Chapter 4

SQL Concepts & Facilities

Is SQL an End User Tool?

When you first look at SQL, each statement makes a lot of sense, and it would be reasonable to conclude that a sharp knowledge worker (end user) in an organization ought to be able to use the SQL tool independently of the IT department. At first brush, you might even look at SQL as a query language for end users. However, it is not.

It would be a big mistake for any organization to forget that SQL is designed for IT professionals. Depending on the IT job function, some IT people might never need to use SQL. Based on the IT function, there would necessarily be a difference in the level of use and the knowledge required. The types of positions in an IT shop that would depend on SQL as a tool include the following:

- ✓ Application developers
- ✓ System programmers
- ✓ Database administrators

One of the leading System i5 database gurus for the past twenty years or more is a former IBMer, Skip Marchesani. Skip goes way back with the System i5 product line. In fact, Skip was one of a number of IBM instructors who conducted IBM internal System/38 database education classes in the 1979 / 1980 time period, even prior to the machine becoming generally available. I am proud to say that I was an eager-to-learn student in Skip's System/38 classes.

I mention Skip in this section because I have what is almost an exact quote (shown below) from him about his perception of end user involvement in SQL. It is very telling:

"Putting SQL in hands of end users is like giving a razor blade to a 3 year old to cut pictures out of magazines …Who gets to clean up the mess?"

AS/400 ANSI SQL Advanced Facilities

There is a whole lot more than just a query engine in SQL. In addition to the basic database facilities expected in any database language, such as table and view creation with DDL, and query and update capabilities supplied by DML, and authority supplied by DCL, the SQL language provides the most advanced facilities available for the System i5 database. For example, the following advanced database functions are provided in the database and usable through the SQL interface:

Declarative Referential Integrity

This integrity constraint provides the ability for developers to define integrity relationships to be enforced at the database level, rather than the program level. The DB2 UDB for System i5 implemented via SQL provides support for the following actions when the defined integrity rules are attempted to be broken:

- ✓ NO ACTION
- ✓ RESTRICT
- ✓ CASCADE
- ✓ SET NULL
- ✓ SET DEFAULT

Triggers

Triggers are also implemented via the SQL language and the DB2 UDB database. When you have defined that certain actions need to occur when and if certain database values change, you implement that function with "Triggers."

Without triggers, you code these actions into all the programs that touch a database. And, as you know, in most shops, there are many programs that cause updates to the same database. That's where DB Triggers help out. When a certain action occurs on the database as a result of an insert, or update, or a delete, a trigger fires and a program that you write gets control. In this program you can code whatever has to be done to protect your business at that time.

For example, one of my clients has an order entry program that was purchased many years ago. The software company no longer supports it and will not provide the source code. They want this client to buy their new version but the client is unwilling. DB Triggers are a way of getting into the logic of a program even if you do not have the source. In my client's case, by law they are prohibited from selling certain products in certain jurisdictions. If they violate the law, they could lose their license to do business within a particular state. Having their backs to the wall, the best solution for them was to write an order entry trigger program that analyzed the DB update to the order file before it was made. In this way, the client company was able to send the order taker a message and not permit the bad line item to be written to the transaction database.

Stored Procedures

Stored procedures consist of compiled code residing on an intelligent database server such as the DB2 UDB for System i5. The major purpose of stored procedures is to reduce the processing burden on the client side of client server as well as to reduce the communication interactions time. These precompiled SQL routines (and other languages such as RPG and COBOL on System i5) are stored on the System i5.

When implemented, they provide major advantages for client server and intelligent Web processing. The major benefit is that the application performs better since the server code is precompiled and because there is minimal back and forth action over the network between the client and the server. Additionally, because the code is on the server, one set of code can be reused for as many clients as necessary.

SQL Basic Facilities

In addition to the advanced facilities noted above, SQL is rich in the type of ease of use capabilities that are necessary to support relational databases from the simple to the complex.

Table Facility

First and foremost, SQL provides a table facility that enables a prompted, intuitive interface for the following functions:

- ✓ Defining databases
- ✓ Populating databases with rows
- ✓ Manipulating databases.

Table Editor

SQL also provides a table editor that makes it easy for you to perform the following functions against rows in table data that is structured in row and column format:.

- ✓ Access
- ✓ Insert
- ✓ Update
- ✓ Delete

Query Facility:

With the Query facility, SQL permits you to interactively define queries and have results displayed in a variety of report formats including the following:

- ✓ Tabular
- ✓ Matrix
- ✓ Free format

For those readers who have a System i5 background, you will notice that SQL brings with it its own naming scheme that is significantly different from corresponding native objects. See table 4-1 for specifics

Table 4-1 The SQL Name Game

SQL Name	System i5 Name
Database	System Name
Entity	File
Table	Unkeyed Physical File
View	Unkeyed Logical File
Row	Tuple or Record
Column	Attribute or Field
Index	Keyed Logical File
Collection	Library w/ SQL info objects
Schema	Library w/ SQL info objects
Log	Journal
Isolation level	Commitment Control Level
Tablespace	Not needed in DB2/400
Storage group	Single Level Storage

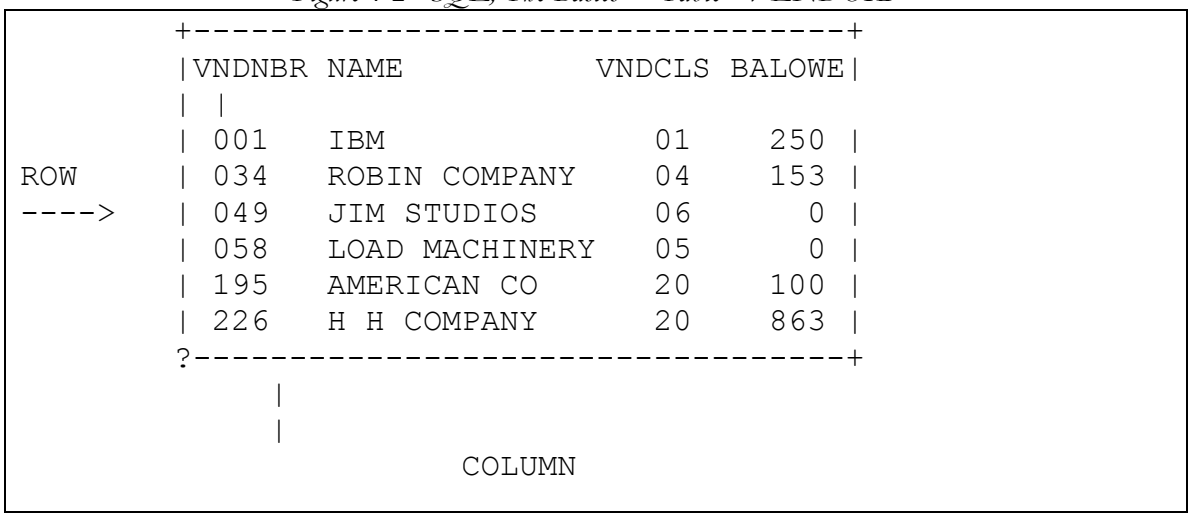
Basic SQL Data Definition Language

Let's start our examination of the SQL language by doing something simple such as constructing a query against a table that already exists in our database. This will help us quickly get a flavor for the conciseness and power of the SQL language syntax. Suppose we have never even defined an SQL table but on the System i5, there is a file defined and in use by the accounts payable application. It's name is Vendorp. It is a physical file and it contains data about vendors. It was created using DDS years ago. For our simple purpose let's say that it contains just four fields -- a vendor number (field name VNDNBR), vendor name (field name NAME), the class of the vendor (field name VNDCLS), and the balance owed to that vendor (field name BALOWE).

Sample SQL Table – Mini-Vendorp

As you recall from Chapter 3, a relational database is a database that is perceived by its users to be a *collection of tables* (and nothing else but tables). An SQL table is defined as a series of rows and columns where each row represents a record and each column represents an attribute of the records contained in the table. With that as a backdrop, let's look at our first table in Figure 4-2 after it has been populated by just a few records:

Figure 4-2 SQL, The Basics -- Table - VENDORP



This simple table in Figure 4-2 with data gives the notion of rows and columns as used by SQL.

Creating a Tables/File with CRTPF and DDS:

Now, that we have a general idea of a file/table, let's say that we are going to start all over with Vendorp, since we left out too many fields to make it worthwhile altering the table. The native coding in DDS for this new, enhanced Vendorp file is shown in Figure 4-3. Notice we added quite a few more fields to the Vendorp physical file. The list of fields in Figure 4-3 is actually the DDS for the Vendorp file. If we were to issue a create physical file command (CRTPF) against this set of DDS, we would create a database object named Vendorp in a to-be specified library. A library on the AS/400 is like a big directory that helps us organize objects.

Figure 4-3 DDS for Expanded Vendorp File Definitions

```
 FMT PF .....A.........T.Name++++++RLen++TDpB..
        ********** Beginning of data ******
0001.00      A          R VENDRF
0002.00                   VNDNBR         5S 0
0003.00                   NAME          25
0004.00                   ADDR1         25
0005.00                   CITY          15
0006.00                   STATE          2
0007.00                   ZIPCD          5  0
0008.00                   VNDCLS         2  0
0009.00                   VNDSTS         1
0010.00                   BALOWE         9  2
0011.00                   SRVRTG         1
        ****************** End of data *********
```

In line 1 (sequence # 1) of the DDS specifications, as you can see, in Figure 4-3, we defined the record format with a name of VENDRF. In high level programming languages (HLL), such as RPG and COBOL, many of the file operations such as read and write are directed at the record format name as opposed to the file name in flat file systems. When the program that references a database file such as Vendorp is compiled, some of the high level language compilers, especially RPG/400 require that the DB file's record format name is different from the name of the file. When tables are created with SQL, the default record format name is always used and for its own reasons, IBM has selected the name of the file for the record format name in SQL database table objects. This creates issues when trying to use SQL created database objects in existing high level programs.

IBM wrote a whole book (Redbook) on considerations for moving to an SQL-only environment and because DDS and SQL's capabilities are not completely the same, this book is helpful if you choose to make the move to SQL from DDS. The Redbook name is Modernizing IBM eServer System i5 Application Data Access -A Roadmap Cornerstone. The Redbook site is www.redbooks.ibm.com. From there, search for some words in this title, and you can download this valuable IBM manual.

In line 2 of Figure 4-3, we defined the vendor number field named as VNDNBR. RPG/400 demands that no more than 6 character field names be used so many System i5 databases are defined with very short names as you can see in the figure. The number five on the line says the field length of VNDNBR is 5 and the S data type stands for unpacked decimal. This means that this field takes 5 positions of storage in the disk record. The zero at the end of the line says that the field is numeric with zero decimal places.

Contrast this with the BALOWE field in line 10. There is no S. But there is a 2 in the decimal positions column. This says that the field is numeric, just as VNDNBR, but without the S for data type, it defaults to a packed decimal data type. Therefore, for the nine positions defined, with packed decimal, this large numeric field can be stored in just five positions in the disk record. That's the nature of packed decimal as a data type. It saves space.

Now, look at the NAME field as defined in line 3 of the DDS. It is barebones, meaning the name NAME, and the field length of 25 are the only two pieces of information specified for this field. This coding means that this field is character or alphanumeric as IBM likes to call it. If a DDS line has no decimals specified, that means the field being defined will hold character data. When read in a program, no mathematical operations can be performed against character data.

Now, let's take the same file as defined in DDS and define it in SQL. SQL tables become physical file objects on System i5 after they are created – with either DDS or SQL. The command to create a table in SQL is Create Table. The full SQL coding for the Vendorp table is shown in Figure 4-4.

Figure 4-4 Creating a Tables/File with SQL

```
CREATE TABLE SQLBOOK/VENDORP
    (VNDNBR       NUMERIC(5,0)     NOT NULL,
     NAME         CHAR(25),
     ADDR1        CHAR(25),
     CITY         CHAR(15),
     STATE        CHAR(2),
     ZIPCD        DEC(5,0),
     VNDCLS       DEC(2,0),
     VNDSTS       CHAR(1),
     BALOWE       DEC(9,2),
     SRVRTG       CHAR(1))
```

Unlike DDS, the SQL data definitions for creation of a table begin after the command to create them, not in a separate screen panel or program. As you can see in Figure 4-4, the SQL statement Create Table starts the processing command. It is going to create the table name specified after the word "Table." So following the Create Table, you specify the schema / collection (SQLBOOK) in which the table will be created, immediately followed by the name of the table that is being created (Vendorp).

After the Create Table Name, the rest of the SQL statement provides the data definition. The first element in the data definition is the field name. This is followed by the data type. Notice that the signed decimal data type from DDS has an equivalent in SQL. In SQL this is the NUMERIC data type. The packed decimal data type from SQL (no type specified with decimals specified) also has its equivalent, as the DECIMAL or DEC data type. Finally, in this example, though there are many data types in SQL, the last data type in this example is CHAR type, meaning character. This is the equivalent of no decimals specified and no data type specified for a field in DDS.

Basic SQL Data Manipulation Language

As noted in Chapter 3, SQL's Data Manipulation Language (DML) has four basic functions provided by four different SQL statements. You may recall these are as follows:

- ✓ Select data from a database
- ✓ Insert data to a database
- ✓ Update data in a database
- ✓ Delete data from a database

Now that we have a database defined, let's assume for the next set of basic examples that we took it all the way. The Vendorp table is created and it is pre-loaded with the data necessary to execute the following examples. The first examples are simple Selects, followed immediately by Insert, Update, and Delete examples.

Basic SQL Select Statement

The verb, *Select* is the query verb in SQL. Whenever you use this verb, you can perform any of the many RDBMS functions such as projection, selection, intersection, join, etc. as defined by Tedd Codd.

Let's start with the most basic example of select. The three parts to a basic select are as follows:

Command	Function
Select	**Select verb starts the statement**
[colums]	**[* for all or column names to select]**
From:	**From clause specifies the library and table**

When we want to select all of the columns and all of the rows, the select statement in its most simple is shown below

```
Select *
From SQLBOOK/vendorp
```

This brings back all of the columns and all of the records in a memory table and it displays the result table if the user is in interactive mode. If this is executed in a program, it brings the whole table into the memory of the program.

Let's bring back the rows that we explored in Figure 4-2 above.

```
Select
VNDNBR, NAME, VNDCLS, BALOWE
From SQLBOOK/vendorp
```

This command provides four fields from the Vendorp table across all of the rows.

Now, if you refer to the data (6 records) in Figure 4-2 above, let's add a row constraint with the SQL Where Clause. Let's display only those vendors whose class value VNDCLS is 20.

Here is what this looks like

```
Select
VNDNBR, NAME, VNDCLS, BALOWE
From SQLBOOK/vendorp
Where VNDCLS = 20
```

The results of the query from this mini database are in Figure 4-5.

Figure 4-5 SQL, The Basics -- Table – VENDORP VNDCLS 20

```
         +---------------------------------+
         |VNDNBR NAME           VNDCLS BALOWE|
ROW      | 195   AMERICAN CO       20    100 |
---->    | 226   H H COMPANY       20    863 |
         ?---------------------------------+
              |
              |
                        COLUMN
```

Let's say you add the fields from Figure 4-4 to the file and that you add a few more records. If you run the same Select Query again, you will see results such as those shown in Figure 4-6:

Figure 4-6 Result of Projection and Selection with Larger File

```
                                    Display Data
                                                Data width . .
Position to line  . . . . .                  Shift to column  .
....+....1....+....2....+....3....+....4....+....5....+..
VENDOR   NAME                          VENDOR       BALANCE
NUMBER                                 CLASS        OWED
    40   SCRANTON INC                    20           250.00
    44   J B EQUIP INC                   20            50.00
    48   DENTON AND BALL                 20         3,500.00
    26   B MACHINERY                     20         1,495.55
    28   C ENGRAVING CO                  20           100.00
    30   D CONTROLS                      20           900.25
    32   I POWER EQUIPMENT               20           250.00
    34   ROBIN COMPANY                   20           153.00
    56   Feenala Grund Mfg.              20         4,260.00
********  End of data  ********
```

As you can see in the result table in Figure 4-6, we have a projected view (not all columns) and a selection view (not all rows – just those with VNDCLS = 20). We query a set of data and a set of data is returned to us in memory.

In the next four SQL statements, insert one record into Vendorp, update several records in Vendorp by increasing the balance owed by 20%, and then delete all the PA state records from Vendorp. As a final short exercise, delete all the remaining records from the Vendorp file.

Basic SQL Insert Statement

```
INSERT INTO SQLBOOK/VENDORP
(VNDNBR, NAME, ADDR1, CITY, STATE, ZIPCD, VNDCLS,
VNDSTS, BALOWE, SRVRTG)
VALUES(
'8020', 'Phillies Phinest',
'391 Carey Avenue',
'Wilkes-Barre',  'PA', '18702',
'20', 'A', '35700', 4)
```

Basic SQL Update Statement

```
UPDATE SQLBOOK/Vendorp
SET balowe =  balowe * 1.2
33 rows updated in VENDORP in SQLBOOK
```

Basic SQL Delete Statement

```
DELETE from SQLBOOK/Vendorp
WHERE STATE = 'PA'
14 rows deleted from VENDORP in SQLBOOK.

DELETE from SQLBOOK/Vendorp
```

Figure 4-7 Confirm Delete All Records

```
                    Confirm Statement

You are about to alter (DELETE or UPDATE) all of the records in your file(s).

Press Enter to confirm your statement to alter the entire file.
Press F12=Cancel to return and cancel your statement.
```

```
33 rows deleted from VENDORP in SQLBOOK.
```

As you can see from Figure 4-7, after we hit the enter key all the records in Vendorp are deleted and the file is empty. So that we can run more SQL statements against Vendorp data later in the book, we have built a data refresh facility that we would now call to perform the data reload function.

Basic SQL Data Control Language (DCL)

There are lots of ways to establish security with SQL and with a System i5 box. Since the System i5 comes standard with capability based addressing, security is built into the operating system at a low level and has been a hallmark of the System i5 since it was a System/38. For a number of years, IBM felt that it was sufficient to use the native security commands such as Grant Object Authority (GRTOBJAUT) and Revoke Object Authority (RVKOBJAUT). However, IBM is now focusing on all of its DB2s functioning in the same way, regardless of innate OS capabilities or not. So, several releases ago, IBM created the standard SQL interface for security and delivered it with the GRANT and REVOKE SQL commands.

In the examples below the GRANT command gives TONYS the full authority to use the Vendorp database. Yes, TONYS can delete Vendorp if he chooses. Rethinking TONYS' ability to delete the Vendorp table, the next command (REVOKE) revokes all of TONYS' authority to the Vendorp object

The Granting

```
GRANT UPDATE ON SQLBOOK/VENDORP
 TO TONYS WITH GRANT OPTION
```

```
GRANT of authority to VENDORP in SQLBOOK completed.
```

The Revoking

```
REVOKE UPDATE ON SQLBOOK/VENDORP FROM TONYS
```

```
REVOKE of authority to VENDORP in SQLBOOK
completed.
```

All data control commands in SQL are initiated with a GRANT or a REVOKE of authority and these commands use the natural AS/400 security within the AS/400 objects to protect them. The "PACKAGE" versions (Grant and Revoke authority to packages) as well as the procedure versions are very similar to the table versions of Grant and Revoke. Their function is more for program security than data security.

The various iterations and parameters of the Grant and Revoke commands are available in the IBM System i5 SQL Reference Manual as well as via the ISQL prompter. We will be studying the ISQL prompter in Chapter 9.

The intention of this book is to enable you to perform SQL functions for application development. Therefore, we will not devote any more time on security since that is an issue in itself and should be attacked by the company's security officer. In many ways, SQL security for its managed objects is no different in concept than how security is invoked across all objects in the native AS/400 environment.

Chapter Summary

Before we took a brief look at some of SQL's basic capabilities in this chapter we positioned SQL as a developer's tool, not an end-user query product. Sine this book is intended to get you up to speed with SQL in 17 easy chapters, we spend little time on the advanced facilities. In this chapter, we introduced referential constraints, triggers, and stored procedures as advanced tools that can take you the extra mile when needed.

To stage us for more to come, we introduced the table facility of SQL and then using SQL's data definition language, DDL, we examined a table created with DDS and then we performed a Create Table command with SQL to show the similarities and differences in coding the two. We created a mini file and a large file with lots of records to test our skills. We used each of the four major data manipulation language statements to work with the file after it was populated with data. We saw our Select statement access data; and our Insert statement insert a record. Then we used the Update statement and we saw how it could be used for single or multiple record updates. Finally, we used the Delete statement and we saw its utility in deleting single, multiple, or all records in a database file.

Once we had used DDL for creation and DML for manipulation, we moved to the Data Control Language facilities of SQL and we explored the Grant and Revoke commands and we demonstrated an example of each.