commands and messages:
  Job 013368/SATID/QPADEV0003 started on 04/30/02 at 10:52:00 in subsystem
  QINTER in QSYS. Job entered system on 04/30/02 at 10:52:00.
> STRSEU SRCFILE(SAMPLEDB90/QSQLSRC) SRCMBR(TOPCOMPAID)
> RUNSQLSTM SRCFILE(SAMPLEDB90/QSQLSRC) SRCMBR(TOPCOMPAID) NAMING(*SQL)
  Printer device PRT01 not found. Output queue changed to QPRINT in library
  QGPL.
  File QSQLSRC created in library QTEMP.
  File QSQLSRC in library QTEMP changed.
  Member TOPCOMPAID added to file QSQLSRC in QTEMP.
  File QSQDSRC created in library QTEMP.
  File QSQDSRC in library QTEMP changed.
  Member TOPCOMPAID added to file QSQDSRC in QTEMP.

  More...

Type command, press Enter.
====>

F3=Exit   F4=Prompt   F9=Retrieve   F10=Exclude detailed messages
F11=Display full   F12=Cancel   F13=Information Assistant   F24=More keys

```

Figure 55. Creating an SQL procedure with `DBGVIEW = *SOURCE` option

\_\_\_ 13. On the OS/400 command line, start the debugger:

```
STRDBG SAMPLEDBXX/TOPCOMPAID
```

You should see the SQL source in the debugger listing view (Figure 56).

```

                                Display Module Source

Program:  TOPCOMPAID      Library:  SAMPLEDB90      Module:  TOPCOMPAID
 1 CREATE PROCEDURE SAMPLEDB90 . TOPCOMPAID ( IN TOP_N INTEGER , OUT TOTA
 2 DETERMINISTIC READS SQL DATA CALLED ON NULL INPUT SET OPTION DBGVIEW =
 3 BEGIN
 4 DECLARE SQLS1 CHAR ( 256 );
 5 DECLARE C1 CURSOR FOR S1;
 6 IF TOP_N IS NOT NULL
 7 THEN
 8 SET SQLS1 = 'SELECT SUM(COMM) FROM (SELECT COMM FROM SAMPLEDB01.STAFF
 9 CONCAT ' ROWS ONLY) AS T1';
10 PREPARE S1 FROM SQLS1;
11 OPEN C1;
12 FETCH C1 INTO TOTAL_COMM;
13 CLOSE C1;
14 END IF;
15 END;

  Bottom

Debug . . .

F3=End program   F6=Add/Clear breakpoint   F10=Step   F11=Display variable
F12=Resume       F17=Watch variable   F18=Work with watch   F24=More keys

```

Figure 56. OS/400 Debugger view of SQL procedure with `DBGVIEW = *SOURCE` option

\_\_\_ 14. As you can see from Figure 56, you are no longer debugging C code but you are debugging SQL code.



\_\_\_ 15. Define a breakpoint in statement 10. Place the cursor anywhere in statement 10 and press F6. Then press F3.

\_\_\_ 16. Type the `STRSQL` command to invoke interactive SQL.

\_\_\_ 17. Type the following command:

```
CALL SAMPLEDEXX/TOPCOMPAID (10,0)
```

\_\_\_ 18. The stored procedure should stop its execution on instruction 10. Once it does, type the following command:

```
EVAL PROC.SQLS1 :X
```

This is good news! We can now debug SQL code instead of the C-generated code.

You can apply the method you just learned to debugging SQL triggers and functions.

You have now completed this lab!



---

## Lab 9. User-defined function

User-defined function is a DB2 UDB feature that helps you extend the existing set of SQL functions in your system with your own customized functions that provide additional flexibility or common business-specific functions through SQL interface. All UDFs created by an SQL programmer can be used by all SQL users in the same system.

UDF also provides additional functions for any User-defined Data Type (UDT) created in the system. Without UDF, UDT would be of limited benefit to SQL programmers.

Once created, a UDF can run in the database engine (as opposed to SQL codes at application level). This can potentially provide improved performance in some cases such as row selection processing and dealing with large objects (LOB).

Since V4R4, you can only create a Scalar UDF that returns only a single value to the calling SQL program. In V5R2, we provide you with a support for User-defined Table Function (UDTF) that returns rows of result. You must only use SELECT with the TABLE function to see the result from a UDTF.

### Objectives

This lab teaches you how to create a Scalar SQL UDF and an SQL UDTF.

### Lab prerequisites

Before you begin this lab, be sure you meet the following prerequisites:

- You must have completed *Lab 1, "iSeries Navigator setup and basic operations" on page 3.*
- You must have completed Task 1 of *Lab 2, "SQL Script Center" on page 11,* because the SAMPLEDBXX schema and SQL Script Center are used in this lab.
- You need to install an OS/400 component named "System Openness Includes" (V5R2 option number 13) in your machine. It contains include files needed when SQL UDFs are created.

### Time required

The time required to efficiently complete this lab project is 15 minutes.

### Naming convention for a 'schema'

Starting in V5R1, the term "schema" is used in the same sense as the term "collection". This is an OS/400 library created with automatic DB journaling enabled and local DB catalog views.

---

## Task 1: Creating and using a scalar SQL UDF

In this task, let's suppose we want to create a scalar UDF that can simplify the following statement:

```
SELECT ID, NAME, DEPTNAME, JOB, SALARY FROM STAFF, ORG WHERE DEPT =  
DEPTNUMB;
```

The DEPTNAME and DEPTNUMB columns are in ORG table, while the rest are in STAFF table. It would be nice if we could select only from the STAFF table. So we are going to create a UDF that receives DEPT (Dept. No.) from STAFF and use it to select a corresponding DEPTNAME from ORG for us.

This helps to simplify your SQL query to STAFF table by not having to explicitly refer to ORG table in the query.

Here's how you do this:

- \_\_\_ 1. In the left panel of the main iSeries Navigator window, expand **I400WS-> Databases-> Libraries** and click your **SAMPLEDBXX** library.  
Your SAMPLEDBXX library should appear in the active library list under the Libraries icon (XX is your team number). If it does not, add it by right-clicking **Libraries** and selecting **Select Libraries to Display**. Refer to Task 4, "Maintaining your active library list for iSeries Navigator" on page 7, in Lab 1, for instructions on how to do this.
- \_\_\_ 2. Right-click the **SAMPLEDBXX** icon and click **New-> Function-> SQL** from the pop-up menu. The New SQL Function in SAMPLEDBXX window appears.
- \_\_\_ 3. In the **General** tab (Figure 57), type the name `DNAME` in the Function input field. In the Description input field, type `RETURN DEPT NAME OF DEPT NO.` This is the name of the SQL UDF that you are going to create and use.  
In the Data returned to invoking statement box, select the **Single value** radio button. This means you create a scalar SQL UDF.  
Select **CHARACTER** from the Type drop-down list and specify a Length of 25. This is the same data type as the DEPTNAME column of the ORG table that this UDF will read from and return it to the calling statement.



Figure 57. Creating a scalar SQL UDF: General tab

At the bottom part of the tab, select the following check boxes:

- Program does not call outside of itself (No External Action)
- Attempt to run in same thread as invoking statement (Not Fenced)

Leave the rest at their defaults.

- \_\_\_ 4. Click the **Parameters** tab (Figure 58) and then click the **Insert** button. A default entry is now added into the parameter list box.

Double-click **New\_Parameter** entry to highlight it and then change it to DEPTNO.

Press the Tab key to move to the next column and select **SMALLINT** from the Type drop-down list. This is the same data type as the DEPTNO column of the STAFF table from which this UDF will receive its input value.

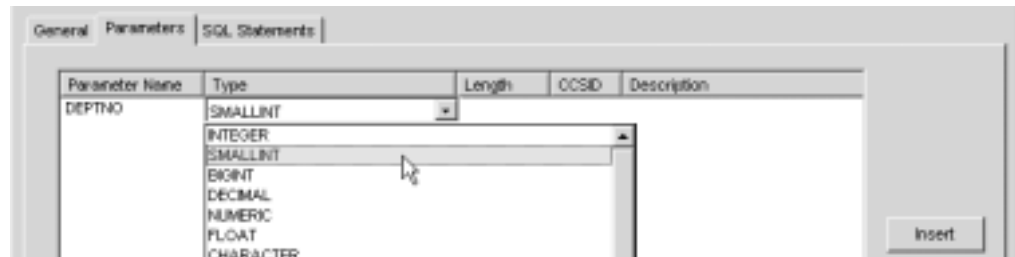


Figure 58. Creating a scalar SQL UDF: Parameters tab

5. Click the **SQL Statements** tab (Figure 59) and enter the following SQL statements:

**Note**

The text indentation format shown here is only for readability. You do not have to follow this exact format. Use care in typing the semicolon (;) as indicated only. Also type your team number for SAMPLEDBXX.

```

BEGIN
DECLARE X CHAR( 25 ) ;
SELECT DEPTNAME INTO X
    FROM SAMPLEDBXX.ORG WHERE DEPTNUMB = DEPTNO;
RETURN X;
END

```

You can see that the UDF uses the DEPTNO value (to be received from DEPT column in STAFF table) to select a corresponding DEPTNAME from the ORG table and put it into a variable X. For each DEPTNAME value read, it RETURNS this result to the calling statement via the variable X.

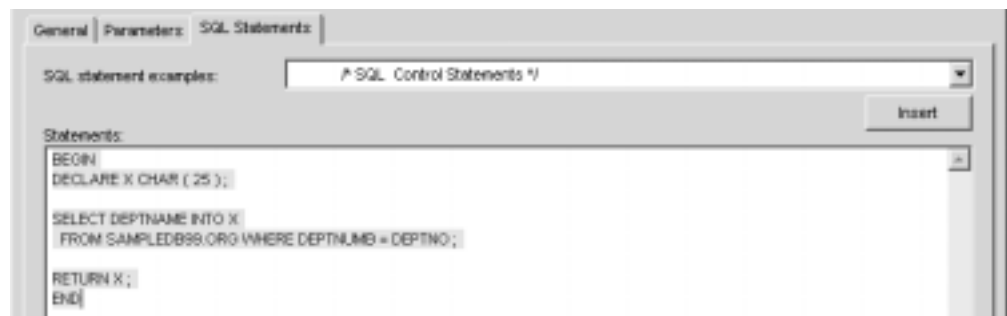


Figure 59. Creating a scalar SQL UDF: SQL Statements tab

6. Click **OK** to create the SQL UDF, DNAME.

You are now back to iSeries Navigator window. Notice the message “Function SAMPLEDB99.DNAME created successfully.” in the status bar located at the bottom-most part of the iSeries Navigator window.

If there is any failure at this point, you should review what you have done so far.

You should also see an icon for the DNAME function added on the right panel of the object list (Figure 60).

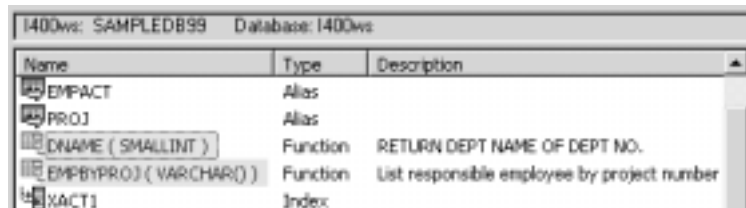


Figure 60. Creating a scalar SQL UDF: The newly created DNAME UDF

\_\_\_ 7. To test the UDF, open the SQL Script Center by right-clicking the **I400WS** icon (right under Databases) in the left panel and selecting **Run SQL Scripts**.

\_\_\_ 8. Type the following statement and run it:

```
SELECT ID, NAME, DNAME(DEPT) AS DEPTNAME, JOB, SALARY FROM STAFF;
```

You should see the result as shown in Figure 61.



Figure 61. Result of the UDF: DNAME and its equivalent normal statement

Running the previous statement with UDF is equivalent to running the following normal statement, where the ORG table and its DEPTNAME and DEPTNUMB columns must be explicitly specified:

```
SELECT ID, NAME, DEPTNAME, JOB, SALARY FROM STAFF, ORG WHERE DEPT = DEPTNUMB ORDER BY ID;
```

Close SQL Script Center when you are finished.

Now you see how an SQL scalar UDF is created and used. Let's move on to the new V5R2 UDTF topic.

## Task 2: Creating and using an SQL UDTF

In this lab, we work on the same statement as in Task 4, "Using basic Visual Explain" on page 18, from Lab 2. "SQL Script Center".

```
SELECT * FROM EMPLOYEE WHERE EMPNO IN ( SELECT EMPNO FROM EMPPROJECT
WHERE PROJNO = '??????' );
```

This statement identifies all employees who are involved in a particular project (identified by the project number). Now let's suppose that this statement is so frequently used that we want to standardize it into a UDTF for more convenient access.

Let's begin the task:

- \_\_\_ 1. Right-click the SAMPLEDBXX icon and click **New-> Function-> SQL** from the pop-up menu. The New SQL Function in SAMPLEDBXX window appears.
- \_\_\_ 2. In the **General** tab (Figure 57), type the name `EMPBYPROJ` in the Function input field. In the Description input field, type `EMPLOYEE LIST BY PROJECT NO.` This is the name of the SQL UDTF that you are going to create and use. In the box, Data returned to invoking statement, select the **Table** radio button. This means you create an SQL UDTF.
- \_\_\_ 3. Click **Insert** button to add the four columns from Table 11 to be returned by the `EMPBYPROJ` UDTF. (Do *not* press Enter key between each column; simply click **Insert** to add each column.)

Table 11. Defining the returned columns of the UDTF: `EMPBYPROJ`

| Column Name | Type      | Length |
|-------------|-----------|--------|
| EMPNO       | CHARACTER | 6      |
| FIRSTNME    | CHARACTER | 20     |
| LASTNAME    | CHARACTER | 20     |
| BIRTHDATE   | DATE      | -      |

### Note

The data Type is selected from the drop-down list.

At the bottom part of the tab, select the following check boxes:

- Program does not call outside of itself (No External Action)
- Attempt to run in same thread as invoking statement (Not Fenced)

Leave the rest at their defaults.

- \_\_\_ 4. Click the **Parameters** tab and the click the **Insert** button. A default entry is now added into the parameter list box.

Double-click the **New\_Parameter** entry to highlight it and then change it to `PROJNBR`.

Press the Tab key to move to the next column and select **VARCHAR** from the Type drop-down list. Then, specify a Length of 6. This is the project

number to be entered in the SQL statement as the input value for the UDTF.

- \_\_\_ 5. Click the **SQL Statements** tab and enter the following SQL statements:

**Note**

The text indentation format shown here is only for readability. You do not have to follow this exact format. Use care in typing the semicolon (;) as indicated only. Make sure you type your team number for SAMPLEDBXX.

```
BEGIN
RETURN
  SELECT EMPNO, FIRSTNME, LASTNAME, BIRTHDATE
  FROM SAMPLEDBXX.EMPLOYEE WHERE EMPNO IN (
  SELECT EMPNO FROM SAMPLEDBXX.EMPPROJECT
  WHERE PROJNO = PROJNBR ) ;
END
```

You can see here that the input parameter PROJNBR is used first, in the subselect of EMPPROJECT table, to identify EMPNO of all the employees who work in the project (identified by its number). Then the result of the subselect is used by the main query to extract the desired columns of EMPLOYEE table. Then it is passed back as a set of rows to the calling SQL statement.

- \_\_\_ 6. Click **OK** to create the SQL UDTF, EMPBYPROJ.

You are now back at the iSeries Navigator window. Notice the message "Function SAMPLEDB99.EMPBYPROJ created successfully." in the status bar located at the bottom-most part of the window.

If there is any failure at this point, you should review what you have done so far.

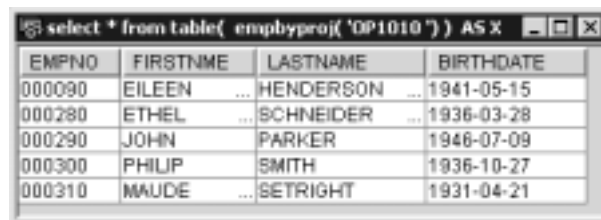
You should also see the icon of the EMPBYPROJ function added to the right panel of the object list (Figure 60).

- \_\_\_ 7. To test the UDTF, open the SQL Script Center by right-clicking the **I400WS** icon (right under Databases) in the left panel and selecting **Run SQL Scripts**.

- \_\_\_ 8. Type the following statement and run it:

```
SELECT * FROM TABLE(EMPBYPROJ('OP1010')) AS X;
```

You can see now that the statement is simpler than the original one. You should see the results as shown in Figure 62.



| EMPNO  | FIRSTNME | LASTNAME  | BIRTHDATE  |
|--------|----------|-----------|------------|
| 000090 | EILEEN   | HENDERSON | 1941-05-15 |
| 000280 | ETHEL    | SCHNEIDER | 1936-03-28 |
| 000290 | JOHN     | PARKER    | 1946-07-09 |
| 000300 | PHILIP   | SMITH     | 1936-10-27 |
| 000310 | MAUDE    | SETRIGHT  | 1931-04-21 |

Figure 62. Result of the UDTF: EMPBYPROJ



### Invoking a UDTF

You invoke a UDTF with the following TABLE built-in function syntax:

```
TABLE ( UDTF_name ( expression) ) correlation clause
```

The (expression) follows the same convention as in scalar or column function. This syntax is most frequently used in the FROM clause (as in the lab exercise).

The Correlation clause takes a form of:

```
AS correlation_name (column)
```

Here AS and (column) are *optional*. This represents an intermediate result table built from the TABLE function.

Close the SQL Script Center when you are finished.

Now you should understand how an SQL UDTF is created and used.

You have completed this lab!

