通配符用*.还是用% 2

DAO.ADO 控件 3

用DAO控件 SQL 4

用ADO控件 SQL 8

用ADO Data 环境 SQL 11

无Microsoft Data Bound Grid Control 控件怎么办, 17

Select [Fields]* FRom [Tables] 22

ORder BY 25、select Distinct 25

select Top 20 * 27 使用Desc倒序 26

select Top 20 percent * 27

where 29

Like 31

Inner Join ...on 37. 47

Right outer join 41、47

Left outer join 42. 47

字段名中可以带函数使用 43

SQL时就可以统计 45. 48

Group BY ... HAVing 46. 49

SQL建立工具: Access 50

SQL建立工具: ADO data environment 52

- SQL can be used with any database management system, not just Visual Basic. Hence, the syntax learned here will help any database programmer. SQL is a set of about 30 statements for database management tasks.

- To query a database, we form a **SQL statement**. A statement is a string of SQL keywords and other information, such as database table and field names. This statement tells the database engine what information we want from the database. You do not have to tell the database engine how to get the information - it does all the hard work for you!

- What can a SQL statement accomplish?

⇒ Sort records
⇒ Choose fields
⇒ Choose records
⇒ Cross reference tables
⇒ Perform calculations
⇒ Provide data for database reports
⇒ Modify data

- Even though we don't even know what a SQL statement looks like yet, we need to set some rules on how to construct such statements. Then, we will look at how to use a SQL statement in a Visual Basic application.

- All SQL **keywords** in a SQL statement will be typed in **upper case** letters. Even though SQL is 'case-insensitive,' this is good programming practice and allows us (and others) to differentiate between keywords and other information in a SQL statement.

最好关键字大写

- SQL uses the term **row** to refer to a database **record** and the term **column** to refer to database **field**. This will not come into play in this class, but you should be aware of this difference if you read other books about SQL.

行→记录　列→字段

- String information imbedded within a SQL statement can be enclosed in double-quotes (") or single-quotes ('). With Visual Basic, you should only use single-quotes to enclose imbedded strings. The reason for this is that the SQL statement is itself a string - so, in Visual Basic code, SQL statements must be enclosed in double-quotes. We enclose imbedded strings with single-quotes to avoid confusion.

因SQL为字符串，用双引号，所以VB中的字符串用单引号.

- SQL supports the use of wildcards in forming data views. The wildcard character for the Jet engine is an asterisk (*). Use of wildcards will be illustrated in many examples. ANSI Standard SQL implementations use the percent sign (%) as a wildcard.

  通配符. Jet engine is *. ANSI is %

- 对于文本字段数据类型字段的 WHERE 标准，最有用的操作符是 LIKE。在 Jet SQL 中称为 LIKE 操作符。下表显示了 LIKE 操作符的 ANSI SQL 语法和用于 SQL 语句 WHERE 子句中的 Jet SQL Like 操作符。

用途	ANSI SQL	Jet SQL	返回
取包含字符的任何文本	LIKE '%am%'	Like "*am*"	ram,rams damsel,amnesty
取以字符打头的任何文本	LIKE 'John%'	Like "John*"	Johnson,Johnsson
取以字符结尾的任何文本	LIKE '%son'	Like "*son"	Johnson,Anderson
取文本和任意单个后缀字符	LIKE 'Glen_'	Like "Glen?"	Glenn,Glens
取文本和任意单个前缀字符	LIKE '_am'	Like "?am"	dam,Pam,ram
取文本和一个前缀及任意个后缀字符	LIKE '_am%'	Like "?am*"	dams,Pam,Ramses

- 注意：为与 ANSI SQL－92 保持一致，本地 OLE DB 数据提供者（Microsoft.JET.OLEDB.3.51）使用%和_通配符而非 Jet 的?和*。这种不一致要求修改已存在的 DAO 代码以适应 ADO 使用 SQL 通配符。

- If a table or field name has an imbedded space, that name must be enclosed in brackets ([]). For example, if the table name is **My Big Table**, in a SQL statement you would use:

  如果表或字段名中有空格，用[ ]括起来

  [My Big Table]

  This notation is not allowed in some SQL implementations. But in implementations that don't recognize brackets, imbedded spaces in table and field names are not allowed, so it should never be a problem.

- To refer to a particular field in a particular table in a SQL statement, use a dot notation:

  TableName.FieldName 引用表中字段的格式：表名.字段名

  If either the table or field name has imbedded spaces, it must be enclosed in brackets.

Now, we're ready to start forming SQL statements and using them with Visual Basic applications. One warning - SQL is a very powerful ally in obtaining and modifying data in a database. But, it can also be very destructive - a single SQL statement can wipe out an entire database! So, be careful and always provide safeguards against such potential destruction. 会导致数据库清空. 使用小心小心

Where Does SQL Fit In Visual Basic?

- Visual Basic uses SQL queries to define a **data source**. SQL statements are processe by the Jet database engine (whether using DAO or ADO technology) to form **recordset** object. This object contains the virtual database table formed as a result o the SQL statement. The resulting object can be used to display and, perhaps, updat the database.

- How do we tell Visual Basic what the SQL statement is? It depends on whether w want to provide the statement in **design** mode or **run** mode. In design mode, w simply set a property for the appropriate DAO or ADO control (or Data Environment) In run mode, how we process the SQL statement depends on the data access technolog being used. This is addressed for each technology in the following sections.

- A result of interest from a SQL query is the number of records returned (if any). With both DAO and ADO technology, the returned recordset has a **RecordCount** property To receive a valid count with this property, however, the Jet database engine mus 'touch' every record in the recordset. This is accomplished by performing a **MoveLas** method once the recordset is formed, followed immediately by a **MoveFirst** method This accomplishes two tasks: provides a valid RecordCount and positions the record pointer at the top record. We will look at obtaining a valid RecordCount with the DAO and ADO technologies next.

- Note we say the database can **perhaps** be updated. How do we know if an update can be performed? It all depends on the particular database access technology and SQL statement used to create the virtual table. This is discussed in later sections.

SQL with the DAO Data Control

- When using the **DAO** (data access object) data control, the SQL statement takes the place of the **RecordSource** property of the control. In design mode, simply go to the Properties Window for the data control, scroll down to the RecordSource property and type in a valid SQL statement. Many times, this will be a very long property. Obviously, it is assumed that the **DatabaseName** property of the data control has been set to the desired database file.

- In run mode, the SQL statement is also assigned to the **RecordSource** property of the data control (**Refresh** the data control after assigning the RecordSource). For example, if we have a SQL statement named **MySQL** (this will be a **string** type variable) we want to use with a data control named **datDAOExample** (again, it is assumed that the DatabaseName property has been appropriately set), the BASIC code syntax is:

```
datDAOExample.RecordSource = MySQL
datDAOExample.Refresh
```

  We usually set the RecordSource property (and DatabaseName property, also) at run-time, rather than in design mode. Reasons for this are discussed in later chapters.

- Whether in design or run mode, a valid SQL statement will return a **Recordset** object containing the selected database records. This object will have its own methods and properties for our use. In particular, to establish a valid **RecordCount** for the Recordset returned using a data control named datDAOExample, use these two lines of code:

```
datDAOExample.Recordset.MoveLast
datDAOExample.Recordset.MoveFirst
```

  Following these lines, the RecordCount is examined using:

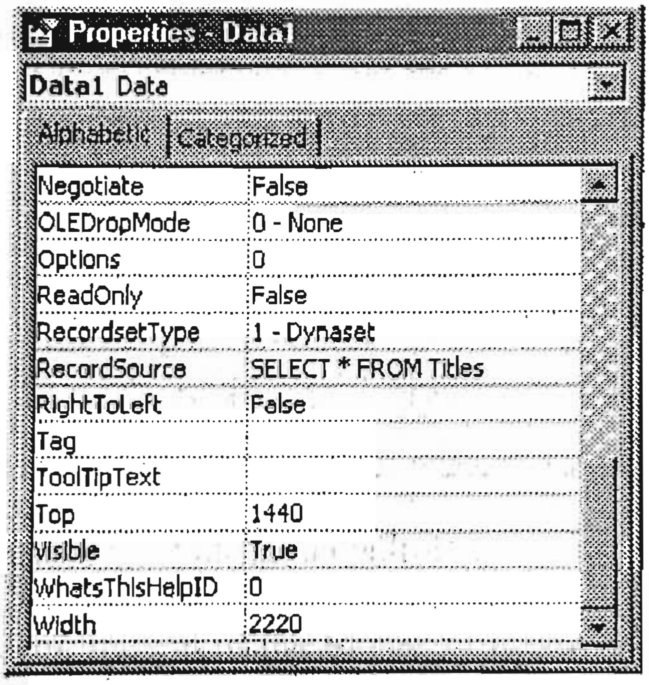
```
datDAOExample.Recordset.RecordCount
```

Quick Example 1 - SQL with the DAO Data Control

1. Start a new project. Add two label controls and a DAO data control. Set two data control properties to:

DatabaseName	BIBLIO.MDB (point to your working copy)
RecordSource	SELECT * FROM Titles

After setting the RecordSource property, the Properties Window should look like this:



Negotiate	False
OLEDropMode	0 - None
Options	0
ReadOnly	False
RecordsetType	1 - Dynaset
RecordSource	SELECT * FROM Titles
RightToLeft	False
Tag	
ToolTipText	
Top	1440
Visible	True
WhatsThisHelpID	0
Width	2220

Yes, this is your first SQL statement! You don't have to recognize this right now, but it's pretty easy to understand. The statement says **SELECT** all fields (the wildcard *) **FROM** the **Titles** table. This has the same result as choosing the Titles table as the RecordSource property.

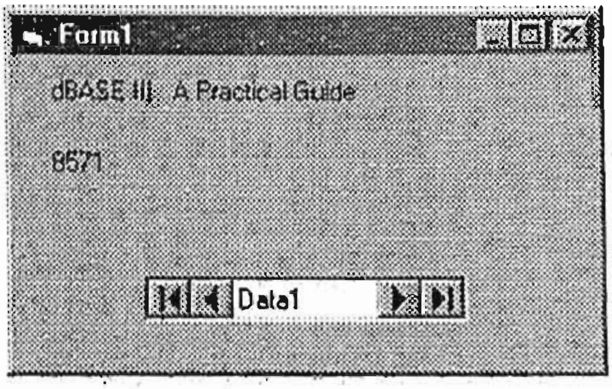
2. Set the following two properties for the first label control (Label1):

DataSource	Data1
DataField	Title

3. Place this code in the **Form_Activate** procedure (this counts and displays the number of records):

```
Private Sub Form_Activate()
Data1.Recordset.MoveLast
Data1.Recordset.MoveFirst
Label2.Caption = Data1.Recordset.RecordCount
End Sub
```

4. Save and run the application. You should see this (the first label control showing a title and the second a number (the number of returned records):



dBASE III: A Practical Guide

8571

Scroll through different titles using the data control arrows..

5. Now, add these two lines at the top of the **Form_Activate** procedure (these lines set the RecordSource at run-time):

```
Data1.RecordSource = "SELECT * FROM Titles ORDER BY Title"
Data1.Refresh
```

The SQL statement (enclosed in quotes since it is a BASIC string variable) is modified so the results are in alphabetical order.

6. Save and rerun the application. The 'in code' SQL statement should produce the same records, but in order:

SQL with the ADO Data Control

- When using the **ADO** (ActiveX data object) data control, the SQL statement takes the place of the **RecordSource** property of the control. In design mode:

  ⇒ Establish the **ConnectionString** property.
  ⇒ Go to the **Properties Window** for the data control, scroll down to the **RecordSource** property and click on the ellipsis that appears. The **RecordSource Property Page** will appear.
  ⇒ Under **Command Type**, select **1 - adCmdText** (this tells the control we will be using a SQL statement). Then, in the **Command Text (SQL)** window, type in a valid SQL statement. When done, click **OK**.

- In run mode, the SQL statement is also assigned to the **RecordSource** property of the data control (**Refresh** the data control after assigning the RecordSource). For example, if we have a SQL statement named **MySQL** (this will be a **string** type variable) we want to use with a data control named **datADOExample** (again, it is assumed that the **ConnectionString** property has been appropriately set), the BASIC code syntax is:

```
datADOExample.RecordSource = MySQL
datADOExample.Refresh
```

  We usually set the RecordSource property (and ConnectionString property, also) at run-time, rather than in design mode. Reasons for this are discussed in later chapters.

- Whether in design or run mode, a valid SQL statement will return a **Recordset** object containing the selected database records. This object will have its own methods and properties for our use. In particular, to establish a valid **RecordCount** for the Recordset returned using a data control named datADOExample, use these two lines of code:

```
datADOExample.Recordset.MoveLast
datADOExample.Recordset.MoveFirst
```

  Following these lines, the RecordCount is examined using:

```
datADOExample.Recordset.RecordCount
```

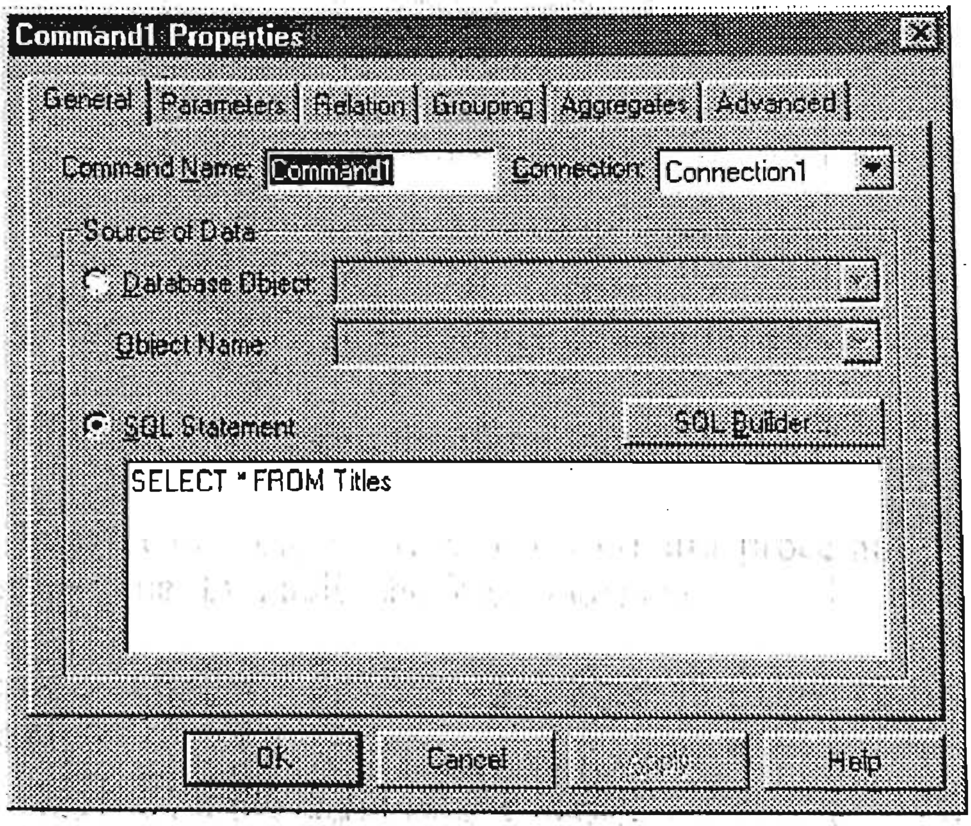
ick Example 2 - SQL with the ADO Data Control

Start a new project. Add two label controls and an ADO data control. Build the data control **ConnectionString** property to point to your working copy of BIBLIO.MDB.

Go to the **Properties Window** and click on the data control's **RecordSource** property. Click the ellipsis. The **RecordSource Property Page** will appear. Under **Command Type**, select **1 - adCmdText**. Then, in the **Command Text (SQL)** window, type:

SELECT * FROM Titles

You should see:



Command1 Properties

Command Name: Command1

Connection: Connection1

○ Database Object

Object Name:

● SQL Statement

```
SELECT * FROM Titles
```

When done, click **OK**.

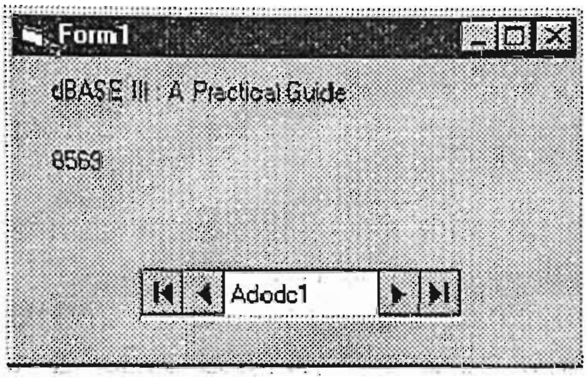
Set the following two properties for the first label control (Label1):

DataSource	Adodc1
DataField	Title

4. Place this code in the **Form_Activate** procedure (this counts and displays the number of records):

```
Private Sub Form_Activate()
Adodc1.Recordset.MoveLast
Adodc1.Recordset.MoveFirst
Label2.Caption = Adodc1.Recordset,RecordCount
End Sub
```

5. Save and run the application. You should see something like this (the first label control showing a title and the second a number (the number of returned records):



dBASE III : A Practical Guide

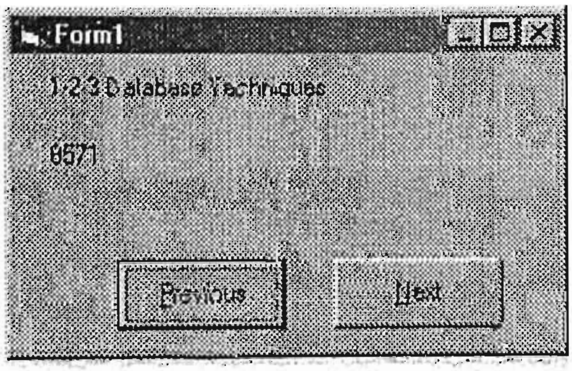
8569

Scroll through different titles using the data control arrows.

6. Add these two lines at the top of the **Form_Activate** procedure (these lines set the RecordSource at run-time) to modify the SQL statement:

```
Data1.RecordSource = "SELECT * FROM Titles ORDER BY Title"
Data1.Refresh
```

7. Save and rerun the application. The 'in code' SQL statement should produce the same records, but in order:



1-2-3 Database Techniques

8571

SQL with the ADO Data Environment

- When using the **ADO** (ActiveX data object) data environment, the SQL statement forms a new **Command** object within an existing **Connection** object. In design mode:

  ⇒ Establish the **Connection** object (connect to a database).
  ⇒ Right-click on the Connection object in the **Data Environment** window and select **Add Command**. A new Command object will appear.
  ⇒ Right click the Command object and select **Properties**. The **Properties** window appears - make sure the **General** tab is selected. Under **Source of Data**, click **SQL Statement**. The SQL window will become enabled. Type a valid SQL statement, then click **OK**.

  (In all these steps, it is assumed that proper conventions were followed in naming all objects.)

- With the ADO data environment, we follow a different approach when using SQL statements in run mode. The recordset created based on design-time parameters is first closed (use the **Close** method). Then, we re-open the recordset using the **Open** method and the new SQL statement. For example, assume we have a data environment named **denExample**, a command object named **comExample** and a new SQL statement named **MySQL** (this will be a **string** type variable). Recall the recordset associated with comExample will be named **rscomExample**. The code to close the current recordset and re-open it with a new SQL statement is:

```
denExample.rscomExample.Close
denExample.rscomExample.Open MySQL
```

We're not done, though. One more step is needed.

- After creating the new recordset, all data bound controls are left bound to the original recordset. Without manually rebinding (in code) the controls to the new recordset, you won't see the new results. Microsoft, in their Knowledge Base Articles, claims this is an intended behavior. We believe it is a bug that will hopefully be addressed as ADO technology matures. To rebind the data bound controls to the ADO data environment, you need to reset each control's **DataSource** property. The code to rebind a control named **ExampleControl** to a data environment named **DataEnvironmentName** is:

  ```
  Set ExampleControl.DataSource = DataEnvironmentName
  ```

  Note use of the **Set** statement. Set must be used when initializing a programming object, as we are here. We will look at some automated techniques for rebinding in a later chapter. You can see that working with the data environment is a little trickier. But, after you've used it a few times, you'll begin to appreciate its great advantages.

- Whether in design or run mode, a valid SQL statement will return a **recordset** object containing the selected database records. Recall, in our example above, the returned recordset is named **rscomExample**. This object will have its own methods and properties for our use. In particular, to establish a valid **RecordCount** for the recordset returned by a data environment named denExample, use these two lines of code:

  ```
  denExample.rscomExample.MoveLast
  denExample.rscomExample.MoveFirst
  ```

  Following these lines, the RecordCount is examined using:

  ```
  denExample.rscomExample.RecordCount
  ```

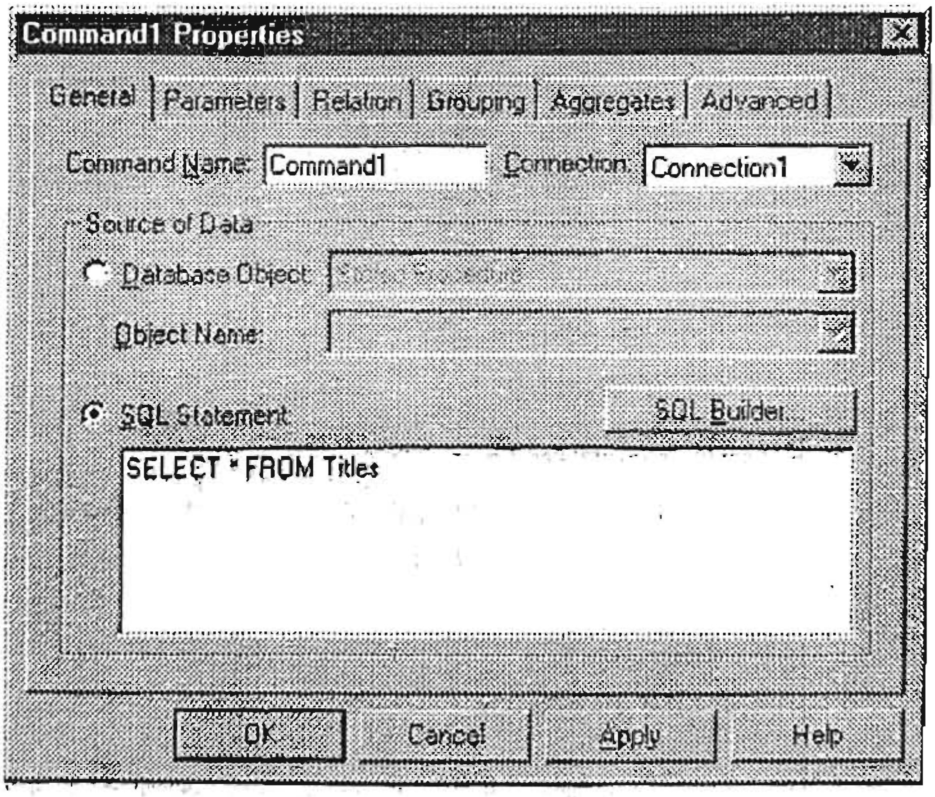
Quick Example 3 - SQL with the ADO Data Environment

1. Start a new project. Add two label controls and two command buttons (needed to allow navigation among records). Add a **Data Environment** in the **Project Explorer** window. Right-click **Connection1** and set **Properties** so it points to your working copy of **BIBLIO.MDB**.

2. Right-click on **Connection1** and select **Add Command**. A new Command object will appear. Right click that object and select **Properties**. The **Properties** window appears - make sure the **General** tab is selected. Under **Source of Data**, click **SQL Statement**. The SQL window will become enabled. Type:

   SELECT * FROM Titles

   You should see:



Command1 Properties

Command Name: Command1

Connection: Connection1

Source of Data

○ Database Object:

Object Name:

● SQL Statement

SELECT * FROM Titles

When done, click **OK**.

3. Set the following properties for the first label control and the two command buttons:

Label1:	
DataSource	DataEnvironment1
DataMember	Command1
DataField	Title
Command1:	
Caption	&Previous
Command2:	
Caption	&Next

4. Place this code in the **Form_Activate** procedure (this counts and displays the number of records):

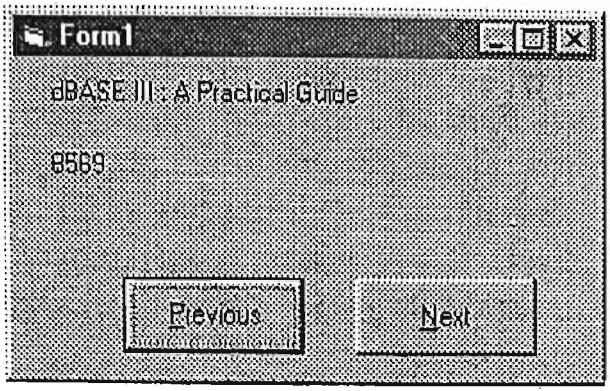
```
Private Sub Form_Activate()
DataEnvironment1.rsCommand1.MoveLast
DataEnvironment1.rsCommand1.MoveFirst
Label2.Caption = DataEnvironment1.rsCommand1.RecordCount
End Sub
```

5. Add this code to the command button **Click** events to allow navigation:

```
Private Sub Command1_Click()
DataEnvironment1.rsCommand1.MovePrevious
If DataEnvironment1.rsCommand1.BOF Then
 DataEnvironment1.rsCommand1.MoveFirst
End If
End Sub

Private Sub Command2_Click()
DataEnvironment1.rsCommand1.MoveNext
If DataEnvironment1.rsCommand1.EOF Then
 DataEnvironment1.rsCommand1.MoveLast
End If
End Sub
```

6. Save and run the application. You should see something like this (the first label control showing a title and the second a number (the number of returned records – this may be a different value for you, depending on the current state of the BIBLIO.MDB database):



dBASE III : A Practical Guide

8569

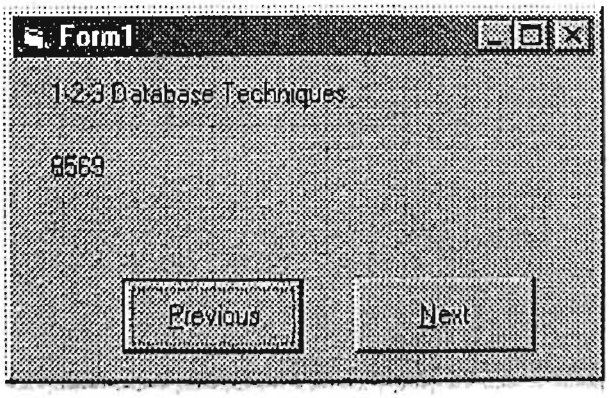
Navigate through the records, if you like.

7. Add these three lines at the top of the **Form_Activate** procedure (these lines set the RecordSource at run-time):

```
DataEnvironment1.rsCommand1.Close
DataEnvironment1.rsCommand1.Open "SELECT * FROM Titles ORDER
BY Title"
Set Label1.DataSource = DataEnvironment1
```

These lines close the old recordset, re-open it with the new SQL statement, and then rebind the label control to the data environment.

8. Save and rerun the application. You obtain the same records, but ordered:



1-2-3 Database Techniques

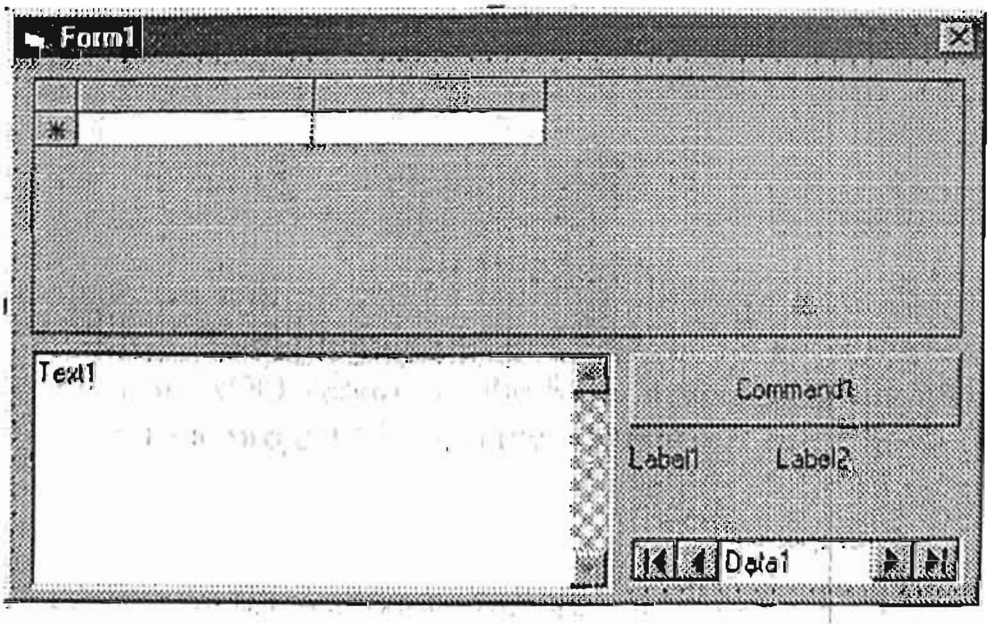
8569

Example 5-1

SQL Tester

Well, now we know some of the rules and syntax of SQL statements and how to use them with Visual Basic, but we still don't know what a SQL statement looks like (well, we saw one in the examples). We correct all that now and start learning more about SQL. To test SQL statements we form, we build this example which allows us to enter SQL statements and see the results of the formed database queries. In this example, we use the DAO data control so both Visual Basic 5 and Visual Basic 6 users can build the same example. You can choose to use the ADO control (or data environment) if you choose.

1. Start a new project. Add a DAO data control, a text box control, two label controls, a command button, and a DBGrid control to the form. Wait, you say, what is a **DBGrid Control** and why isn't it in the toolbox? It is a DAO data bound control we haven't looked at yet, but it is very powerful. The DBGrid control allows us to view and edit an entire database table by setting just one property (**DataSource**). It is a custom control that must be added to the toolbox. To do this, select **Components** under the **Project** menu item. In the window that appears, check the box next to **Microsoft Data Bound Grid Control**, then click **OK**. It is then available for selection from the toolbox. We will look further at this control in Chapter 6. Resize and position the controls so your form looks something like this:



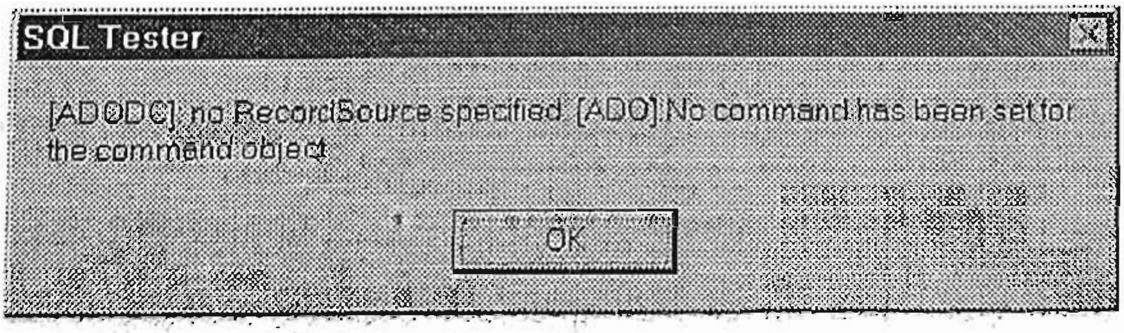
A Brief (Hopefully) Interlude for Visual Basic 6 Users:

When you selected the **Components** tab, the choice for the **Microsoft Data Bound Control** may not have been there. You will see a choice for **Microsoft DataGrid C** This is not the same control – this is the version of the control that works with technology. So what can you do? There are two solutions: a quick one 'not-so-quick' one. We recommend the latter.

Solution 1 – The Quick Solution:

Use the ADO DataGrid control (make sure it has been added to the toolbox) in place DAO data bound grid control. You will also have to replace the DAO data contro the ADO data control. Use the same properties for the grid control and the data c with one exception. Recall the ADO data control does not have a **Database** property. If using the ADO control, set the **ConnectionString** property such that it to your working copy of the BIBLIO.MDB database. No code changes are neces: the code that works for the DAO data control will work for the ADO data control.

When you attempt setting the **DataSource** property for the grid control, you will g error message:


SQL Tester

[ADODC]: no RecordSource specified. [ADO]:No command has been set for the command object

This is acceptable since we will be setting the data control's **RecordSource** at run You may also get this error when running the application. If so, just click **OK**. Fo reference, we have built an ADO version of the **SQL Tester** program and included i the example files (look for the project file with the **AD** suffix).

Solution 2 – The 'Not-So-Quick' Solution:

Here, we will install the desired DAO data grid control (and other DAO-based controls, if desired) onto your computer. The steps are many, but the effort is worth it, especially if you ever plan to use or build applications that employ DAO database technology. The information provided here was taken from Microsoft's website. You will need your installation CD for Visual Basic 6. You will also have to be familiar with issuing DOS command line statements. Ask for help from someone if this is unfamiliar.

Look in the **\COMMON\TOOLS\VB\CONTROLS** directory on the VB6 CD. This directory contains controls that shipped with Visual Basic 4/5 Professional and Enterprise Editions, which are no longer shipping with Visual Basic 6:

AniBtn32.ocx, Gauge32.ocx, Grid32.ocx (the file we are interested in here), KeySta32.ocx, MSOutl32.ocx, Spin32.ocx, Threed32.ocx, MSChart.ocx

To install these files on your computer, follow these steps:

1. Copy all of the files in this directory to your **\WINDOWS\SYSTEM** directory.

2. Register the controls by either Browsing to them in Visual Basic itself (select the **Browse** option when selecting **Components**), or manually register them using **RegSvr32.Exe**. RegSvr32.EXE can be found in the **\COMMON\TOOLS\VB\REGUTILS** directory. The DOS command line is:

   regsvr32.exe grid32.ocx

3. Register the design time licenses for the controls. To do this, merge the **vbctrls.reg** file found in the **\COMMON\TOOLS\VB\CONTROLS** directory into your registry. You can merge this file into your registry using **RegEdit.Exe** (Win95, Win98, WinMe, Win2000 or WinNT4) or RegEd32.Exe (WinNT3.51):

   regedit vbctrls.reg (or other reg files associated with the controls)

The DAO files (including the DAO data grid control) should now appear in the **Components** listing when choosing controls to add to your toolbox. Now back to our example.

Set properties for the form and controls:

Form1:

Name	frmSQLTester
BorderStyle	1-Fixed Single
Caption	SQL Tester

Data1:

Name	datSQLTester
Caption	SQL Tester
DatabaseName	BIBLIO.MDB (point to your working copy)

Label1:

Caption	Records Returned

Label2:

Name	lblRecords
Alignment	2-Center
BackColor	White
BorderStyle	1-Fixed Single
Caption	0
FontSize	12

Command1:

Name	cmdTest
Caption	Test SQL Statement
TabStop	False

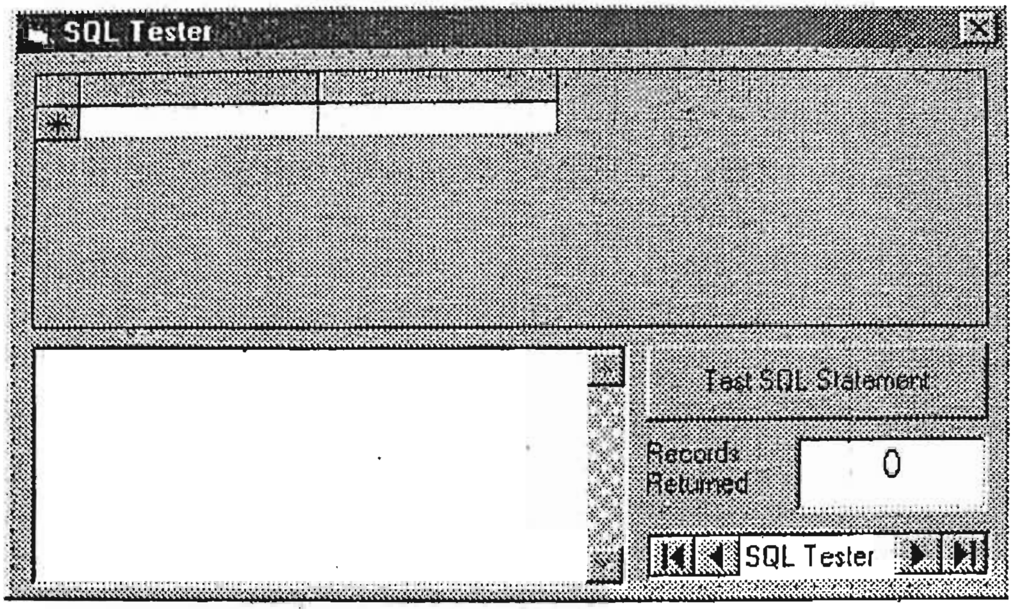
DBGrid1:

Name	grdSQLTester
DataSource	datSQLTester
TabStop	False

Text1:

Name	txtSQLTester
MultiLine	True
ScrollBars	2-Vertical

When done, the form should look like this:



With this example, we will type SQL statements in the text box area, then click the Test SQL Statement button. The data grid will display the returned records, while the label control will display the number of records returned. We need some code to do all of this.

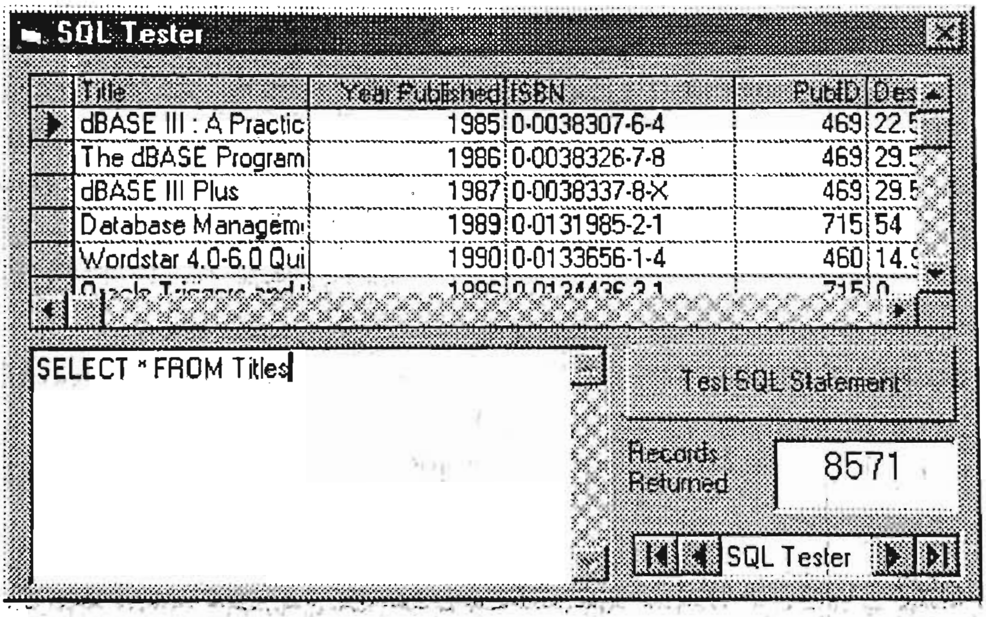
All the code goes in the **cmdTest_Click** event:

```
Private Sub cmdTest_Click()
'Enable error handling
On Error GoTo SQLError
'Read SQL statement and establish Recordsource
datSQLTester.RecordSource = txtSQLTester.Text
datSQLTester.Refresh
If datSQLTester.Recordset.RecordCount <> 0 Then
  datSQLTester.Recordset.MoveLast
  datSQLTester.Recordset.MoveFirst
  lblRecords.Caption = datSQLTester.Recordset.RecordCount
Else
  lblRecords.Caption = "0"
End If
txtSQLTester.SetFocus
Exit Sub
'If error occurs, report it in message box
SQLError:
MsgBox Error(Err.Number), vbExclamation + vbOKOnly, "SQL Error"
Exit Sub
End Sub
```

Let's spend some time seeing what's going on in this code. The first thing we do is turn on error trapping. Without it, if we make a small error in a SQL statement, the program will stop. With it, we get a message indicating our mistake and are allowed to continue. Following error control, the SQL statement (from txtSQLTester) is processed and the Recordset established. The records are then counted and displayed.

Be careful in typing SQL statements. Although we have error trapping in SQL Tester, if you make a mistake, the returned error messages are (many times) not of much help. If you get an error, the best thing to do is retype the SQL command, paying attention to spacing, spelling, and proper punctuation.

3. Save the application and run it. Type the only SQL statement you know at this time in the text box (**SELECT * FROM Titles**). Click **Test SQL Statement** and you should see:



Title	Year Published	ISBN	PubID	Des
dBASE III : A Practic	1985	0-0038307-6-4	469	22.5
The dBASE Program	1986	0-0038326-7-8	469	29.5
dBASE III Plus	1987	0-0038337-8-X	469	29.5
Database Managem	1989	0-0131985-2-1	715	54
Wordstar 4.0-6.0 Qui	1990	0-0133656-1-4	460	14.9
Oracle Triggers and	1995	0-0134436-3-1	715	0

```
SELECT * FROM Titles
```

Records Returned: 8571

SQL Tester

Note the DB grid control display the entire table. You can scroll through the table or edit any values you choose. Any changes are automatically reflected in the underlying database. Column widths can be changed at run-time. Multiple row and column selections are possible. As we said, it's a very powerful tool. Please note **Records Returned** values for your results may be different, depending on the current data in the database.

Change the word **SELECT** to **SLECT** to make sure the error trapping works. Now, let's use this SQL Tester to examine many kinds of SQL statements. When typing the statements, use upper case letters for the SQL keywords. Statements do not necessarily have be on a single line - multiple line SQL statements are fine and usually make them easier to read and understand.

SELECT/FROM SQL Statement

- The most commonly used SQL statement is the one we've been using as an example: the **SELECT/FROM** statement. This statement allows you to pick fields from one or more tables.

- The syntax for a SELECT/FROM SQL statement is:

  SELECT [Fields] **FROM** [Tables]

  where [Fields] is a list of the fields desired and [Tables] is a list of the tables where the fields are to be found. The wildcard character (*) can be used for the fields list to select all fields from the listed table(s). For example, the statement we have been using:

  SELECT * FROM Titles

  selects and returns all fields from the BIBLIO.MDB database Titles table. Look at all fields in the other tables (Authors, Publishers, Title Author) using similar statements. When looking at the Title Author table, you need to write:

  SELECT * FROM [Title Author]

  Recall field and table names with imbedded spaces must be enclosed in brackets. Looking at each table will reacquaint you with the structure of the BIBLIO.MDB database tables and fields. We will use a lot in the rest of this chapter.

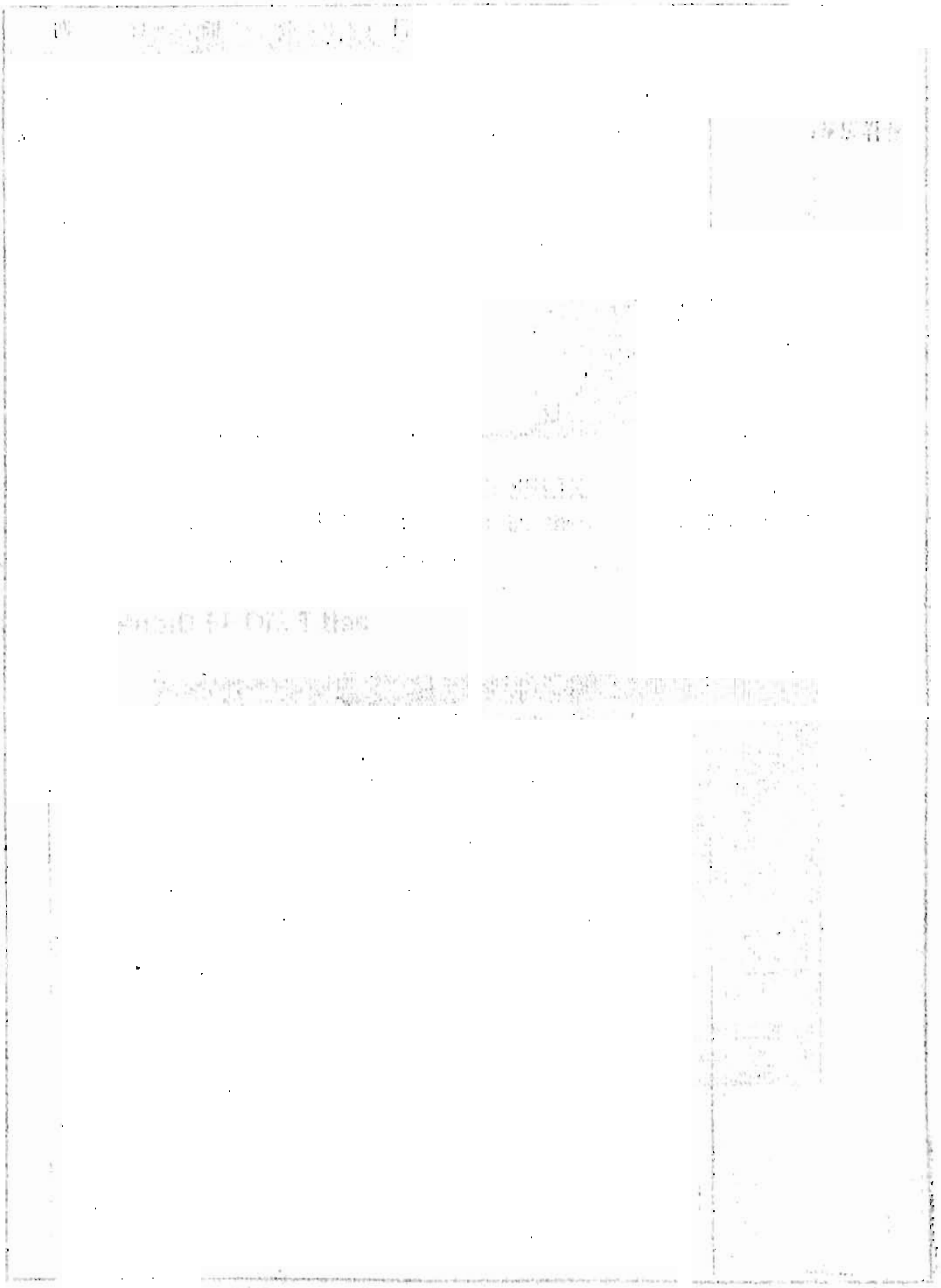
- If we only want selected fields from a table, we use a **field list**, which is a comma-delimited list of the fields desired, or:

  SELECT Field1, Field2, Field3 **FROM** Table

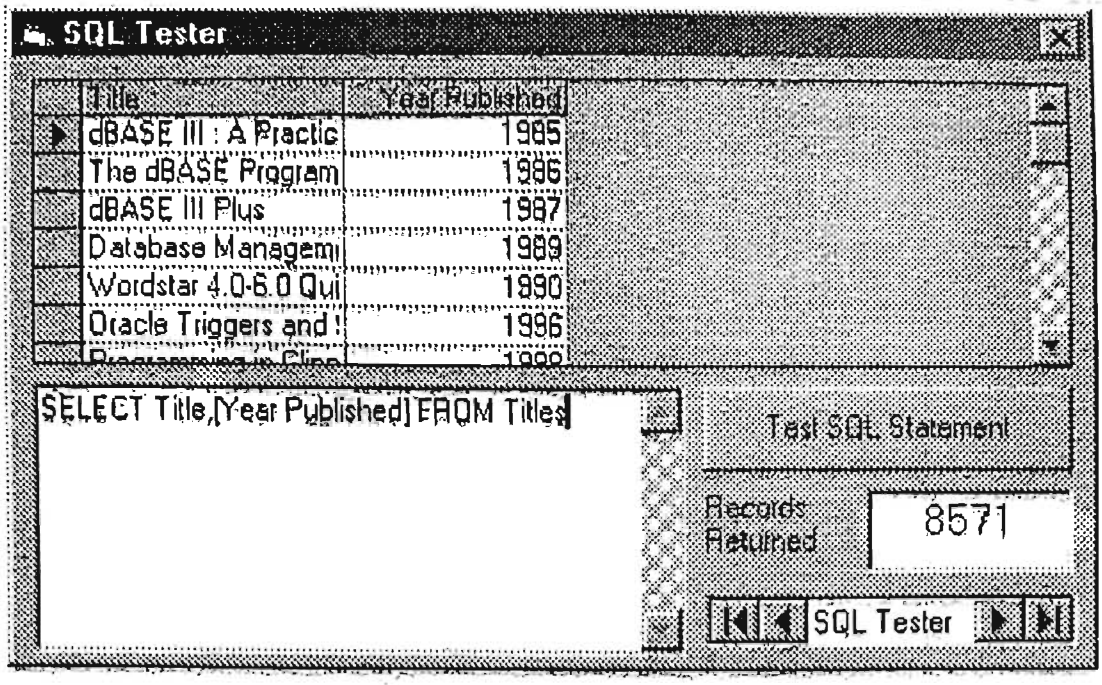
  will return three named fields from Table. Make sure you do not put a comma after the last field name. To obtain just the Title and Year Published (name must be enclosed in brackets because of imbedded space) fields from the books database Titles table, use:

  SELECT Title,[Year Published] FROM Titles

  Note the field names are not written using the prescribed dot notation of **Table.Field**. The table name omission is acceptable here because there is no confusion as to where the fields are coming from. When using multiple tables, we must use the dot notation.



Try this with the SQL tester and you will see just two fields are returned.



Title	Year Published
dBASE III : A Practic	1985
The dBASE Program	1986
dBASE III Plus	1987
Database Managem	1989
Wordstar 4.0-6.0 Qui	1990
Oracle Triggers and !	1996

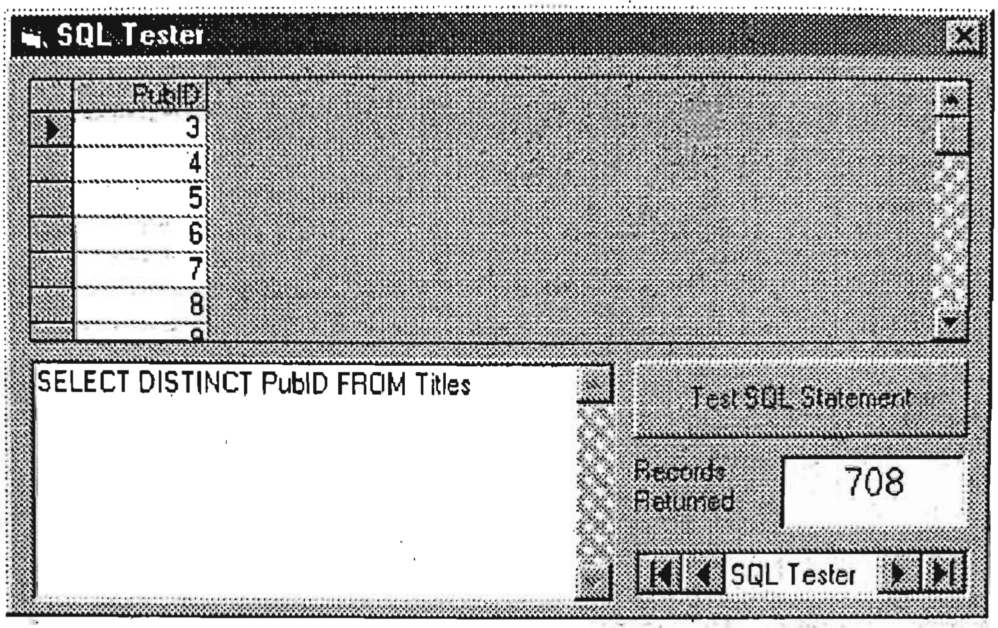
```
SELECT Title,[Year Published] FROM Titles
```

- The **DISTINCT** keyword can be used with SELECT to restrict the returned rec one per unique entry for the field. That is, there are no duplicate entries. example, first try this with the SQL tester:

SELECT PubID FROM Titles

Now, try:

SELECT DISTINCT PubID FROM Titles



PubID
3
4
5
6
7
8
9

```
SELECT DISTINCT PubID FROM Titles
```

Records Returned: 708

You should see far fewer records are returned - only distinct publishers are returned.

ORDER BY Clause

- When you use a SELECT/FROM statement, the records are returned in the order they are found in the selected table(s). To sort the returned records in some other order, you use the **ORDER BY** clause. The syntax is:

SELECT [Fields] **FROM** [Tables] **ORDER BY** FieldSort

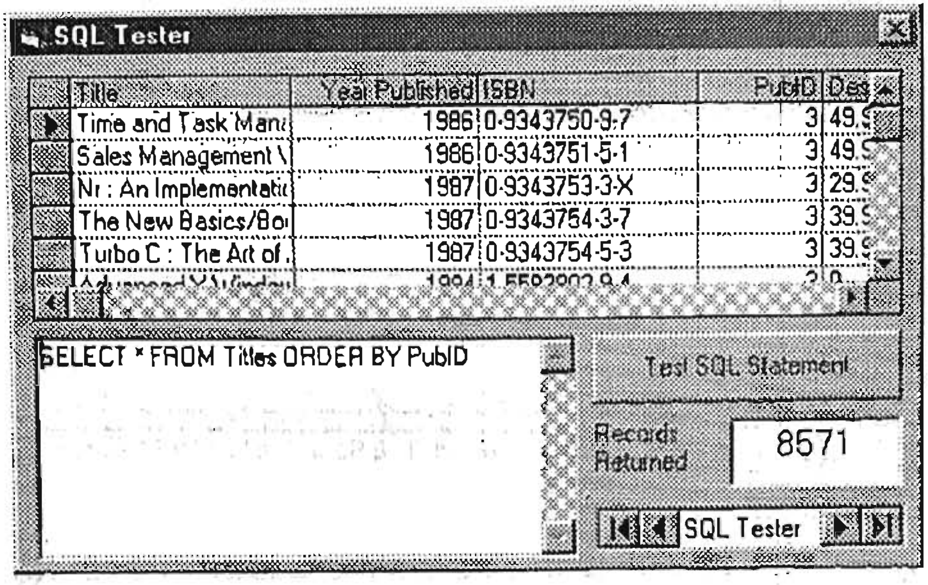
This statement selects the listed fields from the listed tables and sorts them by the field named FieldSort, By default, the ordering is in ascending order. If you want the sort to be in descending order, the FieldSort name is followed by the keyword **DESC**.

支持多个字段、倒序排序，用逗号分隔，如：
ORDER BY [表].[日期] DESC,[表].[时间]

- Try this statement with the SQL Tester:

SELECT * FROM Titles ORDER BY PubID

All records in the Titles table will be returned in order of Publisher ID.



Title	Year Published	ISBN	PubID	Des
Time and Task Man	1986	0-9343750-9-7	3	49.9
Sales Management \	1986	0-9343751-5-1	3	49.9
Nr : An Implementati	1987	0-9343753-3-X	3	29.9
The New Basics/Bo	1987	0-9343754-3-7	3	39.9
Turbo C : The Art of	1987	0-9343754-5-3	3	39.9

```
SELECT * FROM Titles ORDER BY PubID
```

Records Returned: 8571

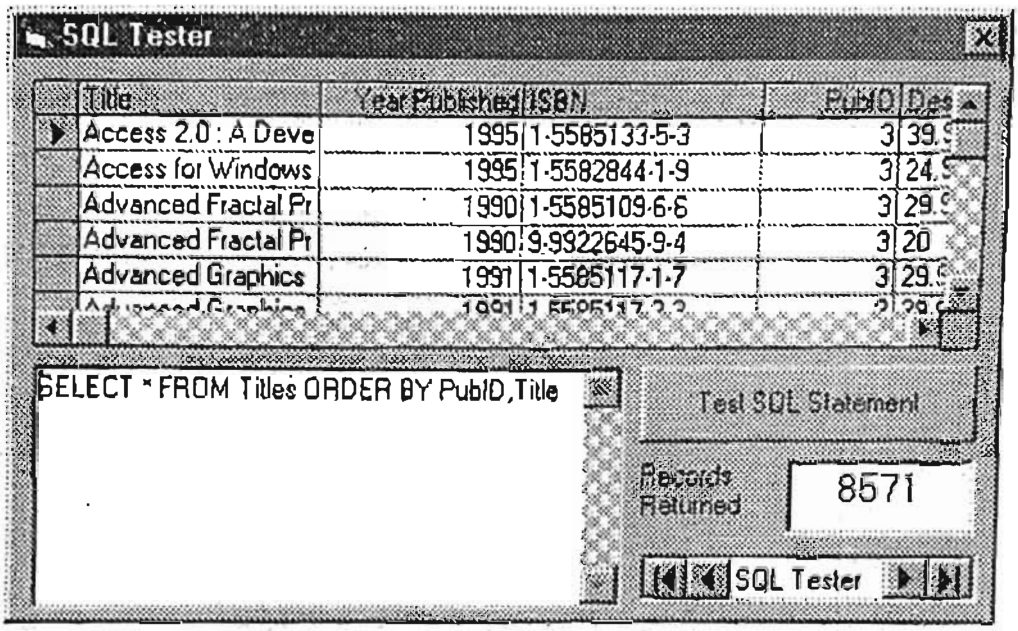
Try this and the order should be reversed:

SELECT * FROM Titles ORDER BY PubID DESC

- You can use more than one field in the ORDER BY clause. SQL will create a recordset based on all requested orderings. Try this with SQL tester:

SELECT * FROM Titles ORDER BY PubID,Title

The returned records will be in order of the publishers, with each publisher's titles in alphabetic order.



SQL Tester

Title	Year Published	ISBN	PubID	Des
Access 2.0 : A Deve	1995	1-5585133-5-3	3	39.
Access for Windows	1995	1-5582844-1-9	3	24.
Advanced Fractal Pr	1990	1-5585109-6-6	3	29.
Advanced Fractal Pr	1990	9-9922645-9-4	3	20
Advanced Graphics	1991	1-5585117-1-7	3	29.

```
SELECT * FROM Titles ORDER BY PubID,Title
```

Records Returned: 8571

- If you want to restrict the number of records returned by a SQL statement that orders the returned records, you can use the **TOP** keyword with SELECT. **TOP n** returns the first n records. **TOP n PERCENT** returns the first n percent of the returned records. If two or more records have the same order value, they are all returned. Use the SQL Tester and try:

SELECT TOP 20 * FROM Titles ORDER BY PubID,Title

Twenty books should be returned. Now, try:

SELECT TOP 20 PERCENT * FROM Titles ORDER BY PubID,Title

Far more books will be returned.

WHERE Clause

- One of the most useful aspects of the SELECT/FROM SQL statement is its ability to limit the returned recordset via the **WHERE** clause. This clause specifies some criteria that must be met in forming the recordset. The syntax is:

  SELECT [Fields] **FROM** [Tables] **WHERE** Criteria

- The WHERE clause limits the number of returned records by allowing you to do logical checks on the value of any field(s). **Operators** used to perfom these checks include:

<	Less than	<=	Less than or equal to
>	Greater than	>=	Greater than or equal to
=	Equal	<>	Not equal

Other operators are:

Between	Within a specified range
In	Specify a list of values
Like	Wild card matching

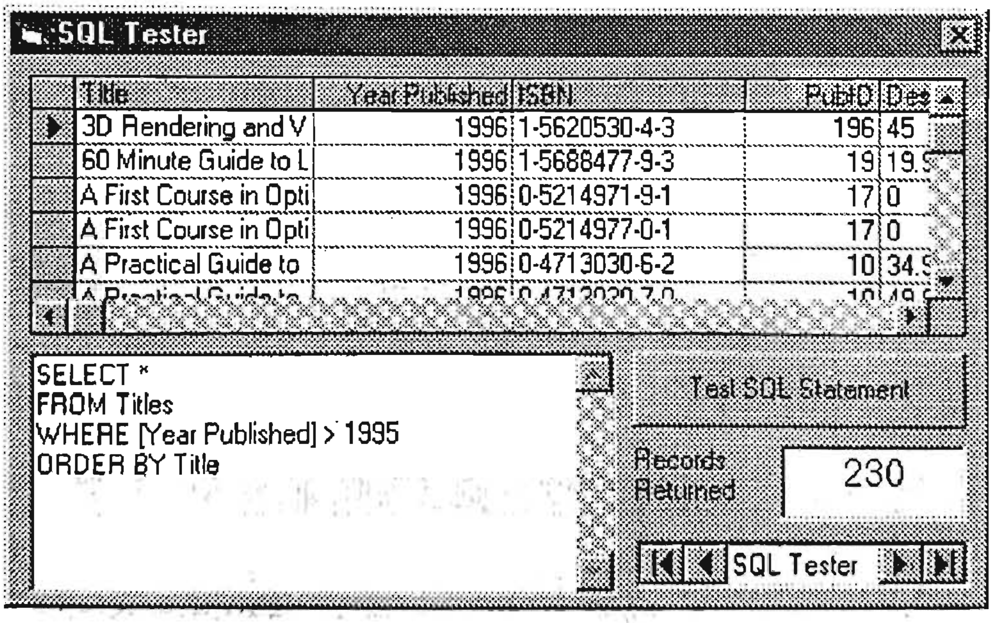
The WHERE clause can limit information displayed from one table or combine information from one or more tables. First, let's do some several single table examples using SQL Tester.

Single Table WHERE Clause

- Say we want to see all fields in the BIBLIO.MDB Titles table for books published after 1995. And, we want the returned records ordered by Title. The SQL statement to do this is (we'll type each clause on a separate line to clearly indicate what is going on - multiple line SQL statements are acceptable and, many times, desirable):

```
SELECT *
FROM Titles
WHERE [Year Published] > 1995
ORDER BY Title
```

This is where the real power of SQL comes in. With this simple statement, the Jet database engine quickly finds the desired records and sorts them - all without any coding on our part!



Title	Year Published	ISBN	PubID	Des
3D Rendering and V	1996	1-5620530-4-3	196	45
60 Minute Guide to L	1996	1-5688477-9-3	19	19.5
A First Course in Opti	1996	0-5214971-9-1	17	0
A First Course in Opti	1996	0-5214977-0-1	17	0
A Practical Guide to	1996	0-4713030-6-2	10	34.5

```
SELECT *
FROM Titles
WHERE [Year Published] > 1995
ORDER BY Title
```

Records Returned: 230

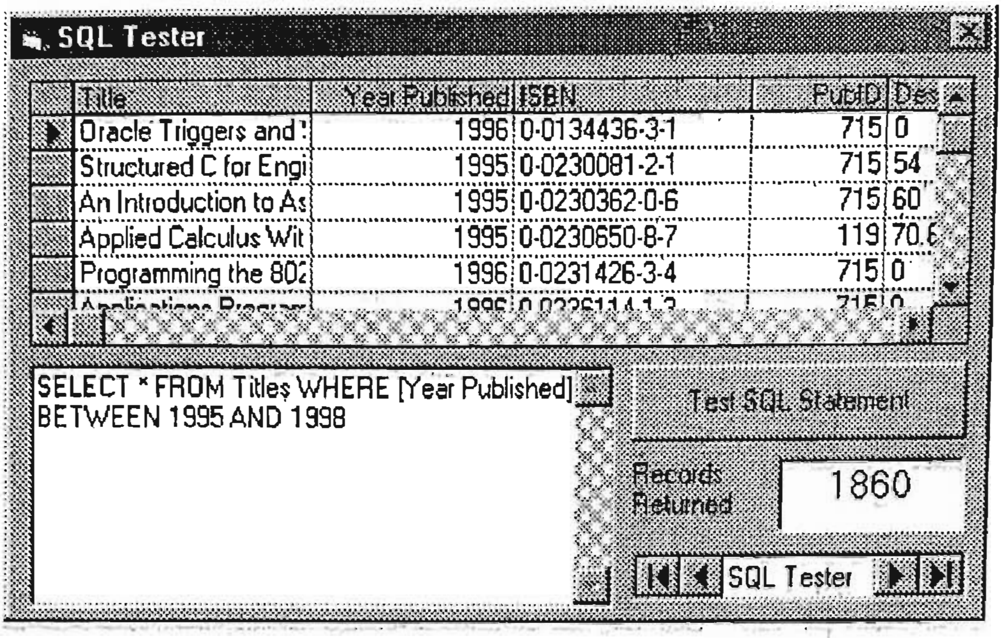
- What if we want to know information about all the book publishers in the state of Washington. Try this SQL statement with the BIBLIO.MDB Publishers table:

```
SELECT * FROM Publishers WHERE State = 'WA'
```

Note we enclosed the state name abbreviation (a string) in single quotes, as discussed earlier in this chapter. Try this SQL statement with the SQL tester and you should find one lonely publisher (BetaV) in the state of Washington! Wonder where Microsoft is?

- The **BETWEEN** keyword allows us to search for a range of values. Want all books published between 1995 and 1998? Use this SQL statement:

SELECT * FROM Titles WHERE [Year Published] BETWEEN 1995 AND 1998



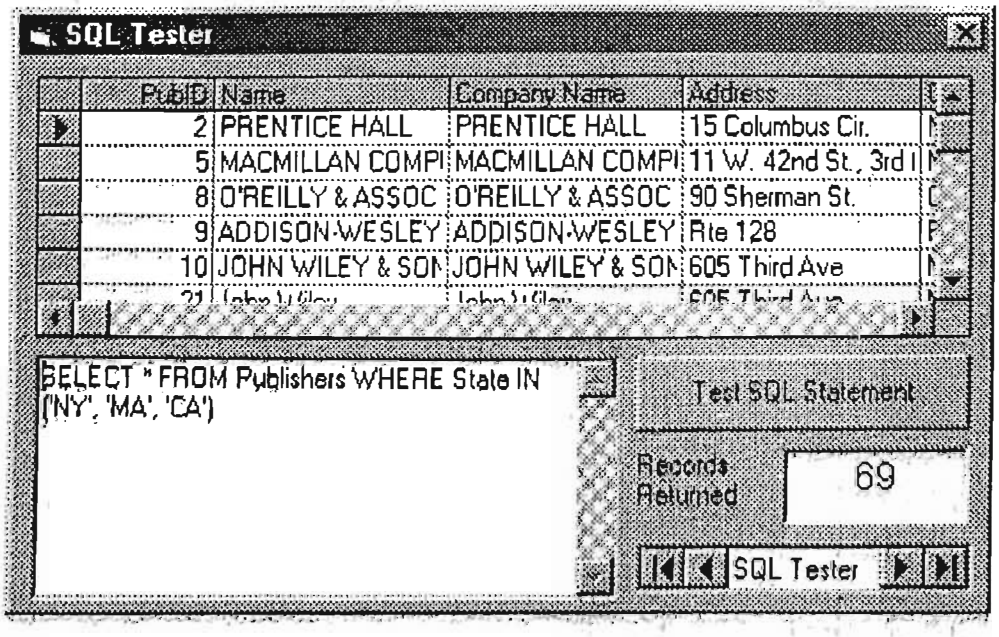
Title	Year Published	ISBN	PubID	Des
Oracle Triggers and	1996	0-0134436-3-1	715	0
Structured C for Engi	1995	0-0230081-2-1	715	54
An Introduction to As	1995	0-0230362-0-6	715	60
Applied Calculus Wit	1995	0-0230650-8-7	119	70.6
Programming the 802	1996	0-0231426-3-4	715	0

```
SELECT * FROM Titles WHERE [Year Published]
BETWEEN 1995 AND 1998
```

Records Returned: 1860

- The **IN** keyword lets us specify a comma-delimited list of desired values in the returned recordset. Say, we want to know the publishers in New York, Massachusetts, and California. This SQL statement will do the trick:

SELECT * FROM Publishers WHERE State IN ('NY', 'MA', 'CA')



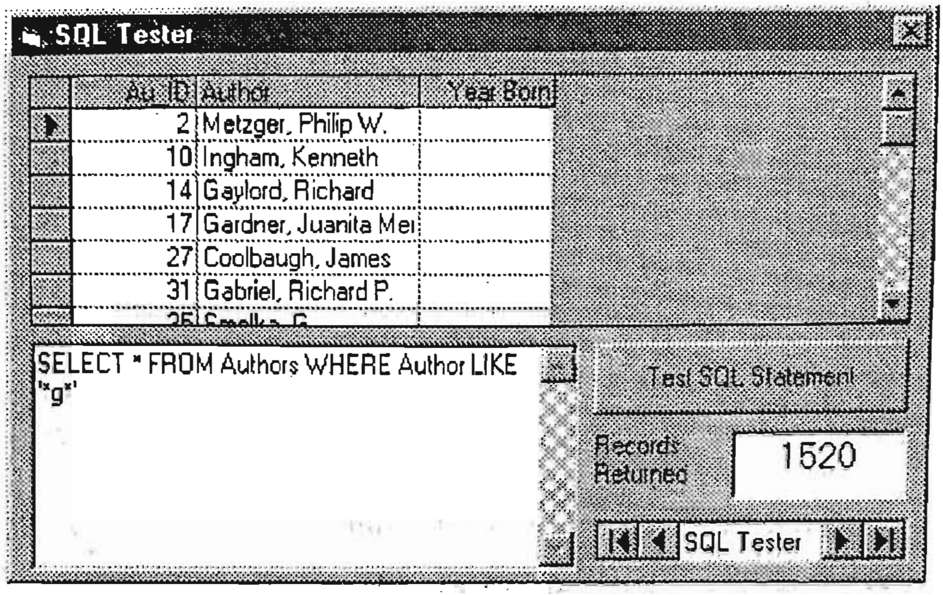
PubID	Name	Company Name	Address
2	PRENTICE HALL	PRENTICE HALL	15 Columbus Cir.
5	MACMILLAN COMPI	MACMILLAN COMPI	11 W. 42nd St., 3rd I
8	O'REILLY & ASSOC	O'REILLY & ASSOC	90 Sherman St.
9	ADDISON-WESLEY	ADDISON-WESLEY	Rte 128
10	JOHN WILEY & SON	JOHN WILEY & SON	605 Third Ave

```
SELECT * FROM Publishers WHERE State IN
('NY', 'MA', 'CA')
```

Records Returned: 69

- The **LIKE** keyword allows us to use wildcards in the WHERE clause. This lets us find similar fields. Recall, the Jet engine wildcard character is the asterisk (*). To find all authors with a 'g' anywhere in the their name, try:

SELECT * FROM Authors WHERE Author LIKE '*g*'



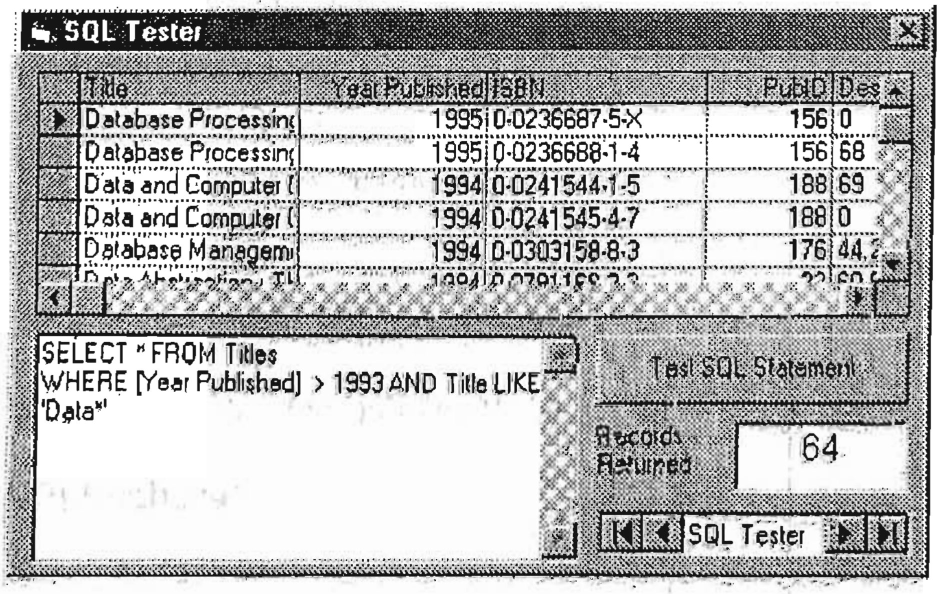
Au_ID	Author	Year Born
2	Metzger, Philip W.	
10	Ingham, Kenneth	
14	Gaylord, Richard	
17	Gardner, Juanita Mei	
27	Coolbaugh, James	
31	Gabriel, Richard P.	

```
SELECT * FROM Authors WHERE Author LIKE
'*g*'
```

Records Returned 1520

- Multiple criteria are possible by using the logical operators **AND** and **OR**. For example, to find all books in the Titles table published after 1993 with a title that starts with the letters Data, we would use the SQL statement:

SELECT * FROM Titles
WHERE [Year Published] > 1993 AND Title LIKE 'Data*'



Title	Year Published	ISBN	PubID	Des
Database Processing	1995	0-0236687-5-X	156	0
Database Processing	1995	0-0236688-1-4	156	68
Data and Computer (	1994	0-0241544-1-5	188	69
Data and Computer (	1994	0-0241545-4-7	188	0
Database Managem	1994	0-0303158-8-3	176	44.2

```
SELECT * FROM Titles
WHERE [Year Published] > 1993 AND Title LIKE
'Data*'
```

Records Returned 64

Multiple Table WHERE Clause

- So far, almost everything we've done in this course has involved looking at a single native (built-in) table in a database. This has been valuable experience in helping us understand database design, learning how to use the Visual Basic database tools, and learning some simple SQL statements. Now, we begin looking at one of the biggest uses of database management systems - combining information from multiple tables within a database. SQL makes such combinations a simple task.

- We still use the same SELECT/FROM syntax, along with the WHERE and ORDER BY clauses to form our new virtual tables:

  SELECT [Fields]
  FROM [Tables]
  WHERE Criteria
  ORDER BY [Fields]

  The only difference here is there's more information in each SQL statement, resulting is some very long statements. The [Fields] list will have many fields, the [Tables] list will have multiple tables, and the Criteria will have several parts. The basic idea is to have the SQL statement specify what fields you want displayed (**SELECT**), what tables those fields are found in (**FROM**), how you want the tables to be combined (**WHERE**), and how you want them sorted (**ORDER BY**). Let's try an example.

- Notice the Titles table does not list a book's publisher, but just publisher identification (PubID). What if we want to display a book's title (**Title** field in **Titles** table) and publisher (**Company Name** in **Publishers** table) in the same recordset? Let's build the SQL statement. First, the SELECT clause specifies the fields we want in our 'virtual' table:

  SELECT Titles.Title,Publishers.[Company Name]

  Note the use of dot notation to specify the desired fields. With multiple tables, this avoids any problems with naming ambiguities.

- The FROM clause names the tables holding these fields:

  FROM Titles,Publishers

- The WHERE clause declares what criteria must be met in combining the two tables. The usual selection is to match a **primary key** in one table with the corresponding **foreign key** in another table. Here, we want the publisher identification numbers from each table to match:

WHERE Titles.PubID = Publishers.PubID

Any records from the tables that do not match the WHERE criteria are not included in the returned recordset.

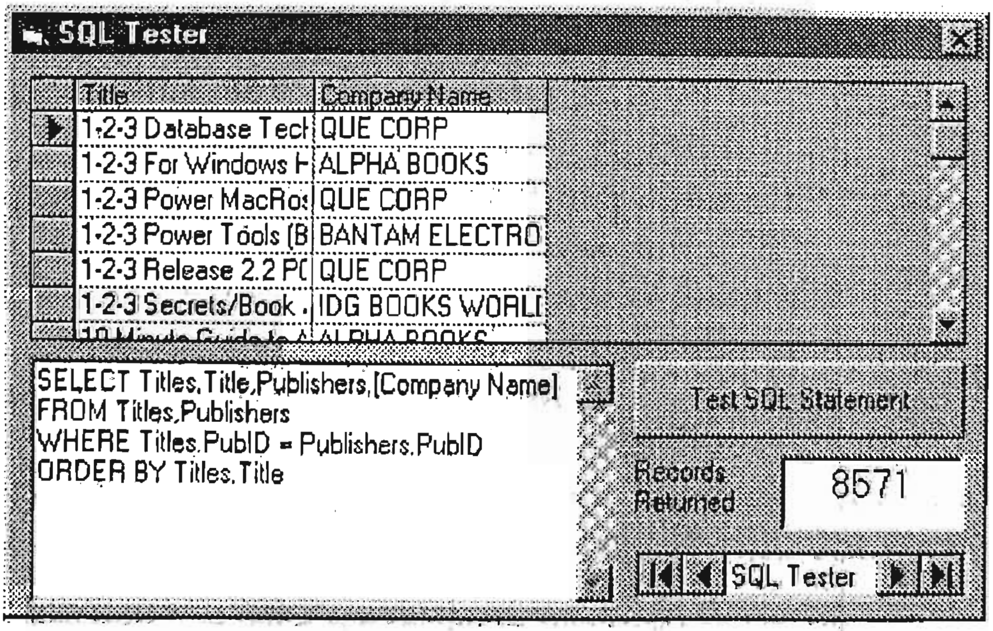
- Lastly, we declare how we want the resulting recordset to be sorted:

ORDER BY Titles.Title

- The complete SQL statement is thus:

```
SELECT Titles.Title,Publishers.[Company Name]
FROM Titles,Publishers
WHERE Titles.PubID = Publishers.PubID
ORDER BY Titles.Title
```

Try this with the SQL tester.



Title	Company Name
1-2-3 Database Tecl	QUE CORP
1-2-3 For Windows H	ALPHA BOOKS
1-2-3 Power MacRo:	QUE CORP
1-2-3 Power Tools (B	BANTAM ELECTRO
1-2-3 Release 2.2 PC	QUE CORP
1-2-3 Secrets/Book .	IDG BOOKS WORLI
10 Minute Guide to A	ALPHA BOOKS

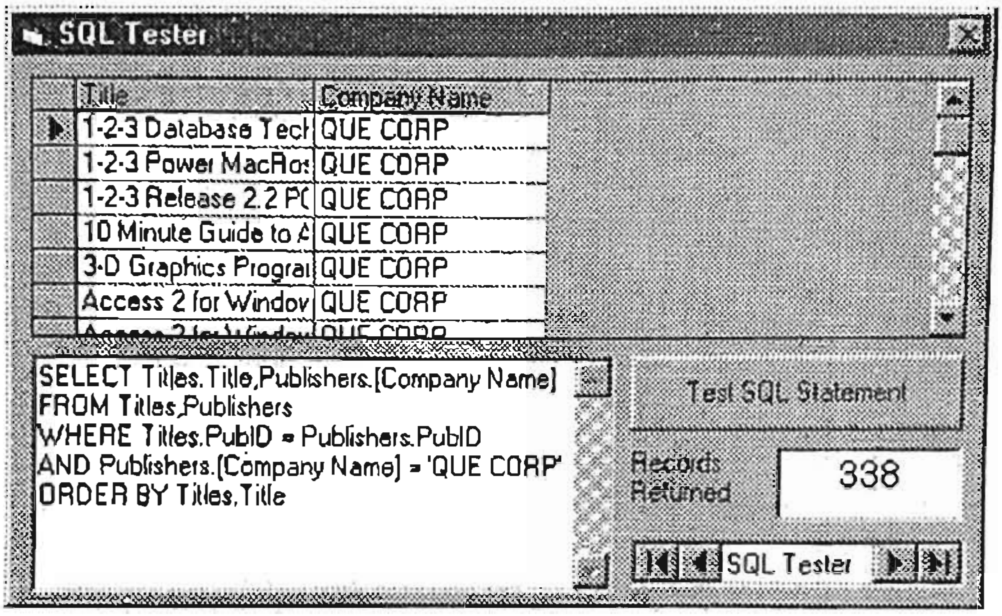
```
SELECT Titles.Title,Publishers.[Company Name]
FROM Titles.Publishers
WHERE Titles.PubID = Publishers.PubID
ORDER BY Titles.Title
```

Records Returned: 8571

- Are you amazed? You have just seen one of the real powers of using SQL with the Jet database engine (or any database system, for that matter). We simply told the engine what we wanted (via the SQL statement) and it did all of the work for us - no coding needed! Let's do some more examples.

- In the previous example, say you just want books published by Que Corporation. Modify the SQL statement to read (we added an AND clause):

SELECT Titles.Title,Publishers.[Company Name]
FROM Titles,Publishers
WHERE Titles.PubID = Publishers.PubID
AND Publishers.|Company Name| = 'QUE CORP'
ORDER BY Titles.Title



Title	Company Name
1-2-3 Database Tech	QUE CORP
1-2-3 Power MacRo:	QUE CORP
1-2-3 Release 2.2 PC	QUE CORP
10 Minute Guide to A	QUE CORP
3-D Graphics Progra	QUE CORP
Access 2 for Window	QUE CORP
Access 2 for Window	QUE CORP

```
SELECT Titles.Title,Publishers.[Company Name]
FROM Titles,Publishers
WHERE Titles.PubID = Publishers.PubID
AND Publishers.[Company Name] = 'QUE CORP'
ORDER BY Titles.Title
```

Records Returned: 338

- What if we want to list a book's title, publisher, <u>and</u> author, ordered by the author names This requires using all four tables in the BIBLIO.MDB database. Let's build the SQL statement. We want three fields:

SELECT Authors.Author,Titles.Title,Publishers.[Company Name]

As mentioned, to retrieve this information requires all four tables:

FROM Authors,Titles,Publishers,|Title Author|

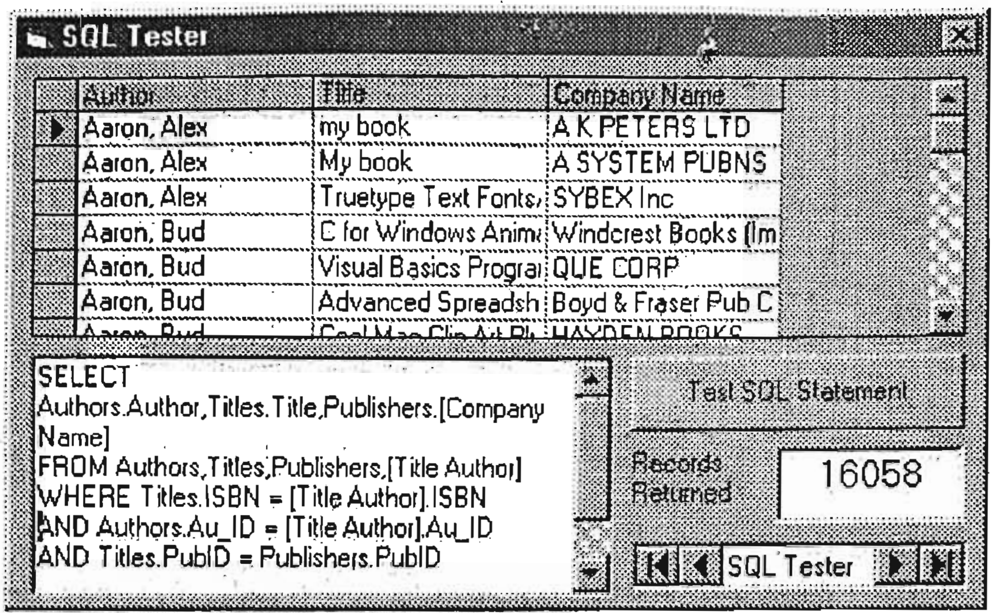
We still need the publisher identification numbers to match, but now also need to make sure book titles (via the ISBN field) and author identification numbers match. The corresponding WHERE clause is:

WHERE Titles.ISBN = [Title Author|.ISBN
AND Authors.Au_ID = [Title Author].Au_ID
AND Titles.PubID = Publishers.PubID

Finally, the results are sorted:

ORDER BY Authors.Author

Putting all this in the SQL tester gives us over 16,000 listings (one entry for every author and every book he or she wrote or co-wrote):



Author	Title	Company Name
Aaron, Alex	my book	A K PETERS LTD
Aaron, Alex	My book	A SYSTEM PUBNS
Aaron, Alex	Truetype Text Fonts,	SYBEX Inc
Aaron, Bud	C for Windows Anim	Windcrest Books (Im
Aaron, Bud	Visual Basics Progra	QUE CORP
Aaron, Bud	Advanced Spreadsh	Boyd & Fraser Pub C
Aaron, Bud	Cool Mac Clip Art Pl	HAYDEN BOOKS

```
SELECT
Authors.Author,Titles.Title,Publishers.[Company
Name]
FROM Authors,Titles,Publishers,[Title Author]
WHERE Titles.ISBN = [Title Author].ISBN
AND Authors.Au_ID = [Title Author].Au_ID
AND Titles.PubID = Publishers.PubID
```

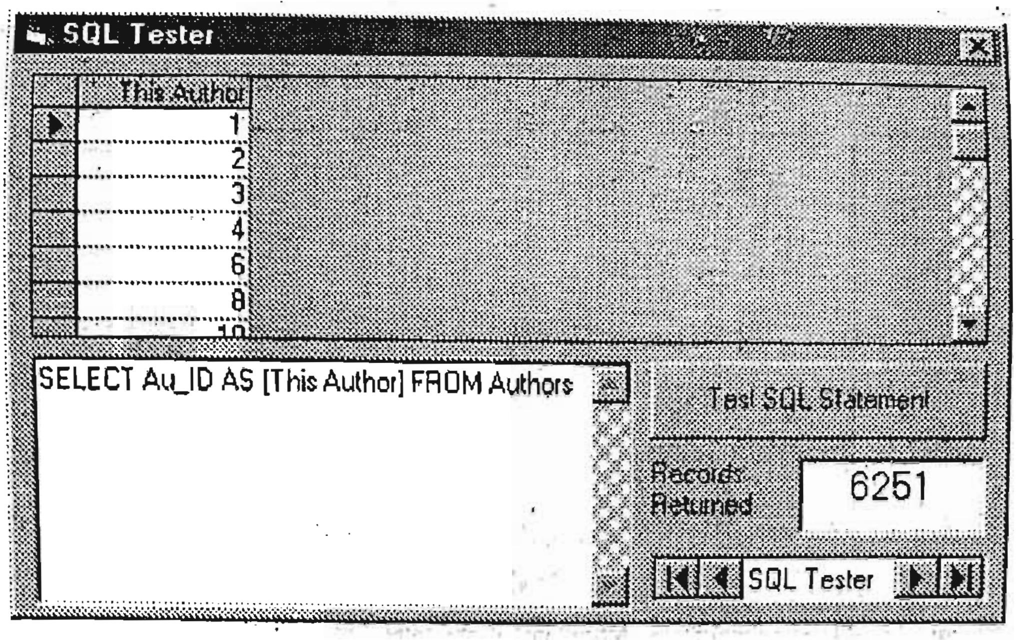
Records Returned: 16058

Such power! Can you imagine trying to write BASIC code to perform this record retrieval task?

If the displayed field name does not clearly describe the displayed information, you can **alias** the name, or change it to something more meaningful using the AS clause. As a simple example, try this:

SELECT Au_ID AS [This Author] FROM Authors

the displayed column is now **This Author**.



```
SELECT Au_ID AS [This Author] FROM Authors
```

Records Returned: 6251

The field name is unaffected by aliasing - only the displayed name changes.

- **Important** - Database tables combined (forming a virtual data view) using DAO technology and the SQL WHERE clause cannot be updated. The data can only be viewed. Go ahead - combine tables using a SQL statement with a WHERE clause and try to change a value in the resulting grid. You can't do it! The ability to update a DAO recordset is established by the read-only **Updatable** property. Is this a problem? Not if you are just displaying information for a user. But, if you need editing capabilities with DAO, do not use the WHERE clause to join tables.

- Any recordset established using ADO technology (even with a combining WHERE clause) can be updated, depending on **locks** placed on the recordset. The use of such locks is discussed in a later chapter,

- To provide editing in DAO recordset, you need to use the SQL **JOIN** clauses. Using JOIN clauses will also work with ADO recordsets. Let's take a look at such a clause.

INNER JOIN Clause

- When combining tables, the SQL **INNER JOIN** clause does the same work as the WHERE clause and it returns a recordset that <u>can</u> be updated (for both DAO and ADO technologies). The syntax for an INNER JOIN is a little different than that of the WHERE clause.

```
SELECT [Fields]
FROM Table1 INNER JOIN Table2 ON Linking Criteria
WHERE Criteria
ORDER BY [Fields]
```

This rather long statement begins by specifying the fields to **SELECT**. The **FROM** clause specifies the fields will come from the first table (Table1) being **INNER JOIN**ed with a second table (Table2). The **ON** clause states the linking criteria (usually a matching of key values) to be used in the join. At this point, the tables are combined. You can still use a **WHERE** clause to extract specific information from this table (you just can't use it to combine tables) and an **ORDER BY** clause, if desired. Let's repeat the examples just done with the WHERE clause.

- To display a book title and publisher name, the SELECT clause is:

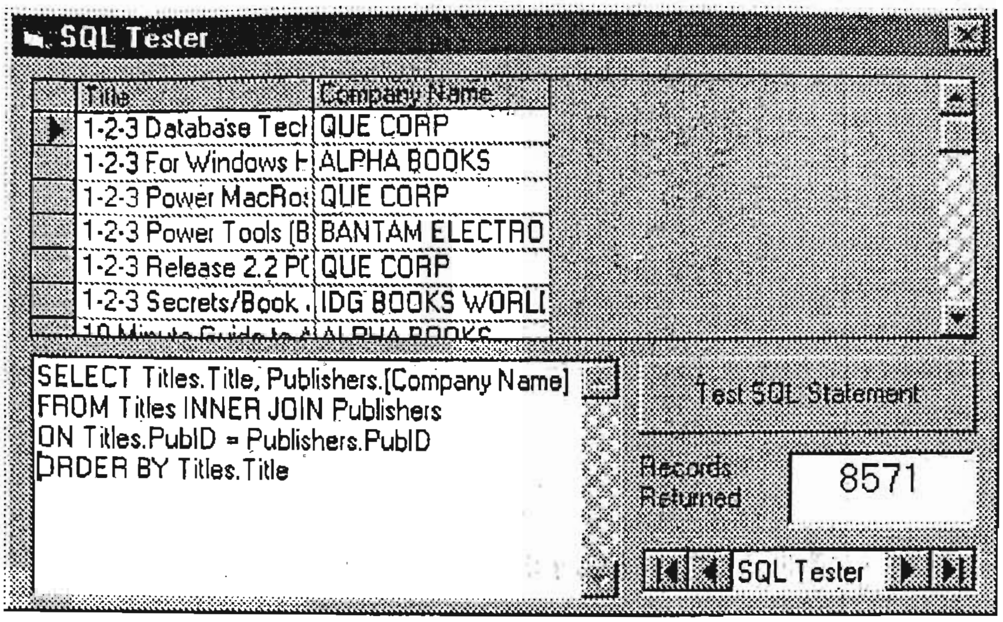
```
SELECT Titles.Title, Publishers.[Company Name]
```

We want to 'join' the Titles table with the Publishers table, making sure the PubID fields match. The corresponding INNER JOIN statement is:

```
FROM Titles INNER JOIN Publishers
ON Titles.PubID = Publishers.PubID
```

Lastly, we order by the Title:

```
ORDER BY Titles.Title
```



Title	Company Name
1-2-3 Database Tecl	QUE CORP
1-2-3 For Windows H	ALPHA BOOKS
1-2-3 Power MacRo:	QUE CORP
1-2-3 Power Tools (B	BANTAM ELECTRO
1-2-3 Release 2.2 P(	QUE CORP
1-2-3 Secrets/Book .	IDG BOOKS WORLI
10 Minute Guide to A	ALPHA BOOKS

```
SELECT Titles.Title, Publishers.[Company Name]
FROM Titles INNER JOIN Publishers
ON Titles.PubID = Publishers.PubID
ORDER BY Titles.Title
```

Records Returned 8571

- Try to change a value in the data grid for this example. You should see that, as expected, use of the INNER JOIN provides an **updatable** recordset. If you leave your change as is, it will be written as a permanent modification to the database! So, we suggest 'undoing' your change. You have just learned one of your first skills in building a complete database management system - how to edit an existing database. It was easy, wasn't it? This ease comes from the power of the Jet database engine. There are times we won't want editing the database to be so easy. Limiting these capabilities are discussed in the next chapter on Visual Basic interfaces.

To illustrate use of the WHERE clause (to limit displayed records) in conjunction with the JOIN clause, try this modified SQL statement with SQL Tester:

SELECT Titles.Title, Publishers.[Company Name]
FROM Titles INNER JOIN Publishers
ON Titles.PubID = Publishers.PubID
WHERE Publishers.[Company Name] = 'QUE CORP'
ORDER BY Titles.Title

Only QUE CORP publishers will be listed. And, the recordset can still be edited (WHERE only affects 'updatability' of DAO recordsets when used to combine information on tables).

Use of the INNER JOIN clause to combine information from more than two tables is a little more complicated. The tables need be joined in stages, nesting the INNER JOIN clauses using parentheses for grouping. Assume we have three tables (**Table1**, **Table2**, **Table3**) we want to combine. Table1 and Table3 have a common key field for linking (**Key13**), as do Table2 and Table3 (**Key23**). Let's combine these three tables using INNER JOIN. In the first stage, we form a temporary table that is a result of joining Table2 and Table3 using Key23 for linking:

```
Table2 INNER JOIN Table3 ON Table2.Key23 = Table3.Key23
```

In the next stage, we join Table1 with this temporary table (enclose it in parentheses) using Key13 for linking:

```
Table1 INNER JOIN
(Table2 INNER JOIN Table3 ON Table2.Key23 = Table3.Key23)
ON Table1.Key13 = Table3.Key13
```

This nested statement is used in the SQL statement to specify the tables for field selection. Notice we've spread this over a few lines to make it clearer - any SQL processor can handle multiple line statements. The multiple table INNER JOIN can be generalized to more tables - just pay attention to what tables link with each other. Always make sure the tables you are joining, whether a temporary joined table or a database table, have a common key.

Remember the example we did earlier where we listed Author, Title, and Publisher in the BIBLIO.MDB database? Let's build that SQL statement. First, SELECT the fields:

```
SELECT Authors.Author,Titles.Title,Publishers.[Company Name]
```

This is the same SELECT we used previously. Now, we need to form the FROM clause by combining four tables in three stages (one for each common key linking). In the first stage, combine the Publishers and Titles tables (PubID is common key):

```
Publishers INNER JOIN Titles
ON Publishers.PubID=Titles.PubID)
```

```
(Publishers INNER JOIN Titles
ON Publishers.PubID=Titles.PubID)
INNER JOIN [Title Author]
ON Titles.ISBN=[Title Author].ISBN
```

In the final stage, join the Authors table with this temporary table (enclose its statement in parentheses) using Au_ID as the key:

```
Authors INNER JOIN
((Publishers INNER JOIN Titles
ON Publishers.PubID=Titles.PubID)
INNER JOIN [Title Author]
ON Titles.ISBN=[Title Author].ISBN)
ON Authors.Au_ID=[Title Author].Au_ID
```

The **FROM** clause needed for the combined data view is now complete. The final line in the SQL statement orders the data:

```
ORDER BY Authors.Author
```

Whew! Try this full statement with the SQL tester and you should get the same results seen earlier using the WHERE clause. The difference, of course, is that the recordset here can be updated.

OUTER JOIN Clause

- The INNER JOIN only retrieves records that have a match on both sides of the JOIN. For example, with BIBLIO.MDB, the INNER JOIN statement:

  Publishers INNER JOIN Titles ON Publishers.PubID = Titles.PubID

  只,返回二者都有ID的记录

  In this statement, if there is a PubID in the Publishers table without a corresponding PubID in the Titles table, that value will not be in the returned recordset. If you want all records returned, whether there is a match or not, you need to use what is called an **OUTER JOIN**. There are two forms for the OUTER JOIN.

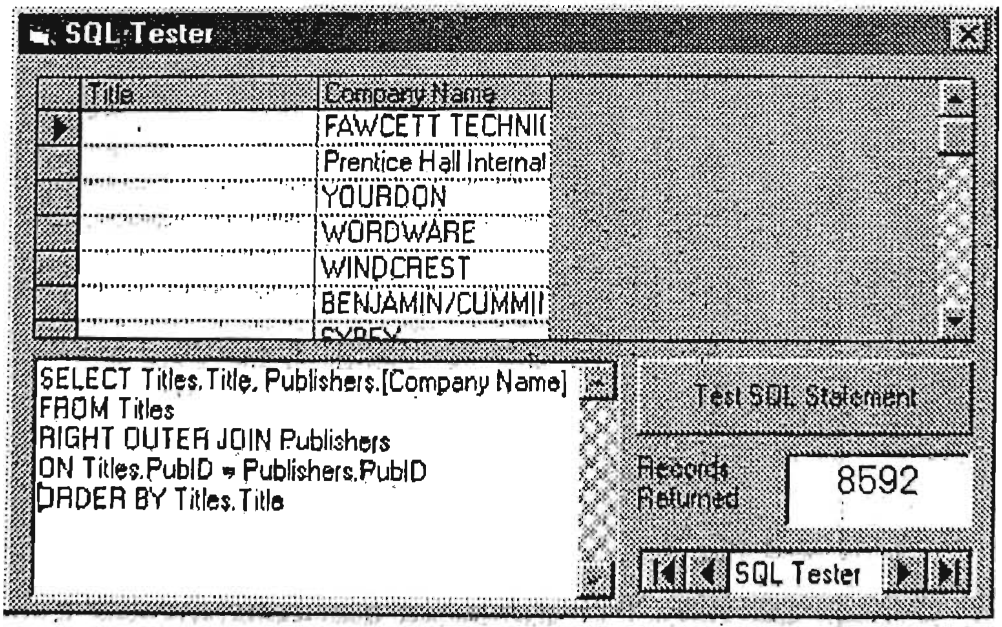
  返回一个表中的全部记录，两种情况

- A **RIGHT OUTER JOIN** includes all records from the second-named table (the right-most table), even if there are no matching values for records in the first-named (left-most table). Try this with SQL Tester:

  → 返回第二个表中的全部记录，而不管第一个表中是否有配匹的ID

```
SELECT Titles.Title, Publishers.[Company Name]
FROM Titles
RIGHT OUTER JOIN Publishers
ON Titles.PubID = Publishers.PubID
ORDER BY Titles.Title
```

→ 第二个表（在右边）



Title	Company Name
	FAWCETT TECHNI(
	Prentice Hall Internat
	YOURDON
	WORDWARE
	WINDCREST
	BENJAMIN/CUMMII
	SYBEX

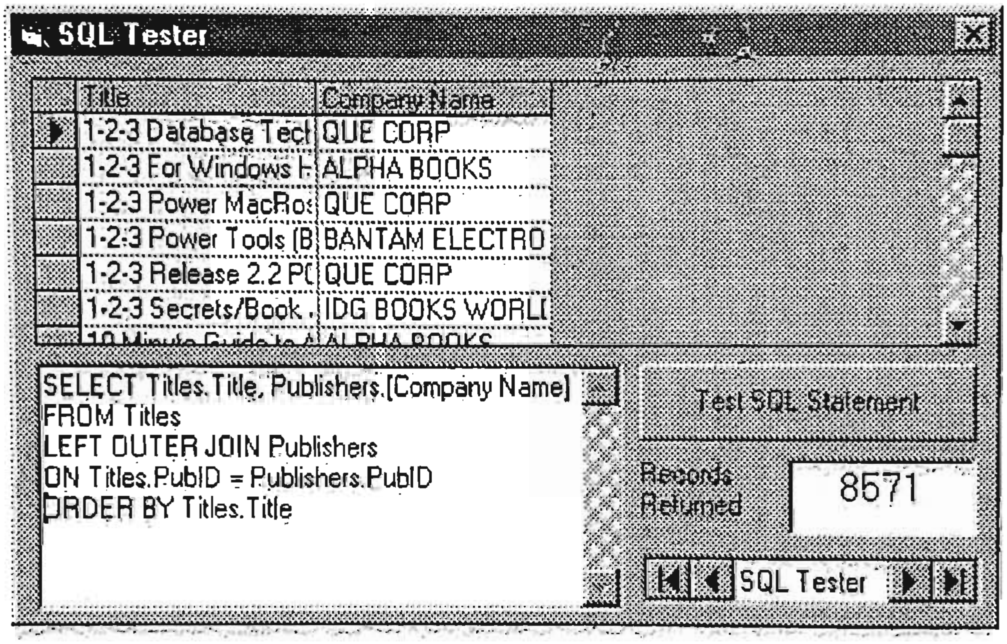
```
SELECT Titles.Title, Publishers.[Company Name]
FROM Titles
RIGHT OUTER JOIN Publishers
ON Titles.PubID = Publishers.PubID
ORDER BY Titles.Title
```

Records Returned 8592

There are several publishers (about 19 or so) without corresponding titles in the database.

- A ***LEFT OUTER JOIN*** includes all records from the first-named table (the left-most table), even if there are not matching values for records in the second-named (right-most table). Try this with SQL Tester; →第一个表要返回全部数据

SELECT Titles.Title, Publishers.[Company Name]
FROM Titles 第一个表（在语句的左边）
LEFT OUTER JOIN Publishers
ON Titles.PubID = Publishers.PubID
ORDER BY Titles.Title



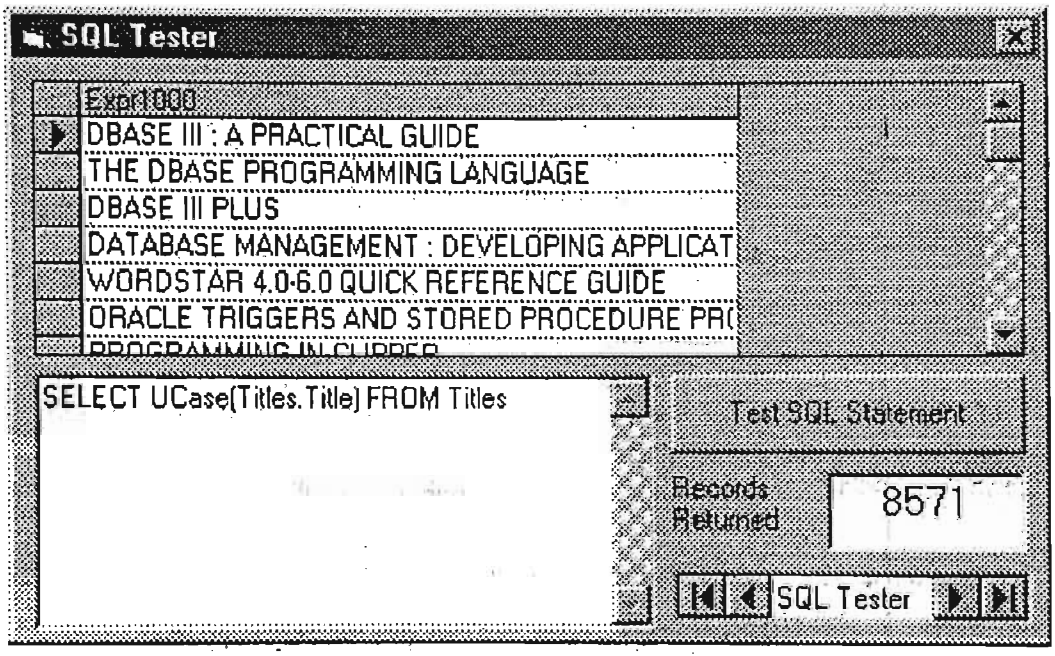
Title	Company Name
1-2-3 Database Tecl	QUE CORP
1-2-3 For Windows H	ALPHA BOOKS
1-2-3 Power MacRo:	QUE CORP
1-2-3 Power Tools (B	BANTAM ELECTRO
1-2-3 Release 2.2 PC	QUE CORP
1-2-3 Secrets/Book .	IDG BOOKS WORLI
10 Minute Guide to A	ALPHA BOOKS

```
SELECT Titles.Title, Publishers.[Company Name]
FROM Titles
LEFT OUTER JOIN Publishers
ON Titles.PubID = Publishers.PubID
ORDER BY Titles.Title
```

The returned recordset is identical to that obtained with the INNER JOIN. Obviously, all books in the database have a corresponding publisher - that's actually a good thing.

- The Jet database engine allows you to use any valid BASIC function as part of statement. This lets you modify the displayed information. It does not aff underlying information in the database. As an example, say you want all book the BIBLIO.MDB Titles database to be listed in upper case letters. Try th: statement with SQL Tester:

SELECT UCase(Titles.Title) FROM Titles



```
Expr1000
DBASE III : A PRACTICAL GUIDE
THE DBASE PROGRAMMING LANGUAGE
DBASE III PLUS
DATABASE MANAGEMENT : DEVELOPING APPLICAT
WORDSTAR 4.0-6.0 QUICK REFERENCE GUIDE
ORACLE TRIGGERS AND STORED PROCEDURE PR(
PROGRAMMING IN CLIPPER
```

```
SELECT UCase(Titles.Title) FROM Titles
```

Notice SQL assigns a heading of Expr1000 to this 'derived' field. We can use t feature of SQL change this heading to anything we want (except the name of an field). Try this:

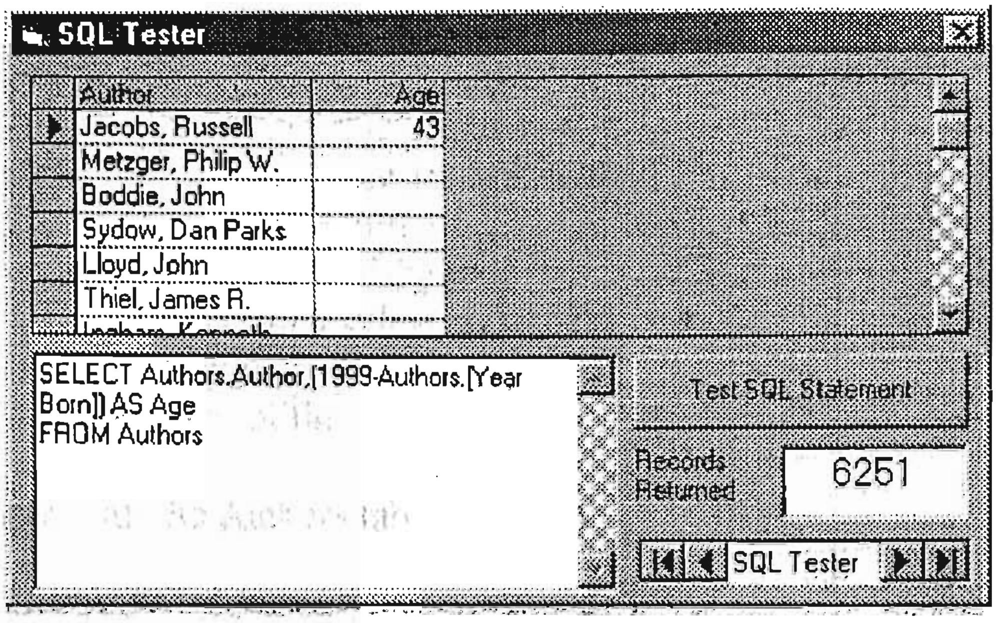
SELECT UCase(Titles.Title) AS Title FROM Titles

- Or, what if we had some process that could only use the 10 left-most character book title. This SQL statement will do the trick:

SELECT UCase(Titles.Title) AS Title FROM Titles

- You can also do BASIC math in a SQL statement. The BIBLIO.MDB database Authors table has **Year Born** as a field. This SQL statement will display each author and their age in 1999 (when this is being written):

SELECT Authors.Author,(1999-Authors.[Year Born]) AS Age
FROM Authors



Author	Age
Jacobs, Russell	43
Metzger, Philip W.	
Boddie, John	
Sydow, Dan Parks	
Lloyd, John	
Thiel, James R.	

```
SELECT Authors.Author,(1999-Authors.[Year
Born]) AS Age
FROM Authors
```

Records Returned: 6251

Note that most of the listings do not have an Age value. The reason for this is because only a few of the author records have birth year entries - the entries are **NULL** (containing no information).

NULL is a special value meaning there is nothing there - this is not the same as an empty string or blank space. In our work, we will avoid placing NULLs in a database, but they may exist in other databases. You need to decide how to handle NULLs in your design. We will see examples where they cause problems. A NULL field can be tested using the SQL functions **IS NULL** and **IS NOT NULL**. We can add this to the SQL statement above to find just the Authors records with a birth year:

SELECT Authors.Author,(1999-Authors.[Year Born]) AS Age
FROM Authors
WHERE Authors.[Year Born] IS NOT NULL

You should now find 20 authors with ages listed.

SQL Aggregate Functions

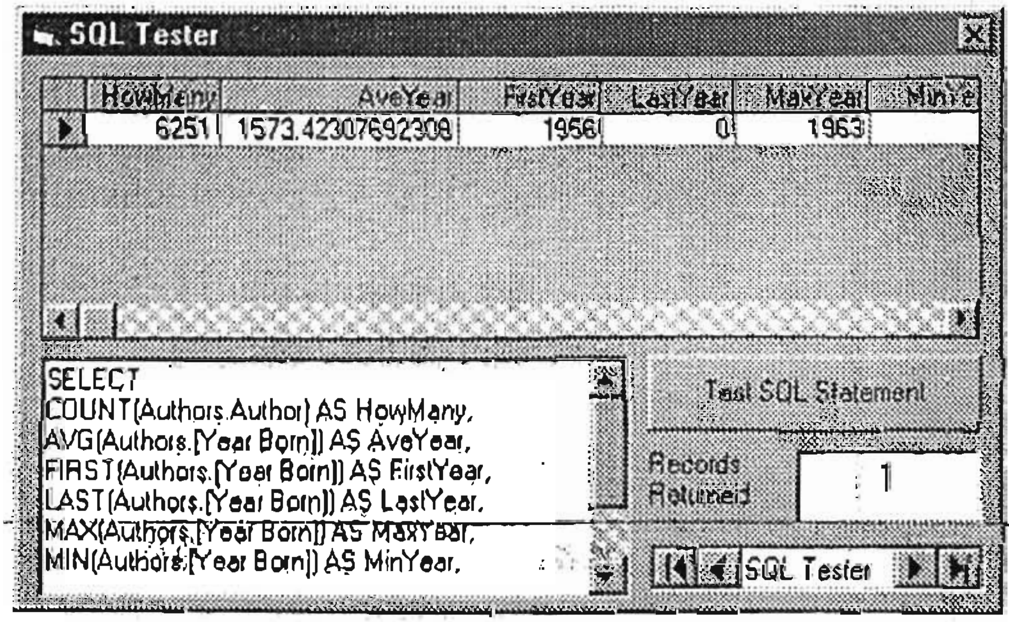
- In addition to BASIC functions, the Jet database engines supports the standard **SQL aggregate functions**. These are functions that let you compute summary statistics for fields in your database, alias the results, and display them in a recordset. NULL fields are ignored by the aggregate functions.

- The aggregate functions and their results are:

AVG(Field)	Average value of the field
COUNT(Field)	Number of entries for the field
FIRST(Field)	First value of the field
LAST(Field)	Last value of the field
MAX(Field)	Maximum value of the field
MIN(Field)	Minimum value of the field
SUM(Field)	Sum of the field values

- Try this example with the Authors table:

```
SELECT
COUNT(Authors.Author) AS HowMany,
AVG(Authors.[Year Born]) AS AveYear,
FIRST(Authors.[Year Born]) AS FirstYear,
LAST(Authors.[Year Born]) AS LastYear,
MAX(Authors.[Year Born]) AS MaxYear,
MIN(Authors.[Year Born]) AS MinYear,
SUM(Authors.[Year Born]) AS SumYear
FROM Authors
```



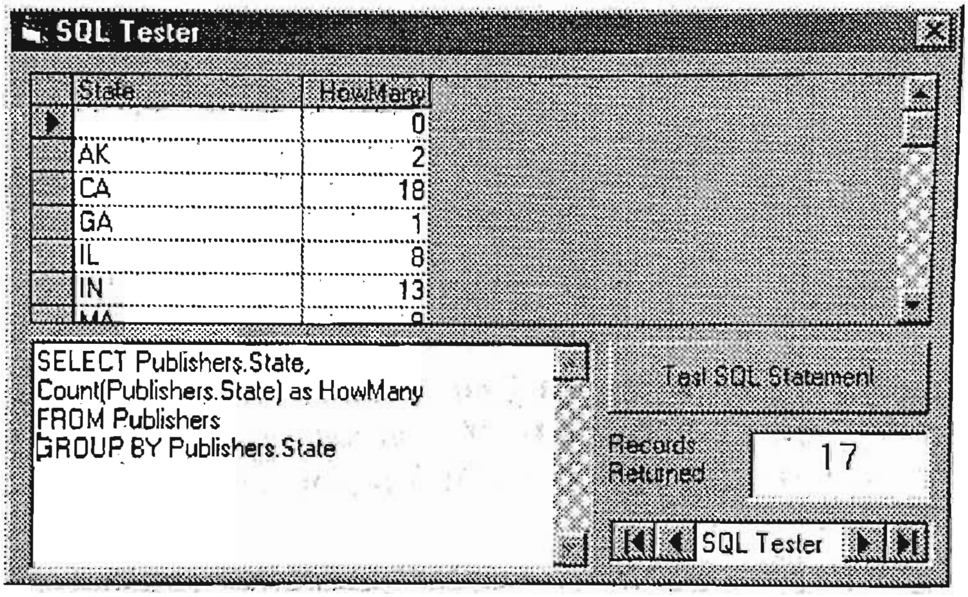
HowMany	AveYear	FirstYear	LastYear	MaxYear	MinYe
6251	1573.42307692308	1956	0	1963	

```
SELECT
COUNT(Authors.Author) AS HowMany,
AVG(Authors.[Year Born]) AS AveYear,
FIRST(Authors.[Year Born]) AS FirstYear,
LAST(Authors.[Year Born]) AS LastYear,
MAX(Authors.[Year Born]) AS MaxYear,
MIN(Authors.[Year Born]) AS MinYear,
```

Note some of the aggregate fields (FirstYear, LastYear) have no values since these are NULL fields.

- Aggregate functions can be used to group results. The **GROUP BY** clause lets you determine records with duplicate field values. Want to know how many publishers in your database are in each state? Try this SQL statement:

SELECT Publishers.State, Count(Publishers.State) as HowMany
FROM Publishers
GROUP BY Publishers.State



State	HowMany
	0
AK	2
CA	18
GA	1
IL	8
IN	13
MA	9

```
SELECT Publishers.State,
Count(Publishers.State) as HowMany
FROM Publishers
GROUP BY Publishers.State
```

Records Returned 17

- You can use the HAVING qualifier to further reduce the grouping obtained with a GROUP BY clause. Say in the above example, you only want to display states starting with the letter M (a strange request, we know). This SQL state will do the trick (try it):

SELECT Publishers.State, Count(Publishers.State) as HowMany
FROM Publishers
GROUP BY Publishers.State
HAVING Publishers.State LIKE 'M*'

连接表

1。常规的内部连接或相等连接

最常见的连接类型是相等连接（或称为 INNER JOIN)。例如：

SELECT Publishers.Name,Titles.ISBN,Titles.Title FROM Publishers INNER JOIN Titles ON Publishers.PubID=Titles.PubID

也可以写成：

SELECT Publishers.Name,Titles.ISBN,Titles.Title FROM Publishers,Titles WHERE Publishers.PubID=Titles.PubID

2。多重相等连接

例如：

SELECT Titles.Title,Publishers.Name,Titles.ISBN,Authors.Author FROM Publishers,Titles,Authors,[Title Author] WHERE Titles.ISBN=[Title Author].ISBN AND Authors.Au_ID=[Title Author].Au_ID AND Publishers.PubID=Tiltes.PubID

3。外部连接

INNER JOIN 只返回与字段值相匹配的行，OUTER JOIN 返回一个表的所有行和另一个表中有匹配值的行。有两种类型的 OUTER JOIN。

LEFT OUTER JOIN 返回 LEFT OUTER JOIN 语句左边表或结果集的所有行和语句右边表具有匹配值的行。WHERE 子句中，*=操作符特指 LEFT OUTER JOIN
RIGHT OUTER JOIN 返回 RIGHT OUTER JOIN 语句右边表或结果集的所有行和语句左边表具有匹配值的行。WHERE 子句中，=*操作符表求 RIGHT OUTER JOIN。通常地，按一到多的形式创建。也就是说，代表关系中“一”方的主要表出现在 JOIN 表达式左边或 WHERE 子句中操作符左边，而关系中“多”方的相关表出现在表达式或操作符的右边。LEFT OUTER JOIN 可以显示主表所有记录，而不管相关表中的匹配记录；RIGHT OUTER JOIN 对寻找孤儿记录很有用。所谓“孤儿记录”就是那些在主表中没有相关记录的相关表中的记录，孤儿记录是违反引用一致性规则的结果。

注意：Jet SQL 不支持 WHERE 子句中的*=和=*操作符。用 Jet 数据库引擎时要用 LEFT

JOIN 或 RIGHT JOIN 保留字来创建外部连接。

4。DISTINCTROW

限定词 DISTINCT 告诉查询只返回在 SELECT 语句指定字段中具有不同值的行，Jet SQL 的限定词 DISTINCTROW 使得返回集包括在两个表任意字段中有异值的行（而不只是 SELECT 语句指定显示的字段）

5。自连接和复合列

自连接是建立在同一表中具有相似字段数据类型两个字段之间的一个连接。第一个字段常为主键字段，而第二个字段常为与主键字段相关的外部键字段，尽管这种结构并不是自连接必需的(为使自连接结果有意义，前一个要求也许是需要的)。

建立自连接时，RDBMS 建立原表的一个副本，然后把副本连接到原表上。以 VB 自带的 Nwind.mdb 为例。其中的 Employees 表包含 ReportsTo 字段来表示雇员主管的 EmployeeID.在 Employees 表上建立自连接以显示雇员主管名字的 Jet SQL 语句如下

```
SELECT Employees.EmployeeID AS EmpID,Employees.LastName & "," &
Employees.FirstName AS Employee,Employees.ReportsTo AS SupId,
EmpCopy.LastName & "," & EmpCopy.FirstName AS Supervisor FROM Employees
Employees AS EmpCopy,Employees INNER JOIN EmpCopy ON
Employees.ReportsTo=EmpCopy.EmployeeID
```

SQL 汇总函数和 GROUP BY 及 HAVING 子句

1。汇总函数

COUNT（field_name）返回 field_name 中包含 NOT NULL 值的行数，COUNT（*）返回表或查询的行数，不考虑字段的 NULL 值。
MAX（field_name）返回集合中 field_name 的最大值
MIN（field_name）返回集合中 field_name 的最小值
SUM（field_name）返回集合中 field_name 值的和
AVG（field_name）返回集合中 field_name 值的算术平均值
下面是使用汇总函数的一个例子：

```
SELECT COUNT(*) AS Count,SUM(PubID) AS Total,AVG(PubID) AS
Average,MIN(PubID) AS Minimum,MAX(PubID) AS Maximum FROM publishers
```

2。GROUP BY 和 HAVING

有重要内容的数据库常常有包含代表分类的字段的表，例如 Nwind.mdb 中的 Products 表把一种外来食品的品种归为 8 个类别。当要为一个对象的每个类获得 SQL 汇总函数值时，可以使用 GROUP BY 子句。GROUP BY 子句创建一个虚表称为分组表。如果想用特殊标准来限制组(类)成员关系，WHERE 子句可以建立标准，然而，WHERE 作用于整个表。HAVING 子句和 WHERE 子句一样，但它作用于组，而且把 SQL 汇总函数的适用性限制在一个组的特定集合上。所以可以加上带有 IN 操作符的 HAVING 子句。 例如：

```
SELECT CategoryID AS Categroy,COUNT(ProductID) AS Item, SUM(UnitsInStock) AS Sum_Stock FROM Products GROUP BY CategoryID HAVING CategoryID IN(1,2)
```

SQL Construction Tools

- We've completed our review of the SQL language. There are other commands we haven't looked at. If you would like to know more, there are numerous references available for both ANSI standard SQL and the Jet database engine version. You now know how to construct SQL statements to extract desired information from a multi-table database and you know how to read other's SQL statements.

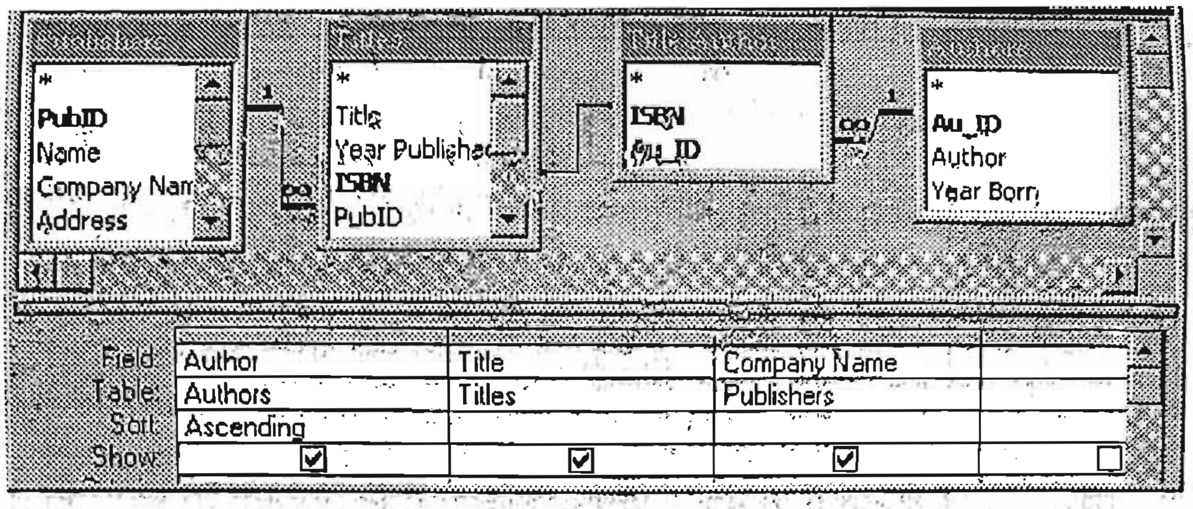
- You have seen that constructing SQL statements is, at times, a tedious process. To aid in the construction of such statements, there are several tools available for our use. We'll discuss two: one in **Microsoft Access** and one available with the **ADO data environment**.

- To build a SQL query using Microsoft Access, you obviously must have Access installed on your computer. As an example, we will build the SQL query that displays Author, Title, and Publisher for each book in the BIBLIO.MDB:

⇒ Start **Access** and open your copy of the **BIBLIO.MDB**. Click the **Queries** tab and select **New**. Select **Design View**, click **OK**.

⇒ Click the **Tables** tab. Add all four tables. When done, click **Close**. A split window appears with the four linked tables at the top (showing the relationships between primary and foreign keys) and a table in the lower portion.

⇒ In the lower portion of the window, click the first **Field** column, click the drop-down arrow and select **Authors.Author**. Under **Sort**, choose **Ascending** (sorting by Author). In the second column, click the drop-down arrow and select **Titles.Title**. In the third column, click the drop-down arrow and select **Publishers.Company Name**. When done, you should see (I moved the tables around a bit):



Field:	Author	Title	Company Name	
Table:	Authors	Titles	Publishers	
Sort:	Ascending			
Show:	☑	☑	☑	□

[illegible] This is [illegible] [illegible] Visual Basic application [illegible] [illegible] helping to minimize your programming headaches.

[illegible] Visual Basic 6 and are using the ADO data environment, [illegible] SQL statement built for you. Again, the process [illegible] steps explained in Chapter 4 to implement the [illegible] steps are similar to those just used with Access [illegible] [illegible] Basic 6.

Add [illegible] environment to the project and set the properties of the Connection object so it is attached to [illegible] of the BIBLIO.MDB database.

[illegible] Connection object and choose Add Command.

[illegible] the Command object and select Properties. A Properties [illegible] appears. Under Source of Data, choose SQL statement, then click the SQL Builder button. A Data View window and Design Window [illegible] appear.

⇒ Click the exclamation point (!) on the Access toolbar to build the recordset. Now, click **View** on the main Access menu and select **SQL View**.

- Like magic, the SQL statement that was used to develop the recordset is displayed:

```
SELECT Authors.Author, Titles.Title, Publishers.[Company Name]
FROM (Publishers INNER JOIN Titles ON Publishers.PubID = Titles.PubID)
INNER JOIN (Authors INNER JOIN [Title Author] ON Authors.Au_ID = [Title
Author].Au_ID) ON Titles.ISBN = [Title Author].ISBN
ORDER BY Authors.Author;
```

Notice a couple of things about this query. First, it uses the **INNER JOIN** clause to combine tables. Hence, this query could be used with DAO (if you need an updatable recordset) or ADO. Second, notice the semicolon (;) at the end of the query. This is not needed and will be ignored by the Jet database engine. You could now cut and paste the above query wherever you need it in your Visual Basic application (setting a design time property or in your BASIC code). You may need to make some adjustments to the query to make sure it does not result in any syntax errors at run-time. Notice this generated query is very much like that developed earlier in these notes. It's similar because the author used Access to generate that query - you, too, should use the Access query building capabilities whenever you can. You are assured of a correct SQL statement, helping to minimize your programming headaches.

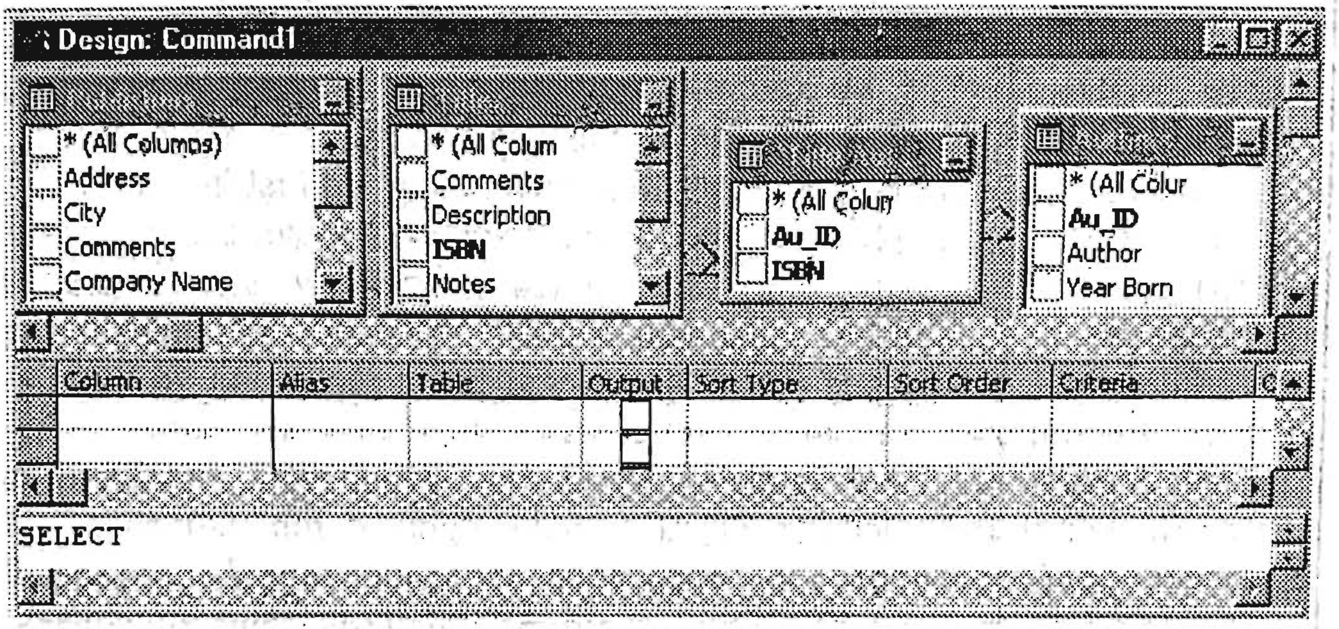
- If you have Visual Basic 6 and are using the ADO data environment, you can also have your SQL queries built for you. Again, this process is best illustrated by example (review the steps explained in Chapter 4 to implement the ADO data environment). The steps are similar to those just used with Access (not unexpected since they probably use the same underlying code). Start a new project in Visual Basic 6.

⇒ Add a Data Environment to the project and set the **Properties** of the **Connection** object so it is attached to your copy of the **BIBLIO.MDB** database.

⇒ Right-click the **Connection** object and choose **Add Command**.

⇒ Right-click the **Command** object and select **Properties**. A **Properties** window will appear. Under **Source of Data**, choose **SQL Statement**, then click the **SQL Builder** button. A **Data View** window and **Design Window** (with several 'panes') appear.

⇒ In the Data View window, expand **Connection**, then expand **Tables**. Drag (in order) these tables from the Data View window to the top pane of the Design Window: **Publishers, Titles, Title Author,** and **Authors**. You need to do them in this order to insure proper connection of keys. If any keys are not correctly connected, you can make manual connections by dragging a key in one table to the corresponding key in another table. At this stage, the Design Window should look like this (I've moved the tables around to show the links):



⇒ Click on the first row under **Column**. Click the drop-down arrow and select **Authors.Author**. Click the first row under **Sort Type**. Choose **Ascending**.

⇒ In the second row, choose **Titles.Title**. In the third row, choose **Publishers.`Company Name`**. In the third pane, you should see the SQL statement that was built for you (close out the Design Window and this statement will be seen in the **Command** object)

Building SQL Commands in Code

- In each example in this chapter, we formed a SQL command and processed it to obtained a returned recordset (our virtual data view). What do you do if you don't know the SQL command prior to implementing it as a Visual Basic property (either at design-time or run-time)? For example, the user of the books database may want to know all the publishers in Chicago. Or, the user may want to search the database for all authors whose name starts with a G.

- In both of the above examples, we have no idea what the user will select. We need to provide the user a method to make a selection then, once the selection is made, build the SQL statement in Visual Basic. Fortunately, the BASIC language (used in all procedures) is rich with string handling functions and building such statements in code is a relatively straightforward process.

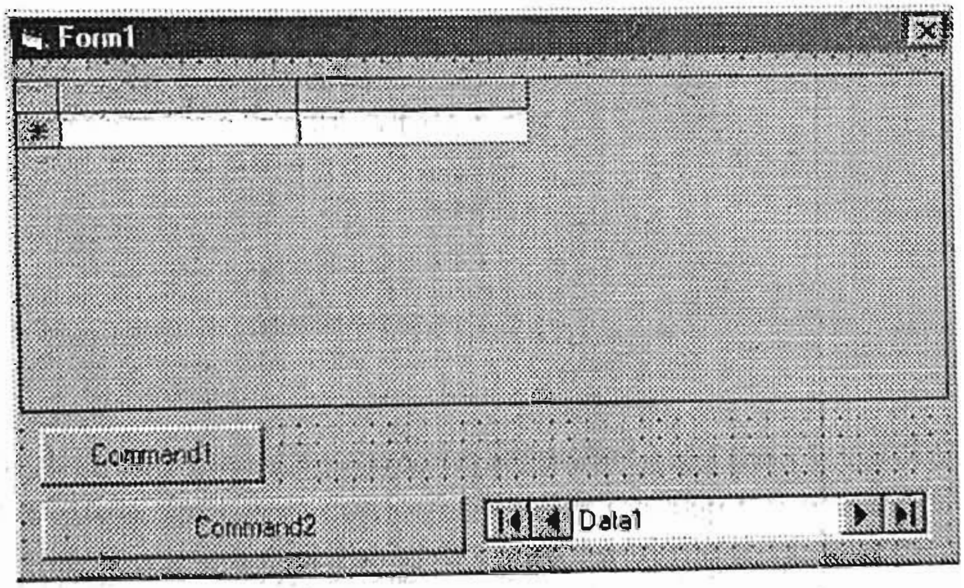
- To build a SQL command in code, form all the known clauses as string variables. Once the user makes the selections forming the unknown information, using string concatenation operators (& or +) to place these selections in their proper position in the complete SQL statement. That statement can then be processed at run-time, using one of the methods discussed earlier in this chapter. The final example in this chapter demonstrates this technique.

Example 5-2

Searching the Books Database

We build an application (using the BIBLIO.MDB books database) that displays a book's author, title, and publisher (none of which are updateable). The user may display all books in the database or, alternately, search the database for books by particular authors (searching by the first letter of the last name, using command buttons for selection). In this application, we use the DAO data control, so it can be built using either Visual Basic 5 or Visual Basic 6. If desired, you could also build it using the ADO data control or ADO data environment. There is a lot to learn from in this example. You'll see how to form a SQL command in code, how get that statement into code, how to set up convenient search mechanisms, and how to build a nice interface, all topics covered in detail in Chapter 6.

1. Start a new project. Add a DAO data control, a DBGrid control and two command buttons. Position and resize the controls until the form looks something like this:



2. Set properties for the form and controls:

Form1:

Name	frmBooks
BorderStyle	1-Fixed Single
Caption	Books Database

Data1:

Name	datBooks
Caption	Books
DatabaseName	BIBLIO.MDB (point to your working copy)

Command1:

Name	cmdLetter
Caption	A
Index	0 (we're building a control array)

Command2:

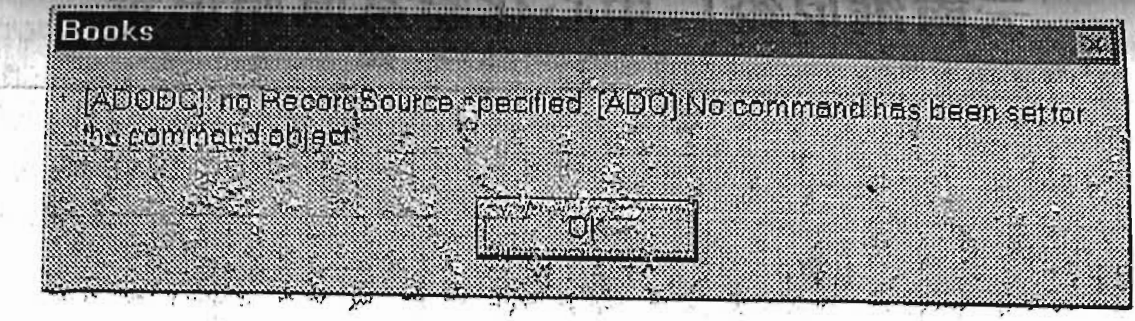
Name	cmdAll
Caption	Show All Records

DBGrid1:

Name	grdBooks
DataSource	datBooks

As in Example 5-1, you may choose to replace the DAO data control with the ADO data control and the DAO data grid control with the corresponding ADO data grid control. If so, use the same properties for the grid control and the data control with one exception Recall the ADO data control does not have a **DatabaseName** property. If using the ADO control, set the **ConnectionString** property such that it points to your working copy of the BIBLIO.MDB database. No code changes are necessary – the code that works for the DAO data control will work for the ADO data control.

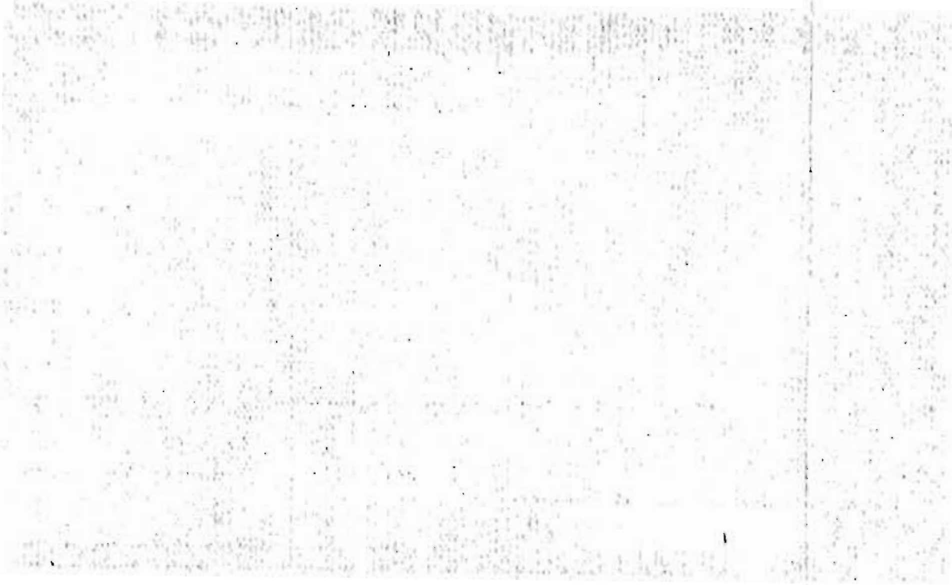
When you attempt setting the **DataSource** property for the grid control, you will get this error message:



Books

[ADODC]: no RecordSource specified. [ADO]: No command has been set for the command object.

[illegible]



[illegible]

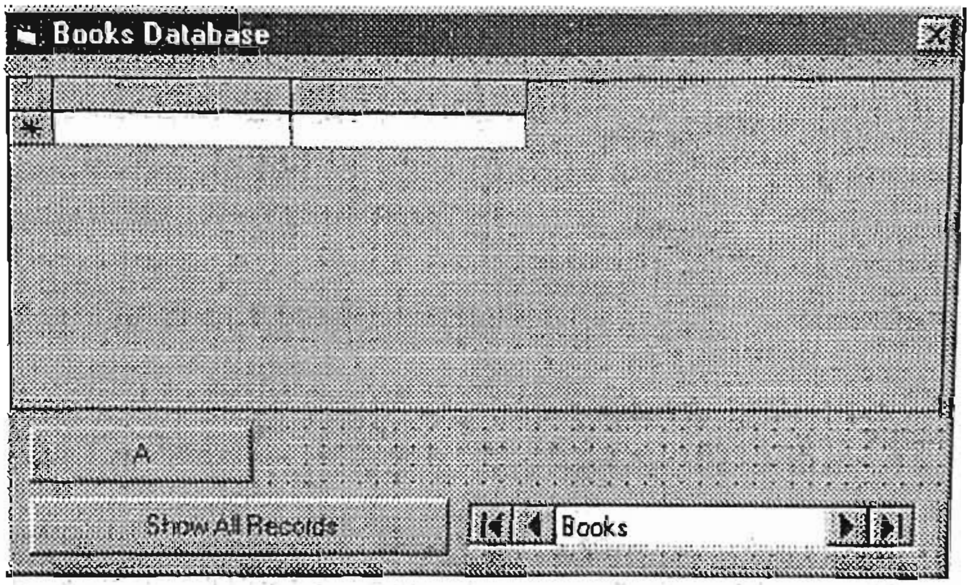
[illegible]

[illegible]

[illegible] default SQL [illegible]

This error is acceptable since we will be setting the data control's **RecordSource** at run-time. You may also get this error when running the application. If so, just click **OK**. For your reference, we have built an ADO version of the **Searching the Books Database** program and included it with the example files (look for the example project file with the **AD** suffix).

At this point, the form should appear similar to this:



3. Place these lines in the **General Declarations** area:

```
Option Explicit
Dim SQLAll As String
```

SQLAll will be the variable that holds the default SQL statement.

```
Private Sub Form_Load()
Dim I As Integer
'Size search buttons
cmdLetter(0).Width = (frmBooks.ScaleWidth - 2 *
cmdLetter(0).Left) / 26
'Create 25 new buttons
'Position new button next to prior button
For I = 1 To 25
  Load cmdLetter(I)
  cmdLetter(I).Left=cmdLetter(I-1).Left+cmdLetter(0).Width
  cmdLetter(I).Caption = Chr(Asc("A") + I)
  cmdLetter(I).Visible = True
Next I
'Build basic SQL statement
SQLAll = "SELECT
Authors.Author,Titles.Title,Publishers.[Company Name] "
SQLAll=SQLAll+"FROMAuthors, Titles, Publishers, [TitleAuthor
"
SQLAll = SQLAll + "WHERE Titles.ISBN = [Title Author].ISBN "
SQLAll = SQLAll + "AND Authors.Au_ID = [Title Author].Au_ID "
SQLAll = SQLAll + "AND Titles.PubID = Publishers.PubID "
End Sub
```

This routine establishes the search buttons A through Z using the cmdLetter control array. It determines button width and places them accordingly. Study the code that does this - it's very useful. This routine also builds the default SQL statement that gets the Author, Title, and Publisher from the database. Note the statement is built in several stages, each stage appending another clause to the statement. Note, particularly, each subsequent clause has a space at the end to make sure there are no 'run-ons' of keywords.

5. Place this code in the **Form_Activate** event procedure:

```
Private Sub Form_Activate()
'Show all records initially
Call cmdAll_Click
End Sub
```

This routine initializes the data grid to the default data view (all records).

6. Place this code in the **cmdAll_Click** event procedure:

```
Private Sub cmdAll_Click()
'Show all records
datBooks.RecordSource = SQLAll + "ORDER BY Authors.Author"
datBooks.Refresh
End Sub
```

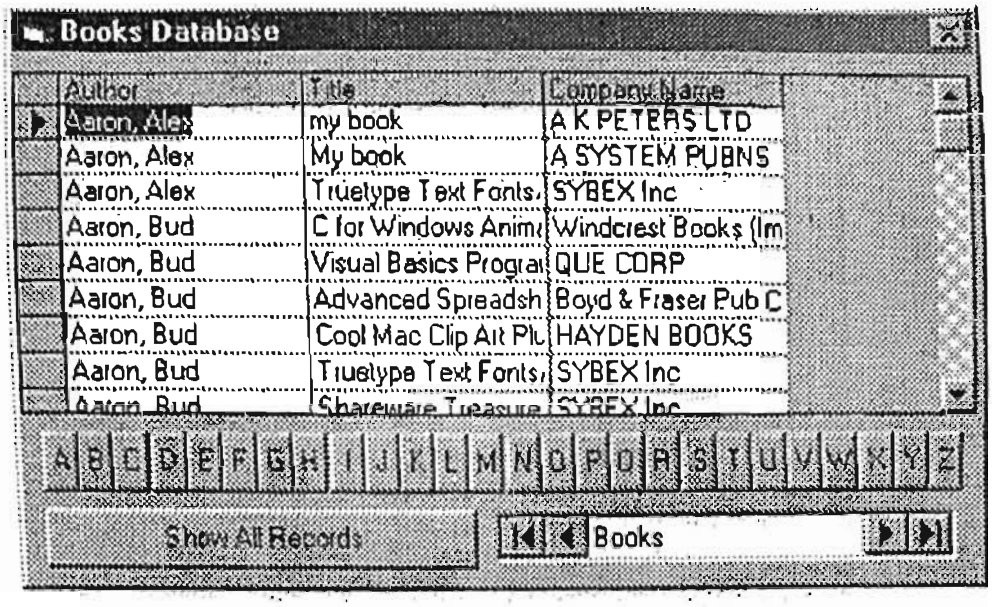
This restores the displayed data to all records using the default SQL statement, appended with the **ORDER BY** clause.

7. Place this code in the **cmdLetter_Click** event procedure:

```
Private Sub cmdLetter_Click(Index As Integer)
If Index <> 25 Then
'Key other than Z clicked
'Append to SQLAll to limit records to letter clicked
  datBooks.RecordSource = SQLAll + "AND Authors.Author > '" +
cmdLetter(Index).Caption + "'"
  datBooks.RecordSource   =   datBooks.RecordSource   +   "AND
Authors.Author < '" + cmdLetter(Index + 1).Caption + "' "
Else
'Z Clicked
'Append to SQLAll to limit records to Z Authors
  datBooks.RecordSource = SQLAll + "AND Authors.Author > 'Z'
"
End If
datBooks.RecordSource  =  datBooks.RecordSource  +  "ORDER  BY
Authors.Author"
datBooks.Refresh
End Sub
```

This routine implements the search on author name. It simply determines what buttor was clicked by the user and appends an additional test (using **AND**) to the WHERE clause in the default SQL statement. This test limits the returned records to author' names between the clicked letter and the next letter in the alphabet. Note that clickin Z is a special case.

8. Save the application. Run it. You should see;



Author	Title	Company Name
Aaron, Alex	my book	A K PETERS LTD
Aaron, Alex	My book	A SYSTEM PUBNS
Aaron, Alex	Truetype Text Fonts,	SYBEX Inc
Aaron, Bud	C for Windows Anim	Windcrest Books (Im
Aaron, Bud	Visual Basics Progra	QUE CORP
Aaron, Bud	Advanced Spreadsh	Boyd & Fraser Pub C
Aaron, Bud	Cool Mac Clip Art Plu	HAYDEN BOOKS
Aaron, Bud	Truetype Text Fonts,	SYBEX Inc
Aaron, Bud	Shareware Treasure	SYBEX Inc

Notice how the search buttons are built and nicely displayed. Notice, too, that all records are displayed. Click one of the search buttons. Only records with author names matching the clicked letter will be displayed.

Summary

- We're now done with our long journey into the world of SQL. This has been a relatively complete overview and you will learn more as you become a more proficient database programmer. SQL is at the heart of obtaining a virtual view of desired database information.

- Forming this virtual view using SQL was seen to be a straightforward, and sometimes complicated, process. Tools such as the Access SQL Builder and the SQL Build function of the ADO data environment can help us build error free SQL queries. Even with such tools, it is important to know SQL so you can understand and modify SQL statements built and implemented by others.

- SQL also has the ability to modify information in a database. You can also use SQL to add records, delete records, and even create new database tables. But, such capabilities are beyond this course. Besides, the same abilities are available to us using Visual Basic. That is the approach we will use for actual database management tasks. Such tasks using are covered in Chapter 8, following a discussion of building a proper Visual Basic interface in Chapter 7.