

2

SQL in iSeries Navigator

In V4R4, IBM added an SQL scripting tool to the standard features included within iSeries Navigator and has continued enhancing it in subsequent releases. Because standard features of iSeries Navigator are available to customers at no charge, this new feature marks a new era in SQL development on the iSeries. For the first time, users can get a mainstream tool to perform interactive SQLs without having to buy additional products. The SQL engine has long been included within the DB2 database that ships with every iSeries, but, the ability to send instructions to it required the installation of additional products. The GUI scripting tool shown in Figure 2.1 can be found within the database component of iSeries Navigator. In addition to providing a free alternative to the STRSQL tool outlined in Chapter 1, this tool provides a much more comfortable environment for developers new to the iSeries. Avoiding the native 5250 interface and having the ability to perform commonplace actions such as cut-and-paste goes a long way towards putting new developers at ease.

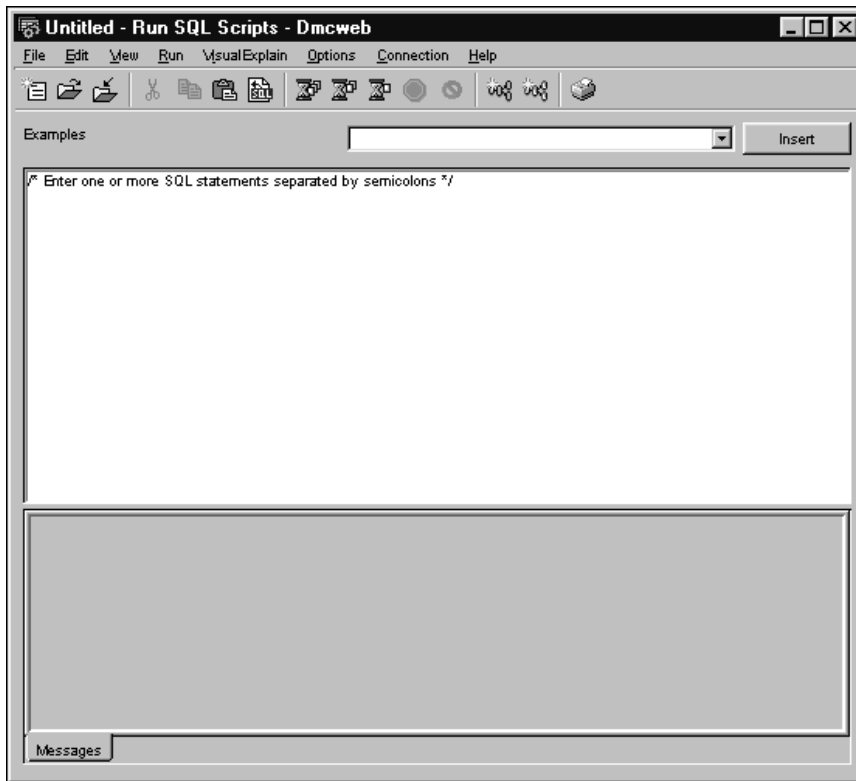


Figure 2.1: Launching the Run SQL Scripts tool.

Installing the Database Component of iSeries Navigator

This component can be added, if it's not already installed. To install the Database component and the SQL scripting tool, open iSeries Navigator and click on **Selective Setup** within the **File** pull-down menu. This launches the Selective setup wizard (Figure 2.2) that controls which features of iSeries navigator are installed on each PC. Before the setup tool itself is launched, this small prompt window is displayed. Select which iSeries server to use. If more than one is present in your network, all servers will be listed in the drop-down box. Generally, I suggest you use the server that is at the highest release level of OS/400. Some compatibility issues may exist if

multiple iSeries servers on your network are running at different levels of OS/400. Some features of iSeries Navigator may work with one server and not others. To eliminate these compatibility problems, however, IBM often provides PTFs for older versions of OS/400. After selecting the appropriate server, you will be prompted to log into the server. The install program then runs: It determines which components are already installed on your PC and which other components are available for installation (Figure 2.3). If the Database component of iSeries Navigator (also referred to as AS/400 Operations Navigator), is not selected, click the check box to select it and click **Next**. The wizard then completes the installation process. Before using the new component, be sure to apply any available patches. IBM provides fixes to the iSeries Navigator application on a regular basis and free of charge. If you want to download patches to iSeries Navigator, see the web site www-1.ibm.com/servers/eserver/series/access/casp.htm.

Now that the database is installed and the latest patches have been applied, the SQL scripting tool can be put to use.

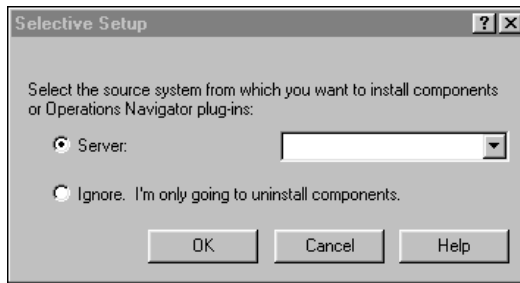


Figure 2.2: Selective Setup tool.

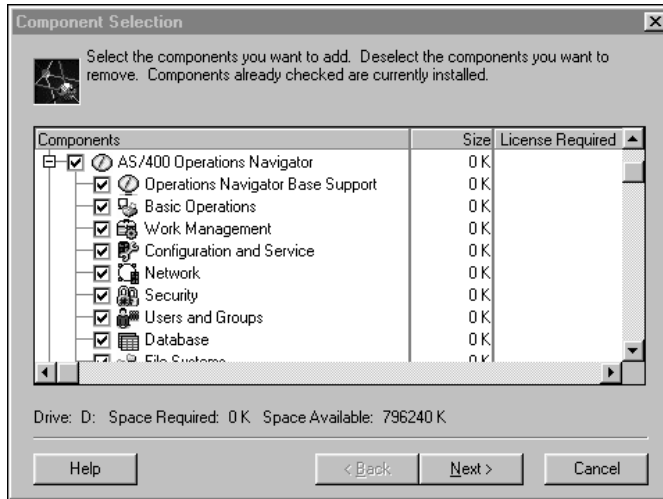


Figure 2.3: Launching the Run SQL Scripts tool.

Getting Started with the SQL Scripting Tool

Once the iSeries Navigator's Database component and its scripting tool have been installed on your PC, you can easily use SQL to interact with the DB2 database on your iSeries. Figure 2.4 demonstrates how to launch the scripting tool. Open the iSeries Navigator and right click on **Database** beneath the server you wish to work with. This opens a list of actions to perform against that server. Click on **Run SQL Scripts...** to launch the GUI SQL scripting tool (Figure 2.1).

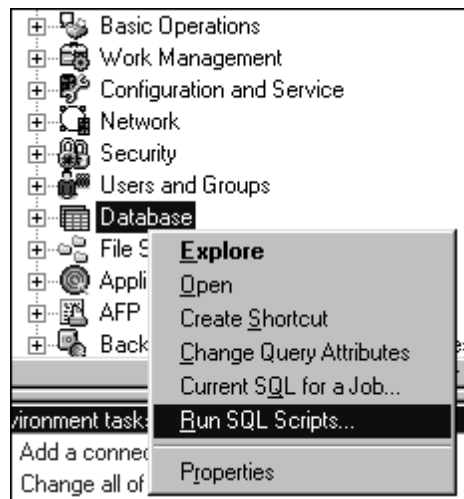


Figure 2.4: Launching the Run SQL Scripts tool.

The primary feature of the scripting tool is a large text box. By default, this text box contains the comment `/* Enter one or more SQL statements separated by semicolons */`. The comment does not have to be deleted, but I find that it often confuses students and interferes with writing code. So, for the sake of avoiding future confusion, delete that comment. The text box should now be completely empty and ready for you to enter your first SQL statement. But before running an SQL statement, take a minute to configure the environment.

Configuring the SQL Script Environment

Click on the **Options** pull-down menu and select those options shown in Figure 2.5. Five options are selected:

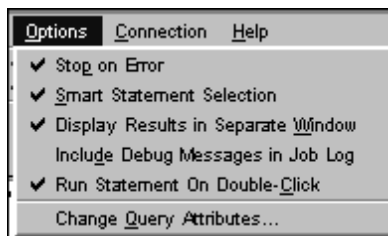


Figure 2.5: Options pull-down menu.

1. **Stop on Error**—Controls the behavior of the scripting tool when multiple SQL statements are being processed in order. If any SQL statement has an error, the processing of the remaining SQL statements is aborted. If this option is not selected, each SQL statement is evaluated independently, and all valid statements are processed.
2. **Smart Statement Selection**—When this option is activated, each time an SQL statement is executed, the entire statement is executed, rather than just the selected portion of the statement.
3. **Display Results in Separate Window**—Each time an SQL statement runs and displays a result table, that result table is displayed as a separate window on the desktop. If this option is not activated, the results are displayed in a separate tab at the bottom of this window.
4. **Include Debug Messages in Job Log**—Selecting this option causes any diagnostic errors that occur to display in the job log for this session. The

job log can be reviewed by selecting **Job Log...** within the **View** pull-down menu.

5. **Run Statement on Double-Click**—This option allows the execution of SQL statements simply by double clicking on them. Semicolons must be used to mark the end of each SQL statement.

After the options have been set, you are ready to execute SQL statements. For a list of example statements that show the basic syntax of a large number of SQL statements, open the Examples drop-down box. To insert one of the example statements into the text box, click to select it, and then click the Insert button. The selected example inserts into the textbox at the current cursor location.

If you already know which SQL statement you wish to run, simply type it in. For example, type the following statement and run it by double clicking:

```
SELECT * FROM KPFSQL/CUST;
```

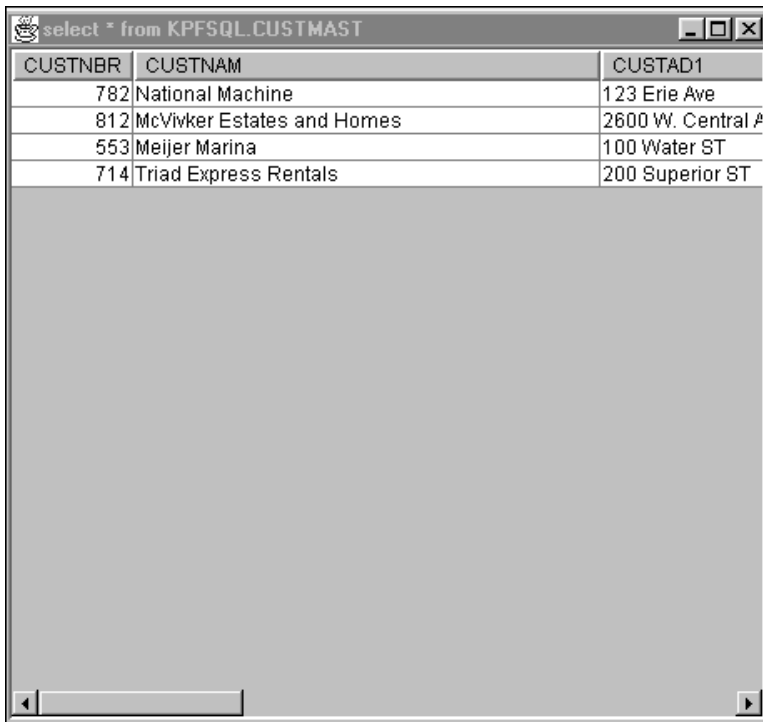
The following error should be displayed at the bottom of the window...

```
> select * from KPFSQL/CUSTMAST;
[SQL5016] Qualified object name CUSTMAST not valid.
Cause . . . . . : One of the following has occurred: - The syntax
used for the qualified object name is not valid for the naming
option specified. With system naming, the qualified form of an
object name is collection-name/object-name. With SQL naming the
qualified form of an object name is authorization-name.
object-name. - The syntax used for the qualified object name is
not allowed. User-defined types cannot be qualified with the
library in the system naming convention on parameters and SQL
variables of an SQL procedure or function. Recovery . . . : Do
one of the following and try the request again: - If you want to
use the SQL naming convention, verify the SQL naming option in
the appropriate SQL command and qualify the object names in the
form authorization-id.object-name. - If you want to use the system
naming convention, specify the system naming option in the
appropriate SQL command and qualify the object names in the form
collection-name/object-name. - With the system naming convention,
ensure the user-defined types specified for parameters and
variables in an SQL routine can be found in the current path.
Processing ended because the highlighted statement did not
complete successfully
```

This error occurred because “/” is not a valid character for use in qualifying a file name. To identify which library a file resides in, use the period (.) instead. This SQL standard is followed on most platforms; the iSeries may be the only system that uses the “/” (back-slash) character. Change the SQL statement as below and double click on it again:

```
SELECT * FROM KPFSQL.CUSTMAST;
```

Figure 2.6 shows the results displayed in a separate window. If the results are larger than the space provided in the window, the window can be resized; scroll bars are provided to display different portions of the result table.



CUSTNBR	CUSTNAM	CUSTAD1
782	National Machine	123 Erie Ave
812	McVivker Estates and Homes	2600 W. Central A
553	Meijer Marina	100 Water ST
714	Triad Express Rentals	200 Superior ST

Figure 2.6: Result window.

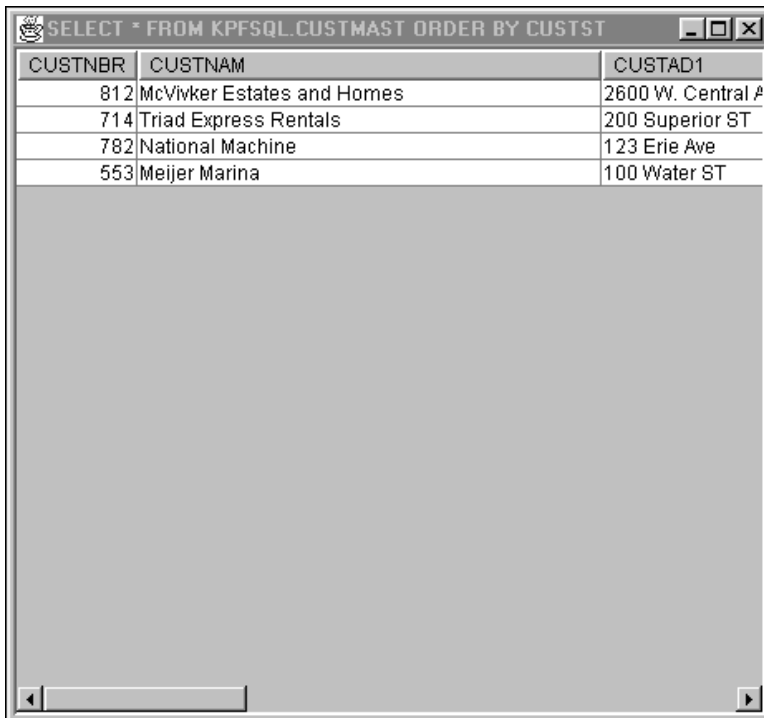
Notice that the error message from the first failed SQL statement is still listed at the bottom of the SQL Script window. To erase the old messages, click on **Clear**

Run History in the **Edit** pull-down menu. (The **Edit** pull-down menu also contains the option **Clear Results**. This option is not used in this book. If we had not selected the option to display results in a separate window, they would be displayed at the bottom of the text box, similarly to the error messages. Select **Clear Results** to erase previous result sets from the bottom of the text box.)

Now that the previous errors have been cleaned up, let's look at sorting the data. To sort the data, we'll add an **ORDER BY** clause to the SQL statement. Enter the statement as shown below and double click on it:

```
SELECT * FROM KPFSQL.CUSTMAST ORDER BY CUSTST;
```

The customers are displayed in alphabetical order by their states in a result window as shown in Figure 2.7.



The screenshot shows a window titled "SELECT * FROM KPFSQL.CUSTMAST ORDER BY CUSTST". The window contains a table with three columns: CUSTNBR, CUSTNAM, and CUSTAD1. The data is sorted by state, with the following rows:

CUSTNBR	CUSTNAM	CUSTAD1
812	McVivker Estates and Homes	2600 W. Central #
714	Triad Express Rentals	200 Superior ST
782	National Machine	123 Erie Ave
553	Meijer Marina	100 Water ST

Figure 2.7: Results sorted by state.

Some SQL statements may require the SQL engine to create temporary access paths to perform certain sorting and selecting logic. In some cases, performance is improved if a permanent access path is built. To determine if the SQL engine is building a temporary access path for this SQL statement click on **Job Log...** in the **View** pull-down menu. The job log shown in Figure 2.8 is displayed.

Message ID	Message	Date sent	Time sent
SQL7963	4 rows fetched from cursor CRSR0017.	08/24/03	14:06:45
SQL7968	DESCRIBE of prepared statement STMT0017 completed.	08/24/03	14:06:45
SQL7962	Cursor CRSR0017 opened.	08/24/03	14:06:45
SQL7916	Blocking used for query.	08/24/03	14:06:45
SQL7912	ODP created.	08/24/03	14:06:45
CPI434B	**** Ending debug message for query .	08/24/03	14:06:45
CPI4321	Access path built for file CUSTMAST.	08/24/03	14:06:45
CPI434A	**** Starting optimizer debug message for query .	08/24/03	14:06:45
CPI433A	Unable to retrieve query options file.	08/24/03	14:06:45
SQL7968	DESCRIBE of prepared statement STMT0017 completed.	08/24/03	14:06:45
SQL7967	PREPARE of statement STMT0017 completed.	08/24/03	14:06:45
SQL7968	DESCRIBE of prepared statement STMT0017 completed.	08/24/03	14:06:45
CPI433A	Unable to retrieve query options file.	08/24/03	14:06:45
SQL7963	2 rows fetched from cursor CRSR0016.	08/24/03	14:05:12
SQL7968	DESCRIBE of prepared statement STMT0016 completed.	08/24/03	14:05:12

Figure 2.8: Viewing the job log.

Any SQL statements that generates the message “Access path built for file....” is a candidate for this performance improvement task. Before building the index, consider how often the SQL statement is executed and how much overhead the access path will add to the database. Sometimes it is better to let the system generate a temporary index for infrequently run SQL statements rather than create an index that will require constant maintenance by the database engine. This topic is reviewed in more detail in Chapter 5.


Running Multiple SQL Scripts

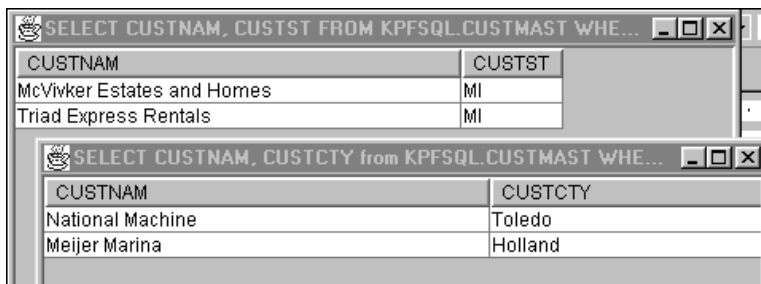
The SQL scripting tool supports the ability to run multiple SQL statements consecutively. Type the following SQL statements and click on the **Run All** icon:

```

SELECT CUSTNAM, CUSTST FROM KPFSQL.CUSTMAST
WHERE CUSTST <> 'OH';
SELECT CUSTNAM, CUSTCTY from KPFSQL.CUSTMAST
WHERE CUSTST = 'OH';

```

Clicking on the **Run All** icon  causes the execution of all SQL statements in the text box. Figure 2.9 shows the results of each SELECT displayed in separate windows.



CUSTNAM	CUSTST
McVivker Estates and Homes	MI
Triad Express Rentals	MI

CUSTNAM	CUSTCTY
National Machine	Toledo
Meijer Marina	Holland

Figure 2.9: Results from Run All.


By default, the windows are displayed directly on top of one another. To view them simultaneously, they must be moved and possibly resized.

As you continue to create and execute SQL statements, at some point you may wish to run many of the SQL statements in the script, but not all. IBM provides the ability to begin execution at a specified point, ignoring all SQL statements above that point in the SQL script. For example, write the following code, then click the cursor on the second SQL statement; then click the **Run from Selected** icon to execute all SQL statements from that point on:

```

SELECT * FROM KPFSQL.CUSTMAST;
SELECT CUSTNAM FROM KPFSQL.CUSTMAST WHERE CUSTST = 'OH';
SELECT CUSTNAM FROM KPFSQL.CUSTMAST WHERE CUSTST <> 'OH' AND
CUSTCTY = 'MONROE';

```

Clicking on the **Run from Selected** icon  causes the execution of all SQL statements in the text box starting with the one on which the cursor is located. Figure 2.10 shows the results of each SELECT displayed in separate windows.

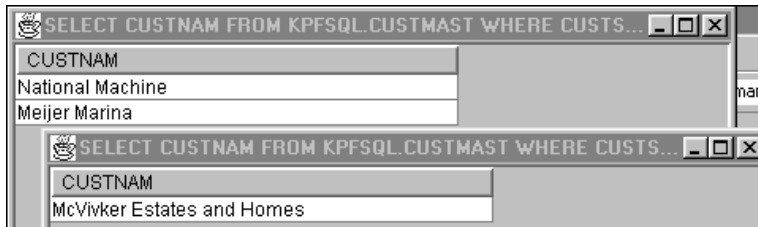



Figure 2.10: Results from Run from Selected.

To force only one SQL statement to execute, either click the **Run Selected** icon.  after placing the cursor on the statement to be executed or double-click on the statement to execute (if the **Run on double-click** option is activated).

Saving and Loading SQL Scripts

After coding a number of SQL statements, or perhaps coding some particularly complex SQL statements, you may want to save them for use at a later time. Unlike the STRSQL tool in the native environment, this STRSQL tool does not automatically remember your previous SQL statements. You must save them manually. To save the three SQL statements written above, click on the **Save As** option within the **File** pull-down menu. The **Save As** dialog box shown in Figure 2.11 is displayed.

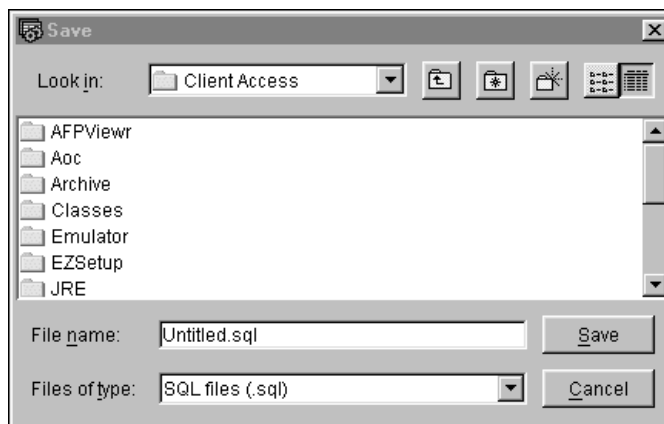


Figure 2.11: Save dialog box.

Select the desired folder and file name. The file suffix defaults to .sql and should not be changed unless absolutely necessary. If the SQL script files are saved into a network folder or a folder on the iSeries Integrated File System (IFS), it can be shared with other programmers or users on the iSeries. If it is stored locally on your PC, only a user at your workstation can use it.

The history log maintained by the STRSQL tool is easy to use because it is automatic. It's harder to search, however, unless various important SQL statements are saved into well-named files. By well-named, I mean that the names need to intuitively reflect the SQL statements stored within them. SQL files named TEST1.SQL, TEST2.SQL, and TEST3.SQL, don't identify their contents. Had they been named PartCost.SQL, TotalSales.SQL, and ProductionSchedule.SQL, the nature of their contents would be clearer, and users would spend less time hunting for the right SQL statement.

Once the script file has been saved, it can be recalled by selecting the **Open...** option within the **File** pull-down menu. The **Open** dialog box shown in Figure 2.12 is displayed.

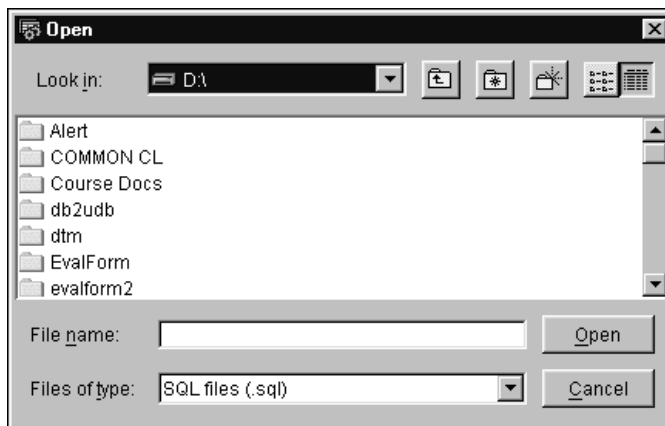


Figure 2.12: Open dialog box.

Select the desired folder and file, then click on **Open**. All SQL statements saved within the selected SQL script file are loaded into the text box. From there, they can be executed as a group or individually as needed.

JDBC Setup

The SQL scripting tool has the ability to modify the user's standard library list to include additional libraries. In all the previous examples using the scripting tools, the library name is hard coded. To avoid hard coding the library name on every statement, the library can be added to the list of libraries used by the JDBC connection to the iSeries. The scripting tool is written in Java and connects to the iSeries DB2 database through a JDBC driver. Configure the JDBC driver through the JDBC Setup Window, shown in Figure 2.13. Open the window by selecting **JDBC Setup...** within the **Connection** pull-down menu.

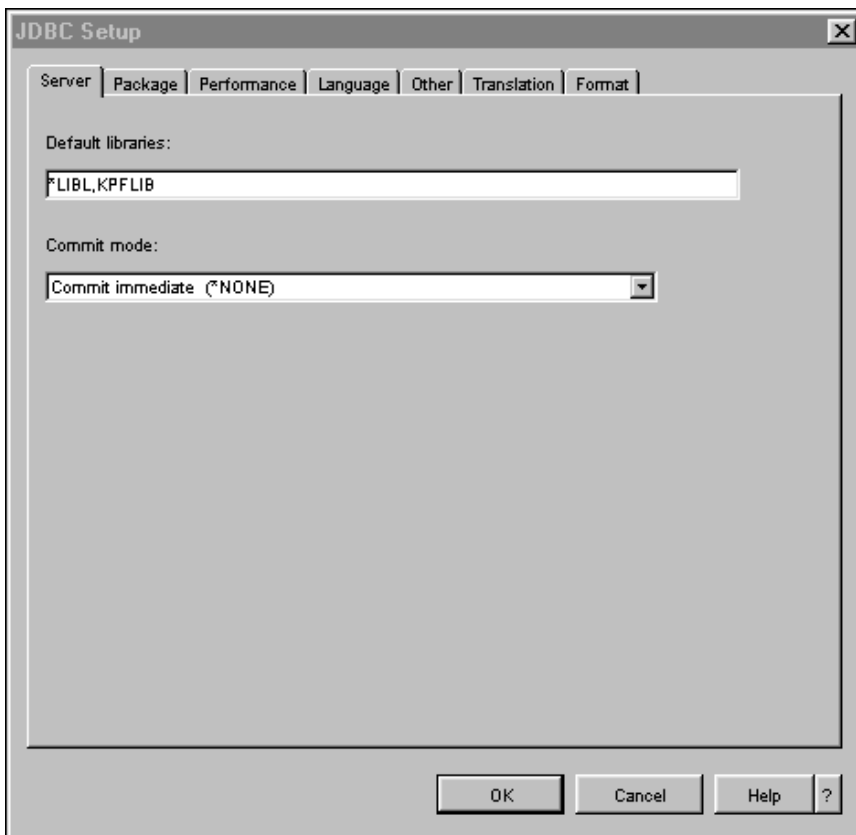


Figure 2.13: JDBC Setup Window.

The JDBC Setup wizard contains a number of different panels, each of which can be displayed by selecting the corresponding tab at the top of the window. By default, the Server tab is displayed. It controls the library list being used when accessing data from the iSeries and the level of commitment control to be used. If you are unfamiliar with library lists, they function similarly to Path statements on a PC. They define the list of libraries to search for objects that are referenced, without requiring the user or programmer to indicate where they are to be found. The commitment control logic is generally only useful when updating the database, and it requires that the files being updated already be journaled. (See the command STRJRNPF in the CL Reference manual for more information.) If commitment control is activated, then file updates can be made temporarily until a COMMIT statement locks them into the database or the ROLLBACK statement removes them.

The default search libraries are listed and can be modified. To eliminate the need to specify the KPFSQL library on all the searches, add it to the end of the library list, separated by a comma:

```
*libl, kpflib, kpfsql
```

The *libl indicates that the user's standard library list should be used, and the libraries KPFLIB and KPFSQL will be added to the end of the list. It does not matter if the library names are coded in upper- or lowercase.

If you perform file updates within your SQL statements over files that are journaled, you can activate commitment control. By setting the Commit mode to *CHG, all changes are held as temporary changes to the database until a COMMIT statement is issued. In addition to holding the changes as temporary, the affected records will be locked until the COMMIT is issued. If a ROLLBACK is issued in place of the COMMIT, then all changes made since the last COMMIT will be removed from the database. (Transaction processing and commitment control are complex topics, beyond the scope of what we can address here. See the SQL Reference manual for more information on Commitment Control.)

What if we wish to prevent the update of the database? The JDBC setup wizard can limit the types of SQL statements that may be performed. To limit the SQL

Scripting tools ability to perform updates against the database, change the Access Type. To set the Access Type, click on the **Other** tab within the JDBC Setup wizard, as shown in Figure 2.14.

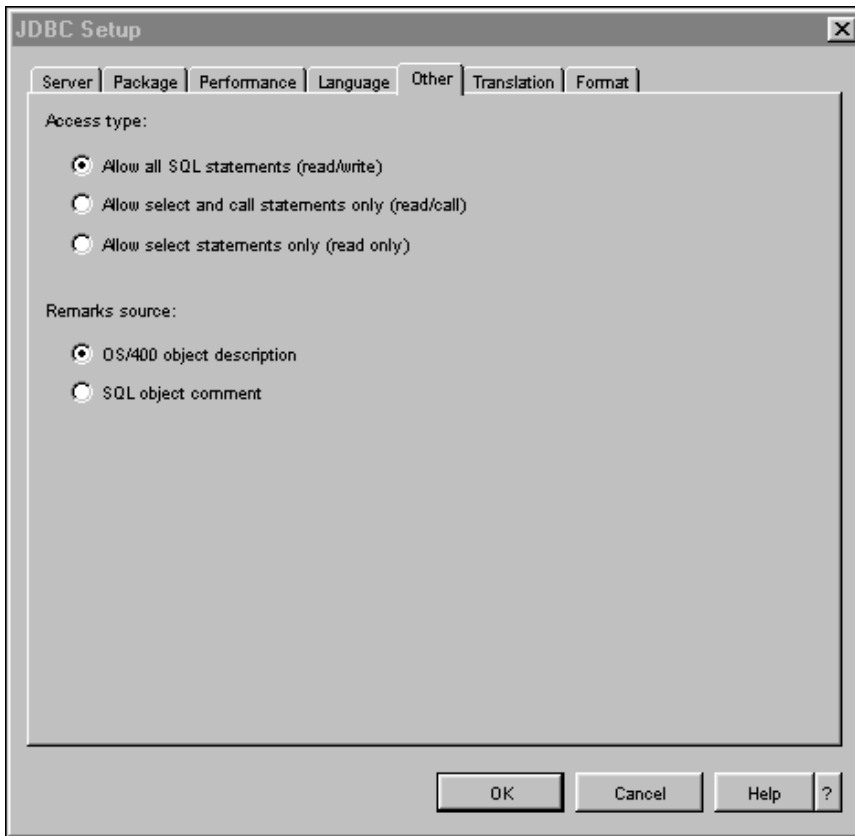


Figure 2.14: Setting the JDBC access type.

To restrict users to read-only access to the database, configure their JDBC connection as read only. If read/call is configured, they will have the authority to call SQL stored procedures on the iSeries. Note that virtually all high-level language programs such as RPG, CL, and COBOL can be called as if they were stored procedures. In Chapter 7, stored procedures are discussed in more detail.

SQL does not easily pull data from program-described or flat files. For example, consider the following SQL statement:

```
SELECT * FROM INVHFLAT;
```

The results shown in Figure 2.15 return the hex translation of the data from the non-database file.

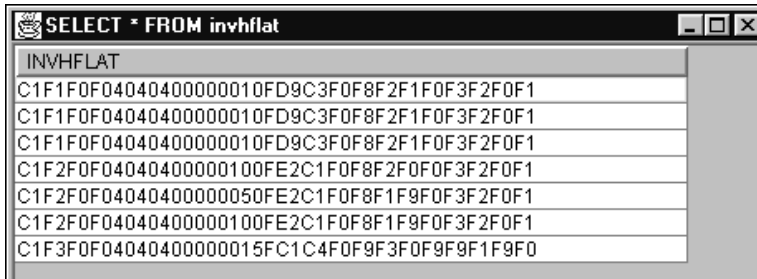


Figure 2.15: Data from a non-database file.

This result is caused by the fact that program-described files are stored using the Coded Character Set ID (CCSID) of 65535 rather than 37, as is used by externally defined database files. To improve the way that data is returned from a program-described file, use the JDBC CCSID Translation window shown in Figure 2.16.

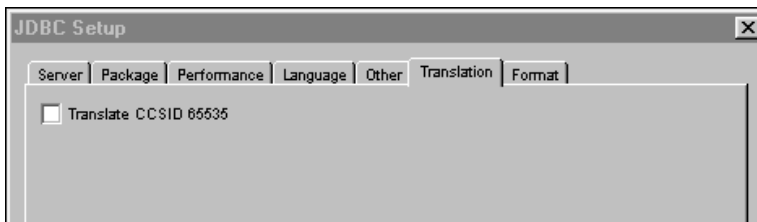
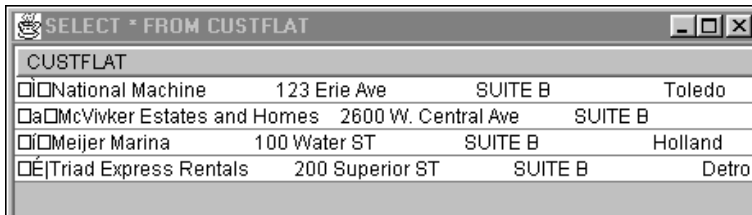


Figure 2.16: Translate the character set.

Open the window by clicking on the **Translation** tab within the JDBC Setup wizard. Select the option to perform the translation of CCSID 65535, and the data will be returned in a much more readable format. Rerun the statement:

```
SELECT * FROM CUSTFLAT;
```



CUSTFLAT			
□□National Machine	123 Erie Ave	SUITE B	Toledo
□a□McVivker Estates and Homes	2600 W. Central Ave	SUITE B	Holland
□i□Meijer Marina	100 Water ST	SUITE B	Holland
□ÉTriad Express Rentals	200 Superior ST	SUITE B	Detro

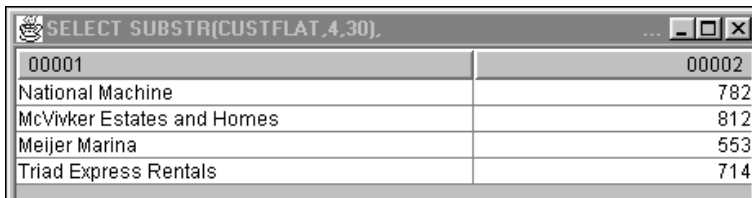
Figure 2.17: Displaying the translated data.

The result in Figure 2.17 now returns character data that is much more readable, although the packed numbers within the record are still difficult to read. Using the techniques explained in the previous chapter, however, the packed data can still be displayed in a clear and readable fashion. For example:

```

SELECT SUBSTR(CUSTFLAT, 4, 30),
DECIMAL(SUBSTR(HEX(SUBSTR(CUSTFLAT, 1, 3)), 1, 5), 5, 0)
* CASE
  WHEN SUBSTR(HEX(SUBSTR(CUSTFLAT, 3, 1)), 2, 1) = 'F'
  THEN 1
  ELSE -1
END
FROM CUSTFLAT;

```



Customer Name	Customer Number
00001	00002
National Machine	782
McVivker Estates and Homes	812
Meijer Marina	553
Triad Express Rentals	714

Figure 2.18: Displaying packed numbers as decimal.

Figure 2.18 displays both the customer name and number in a readable format. If the translation of CCSID 65535 is not selected, however, the customer name will not display correctly, although the packed number is displayed correctly either way. The HEX function used to format the Customer Number column forces the data to be displayed correctly whether the character set is translated or not. But the Customer Name column does not use the HEX function, so it requires the character set to be translated.

A number of other formatting issues can be controlled on the Format tab of the JDBC Setup wizard, as shown in Figure 2.19.

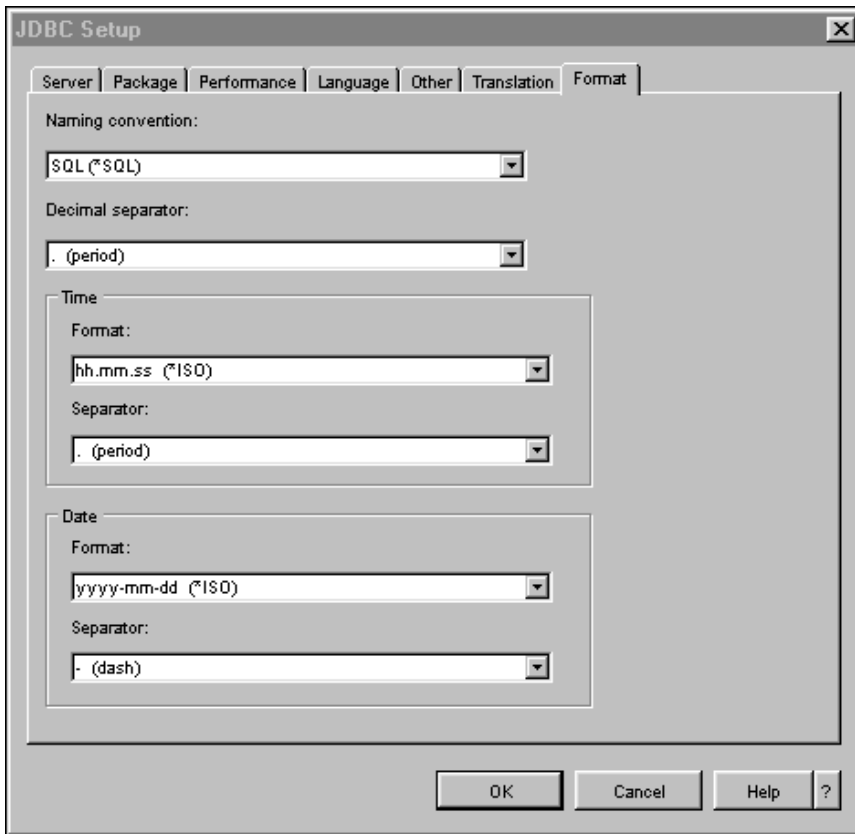


Figure 2.19: JDBC format options.

The most significant of these formatting options is the naming convention. By default, the SQL scripting tool uses the SQL naming convention that requires a decimal point (.) between library and file names. To change the naming convention to match that used in the native iSeries environment, set the naming convention to *SYS. Then the back-slash (/) used to qualify the file names in Chapter 1, will also be valid within the SQL scripting tool. Other formatting options, such as default date and time formats, also can be set here.

By default, the SQL Scripting tool connects to the server you selected when launching the tool (the database icon is associated with a particular server). If multiple servers exist within your network, the server you are connected to can be changed dynamically by clicking on **Connect to Server...** within the **Connection** pull-down menu. The Connect to Server dialog, shown in Figure 2.20, allows you to change the active server. When changing servers, a new user ID and password will be required before the connection to the new server is complete.

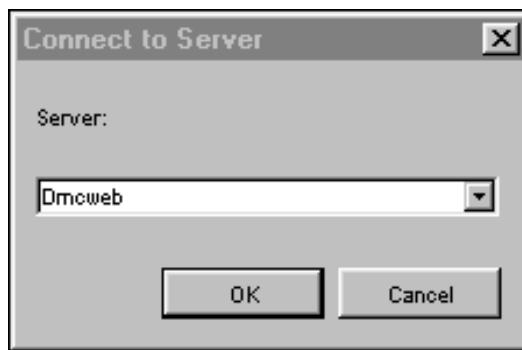
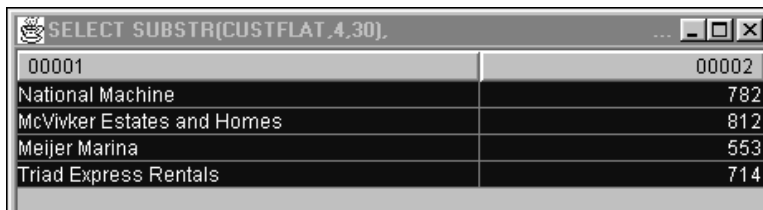


Figure 2.20: Choosing the server to which to connect.

Redirecting the Output

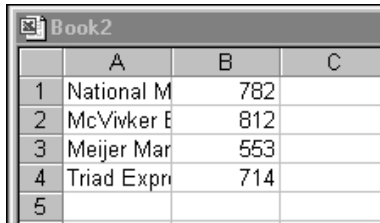
Unlike the interactive tool in the native environment, the GUI SQL Scripting tool does not provide specific tools for sending the output of SELECT statements anywhere except to the screen. Windows, however, does provide some of this capability. When displaying a result table, simply select (click and drag across) all the desired rows and columns of data, as shown in Figure 2.21.

A screenshot of a window titled 'SELECT SUBSTR(CUSTFLAT,4,30)'. The window displays a table with two columns. The first column contains text values, and the second column contains numerical values. The rows are: '00001' with '00002', 'National Machine' with '782', 'McVivker Estates and Homes' with '812', 'Meijer Marina' with '553', and 'Triad Express Rentals' with '714'. The entire table content is highlighted with a dark background, indicating it is selected for copying.

00001	00002
National Machine	782
McVivker Estates and Homes	812
Meijer Marina	553
Triad Express Rentals	714

Figure 2.21: Selecting data to copy.

Then press **Ctrl-C** to copy that data. Once it has been copied into the clipboard, it can be pasted into another object, such as an Excel spreadsheet, Word document, or email. Figure 2.22 shows the data inserted into an Excel spreadsheet.



	A	B	C
1	National M	782	
2	McVivker E	812	
3	Meijer Mar	553	
4	Triad Expre	714	
5			

Figure 2.22: Pasting the copied data.

Its easy to underestimate the importance of this feature. You can run SQLs interactively from the GUI editor until they work correctly and returned the desired results. Then, paste those results into applications such as an Excel spreadsheet. This is far more efficient than testing download after download until the selection criteria are correct, or running SQL after SQL in the native environment to create work files for download.

Using Visual Explain

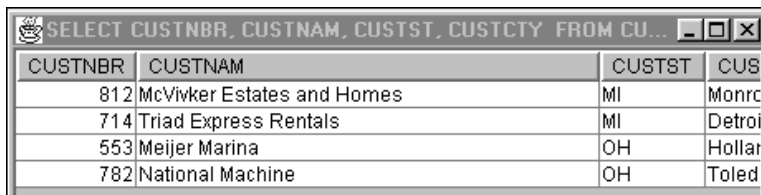
In addition to allowing the user to perform SQL statements, the GUI SQL Scripting tool now includes a powerful diagnostic tool, Visual Explain. This feature analyzes selected SQL statements and collects important statistical information about them. A careful review of this information allows database administrators to better optimize their applications. The Visual Explain tool is complex and requires a fair amount of resources from the PC on which it runs. This tool may overtax those PCs that only marginally meet the requirements for running iSeries navigator. (In our examples, during testing, the translation of CCSID 65535 interfered with the Visual Explain tool. IBM documents this problem in their Redpaper “Using AS/400 Database Monitor and Visual Explain.” The recommended solution is to make sure that the translation option is not selected before performing a Visual Explain. Temporarily turning off the translation is inconvenient, but not impractical.)

Launching Visual Explain

To launch Visual Explain, enter an SQL statement such as:

```
SELECT * FROM CUSTMAST ORDER BY CUSTST, CUSTNAM;
```

Place the cursor on the statement being run, and then select **Run and Explain** from the Visual Explain pull-down menu. The statement executes; Figure 2.23 shows the results displayed in a separate window, as in the previous examples.



CUSTNBR	CUSTNAM	CUSTST	CUS
812	McVivker Estates and Homes	MI	Monrc
714	Triad Express Rentals	MI	Detroit
553	Meijer Marina	OH	Hollat
782	National Machine	OH	Toled

Figure 2.23: Select to be explained.

The results of the Visual Explain are also displayed in a separate window, shown in Figure 2.24.

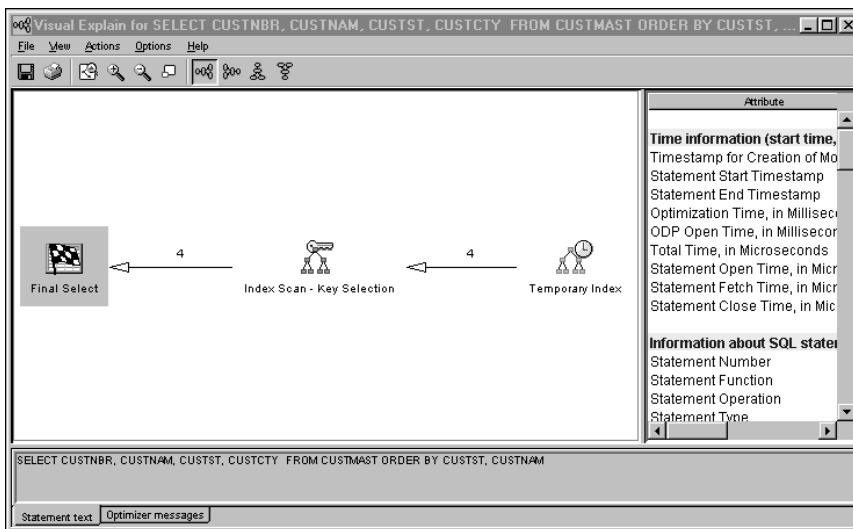


Figure 2.24: Visual Explain.

The window contains extensive information on the SQL statement being analyzed, and it may take a moment to display. Sometimes it may be necessary to run Visual Explain without actually performing the SQL statement. This is the case with some file updates, which cannot be performed multiple times without corrupting the data in the database. In such cases, Visual Explain can still be run against the statement without executing it, by selecting **Explain...** from the **Visual Explain** pull-down menu.

Understanding Visual Explain

The Visual Explain window has two sections: a graphical map of the statement in the left-hand pane and a detailed statistical section in the right-hand pane. Within the left-hand pane are icons for the three phases of the SQL statement just run: the Temporary Index, Index Scan, and Final Select. Each of these sections has corresponding statistics collected for them. Click on each icon and review the statistics in the right-hand pane. Other phases of SQL statements are used at other times. The three phases used for this Statement are:

1. **Temporary Index**—The statistics include information about the name of the index and what files it is built over, which fields to include in the index, the amount of processing time need to create the index, the type of index, the estimated number of entries in the index, and whether or not the SQL engine recommends creating a permanent index for this SQL statement.
2. **Index Scan**—The statistics include information about the name of the table being used, the index being used to process the table, estimated processing time, estimated number of rows returned, whether the optimizer timed out, whether a permanent index should be created, the sorted order of the data, and more.
3. **Final Select**—The statistics include information such as start time, end time, processing time, number of rows returned, and estimated time spent optimizing the statement.

If the creation of a permanent index is advised, it does not mean that it is a good idea: When running Visual Explain on an SQL statement that is only used once a month, the database administrator must consider the overhead involved in

having the database engine maintain that index all month long, simply to improve the performance of one statement each month.

Some programmers have been know to create an index, run an SQL statement that utilizes the index, and then delete the index. This should be unnecessary if the system is already creating a temporary index that is designed to optimize the SQL statement. Only create an index if you want a permanent one, or unless for some reason the system is not creating an effective temporary index.

If creating an index based on the advice from Visual Explain, click on the **Temporary Index** icon in the left-hand pane, then select Create Index from the **Actions** pull-down menu. The New Index wizard shown in Figure 2.25 is displayed.

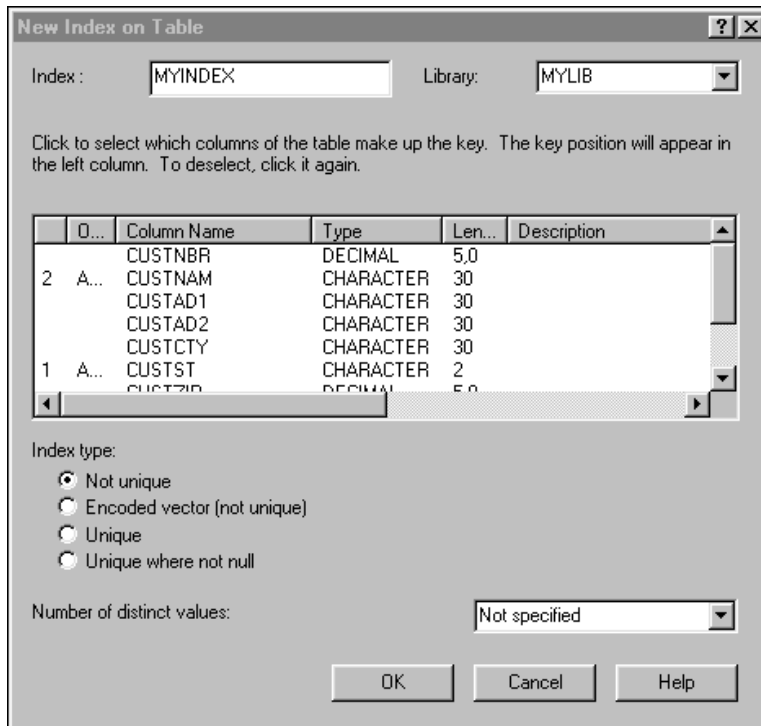


Figure 2.25: Create Index wizard.

The appropriate key fields are preselected, and other settings can be changed as needed. See Chapter 5 for information on how to optimize SQL statements with indexes.

Approximately thirty different phases of SQL statements are represented by icons in the graphical portion of Visual Explain. More information on these can be found within the help text.

When certain phases of the graphical map are selected, the **Actions** pull-down menu provides the ability to display **Table Properties** (Figure 2.26) and **Table Descriptions** (Figure 2.27).

Column Name	Type	Length	Description
CUSTNBR	DECIMAL	5,0	
CUSTNAM	CHARACTER	30	
CUSTAD1	CHARACTER	30	
CUSTAD2	CHARACTER	30	
CUSTCTY	CHARACTER	30	
CUSTST	CHARACTER	2	
CUSTZIP	DECIMAL	5,0	

The properties below apply to the column definition currently selected above.

Short column name: CUSTNBR

Heading: CUSTNBR

Must contain a value (not null)

Default value: 0

Figure 2.26: Table properties.

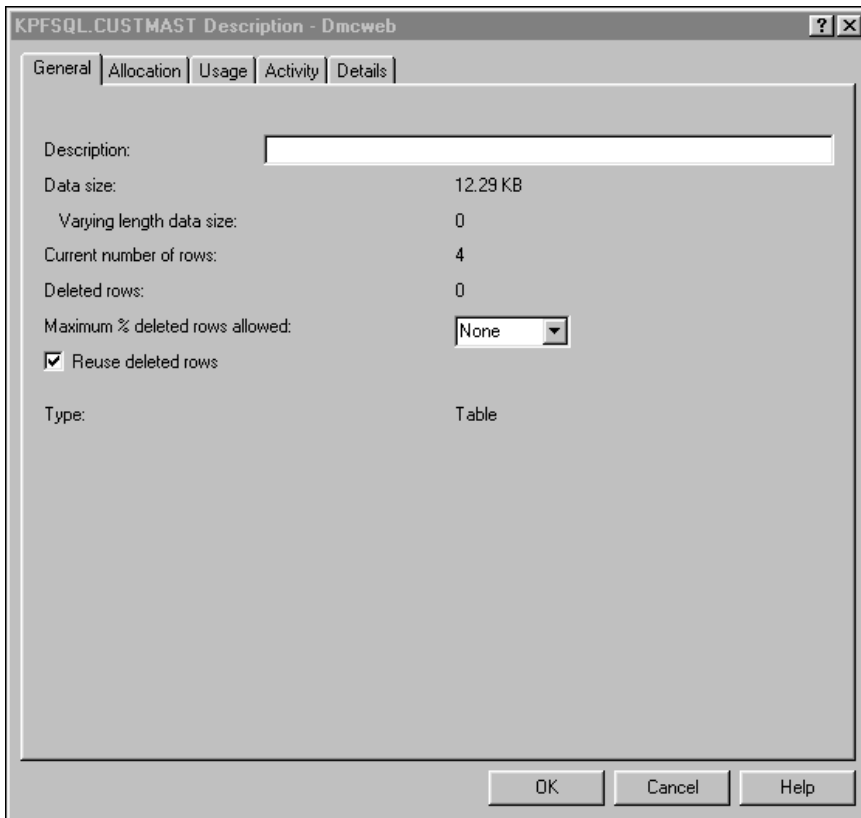


Figure 2.27: Table description.

These displays include extensive information about the properties of the table being processed, including information about column definitions, constraints and triggers, record format level check, reusability of deleted records, and usage and activity level information.

Becoming more familiar with Visual Explain and tapping into its capabilities will greatly enhance the capabilities of any database administrator or programmer. This chapter only briefly reviewed the topic, explaining the Run SQL Scripts and Visual Explain tools well enough for you to take advantage of these powerful new features from IBM. Look for continual improvement and enhancement to these tools with each new release of OS/400.

Because this chapter focused on introducing the GUI tool, the examples mostly used simple SELECT statements because more sophisticated SQL statements might have obscured the significance of the tool's features. But rest assured that virtually any SQL statement you can run in the native interactive environment will run here as well. (More sophisticated SQL statements are discussed in Chapter 4.)