Fundamental Microsoft Jet SQL for Access 2000

Acey James Bunch
Microsoft Corporation

February 2000

Applies To: Microsoft® Access 2000

Summary: This article discusses the basic mechanics of using Jet SQL to work with data in an Access 2000 database. It also delves into using SQL to create and alter a database's structure. If you're new to manipulating data with SQL in an Access database, this article is a great place to start. (21 printed pages)

Download AcFundSQL.exe.

Contents

Introduction

This is the first in a series of articles that explain what SQL is and how you can use it in your Microsoft® Access 2000 applications. There are three articles in all: a fundamental, an intermediate, and an advanced article. The articles are designed to progressively show the syntax and methods for using SQL, and to bring out those features of SQL that are new to Access 2000.

SQL Defined

To really gain the benefit and power of SQL, you must first come to a basic understanding of what it is and how you can use it.

What Is Structured Query Language?

SQL stands for *Structured Query Language* and is sometimes pronounced as "sequel." At its simplest, it is the language that is used to extract, manipulate, and structure data that resides in a relational database management system (RDBMS). In other words, to get an answer from your database, you must ask the question in SQL.

Why and Where Would You Use SQL?

You may not know it, but if you've been using Access, you've also been using SQL. "No!" you may say. "I've never used anything called SQL." That's because Access does such a great job of using it for you. The thing to remember is that for every data-oriented request you make, Access converts it to SQL under the covers.

SQL is used in a variety of places in Access. It is used of course for queries, but it is also used to build reports, populate list and combo boxes, and drive data-entry forms. Because SQL is so prevalent throughout Access, understanding it will greatly improve your ability to take control of all of the programmatic power that Access gives you.

> **Note** The particular dialect of SQL discussed in this article applies to version 4.0 of the Microsoft Jet database engine. Although many of the SQL statements will work in other databases, such as Microsoft SQL ServerÂ™, there are some differences in syntax. To identify the correct SQL syntax, consult the documentation for the database system you are using.

Data Definition Language

Data definition language (DDL) is the SQL language, or terminology, that is used to manage the database objects that contain data. Database objects are tables, indexes, or relationships—anything that has to do with the structure of the database—but not the data itself. Within SQL, certain keywords and clauses are used as the DDL commands for a relational database.

Data Manipulation Language

Data manipulation language (DML) is the SQL language, or terminology, that is used to manage the data within the database. DML has no effect on the structure of the database; it is only used against the actual data. DML is used to extract, add, modify, and delete information contained in the relational database tables.

ANSI and Access 2000

ANSI stands for the *American National Standards Institute*, which is a nationally recognized standards-setting organization that has defined a base standard for SQL. The most recently defined standard is SQL-92, and Access 2000 has added many new features to conform more closely to the standard, although some of the new features are available only when you are using the Jet OLE DB provider. However, Access has also maintained compliance with previous versions to allow for the greatest flexibility. Access also has some extra features not yet defined by the standard that extend the power of SQL.

To understand more about OLE DB and how it fits into the Microsoft Universal Data Access strategy, visit the Universal Data Access Web site at http://www.microsoft.com/data/.

SQL Coding Conventions

Throughout this article, you will notice a consistent method of SQL coding conventions. As with all coding conventions, the idea is to display the code in such a way as to make it easy to read and understand. This is accomplished by using a mix of white space, new lines, and uppercase keywords. In general, use uppercase for all SQL keywords, and if you must break the line of SQL code, try to do so with a major section of the SQL statement. You'll get a better feel for it after seeing a few examples.

Poorly formatted SQL code

```
CREATE TABLE tblCustomers (CustomerID INTEGER NOT NULL,[Last Name] TEXT(50) NOT NULL,
[First Name] TEXT(50) NOT NULL,Phone TEXT(10),Email TEXT(50))
```

Well-formatted SQL code

```
CREATE TABLE tblCustomers
    (CustomerID INTEGER NOT NULL,
    [Last Name] TEXT(50) NOT NULL,
    [First Name] TEXT(50) NOT NULL,
    Phone TEXT(10),
    Email TEXT(50))
```

Using Data Definition Language

When you are manipulating the structure of a database, there are three primary objects that you will work with: tables, indexes, and relationships.

- Tables are the database structure that contains the physical data, and they are organized by their *column*s (or *fields*) and *rows* (or *records*).
- Indexes are the database objects that define how the data in the tables is arranged and sorted in memory.
- Relationships define how one or more tables relate to one or more other tables.

All three of these database objects form the foundation for all relational databases.

Creating and Deleting Tables

Tables are the primary building blocks of a relational database. A table contains rows (or records) of data, and each row is organized into a finite number of columns (or fields). To build a new table in Access by using Jet SQL, you must name the table, name the fields, and define the type of data that the fields will contain. Use the

CREATE TABLE statement to define the table in SQL. Let's suppose that we are building an invoicing database, so we will start with building the initial customers table.

```
CREATE TABLE tblCustomers
    (CustomerID INTEGER,
    [Last Name] TEXT(50),
    [First Name] TEXT(50),
    Phone TEXT(10),
    Email TEXT(50))
```

Notes

- If a field name includes a space or some other nonalphanumeric character, you must enclose that field name within square brackets ([ ]).
- If you do not declare a length for text fields, they will default to 255 characters. For consistency and code readability, you should always define your field lengths.
- For more information about the types of data that can be used in field definitions, type **SQL data types** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then click **Search**.

You can declare a field to be NOT NULL, which means that null values cannot be inserted into that particular field; a value is always required. A null value should not be confused with an empty string or a value of 0, it is simply the database representation of an unknown value.

```
CREATE TABLE tblCustomers
    (CustomerID INTEGER NOT NULL,
    [Last Name] TEXT(50) NOT NULL,
    [First Name] TEXT(50) NOT NULL,
    Phone TEXT(10),
    Email TEXT(50))
```

To remove a table from the database, use the DROP TABLE statement.

```
DROP TABLE tblCustomers
```

Working with Indexes

An *index* is an external data structure used to sort or arrange pointers to data in a table. When you apply an index to a table, you are specifying a certain arrangement of the data so that it can be accessed more quickly. However, if you apply too many indexes to a table, you may slow down the performance because there is extra overhead involved in maintaining the index, and because an index can cause locking issues when used in a multiuser environment. Used in the correct context, an index can greatly improve the performance of an application.

To build an index on a table, you must name the index, name the table to build the index on, name the field or fields within the table to use, and name the options you want to use. You use the CREATE INDEX statement to build the index. For example, here's how you would build an index on the customers table in the invoicing database mentioned earlier.

```
CREATE INDEX idxCustomerID
    ON tblCustomers (CustomerID)
```

Indexed fields can be sorted in one of two ways: ascending (ASC) or descending (DESC). The default order is ascending, and it does not have to be declared. If you use ascending order, the data will be sorted from 1 to 100. If you specify descending order, the data will be sorted from 100 to 1. You should declare the sort order with each field in the index.

```
CREATE INDEX idxCustomerID
    ON tblCustomers (CustomerID DESC)
```

There are four main options that you can use with an index: PRIMARY, DISALLOW NULL, IGNORE NULL, and

UNIQUE. The PRIMARY option designates the index as the primary key for the table. You can have only one primary key index per table, although the primary key index can be declared with more than one field. Use the WITH keyword to declare the index options.

```
CREATE INDEX idxCustomerID
    ON tblCustomers (CustomerID)
    WITH PRIMARY
```

To create a primary key index on more than one field, include all of the field names in the field list.

```
CREATE INDEX idxCustomerName
    ON tblCustomers ([Last Name], [First Name])
    WITH PRIMARY
```

The DISALLOW NULL option prevents insertion of null data in the field. (This is similar to the NOT NULL declaration used in the CREATE TABLE statement.)

```
CREATE INDEX idxCustomerEmail
    ON tblCustomers (Email)
    WITH DISALLOW NULL
```

The IGNORE NULL option causes null data in the table to be ignored for the index. That means that any record that has a null value in the declared field will not be used (or counted) in the index.

```
CREATE INDEX idxCustomerLastName
    ON tblCustomers ([Last Name])
    WITH IGNORE NULL
```

In addition to the PRIMARY, DISALLOW NULL, and IGNORE NULL options, you can also declare the index as UNIQUE, which means that only unique, non-repeating values can be inserted in the indexed field.

```
CREATE UNIQUE INDEX idxCustomerPhone
    ON tblCustomers (Phone)
```

To remove an index from a table, use the DROP INDEX statement.

```
DROP INDEX idxName
    ON tblCustomers
```

Defining Relationships Between Tables

Relationships are the established associations between two or more tables. Relationships are based on common fields from more than one table, often involving primary and foreign keys.

A *primary key* is the field (or fields) that is used to uniquely identify each record in a table. There are three requirements for a primary key: It cannot be null, it must be unique, and there can be only one defined per table. You can define a primary key either by creating a primary key index after the table is created, or by using the CONSTRAINT clause in the table declaration, as shown in the examples later in this section. A constraint limits (or constrains) the values that are entered in a field. For more information about constraints, see the article "Intermediate Microsoft Jet SQL for Access 2000."

A *foreign key* is a field (or fields) in one table that references the primary key in another table. The data in the fields from both tables is exactly the same, and the table with the primary key record (the *primary table*) must have existing records before the table with the foreign key record (the *foreign table*) has the matching or related records. Like primary keys, you can define foreign keys in the table declaration by using the CONSTRAINT clause.

There are essentially three types of relationships:

- **One-to-one** For every record in the primary table, there is one and only one record in the foreign table.
- **One-to-many** For every record in the primary table, there are one or more related records in the foreign table.

- **Many-to-many** For every record in the primary table, there are many related records in the foreign table, and for every record in the foreign table, there are many related records in the primary table.

For example, let's add an invoices table to our invoicing database. Every customer in our customers table can have many invoices in our invoices table—this is a classic one-to-many scenario. We will take the primary key from the customers table and define it as the foreign key in our invoices table, thereby establishing the proper relationship between the tables.

When defining the relationships between tables, you must make the CONSTRAINT declarations at the field level. This means that the constraints are defined within a CREATE TABLE statement. To apply the constraints, use the CONSTRAINT keyword after a field declaration, name the constraint, name the table that it references, and name the field or fields within that table that will make up the matching foreign key.

The following statement assumes that the tblCustomers table has already been built, and that it has a primary key defined on the CustomerID field. The statement now builds the tblInvoices table, defining its primary key on the InvoiceID field. It also builds the one-to-many relationship between the tblCustomers and tblInvoices tables by defining another CustomerID field in the tblInvoices table. This field is defined as a foreign key that references the CustomerID field in the Customers table. Note that the name of each constraint follows the CONSTRAINT keyword.

```
CREATE TABLE tblInvoices
    (InvoiceID INTEGER CONSTRAINT PK_InvoiceID PRIMARY KEY,
    CustomerID INTEGER NOT NULL CONSTRAINT FK_CustomerID
        REFERENCES tblCustomers (CustomerID),
    InvoiceDate DATETIME,
    Amount CURRENCY)
```

Note that the primary key index (PK_InvoiceID) for the invoices table is declared within the CREATE TABLE statement. To enhance the performance of the primary key, an index is automatically created for it, so there's no need to use a separate CREATE INDEX statement.

Now let's create a shipping table that will contain each customer's shipping address. Let's assume that there will be only one shipping record for each customer record, so we will be establishing a one-to-one relationship.

```
CREATE TABLE tblShipping
    (CustomerID INTEGER CONSTRAINT PK_CustomerID PRIMARY KEY
        REFERENCES tblCustomers (CustomerID),
    Address TEXT(50),
    City TEXT(50),
    State TEXT(2),
    Zip TEXT(10))
```

Note that the CustomerID field is both the primary key for the shipping table and the foreign key reference to the customers table.

> **Note** When you are creating a one-to-one relationship by using DDL statements, the Access user interface may display the relationship as a one-to-many relationship. To correct this problem, after the one-to-one relationship has been created, open the Relationships window by clicking **Relationships** on the **Tools** menu. Make sure that the affected tables have been added to the Relationships window, and then double-click the link between the tables to open the **Edit Relationships** dialog box. Click the **Join Type** button to open the **Join Properties** dialog box. You don't have to select an option, just click **OK** to close the dialog box, and then click **OK** to close the **Edit Relationships** dialog box. The one-to-one relationship should now be displayed correctly.

For more information about relationships and how they work, type **relationships** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then click **Search**.

Using Data Manipulation Language

DML is all about working with the data that is stored in the database tables. Not only is DML used for retrieving the data, it is also used for creating, modifying, and deleting it.

Retrieving Records

The most basic and most often used SQL statement is the SELECT statement. SELECT statements are the workhorses of all SQL statements, and they are commonly referred to as *select queries*. You use the SELECT statement to retrieve data from the database tables, and the results are usually returned in a set of records (or rows) made up of any number of fields (or columns). You must designate which table or tables to select from with the FROM clause. The basic structure of a SELECT statement is:

```
SELECT field list
    FROM table list
```

To select all the fields from a table, use an asterisk (*). For example, the following statement selects all the fields and all the records from the customers table:

```
SELECT *
    FROM tblCustomers
```

To limit the fields retrieved by the query, simply use the field names instead. For example:

```
SELECT [Last Name], Phone
    FROM tblCustomers
```

To designate a different name for a field in the result set, use the AS keyword to establish an alias for that field.

```
SELECT CustomerID AS [Customer Number]
    FROM tblCustomers
```

Restricting the Result Set

More often than not, you will not want to retrieve all records from a table. You will want only a subset of those records based on some qualifying criteria. To qualify a SELECT statement, you must use a WHERE clause, which will allow you to specify exactly which records you want to retrieve.

```
SELECT *
    FROM tblInvoices
    WHERE CustomerID = 1
```

Note the `CustomerID = 1` portion of the WHERE clause. A WHERE clause can contain up to 40 such expressions, and they can be joined with the **And** or **Or** logical operators. Using more than one expression allows you to further filter out records in the result set.

```
SELECT *
    FROM tblInvoices
    WHERE CustomerID = 1 AND InvoiceDate > #01/01/98#
```

Note that the date string is enclosed in number signs (#). If you are using a regular string in an expression, you must enclose the string in single quotation marks ('). For example:

```
SELECT *
    FROM tblCustomers
    WHERE [Last Name] = 'White'
```

If you do not know the whole string value, you can use wildcard characters with the **Like** operator.

```
SELECT *
    FROM tblCustomers
    WHERE [Last Name] LIKE 'W*'
```

There are a number of wildcard characters to choose from, and the following table details what they are and what they can be used for.

Wildcard character	Description
* or %	Zero or more characters
? or _ (underscore)	Any single character

#	Any single digit (0-9)
[*charlist*]	Any single character in *charlist*
[!*charlist*]	Any single character not in *charlist*

> **Note** The % and _ (underscore) wildcard characters should be used only through the Jet OLE DB provider and ActiveX® Data Objects (ADO) code. They will be treated as literal characters if they are used though the Access SQL View user interface or Data Access Objects (DAO) code.

For more information about using the **Like** operator with wildcard characters, type **wildcard characters** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then click **Search**.

Sorting the Result Set

To specify a particular sort order on one or more fields in the result set, use the optional ORDER BY clause. As explained earlier in the "Working with Indexes" section, records can be sorted in either ascending (ASC) or descending (DESC) order; ascending is the default.

Fields referenced in the ORDER BY clause do not have to be part of the SELECT statement's field list, and sorting can be applied to string, numeric, and date/time values. Always place the ORDER BY clause at the end of the SELECT statement.

```
SELECT *
    FROM tblCustomers
    ORDER BY [Last Name], [First Name] DESC
```

You can also use the field numbers (or positions) instead of field names in the ORDER BY clause.

```
SELECT *
    FROM tblCustomers
    ORDER BY 2, 3 DESC
```

For more information about using the ORDER BY clause, type **ORDER BY clause** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then click **Search**.

Using Aggregate Functions to Work with Values

Aggregate functions are used to calculate statistical and summary information from data in tables. These functions are used in SELECT statements, and all of them take fields or expressions as arguments.

To count the number of records in a result set, use the **Count** function. Using an asterisk with the **Count** function causes **Null** values to be counted as well.

```
SELECT Count(*) AS [Number of Invoices]
    FROM tblInvoices
```

To count only non-**Null** values, use the **Count** function with a field name:

```
SELECT Count(Amount) AS
    [Number of Valid Invoice Amounts]
    FROM tblInvoices
```

To find the average value for a column or expression of numeric data, use the **Avg** function:

```
SELECT Avg(Amount) AS [Average Invoice Amount]
    FROM tblInvoices
```

To find the total of the values in a column or expression of numeric data, use the **Sum** function:

```
SELECT Sum(Amount) AS [Total Invoice Amount]
    FROM tblInvoices
```

To find the minimum value for a column or expression, use the **Min** function:

```
SELECT Min(Amount) AS [Minimum Invoice Amount]
    FROM tblInvoices
```

To find the maximum value for a column or expression, use the **Max** function:

```
SELECT Max(Amount) AS [Maximum Invoice Amount]
    FROM tblInvoices
```

To find the first value in a column or expression, use the **First** function:

```
SELECT First(Amount) AS [First Invoice Amount]
    FROM tblInvoices
```

To find the last value in a column or expression, use the **Last** function:

```
SELECT Last(Amount) AS [Last Invoice Amount]
    FROM tblInvoices
```

For more information about using the aggregate functions, type **SQL aggregate functions** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then click **Search**.

Grouping Records in a Result Set

Sometimes there are records in a table that are logically related, as in the case of the invoices table. Since one customer can have many invoices, it could be useful to treat all the invoices for one customer as a group, in order to find statistical and summary information about the group.

The key to grouping records is that one or more fields in each record must contain the same value for every record in the group. In the case of the invoices table, the CustomerID field value is the same for every invoice a particular customer has.

To create a group of records, use the GROUP BY clause with the name of the field or fields you want to group with.

```
SELECT CustomerID, Count(*) AS [Number of Invoices],
    Avg(Amount) AS [Average Invoice Amount]
    FROM tblInvoices
    GROUP BY CustomerID
```

Note that the statement will return one record that shows the customer ID, the number of invoices the customer has, and the average invoice amount, for every customer who has an invoice record in the invoices table. Because each customer's invoices are treated as a group, we are able to count the number of invoices, and then determine the average invoice amount.

You can specify a condition at the group level by using the HAVING clause, which is similar to the WHERE clause. For example, the following query returns only those records for each customer whose average invoice amount is less than 100:

```
SELECT CustomerID, Count(*) AS [Number of Invoices],
    Avg(Amount) AS [Average Invoice Amount]
    FROM tblInvoices
    GROUP BY CustomerID
    HAVING Avg(Amount) < 100
```

For more information about using the GROUP BY clause, type **GROUP BY clause** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then click **Search**.

Inserting Records into a Table

There are essentially two methods for adding records to a table. The first is to add one record at a time; the second is to add many records at a time. In both cases, you use the SQL statement INSERT INTO to accomplish the task. INSERT INTO statements are commonly referred to as *append queries*.

To add one record to a table, you must use the field list to define which fields to put the data in, and then you must supply the data itself in a value list. To define the value list, use the VALUES clause. For example, the following statement will insert the values "1", "Kelly", and "Jill" into the CustomerID, Last Name, and First Name fields, respectively.

```
INSERT INTO tblCustomers (CustomerID, [Last Name], [First Name])
    VALUES (1, 'Kelly', 'Jill')
```

You can omit the field list, but only if you supply all the values that record can contain.

```
INSERT INTO tblCustomers
    VALUES (1, Kelly, 'Jill', '555-1040', 'someone@microsoft.com')
```

To add many records to a table at one time, use the INSERT INTO statement along with a SELECT statement. When you are inserting records from another table, each value being inserted must be compatible with the type of field that will be receiving the data. For more information about data types and their usage, see "Intermediate Microsoft Jet SQL for Access 2000."

The following INSERT INTO statement inserts all the values in the CustomerID, Last Name, and First Name fields from the tblOldCustomers table into the corresponding fields in the tblCustomers table.

```
INSERT INTO tblCustomers (CustomerID, [Last Name], [First Name])
    SELECT CustomerID, [Last Name], [First Name]
    FROM tblOldCustomers
```

If the tables are defined exactly alike, you leave can out the field lists.

```
INSERT INTO tblCustomers
    SELECT * FROM tblOldCustomers
```

For more information about using the INSERT INTO statement, type **INSERT INTO statement** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then click **Search**.

Updating Records in a Table

To modify the data that is currently in a table, you use the UPDATE statement, which is commonly referred to as an *update query*. The UPDATE statement can modify one or more records and generally takes this form:

```
UPDATE table name
    SET field name = some value
```

To update all the records in a table, specify the table name, and then use the SET clause to specify the field or fields to be changed.

```
UPDATE tblCustomers
    SET Phone = 'None'
```

In most cases, you will want to qualify the UPDATE statement with a WHERE clause to limit the number of records changed.

```
UPDATE tblCustomers
    SET Email = 'None'
    WHERE [Last Name] = 'Smith'
```

For more information about using the UPDATE statement, type **UPDATE statement** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then click **Search**.

Deleting Records from a Table

To delete the data that is currently in a table, you use the DELETE statement, which is commonly referred to as a *delete query*, also known as truncating a table. The DELETE statement can remove one or more records from a table and generally takes this form:

```
DELETE FROM table list
```

The DELETE statement does not remove the table *structure*, only the *data* that is currently being held by the table structure. To remove all the records from a table, use the DELETE statement and specify which table or tables you want to delete all the records from.

```
DELETE FROM tblInvoices
```

In most cases, you will want to qualify the DELETE statement with a WHERE clause to limit the number of records to be removed.

```
DELETE FROM tblInvoices
    WHERE InvoiceID = 3
```

If you want to remove data only from certain fields in a table, use the UPDATE statement and set those fields equal to NULL, but only if they are nullable fields. For more information about nullable fields, see "Intermediate Microsoft Jet SQL for Access 2000."

```
UPDATE tblCustomers
    SET Email = Null
```

For more information about using the DELETE statement, type **DELETE statement** in the Office Assistant or on the **Answer Wizard** tab in the Microsoft Access Help window, and then click **Search**.

Using SQL in Access

Now that we've had a basic overview of the SQL syntax, let's look at some of the ways we can use it in an Access application. To do this, we'll use the sample database included with this article. Through queries and sample code, the acFundSQL.mdb sample demonstrates the different SQL statements discussed in this article.

> **Note** Many of the sample queries used in acFundSQL.mdb depend on certain tables existing and containing data. Because some of the queries in acFundSQL.mdb alter the data or the database structure, you may eventually have difficulty running other queries due to missing or altered data, tables, or indexes. If this problem occurs, open the frmResetTables form and click the Reset Tables button to re-create the tables and their original default data. To manually step through the reset-table process, execute the following queries in the order they are listed:
>
> Drop Shipping Table
> Drop Invoices Table
> Drop Customers Table
> Create Customers Table
> Create Invoices Table
> Create Shipping Table
> Populate Customers
> Populate Invoices
> Populate Shipping

Building Queries

Queries are SQL statements that are saved in an Access database and can be used at any time, either directly from the Access user interface or from the Visual Basic® for Applications (VBA) programming language. You can build queries by using query Design view, which greatly simplifies the building of SQL statements, or you can build queries by entering SQL statements directly in the SQL view window.

As mentioned at the beginning of this article, Access converts all data-oriented tasks in the database into SQL statements. To demonstrate this behavior, let's build a query in query Design view.

1. Open the acFundSQL.mdb database.

1. Make sure that the tblCustomers table has been created and that it contains some data.
2. In the Database window, click **Queries** under **Objects**, and then click **New** on the Database window toolbar.
3. In the **New Query** dialog box, click **Design View**, and then click **OK**.

4. In the **Show Table** dialog box, click **tblCustomers**, click **Add**, and then click **Close**.
5. In the tblCustomers field list, click the asterisk (*****) and drag it to the first field in the query design grid.
6. On the **View** menu, click **SQL View**. This opens the SQL view window and displays the SQL syntax that Access is using for this query.

   Note This query is similar to the Select All Customers query already saved in the acFundSQL database.

Specifying a Data Source

To make a connection to data in the database's tables, Access objects use data source properties. For example, a form has a **RecordSource** property that connects it to a particular table in the database. Anywhere that a data source is specified, you can use an SQL statement (or a saved query) instead of the name of a table. For example, let's build a new form that connects to the customers table by using an SQL SELECT statement as the data source.

1. Open the acFundSQL.mdb database and make sure that the tblCustomers table has been created and that it contains some data.
2. In the **Database** window, click **Forms** under **Objects**, and then click **New** on the Database window toolbar
3. In the **New Form** dialog box, click **Design View**, and then click **OK**. A blank form is now open in Design view.
4. On the **View** menu, click **Properties** to open the form's property sheet.
5. In the **RecordSource** property text box, type the following SQL statement:

   ```
   SELECT * FROM tblCustomers
   ```

6. Press the ENTER key on your keyboard. The field list appears, and it lists all of the available fields from the tblCustomers table.
7. Select all of the fields by holding down the SHIFT key and clicking the first and then the last field listed.
8. Drag the selected fields to the center of the Detail section on the blank form and then release the mouse button.
9. Close the property sheet.
10. On the **View** menu, click **Form View**, and then use the record selectors at the bottom of the form to scroll through all the records in the tblCustomers table.

Another great place to use an SQL statement is in the **RowSource** property for a list or combo box. Let's build a simple form with a combo box that uses an SQL SELECT statement as its row source.

1. Open the acFundSQL.mdb database and make sure that the tblCustomers table has been created and that it contains some data.
2. Create a new form and open it in Design view.
3. On the **View** menu, click **Toolbox**.
4. Make sure that the **Control Wizards** (upper rightmost) button in the toolbox is not pressed in.
5. Click the **Combo Box** button and then click in the center of the blank form's Detail section.
6. Make sure that the combo box in the form is selected, and then click **Properties** on the **View** menu.
7. In the **RowSource** property text box, type the following SQL statement:

   ```
   SELECT [Last Name] FROM tblCustomers
   ```

8. Press ENTER, and then close the property sheet.
9. On the **View** menu, click **Form View**. In the form, click the down arrow next to the combo box. Note that all the last names from the customers table are listed in the combo box.

Using SQL Statements Inline

The process of using SQL statements within VBA code is referred to as using the statements "inline." Although a deep discussion of how to use VBA is outside the scope of this article, it is a straightforward task to execute SQL statements in VBA code.

Suppose we need to run an UPDATE statement from code, and we want to run the code when a user clicks a button on a form.

1. Open the acFundSQL.mdb database and make sure that the tblCustomers table has been created and that it contains some data.
2. Create a new form and open it in Design view.
3. On the **View** menu, click **Toolbox**.
4. Make sure that the **Control Wizards** (upper rightmost) button in the toolbox is not pressed in.
5. Click the **Command Button** button and then click in the center of the blank form's Detail section.
6. Make sure that the command button in the form is selected, and then click **Properties** on the **View** menu.
7. Click in the following property text boxes and enter the values given:

   Name: cmdUpdatePhones

   Caption: Update Phones
8. Click the **OnClick** property text box, click the **Build** button (**Â...**), and then click **Code Builder** to open the Visual Basic Editor.
9. Type or paste the following lines of code in the cmdUpdatePhones_Click subprocedure:

```
Dim conDatabase As ADODB.Connection
Dim strSQL As String

Set conDatabase = CurrentProject.Connection

strSQL = "UPDATE tblCustomers SET Phone = 'None'"
conDatabase.Execute strSQL

MsgBox "All phones have been set to ""None""."

conDatabase.Close
Set conDatabase = Nothing
```

10. Close the Visual Basic Editor, close the property sheet, and then click **Form View** on the **View** menu.
11. Click the Update Phones button. You should see a message box that says all the phone numbers have been set to "None." You can verify this by opening the tblCustomers table.

Although using SQL statements inline is great for *action queries* (that is, append, delete, and update), they are most often used in select queries to build sets of records. Let's suppose that we want to loop through a results-based set to accomplish what the UPDATE statement did. Following a similar procedure for the UPDATE example, use the following code in the cmdSelectPhones_Click subprocedure:

```
Dim conDatabase As ADODB.Connection
Dim rstCustomers As ADODB.Recordset
Dim strSQL As String

Set conDatabase = CurrentProject.Connection
strSQL = "SELECT Phone FROM tblCustomers"

Set rstCustomers = New Recordset
rstCustomers.Open strSQL, conDatabase, _
    adOpenDynamic, adLockOptimistic

With rstCustomers
    Do While Not .EOF
    !Phone = "None"
    .Update
    .MoveNext
    Loop
End With

MsgBox "All phones have been set to ""None""."

rstCustomers.Close
conDatabase.Close

Set rstCustomers = Nothing
```

```
Set conDatabase = Nothing
```

In most cases, you will achieve better performance by using the UPDATE statement because it acts on the table as a whole, treating it as a single set of records. However, there may be some situations where you simply must loop through a set of records in order to achieve the results you need.

One Last Comment

Although it may be difficult to believe, this article has only scratched the surface of the SQL language as it applies to Access. By now you should have a good basic understanding of SQL and how you can use it in your Access 2000 applications. Try out your new skills by using SQL in any **RecordSource** or **RowSource** property you can find, and use the resources listed in the next section to further your knowledge of SQL and Access.

Additional Resources

Resource	Description
Intermediate Microsoft Jet SQL for Access 2000	This article builds on the fundamental concepts already covered here, and gives a much more detailed picture as to what can be accomplished with Microsoft Jet SQL in Access.
Advanced Microsoft Jet SQL for Access 2000	Third in the series of SQL articles, "Advanced Microsoft Jet SQL" builds on the concepts covered in the first two articles, this time focusing on the SQL syntax that is most often used in a multiuser environment.
Microsoft Jet SQL Reference Help	This is the definitive source for the SQL language as it applies to Access 2000. It can be found in the Contents section of Microsoft Access 2000 Help.
Microsoft Access 2000 Help	An irreplaceable source of Access 2000 programming topics.
Microsoft Office 2000/Visual Basic Programmer's Guide	This comprehensive book covers how to use VBA to program Office 2000 applications.
Microsoft Developer Network, http://msdn.microsoft.com/	This Web site always has the latest information for developing solutions with Microsoft platforms and languages.
MSDN Office Developer Center, http://msdn.microsoft.com/office/default.asp	This Web site contains the latest information about developing applications with Microsoft Office.
MSDN Training, Career, and Events	Look to Microsoft's MSDN Training courses to provide the soundest techniques for developing Access 2000 applications.