4

Simple Database Features

Now we come to the largest use of iSeries Navigator for programmers—the Databases function. IBM is no longer developing DDS (Data Description Specifications) for database definition, and all future database enhancements will use DDL (Data Definition Language) in SQL (Structured Query Language). Already, some database functions are only available through SQL.

If you want to keep up to date with the changes being made to UDB DB2 for iSeries, you will have to become familiar with DDL, because DDS is not going to do it for you anymore. iSeries Navigator has an enormous role to play in this learning process, because it provides an easy-to-use graphical interface to most of the DDL requirements without you having to become an expert in DDL. It also provides a starting point for learning DDL.

The reason IBM is moving from DDS to DDL is that SQL is the industry standard for database on all platforms. In the current and developing environment, in which are multiple platforms are present in an organization, it is important that a

common tool be used to define the most important asset across the platforms—
the database.

Figure 4.1 shows the Databases function accessed in iSeries Navigator. It allows
you to individually view and maintain all the components of a database. Prior to
V5R3 of OS/400, you could only view all the components as one group (the
equivalent of *All Objects*). The Databases function has an entry for each data-
base defined on your system. For most of us, only one database is identified by
the serial number of the system; there may be other entries if remote databases
have been defined on your system. See the Work with Relational Database
Directory Entries (WRKRDBDIRE) command or use New → Relational
Database Directory Entry from the context menu of Databases for more informa-
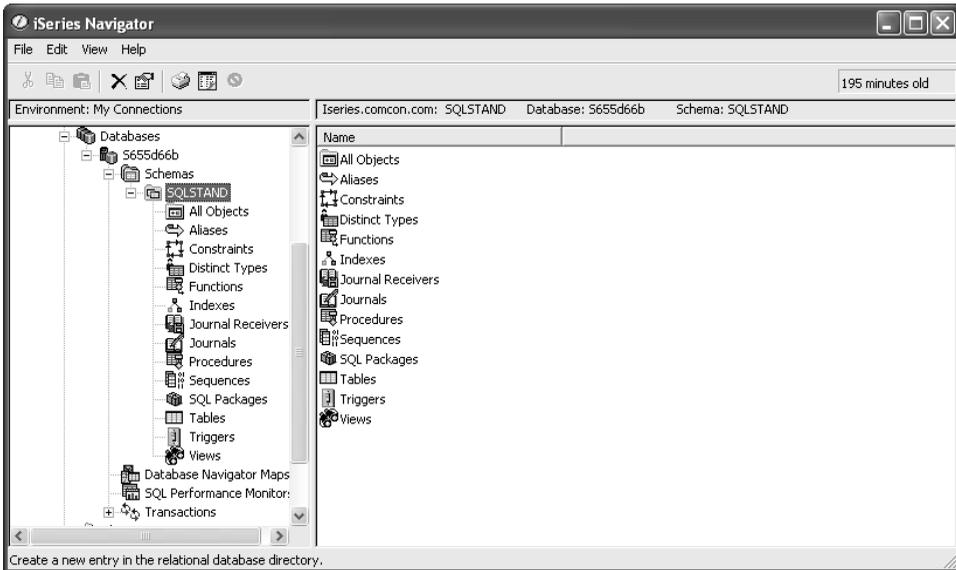tion on defining remote databases.

Figure 4.1: Databases in iSeries Navigator.

Terminology

The first thing you must get used to is SQL terminology. Table 4.1 lists the tradi-
tional OS/400 terms and their corresponding SQL terms; although enormous
similarities are obvious between the two, quite a few differences also exist,

especially with views and indexes. I will discuss the pros and cons of SQL versus DDS later in Chapter 9, once you have had an opportunity to become familiar with the features in Navigator.

Table 4.1: OS/400 terms and the corresponding SQL terms

OS/400	SQL
Library	Collection or Schema
Physical File	Table
Record	Row
Field	Column
Logical File (Keyed)	Index
Logical File (Non – Keyed)	View

Although the terms may be different, the Databases function in iSeries Navigator identifies all objects using the SQL terminology. So, a physical file that was created from a DDS source member is identified as a table.

Schemas

A Schema (or Collection) is the SQL term for a collection of database objects. Creating a schema on the iSeries, results in the creation of a library containing a few predefined objects, as shown in Figure 4.2. iSeries Navigator does not restrict you by only allowing access to a true schema, but also allows you to access normal iSeries libraries. Database objects still will be displayed correctly, even those defined using DDS or those created using SQL in a green-screen environment.
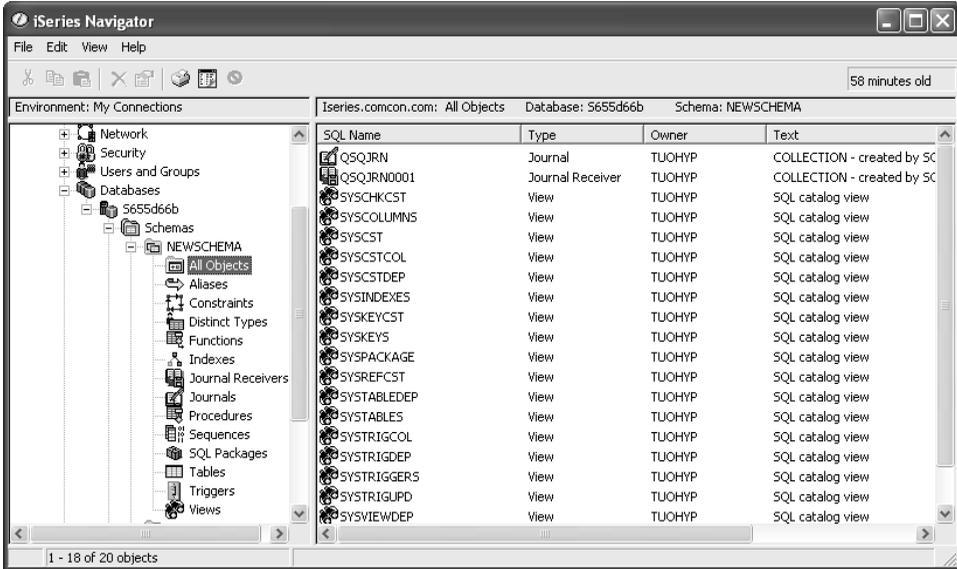
Figure 4.2: Objects created in a schema.

A new schema contains a journal (QSQJRN) and a journal receiver (QSQJRN0001). Any tables created in the schema are automatically journaled to QSQJRN. This is very useful during development, when using a test database, but it warrants some serious consideration when moving into a production environment. Having one journal per library would not be considered the norm in a production environment.

The rest of the objects are SQL catalog views. These are views over the system catalog files, with selection criteria for objects in this library.

Table 4.2: Description of catalog views

Catalog View	Description	Catalog Files
SYSCHKCST	Check constraints	QADBFCST
SYSCOLUMNS	Column attributes	QADBIFLD QADBXSFLD
SYSCST	All constraints	QADBFCST

Table 4.2: Description of catalog views *(continued)*

Catalog View	Description	Catalog Files
SYSCSTCOL	Columns referenced in a constraint	QADBCCST QADBIFLD
SYSCSTDEP	Constraint dependencies on tables	QADBFCST QADBXREF
SYSINDEXES	Indexes	QADBXREF QADBFDEP
SYSKEYCST	Key constraints (unique, primary and foreign)	QADBCCST QADBIFLD
SYSKEYS	Index keys	QADBIFLD QADBKFLD
SYSPACKAGE	SQL Packages	QADBPKG
SYSREFCST	Referential Constraints	QADBFCST
SYSTABLEDEP	Materialized query table dependencies	QADBXREF QADBFDEP
SYSTABLES	Tables and Views	QADBXREF QADBXMQT
SYSTRIGCOL	Columns used in a trigger	QADBXTRIGD QADBXREF
SYSTRIGDEP	Objects used in a trigger	QADBXTRIGD QADBXREF
SYSTRIGGERS	Triggers	QADBXTRIGB
SYSTRIGUPD	Columns in the WHEN clause of a trigger	QADBXTRIGB QADBXTRIGC
SYSVIEWDEP	View dependencies on tables	QADBXREF QADBFDEP QADBXREF
SYSVIEWS	Definition of a view	QADBXREF

The system catalog files are stored in QSYS and contain cross-reference information about every database object on the system, regardless of whether it was generated using SQL or DDS. You can view a list of the catalog files using the command:

```
WRKOBJPDM LIB(QSYS) OBJ(QADB*) OBJTYPE(*FILE) OBJATR('pf-dta')
```

Schemas Displayed

When you first use the Databases function, the libraries QTEMP and QGPL are listed under Schemas, and you may be inclined to think that a library list is being used. This is not the case—you determine which schemas (or libraries) are to be listed.

Select **Databases ➜ Database Name** and then **Select Schemas to Display** from the context menu of **Schemas** to see the selection window shown in Figure 4.3. You can enter the names of the schemas to add to the list, or you can select and add them from a filtered list.
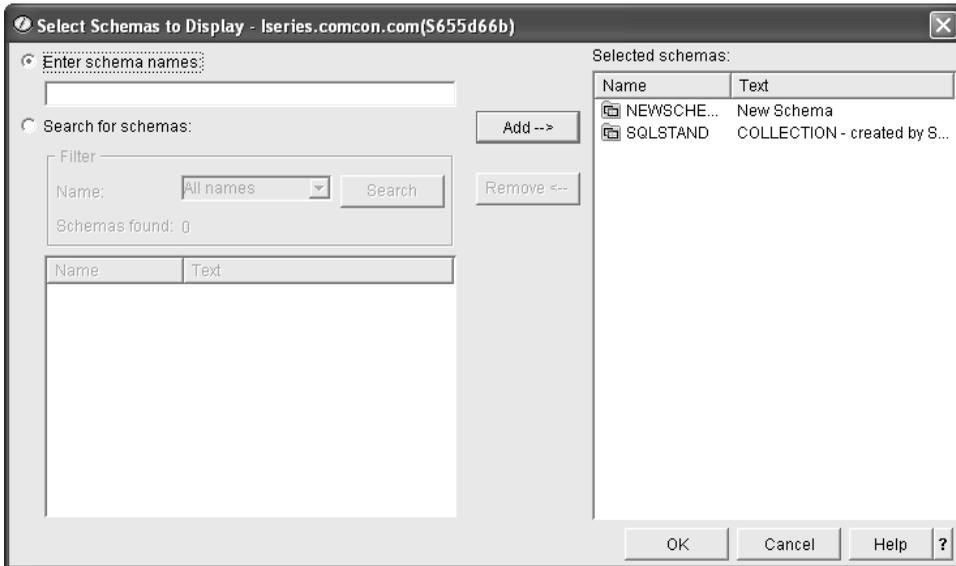

Figure 4.3: Select Schemas to Display.

When you select a schema, you have the choice of listing All Objects or selecting the individual type of database objects to list. Be aware that All Objects does not refer to actual iSeries objects, but to database objects. For example, constraints are listed—and constraints are not iSeries objects. Also, nondatabase objects, such as programs, are not listed.

Tables

Tables are the base building blocks for a database. Select **New ➜ Table** from the context menu of **Tables** to see the window shown in Figure 4.4. The window has tabs for Table, Columns, Key Constraints, Foreign Key Constraints, Check Constraints and Partitioning. Select **Definition** from the context menu for a table to change its definition.
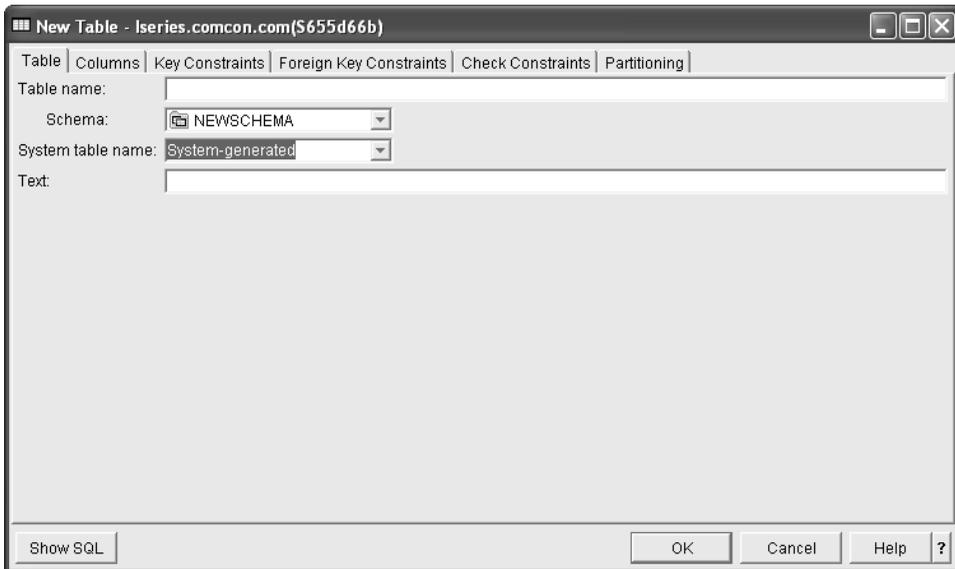

Figure 4.4: Creating a new table.

Table

You name the table in the Table tab, and it is also where you see one of the first differences between DDS and SQL. DDS is restricted by the normal ten-character

93

restriction on names, but this is not so for SQL: SQL names can be up to 128 characters long.

When you specify a table name, you are specifying the SQL name. In Figure 4.4, note the *System table name* entry with a default value of System-generated. When the table name exceeds ten characters, the system generates a ten-character *System table name* by taking the first five characters of the name and adding a five-digit sequence number.

Just in case you are looking for an entry that allows you to specify the record format name—there isn't one. The format name and the table name are the same when you create a table. This is a problem for RPG programs, because the RPG compiler does not allow the file name and the format name to be the same. Two solutions are possible: In the RPG program, use the RENAME keyword on the file specification to rename the record format, or create the table with the name of the format and then rename the table object.

Columns

You specify a table's columns in the *Columns* tab (Figure 4.5 shows a completed column list). You use the Add and Definition buttons to define columns, the Remove, Move Up, and Move Down buttons to sequence columns. You can use the Browse button to select columns from other tables. The Move Up and Move Down buttons are not available when changing the definition of a new table, and new columns may only be added to the end of the list. The Browse function is not the same as using a field reference file. It only provides a simple copy-and-paste function. I will be discussing the equivalent of a field reference file in Chapter 5.
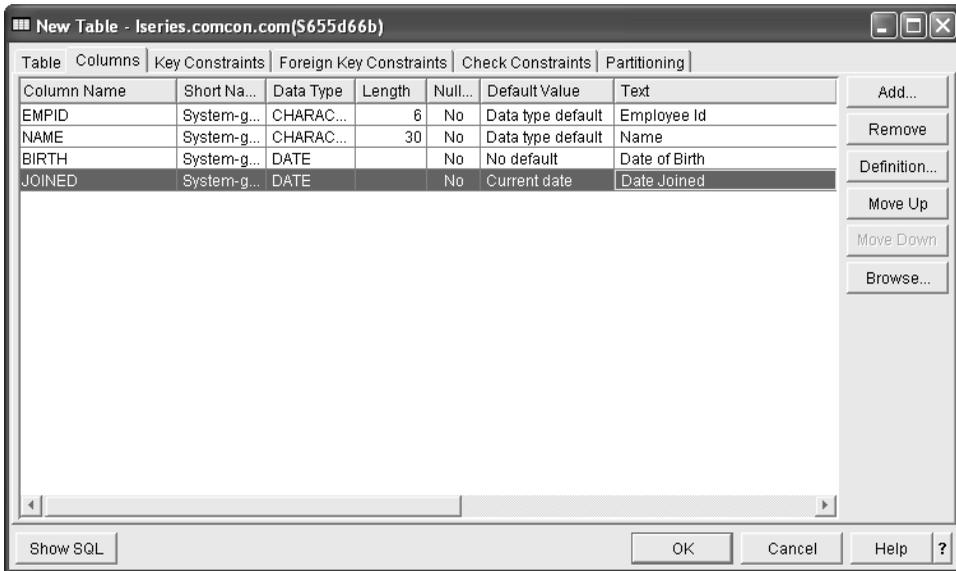
Figure 4.5: Defining columns for a table.

Figure 4.6 shows the window displayed when you add or define a column. When adding columns, the Add window stays in place until you select the Close button.
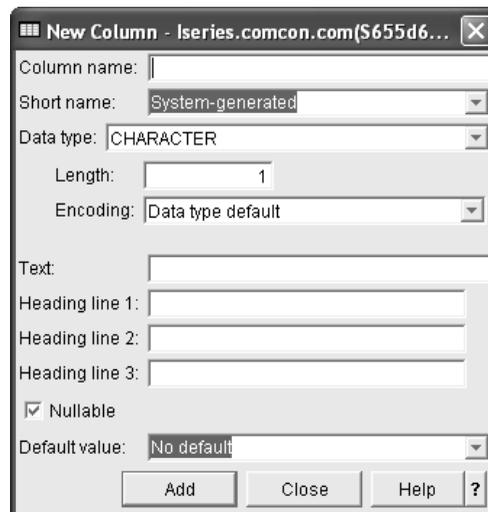

Figure 4.6: Defining a column.

Again, you must be careful of the length of the name: A name exceeding ten characters in length will default to a system-generated name unless you specify a short name.

A few more *Data Types* are available in SQL than in DDS, even to the extent that you can define your own data types (more about this in Chapter 5). Certain data types are not allowed in high-level language programs. Table 4.3 lists the data types available in SQL and their DDS equivalents, and indicates whether the data types are available in high-level languages. The requirement to specify Length, Precision, and/or Encoding depends on the Data Type selected.

Table 4.3: Comparison of data types and usage in high-level languages

SQL	DDS	Allowed in HLL
INTEGER	Binary (9, 0)	Yes
SMALLINT	Binary (4, 0)	Yes
BIGINT	Binary (18, 0)	Yes
DECIMAL	Packed	Yes
NUMERIC	Zoned	Yes
FLOAT	Float	Yes
CHARACTER	Character	Yes
VARCHAR	Character Varying	Yes
GRAPHIC	Graphic	Yes
VARGRAPHIC	Graphic Varying	Yes
DATE	Date	Yes
TIME	Time	Yes
TIMESTAMP	TimeStamp	Yes
DATALINK	N/A	No
CLOB	N/A	No

Table 4.3: Comparison of data types and usage in high-level languages *(continued)*

SQL	DDS	Allowed in HLL
BLOB	N/A	No
DBCLOB	N/A	No
BINARY	Binary Character	No
VARBINARY	Binary Character Varying	No
ROWID	Hexadecimal	Yes

You also must take care with the *Nullable* check box. In SQL, the default is that columns are null capable, which is not the default in DDS, in which you have to explicitly indicate if a column is null capable. Pay special attention when you are adding columns, because the *Nullable* box is rechecked when you switch Data Type.

When you add a column with a data type of SMALLINT, INTEGER, BIGINT, DECIMAL, or NUMERIC, you are given the option of setting the column as an *Identity Column*. An Identity Column provides a means to uniquely identify every row in a table. Every time that a new row is added to a table having an identity column, the identity column value in the new row is incremented (or decremented) by the system.

Constraints

Constraints will be discussed in more detail in Chapter 6. For the moment, it suffices to say that Constraints are a key component to the development of any database, and they are an absolute necessity if your database is going to be accessed from sources other than your iSeries.

Partitioning

Partitioning allows you to divide a table into a maximum of 256 partitions, each of which can contain the maximum number of rows for a table (approximately 4,294,000,000).

Other Differences

Two important defaults are different when creating a table, as opposed to creating a physical file. The maximum size of a table is No Maximum (SIZE(*NOMAX)), and the assumption is to reuse deleted records (REUSEDLT(*YES)). If required, these values can be changed under the General and Allocation tabs, when you select Description from the context menu for a table.

Edit Contents

Once a table has been created, you can insert, update, or delete rows by using the Edit Contents option (the default) from the context menu of the table. Figure 4.7 shows an example of maintaining data in a table; the value in any column can be changed by simply overtyping it. Rows may be inserted or deleted by making the relevant selection from the Rows option on the menu.
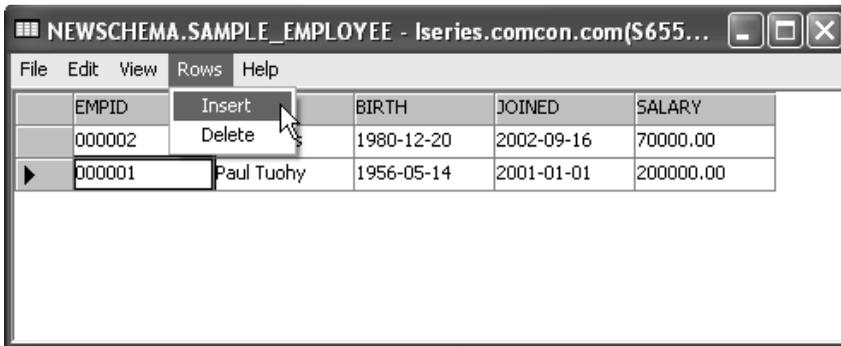

Figure 4.7: Editing the contents of a table.

If the table is journaled (the default for a table created in a schema), then any changes you make are not actually applied until you close the file editor. That is, the editor does not commit the changes until you exit. If the table is not journaled, then the changes to rows are immediate, and you will receive a warning message for the first row that you try to insert, update, or delete.

The ability to edit the contents of a file should be considered as a replacement for the Data File Utility (DFU) and should only be used to maintain data in test

tables—it should not be used to maintain data in production tables, not that the thought would ever cross your mind.

Indexes

An Index is the equivalent of a keyed logical file with no column or record selection defined.

The easiest way to define a new index is to select **New** ➜ **Index** from the context menu of a table. You can also select **New** ➜ **Index** from the context menu for Indexes. Figure 4.8 shows the definition window f or a new index.
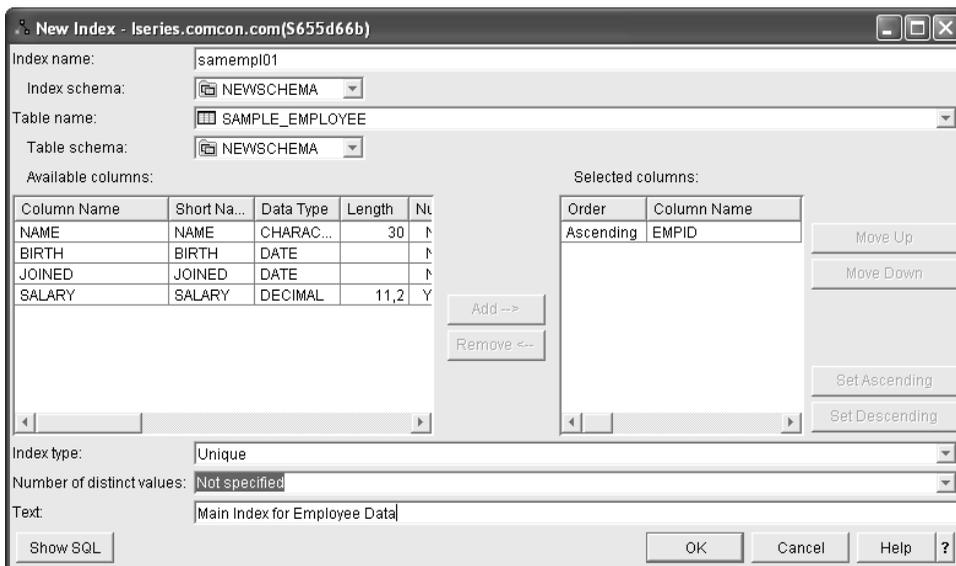

Figure 4.8: Creating a new index.

You must be even more careful when naming indexes than you were when naming tables and columns. For some strange reason, you do not have the option of defining a short name for an index, so it will be system-generated if the length of the *Index name* exceeds ten characters.

Use the Add button to add the required columns that make up the key, and use the Move and Set buttons to ensure the sequence is correct. Be careful when adding columns: They are always added to the top of the list. You soon get used to defining your keys in reverse.

The *Index type* can be Unique, Not unique, Unique where not null, or an Encoded Vector. An Encoded Vector Indexes (EVI) keeps track of the distinct values that can be found in the key columns of a table. An EVI can improve data warehouse performance queries, as well as business applications queries, but an EVI cannot be used to ensure any expected ordering of records and cannot be used to position an open data path. In other words, an EVI may be used by the Query Optimizer when running a selection against the database, but it may not be used in a high-level language. You only use an EVI to enhance performance when ad-hoc queries are present against the database.

The *Number of distinct values* entry is primarily for EVIs and is used to determine the size of each entry. For other index types, the entry can be an estimate of the number of entries expected in the index that may (or may not) be of use to the Query Optimizer.

The following code is the DDS equivalent of the index defined in Figure 4.8. DDS does not have the ability to define an EVI.

```
A                                                   UNIQUE
A             R SAMEMPL01             PFILE(SAMPL00001)
A             K EMPID
```

Indexes for a Table

Select **Indexes** from the context menu of a table to see a window listing all indexes built over a table, as shown in Figure 4.9. One of the disconcerting features is that the text description of the indexes seems to have disappeared. The text description *is* there when you list the indexes using the Indexes selection for the Schema, as shown in Figure 4.10.

The reason for this is that the text you define for the index is placed in the system catalog file as the long comment for the index (look at the contents of SYSINDEXES), but it is not duplicated as the text for the *FILE object created for the index. The index window for the table (Figure 4.9) shows the text from the object description while the indexes for a schema (Figure 4.10) shows the text from the system catalog file.

You can change the object description by selecting **Description** from the context menu of the index and changing the value of the *Description* entry under the **Details** tab. But it is a pity that this is not done automatically, as it is for tables.
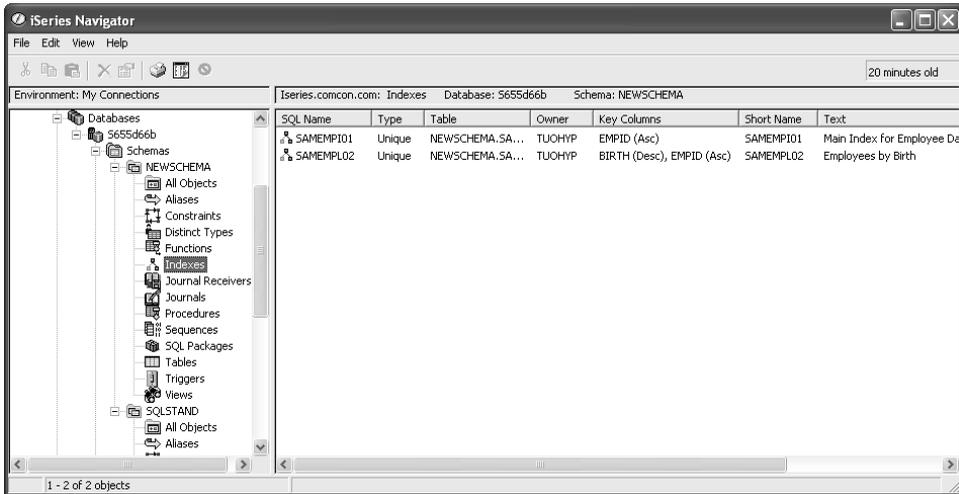

Figure 4.9: Indexes defined for a table.

Figure 4.10: Indexes for a schema.

Other Differences

Two other differences are apparent between creating indexes and creating keyed logical files from DDS.

Indexes have a larger page size in memory (64K as opposed to 4K or 8K), which leads to a faster processing time for an index when processing sequentially by key.

An index will only share an access path if all the key fields match, whereas a logical file can share an access path if its key fields are a subset of an existing access path.

Views

A View is a nonkeyed logical file; in other words, you can define everything except a key. Views highlight another major difference between SQL and DDS. In this chapter, we'll just look at the basic definition of a view. Some of the more advanced features are described in Chapter 5, so not all options and buttons available will be discussed here.

Let's have a look at creating a view that gives us the equivalent for the following DDS:

```
A              R SAMEMPL01                 PFILE(SAMPL00001)
A                EMPID
A                NAME
A                SALARY
A              S SALARYK                   COMP(GT 1000000)
```

To create a new view select **New → View** from the context menu for **Views** in the Schema. Figure 4.11 shows the New View window, where you provide a *Name* and *Description*. As with indexes, you must be very careful with the name; there is no option to provide a short name, so if the name exceeds ten characters, you will end up with a system-generated name for the object name.
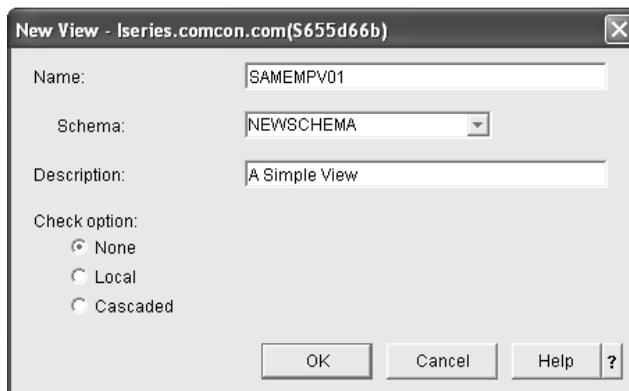

Figure 4.11: Naming a new View.

Figure 4.12 shows the completed definition of a view that selects the Employee Id, Name, and Salary from the SAMPLE_EMPLOYEE table and only selects rows where the salary is greater than 100,000.00. The view is constructed using the Select Tables and Select Rows buttons.
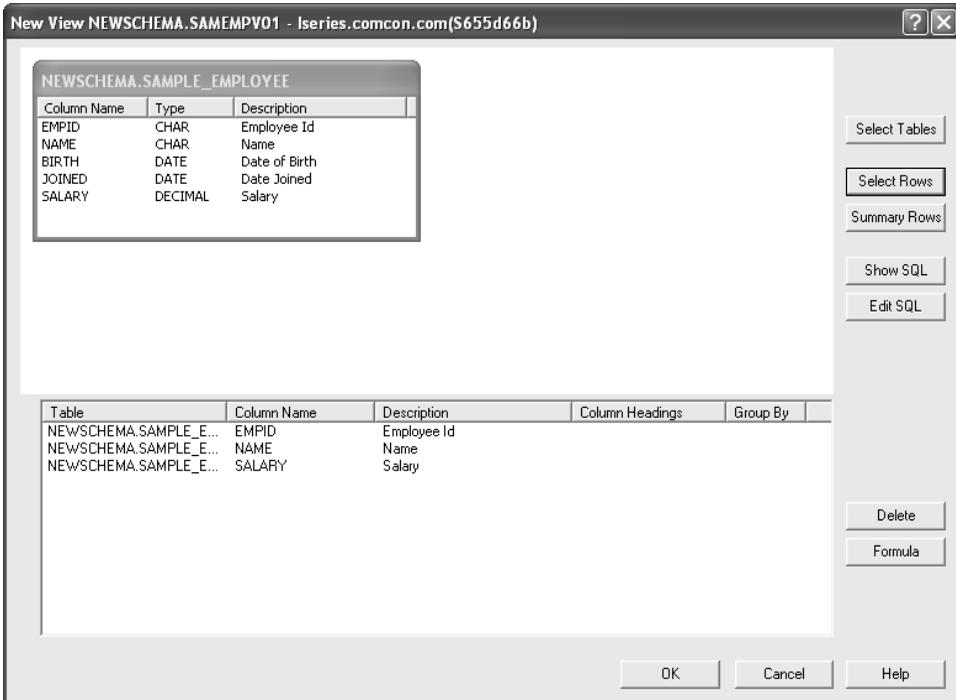

Figure 4.12: Defining a View.

Select Tables

The Select Tables button is a slight misnomer, since it actually allows you to select views as well as tables. This is one of the major benefits of SQL: You can define a view of a view. The Select Tables button presents you with a window that allows you to select tables and views from any of the selected schema, as shown in Figure 4.13. Select a table or view, and use the Add button to add it to the View.
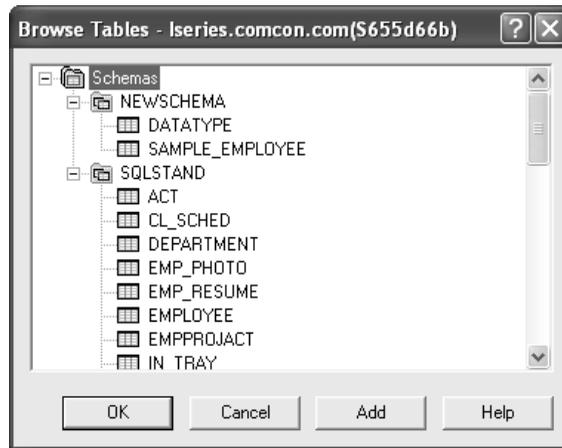
Figure 4.13: Selecting tables and views for a View.

Select Columns

To select columns, you simply drag and drop them from the table (or view) window in the upper pane to the column list in the bottom pane, as shown in Figure 4.12. You can change the sequence of selected columns by simply dragging and dropping them to their new position.

Select Rows

The Select Rows button presents you with a window similar to that shown in Figure 4.14. This example is a simple selection where the salary is greater than 100,000.00.

The *Columns* pane lists all columns available; these are all the columns from the selected tables and views, not just the columns selected for the view. You specify the selection criteria by entering an SQL WHERE clause in the Clause pane, if you are familiar with SQL, or you can select columns and operators by double clicking them. You also may make use of any of the SQL functions listed.
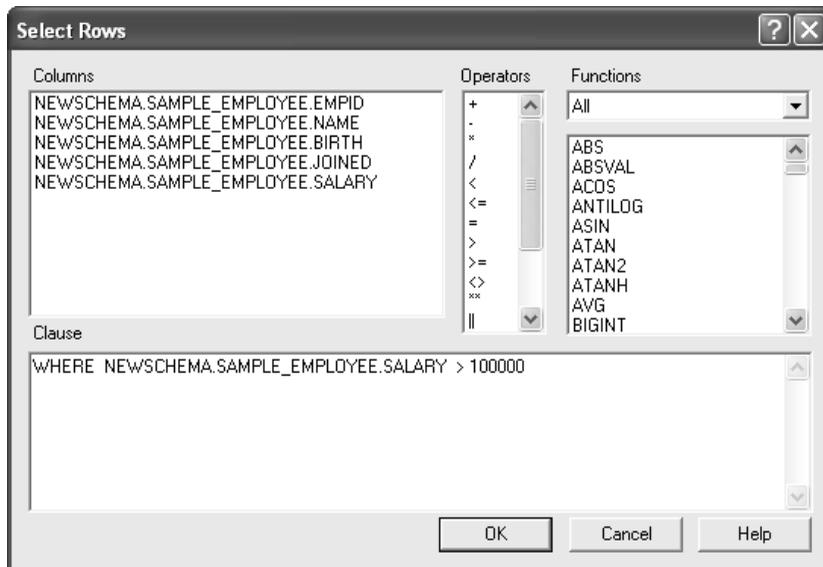

Figure 4.14: Selecting Rows for a View.

Selecting rows for a view is another area in which SQL offers significant advantages over the select/omit logic in logical files. For example, you can specify a view that returns rows where the number of years difference between the date of birth and today's date is greater than 30 years, or you can specify a view that returns rows where the salary is greater than the average salary for the company. These are two criteria that you would not even consider in a logical view.

Changing Views

iSeries Navigator provides an excellent interface for creating a new view, but it does not provide one that allows you to maintain a view. When you select **Definition** from the context menu for a view, you are presented with a window that allows you to change little or none of the details of the view. It is better to delete and recreate the view or to resort to using actual SQL.

Basics Done

This chapter has given you an overview of how you can use iSeries Navigator to define the tables, indexes, and views that emulate the creation of physical and logical files from DDS source. It has also shown some of the differences between DDL and DDS—some good and some bad.

But what about Field Reference files, and where is the source for what you have created? And what about join logicals? And what else do DDL and the Databases function in iSeries Navigator have to offer? Let's move on to Chapter 5.