Tutorial

Fundamentals of SQL

SQL is an acronym for Structured Query Language. A query language is one that allows you to make a request upon a database and retrieve data that satisfies that request. For example, you can request that a database "show you all the records in the a table named Teachers that begins with the letter *B*." Or you can request that a database "Insert a record of data into a table named, Courses."

Both the DAO and ADO data access technologies that you use with Visual Basic support SQL. In fact, SQL is the way by which you work with data in ADO. A basic working knowledge of SQL is essential if you want to be at all proficient when working with databases in Visual Basic.

SQL is a language that allows you to manipulate databases to a very fine degree or granularity. SQL allows you to retrieve records, to add records, to delete records, to join tables, as well as a host of other activities.

General Syntax

The syntax of SQL is based on the use of keyword clauses used with code that identifies tables and fields within a given database.


EXAMPLE

The following example shows a SQL statement that queries a database to retrieve all the records from the table named `tblTeachers`, in which the field `LastName` begins with the character, 'h' or 'H':

```
SELECT * FROM tblTeachers WHERE tblTeachers.LastName LIKE "h*"
```

Most of the SQL keywords have meanings that correspond with their natural language equivalents. In the example above, `SELECT`, `FROM`, `WHERE`, and `LIKE` are the SQL keywords. The "*" character indicates all records. The SQL meaning of `SELECT` and `FROM` is the same as that in day to day language. `WHERE` indicates a condition. In this case the condition is `(((tblTeachers.LastName) Like h*))`, which means the data in the `LastName` field begins with the characters 'h' or 'H'.

As queries grow to handle working with data among multiple tables and fields, the SQL statements written to express them become longer. For example, the following SQL statement asks for a list of records that contains the field `LoginID` from the table `tblLogin` and fields `FirstName` and `LastName` from the table, `tblTeachers`. This query relates the tables, `tblLogin` and table, `tblTeachers` by using `LoginID` as a key field. You'll read about key fields later in this appendix.

EXAMPLE

```
SELECT tblLogin.LoginID, tblLogin.FirstName,
tblTeachers.LastName FROM tblLogin INNER JOIN
tblTeachers ON tblLogin.LoginID = tblTeachers.LoginID;
```

No question about it, the SQL statement above is quite complex. However, the complexity is not due to the language keywords, but rather to the table relationships that the keywords represent. Don't be confused right now if this SQL statement is hard to grasp. You'll build up to constructing a statement of this type. For now the thing to understand is that SQL is made up of a limited number of keywords that you use in conjunction with table structures to perform a given defined tasks.

The SQL words that are important to know are shown in Table 1.

Table 1: Important SQL Keywords

Keyword(S)	Description
*	All Records
SELECT	Select, retrieve, get
FROM	Indicates the tables from which data is to be manipulated
WHERE	Sets a condition
LIKE	Similar to
ORDER BY	The sort criteria
INNER|LEFT|RIGHT JOIN	Creates a new table from data in two related tables
AND|OR	Indicates multiple conditions
DELETE	Delete a record for a table or tables
ON	Defines a key field relationship
INSERT INTO	Add a record to a table or table
VALUES	Indicates a group of values to be added to a table

Table / Field Syntax

You indicate a field in SQL by using a notation that describes the field as part of its table, as shown in Figure 1:

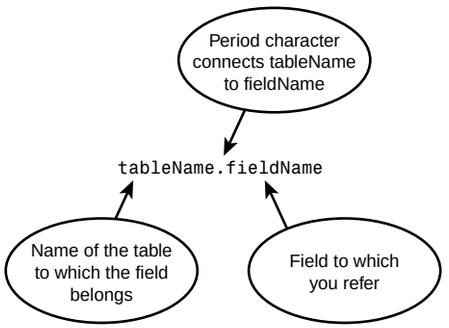

Figure 1: *In SQL you express a field name as part of a table.*

Thus, in SQL you write the field, `CustomerName` in the table, `tblCustomers`, as

```
tblCustomers.CustomerName
```

If you want to indicate the field, `SSN`, in the table `Employees`, you write

```
Employees.SSN
```

Now that you have an overview of how SQL works, let's get down to the nitty-gritty and learn how to write a simple SQL statement.

Creating a Query Using `SELECT` and `FROM` Keywords

The simplest SQL statement to write is a clause using the `SELECT` keyword. `SELECT` means "show me" or "get me." The `SELECT` keyword must be used with the `FROM` clause. Together they make the complete statement. The `FROM` keyword indicates the table or tables from which to retrieve records. The syntax for a SQL statement that uses the `SELECT` and `FROM` keywords is

```
SELECT table1.field1, table1.field2, table1.fieldn FROM table1
```

where

`SELECT` is a SQL keyword.

`table1.field1`, *`table1.field2, table1.fieldn`* are the fields that you want to display in the query.

`FROM` is a SQL keyword.

`table1` is the table from which you want to retrieve records.

Selecting All Fields

If you want to write a SQL statement that returns all fields from a given table, you use the asterisk in the place of a field name. The following example queries the table `Employees` to list all the fields within it:

```
SELECT * FROM Employees
```

Selecting One Field

You can choose to display one or many fields within a given table when using the SELECT statement. If you want to list only one field from a given table, simply indicate the field after the SELECT keyword. The following example shows you how to list only the Model field from the table, Autos:

```
SELECT Autos.Model FROM Autos
```

Selecting Many Fields

If you want to list multiple fields from a table, you write each field followed by a comma after the SELECT keyword. The following example shows you how to list the fields FirstName and LastName from the table, Students:

```
SELECT Students.FirstName, Students.LastName FROM Students
```

The next example shows you how to list the fields Make, Model, and Year from the table, Autos:

```
SELECT Autos.Make, Autos.Model, Autos.Year FROM Autos.
```

Filtering Data

If you want to retrieve records that meet a condition, an exact word match for example, you use the = sign. The following example shows you how to retrieve all records from a table, tblCourses in which the Subject is History:

```
SELECT * FROM tblCourses WHERE tblCourses. Subject)=' History '
```

Setting General Retrieval Criteria Using a LIKE Clause

You retrieve records according to a general condition by using the LIKE keyword in conjunction with using the asterisk to indicate a general search criteria. In a LIKE clause of SQL statement, the asterisk means, "any number of characters of any type." Table 2 shows examples of the different uses of the asterisk character to set a retrieval condition.

Table 2: Ways to Construct the LIKE *Keyword*

Structure	Meaning
p*	Any word that begins with and contains the character, p
*p	Any word that ends with and contains the character, p
p	Any word that contains the character, p

The following example shows you how to construct a SQL statement that returns all of the records table, tblStudents in which the field, LastName contains the character o:

```
SELECT * FROM tblStudents WHERE tblStudents.LastName LIKE '*o*'
```

CAUTION

Please notice that in SQL, `LIKE` criteria is bounded by apostrophe characters. You indicate string characters internal to a SQL statement string by using apostrophes. Some databases have SQL engines that support the use of quotations marks instead of apostrophes. However, many don't. Beware of this possibility when you are programming outside of Visual Basic/VBScript or DAO and ADO.

Using the `NOT` Keyword

If you want to retrieve data from fields that contain string data and you want to apply filtering criteria which means, "Show me all the records in which this field contains data NOT like this...", you use the `NOT` keyword in conjunction with the `LIKE` keyword. The following example shows a SQL statement that asks to see all fields in the table, `tblStudents`, in which data in the field `LastName` does not contain the character `y`:

```
SELECT * FROM tblStudents WHERE tblStudents. LastName NOT LIKE '*y*'
```

Using Inequality Operators

If you want to retrieve records according to criteria in a field that contains numeric data or dates, you can use the inequality operators as shown in Table 3:

Table 3: SQL Inequality Operators

Operator	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

The following SQL statement queries the table, `tblStudents` to show all records in which data in the field `Age` is less than 10:

```
SELECT * FROM tblStudents WHERE tblStudents.Age<10
```

If you want to see all records in which `Age` is greater than 12, you write

```
SELECT * FROM tblStudents WHERE tblStudents.Age>12
```

The following SQL statement asks to see all the records in which the data in field `Age` is not equal to 21:

```
SELECT * FROM tblStudents WHERE tblStudents.Age)<>21
```

Using Dates

You need to add a piece of special notation when you work with dates in SQL. The form to express a date is to put the date between # characters. This is how SQL knows the notation is of data type, Date. Thus if you want to express the date January 1, 2000, you write

```
#1/1/2000#
```

The following example shows a SQL statement that requests all records from the table, tblAttendance, in which data from the Date field occurred after the date July 1, 1999:

```
SELECT * FROM tblAttendance WHERE tblAttendance.Date > #7/1/1999#
```

Using the AND Keyword

It's possible to have multiple conditions in a WHERE clause of a SQL statement. You use the AND and OR keyword to combine conditions.

You use the AND keyword to filter records in which all criteria must be true for the records to be listed. For example, if you want to retrieve all records from the table, tblAttendance, in which the values in the Date fields occurs after July 1, 1999, and the values in the Status field are greater than 0, you write

```
SELECT * FROM tblAttendance
WHERE tblAttendance.Date > #7/1/1999#
AND tblAttendance.Status > 0
```

You can chain AND clauses together to accommodate more than two conditions. The following SQL statement asks for all records from the table, tblAttendance, in which the data in the Date field occurred after July 1, 1999, the values in the Status field are greater than 0, and the StudentID is "reselbob01":

```
SELECT * FROM tblAttendance
WHERE tblAttendance.Date >#7/1/1999#
AND tblAttendance.Status)>0 AND
tblAttendance.StudentID="reselbob01"
```

Using the OR Keyword

You use the OR keyword to select records in which any criteria is in force. For example, if you want to show any record in the table tblStudents in which the Age value is greater than 10 or less than 20, you write the following statement:

```
SELECT * FROM tblStudents WHERE tblStudents.Age > 10) OR tblStudents.Age < 20)
```

Be careful when using either the AND or OR keywords. The syntax is a straightforward enough affair. Nevertheless, most errors occur due to inconsistent logic. Consider the following SQL statement:

```
SELECT * FROM tblOpenInvoices
WHERE tlbStudents.Age < 10 AND tlbStudents.Age > 20
```

This statement would not return any recordset whatsoever. It is logically impossible for any record to have a value in the Age field that is both less than 10 and greater than 20.

Using the DISTINCT Keyword

The DISTINCT keyword is the way the SQL says to a *datasource*, "Show me all the records in a field without showing me duplicate values."

Consider the following database table:

ID	Beatle	Instrument	Song
1	John	Guitar	Oh Darling
2	John	Vocals	Oh Darling
3	Paul	Piano	Long and Winding Road
4	Paul	Guitar	Maybe I'm Amazed
5	Paul	Bass	Get Back
6	Paul	Vocals	Michelle
7	George	Guitar	Day Tripper
8	George	Vocals	Blue Jay Way
9	Ringo	Drums	In the End
10	Ringo	Vocals	With a Little Help from My Friends
11	John	Vocals	Julia
12	Ringo	Drums	Come Together
13	John	Vocals	Come Together
14	Paul	Vocals	Yesterday
15	Paul	Guitar	Yesterday
16	Paul	Vocals	When I'm Sixty-Four


EXAMPLE

Now, run the following SQL statement against the table:

```
SELECT Beatles.Beatle FROM Beatles
```

That SQL statement says "Show me the all the records in the field `Beatle` from the table, `Beatles`." The return recordset is


OUTPUT

Beatle
John
John
Paul
Paul
Paul
Paul
George
George
Ringo
Ringo
John
Ringo
John
Paul
Paul
Paul


EXAMPLE

Notice that although the logic is sound, the return recordset has redundant data. If we want to eliminate these redundant values and show unique values only, we use the `DISTINCT` keyword after the `SELECT` keyword, like so:

```
SELECT DISTINCT Beatles.Beatle FROM Beatles
```

When you use the `DISTINCT` keyword against the `Beatle` field only, the database returns the following table:


OUTPUT

Beatle
George
John
Paul
Ringo


EXAMPLE

When we run the `DISTINCT` keyword against the `Song` fields, we retrieve the following records:

```
SELECT DISTINCT Beatles.Song FROM Beatles
```

OUTPUT

Song
Blue Jay Way
Come Together
Day Tripper
Get Back
In the End
Julia
Long and Winding Road
Maybe I'm Amazed
Michelle
Oh Darling
When I'm Sixty-Four
With a Little Help from My Friends
Yesterday

EXAMPLE

Using the DISTINCT Keyword Against Multiple Fields

You can use the `DISTINCT` keyword against multiple fields. Let's use the `Beatle` table again. Let's say you want to see a list that shows each Beatle with the instruments each musician plays. If you ran the SQL statement without the `DISTINCT` keyword, here is the return recordset you would get:

```
SELECT Beatles.Beatle, Beatles.Instrument FROM Beatles
```

OUTPUT

Beatle	Instrument
John	Guitar
John	Vocals
Paul	Piano
Paul	Guitar
Paul	Bass
Paul	Vocals
George	Guitar
George	Vocals
Ringo	Drums
Ringo	Vocals
John	Vocals
Ringo	Drums

continues

continued

Beatle	Instrument
John	Vocals
Paul	Vocals
Paul	Guitar
Paul	Vocals

Now, let's run the SQL statement using the `DISTINCT` keyword:

```
SELECT DISTINCT Beatles.Beatle, Beatles.Instrument FROM Beatles
```

Beatle	Instrument
George	Guitar
George	Vocals
John	Guitar
John	Vocals
Paul	Bass
Paul	Guitar
Paul	Piano
Paul	Vocals
Ringo	Drums
Ringo	Vocals

Notice that the list is shortened considerably. That's because the logic of the `DISTINCT` keyword says, "Show me all records in both the `Beatle` and `Instrument` fields, but make sure that you do not show data that is redundant with *respect to both fields*. If a record in the field `Beatle` is redundant with another record in the `Beatle` field, that's all right as long the data in the associated `Instrument` field is different, and vice versa."

Using the INSERT Keyword

Up until now, we've been working with SQL statements that list data only. We haven't changed the structure of any underlying table in terms of adding or removing records, which is something that SQL is equipped to do quite well. The point of this section is to teach you how to use SQL to add records to a table within a given database.

You add a record to a table or recordset using the SQL INSERT INTO keywords. The syntax for the statement is as follows:

```
INSERT INTO tableName (strField, numField, boolField)
VALUES ("strValue", 123, TRUE)
```

where

INSERT INTO are the SQL keywords indicating a record is to be added to a table.

tableName is the table into which data is going to be inserted.

strField, numField, boolField are fields in the table into which data is going to be inserted.

VALUE is the SQL keyword indicating the values to add.

"strValue", 123, TRUE are the values of data that are being inserted into the table. Please notice that the values' data types must be consistent with the data types of the fields defined in the INSERT INTO clause.

The following example shows you the SQL statement that adds student data to the fields, FirstName, LastName, and Age of the table, tblStudents:

```
INSERT INTO tblStudnets (FirstName, LastName, Age)
VALUES ('Bugglesworth', 'Bunny', 11)
```

The next example adds a record with the string value "Peter" to the Beatle field of the Beatles table:

```
INSERT INTO Beatles (Beatle) VALUES ('Peter')
```

You need to know some things about using INSERT INTO. First, the data you are going to insert into the recordset must be in the same order as the fields listing. In the students example above, the fields of the table, tblStudents are listed in the order, FirstName, LastName and Age. The data in the VALUES clause is 'Bugglesworth', 'Bunny', and 11. The result is that the string values Bugglesworth and Bunny are added to the fields FirstName and LastName, first and last names respectively, and the numeric value 11 is added to the field Age.

The second item you need to know is that both fields names and corresponding values must be enclosed in parentheses.

The last item that you must remember is to enclose VALUE data of type String in single quote marks, not standard quotation marks. Do not enclose the field names within the INSERT INTO clause with quotation marks of any kind!

Removing Records Using the DELETE Keyword

You permanently remove a record or records from a table or recordset within a database by using the SQL keyword DELETE.

The syntax for the DELETE clause is

```
DELETE FROM tableName WHERE tableName.fieldName = criteria
```

where

DELETE FROM are the SQL keywords indicating the table from which to remove records.

tableName is the table from which records will be removed.

WHERE is the SQL keyword indicating a condition.

tableName.fieldName = criteria is the condition criteria.

You can think of the DELETE keyword as the inverse of the SELECT keyword.

The following SQL statement deletes are records from the table, Beatles in which the value in the field Beatle is Peter:

```
DELETE FROM Beatles WHERE Beatle='Peter'
```

The DELETE keyword is straightforward. There is not a lot to it. You just define the record(s) you want to delete and then delete them.

Working with Relational Fields

As you begin to work with more advanced databases, you'll find that many times the information you need is spread out over many tables. In scenarios such as this, the bulk of your programming activity is to create queries that combine the various tables to get the information that you want. The formal term for combining tables is *relating* tables. Databases that spread information over multiple tables are called *relational databases*. In this section, you are going to learn how to use SQL to create queries that return data from multiple tables.

SQL allows you write queries that relate data over multiple tables by using the JOIN and ON keywords. Before we go into the nuts and bolts of the JOIN statement, let's take a moment to understand how you relate tables to each other.

Working with a Key Field

A relational database allows you to combine data in two tables in order to create a third table that contains new information. The mechanism by which you join the two fields is the *key field*. A key field is a field that contains data that is common to both tables that you are joining. When using

key fields to relate data, at least one table's key field must contain unique values. If it doesn't, the tables have no reliable way to combine.

A good example of a key field that is in most common use in day to day business is the social security number. It's a field that is common to many types of tables—tax tables, driver's licenses, bank accounts, and so on. If one table has a social security number, it's easy to relate it to records in any other table in which that social security number exists. For instance, if a bank wants to report banking activity over $10,000 to the government, it simply passes the social security number of the account holder in question to the IRS. The IRS looks up the SSN against its records to do the rest.

Figure 2 shows you how a social security number is used as a key field to relate two tables together.

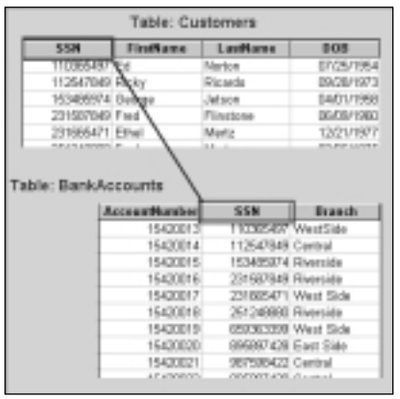

Figure 2: *You use key fields to relate tables in a database.*

Using the `JOIN` Keyword

Now that you understand what relational databases are about and how to use key fields to relate tables, let's take a look at the SQL keywords you use to create statements that can relate tables within a database.

You use the `JOIN` and `ON` keywords to relate tables in database. Also, there are few flavors to the `JOIN` keyword. The illustrations below show you each.

`INNER JOIN`

An `INNER JOIN`, which is the most common, means you list only the records from both tables in which the related records have matching key field values (see Figure 3).

RIGHT, LEFT JOIN

RIGHT JOIN and LEFT JOIN mean that all records from one table are shown even if there are no matching records in the other table that match the key field value. Figure 4 illustrates a RIGHT JOIN. Figure 5 illustrates a LEFT JOIN).

Figure 3: *INNER JOIN shows only those records that match the key fields.*

Figure 4: *A RIGHT JOIN between tables means all the records in the table to the left will be displayed even if the table on the right does not have a matching key field.*

Figure 5: *A `LEFT JOIN` between tables means all the records in the table to the right will be displayed even if the table on the left does not have a matching key field.*

Relating Tables Using SQL

The SLQ clause that relates two tables uses the `(INNER|RIGHT|LEFT)`, `JOIN`, and `ON` keywords.

The syntax for a `JOIN` clause is

```
SELECT tableOne.fieldOne, tableOne.fieldOne,
TableTwo.fieldOne
FROM tableOne INNER|LEFT|RIGHT JOIN tableTwo
ON tabelOne.KeyField = tableTwo.KeyField
```

> `SELECT tableOne.fieldOne, tableOne.fieldOne, TableTwo.fieldOne` indicates the list of fields to display.
>
> `FROM tableOne INNER|LEFT|RIGHT JOIN tableTwo` indicates the tables to join.
>
> `ON tableOne.KeyField = tableTwo.KeyField` indicates the names of the keyfields in each table by which to relate the tables.

To indicate the fields you want in the recordset that results from joining two tables, you create a `SELECT` clause as you would for any SQL statement. Then, you use the `JOIN` key word to indicate the tables to join. You use the `INNER`, `RIGHT`, or `LEFT` keyword to indicate the type of `JOIN`. You use the `ON`

keyword to indicate the key field. The following example shows you how to relate the Customers table to the BankAccounts table using the SSN field as the key field. This example is an INNER JOIN. Therefore, only records in both the Customers and BankAccounts field that have a common SSN are listed (see Figure 3):


EXAMPLE

```
SELECT Customers.FirstName, Customers.LastName, BankAccounts.Branch
FROM BankAccounts INNER JOIN Customers
ON BankAccounts.SSN = Customers.SSN
```

In the above example, only the FirstName, LastName, and SSN fields from the Customer table are listed as well as the Branch field from the BankAccounts field. Notice, too, that although the SSN field is used as the key field, it is not displayed. There is no hard and fast rule that says you need to display a key field.


EXAMPLE

The following JOIN statement is a modification of the previous one. In this case a RIGHT JOIN is used to show all the records in the Customers table that has an associated SSN, even if the BankAccounts table does not contain the given SSN (see Figure 4):

```
SELECT Customers.FirstName, Customers.LastName, BankAccounts.Branch
FROM BankAccounts RIGHT JOIN Customers
ON BankAccounts.SSN = Customers.SSN
```

Working with Table Position in a JOIN

You need to be aware that the order in which tables appear in a SQL statement determines the way the JOIN keyword affects them. In the example above, the table, BankAccounts appears before the RIGHT JOIN keywords. The table Customers appears after. SQL interprets this to mean that Customers will be the table in which all records will be shown. The RIGHT JOIN will be invoked upon Customers. It's as if the table Customers is "looking" to the right for the key field. Were the order to be reversed as

```
FROM Customers RIGHT JOIN BankAccounts
```

the RIGHT JOIN would be invoked upon BankAccounts. This means that the recordset would display all records in the BankAccounts table, even if there were no records with matching SSN values in the Customers table.

Another way to think about it is that changing the order of tables in a RIGHT JOIN clause is the same as changing a RIGHT JOIN to a LEFT JOIN.

Sorting Data Using SQL

The last piece of business that you need to learn about SQL is how to sort. You sort in SQL by writing an `ORDER BY` clause, which you add on to a `SELECT` statement. The syntax for the `ORDER BY` clause is

```
ORDER BY tableName.fieldName ASC
```

where

`ORDER BY` are the SQL key words indicating sorting.

`TableName.fieldName` is the field upon which you want to sort.

`ASC` is a key work indicating ordering for A–Z. If you want to order from Z–A, use the keyword `DESC`. The default is `ASC`. Numeric values are sort lowest to highest when you use `ASC`. `DESC` sorts from highest to lowest.

The following SQL statement is a modification of the recordsets created in the last section. The SQL statement returns a recordset that is created by combining two tables, `Customers` and `BankAccounts`, on an `INNER JOIN` using the `SSN` field as the key field. The recordset is sorted from A–Z according to the values in the `Customers.LastName` field.


EXAMPLE

```
SELECT Customers.FirstName, Customers.LastName, BankAccounts.Branch
FROM BankAccounts INNER JOIN Customers
ON BankAccounts.SSN = Customers.SSN
ORDER BY Customers.LastName ASC
```

Figure 6 displays the results of the query.


OUTPUT

FirstName	LastName	Branch
Fred	Flintstone	Central
Fred	Flintstone	East Side
Fred	Flintstone	Riverside
George	Jetson	Central
George	Jetson	Riverside
Ralph	Kramden	Central
Ralph	Kramden	East Side
Fred	Mertz	Riverside
Ethel	Mertz	West Side
Ed	Norton	West Side
Ricky	Ricardo	Central
Barney	Rubble	West Side
Cosmo	Spacely	Central

Figure 6: *The `ORDER BY` keywords allows you to sort data on a particular field.*

Multiple Level Sorts

You can have more than one set of sort filters in force in a SQL sort. The way you do multiple level sorts is to keep adding fields to the `ORDER BY` clause. The first field added is the first sort. The second field added is the second sort. The third field added is the third sort, and so forth

For example, the following SQL statement sorts all the records in the resulting recordset, first by the `Customers.LastName` field from A–Z. Then, if multiple branches occur for any given customer, the values in the `BankAccounts.Branch` field are sorted Z–A.

```
SELECT Customers.FirstName, Customers.LastName, BankAccounts.Branch
FROM BankAccounts INNER JOIN Customers
ON BankAccounts.SSN = Customers.SSN
ORDER BY Customers.LastName ASC, BankAccounts.Branch DESC
```

The results of this sort are shown in Figure 7:

FirstName	LastName	Branch
Fred	Flintstone	Riverside
Fred	Flintstone	East Side
Fred	Flintstone	Central
George	Jetson	Riverside
George	Jetson	Central
Ralph	Kramden	East Side
Ralph	Kramden	Central
Fred	Mertz	Riverside
Ethel	Mertz	West Side
Ed	Norton	West Side
Ricky	Ricardo	Central
Barney	Rubble	West Side
Cosmo	Spacely	Central

Figure 7: *You can use `ORDER BY` to do sorts over more than one field in a recordset.*

Sorting is not rocket science but it does take practice. You need to spend as much time understanding the logic of a given sort order as you do to making sure your SQL syntax is correct. Again, practice makes perfect.

TIP

For a complete online glossary of SQL terms, see the Microsoft Developer Network Platform SDK documentation at `http://msdn.microsoft.com/library/psdk/sql/gloss01.htm`.