UNIT-III

SYLLABUS

The Multiple Document Interface-Databases: Architecture and Basic Concepts-Building Database Application with ADO.NET-Programming with ADO.NET

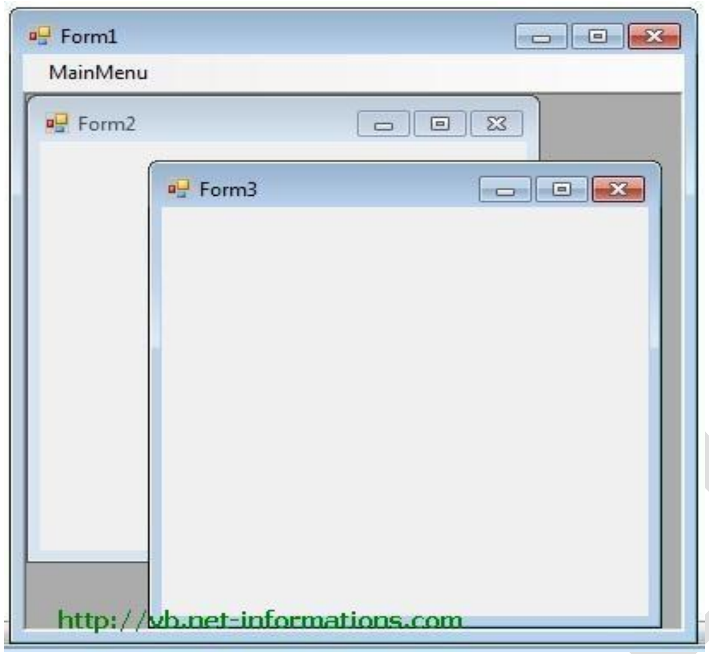
The Multiple Document Interface

A **multiple document interface** (**MDI**) is a graphical user interface in which multiple windows reside under a single parent window. Such systems often allow child windows to embed other windows inside them as well, creating complex nested hierarchies.

MDI applications must have at least two forms, the parent form and one or more child forms. There may be many child forms contained within the parent form, but there can be only one parent form. The parent form is the MDI form, or MDI container, because it contains all child forms.

The parent form may not contain any controls. While the parent form is open in design mode, the icons on the Toolbox aren’t disabled, but you can’t place any controls on the form. The parent form can, and usually does, have its own menu. While one or more child forms are displayed, the menu of the child forms takes over and it’s displayed on the MDI form’s menu bar.

A Multiple Document Interface (MDI) programs can display multiple child windows inside them.



This is in contrast to single document interface (SDI) applications, which can manipulate only one document at a time. Visual Studio Environment is an example of Multiple Document Interface (MDI) and notepad is an example of an SDI application, opening a document closes any previously opened document. Any windows can become an MDI parent, if you set the IsMdiContainer property to True.

```
IsMdiContainer = True
```

Accessing Child Forms

There are two different methods to access the child forms.

The **first method** is to use the Me keyword. This keyword refers to the form in which the code resides, and since the bulk of the code is on the child form, you can use the Me keyword to access the controls on the child form. The MDI child forms of a text editor, for example, contain a TextBox control where the user can enter and edit text. The following expression returns the text on the active child form:

```
Me.TextBox1.Text
```

To select all the text on the active child window, call the TextBox control’s SelectAll method with the following statements:

Me.TextBox1.SelectAll

To access the child form from within the MDI parent form's code, you can use the ActiveMdiChild property, which represents the active child form. The following statement returns the caption of the active child form:

Me.ActiveMdiChild.Text

To access the contents of a TextBox control on the child form from within the MDI form's code, use the following statement:

Me.ActiveMdiChild.TextBox1.Text

Using the second method, access all child windows through the MdiChildren property of the parent form. This property returns an array whose elements represent the child forms open on the MDI form at any one time. To find out the number of open child forms, read the Length property of this array:
Console.WriteLine("There are " & Me.MdiChildren.Length & " child Forms open")

This statement works only if it appears in the MDI parent form's code. To access a child form from within another child form's code, you must first access the parent form (Me.ParentForm) and then the parent form's MdiChildren property. The following statements return the values shown in bold if executed from within a child form's code:

Console.WriteLine(Me.ParentForm.MdiChildren.GetLength(0))
3
Console.WriteLine(Me.ParentForm.ActiveMdiChild)

Tracking the Active Child Form

On an MDI form, all child forms are usually instances of one basic form. All forms all have the same behavior, but the operation of each one doesn't affect the others. When a child form is loaded, for example, it will have the same background color as its prototype, but you can change this from within your code by setting the form's BackColor property. No other child form will be affected by that change.

Each child form is totally independent of any other child form, and you can access it from within your code through the Me keyword. This keyword identifies the current form, as long as you program it from within its own form.

Database

A database is an object for storing complex, structured information. It is maintained by special programs, such as Access and SQL Server. These programs are called database management systems (DBMS).

It isolates much of the complexity of the database from the developer. To access the data stored in the database and to update the database, you use a special language, Structured Query Language (SQL).

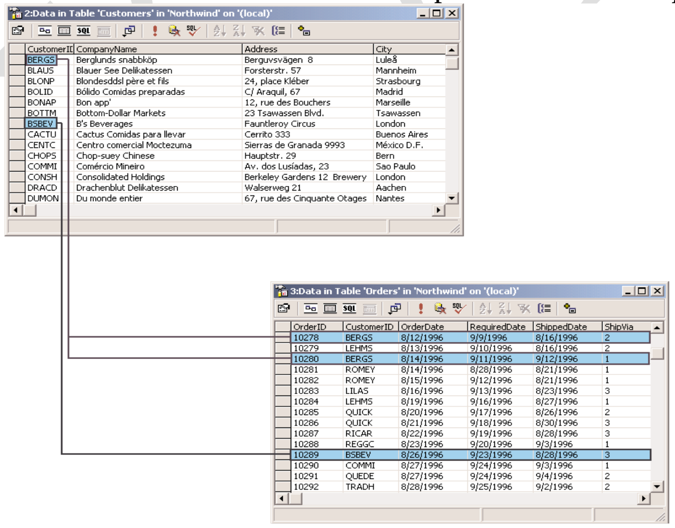
Data are stored in tables, and each table contains entities of the same type.

Relational Database (RDB)

A relational database (RDB) is a collective set of multiple data sets organized by tables, records and columns. RDBs establish a well-defined relationship between database tables. Tables communicate and share information, which facilitates data searchability, organization and reporting.

RDBs use Structured Query Language (SQL), which is a standard user application that provides an easy programming interface for database interaction.

Entities are not independent of each other. For example, orders are placed by specific customers, so the rows of the Customers table must be linked to the rows of the Orders table that store the orders of the customers. Figure shows a segment of a table with customers (top left) and the rows of a table with orders that correspond to one of the customers (bottom right). Thelines that connect the rows of the two tables represent *relationships*.



2:Data in Table 'Customers' in 'Northwind' on '(local)'

CustomerID	CompanyName	Address	City
BERGS	Berglunds snabbköp	Berguvsvägen 8	Luleå
BLAUS	Blauer See Delikatessen	Forsterstr. 57	Mannheim
BLONP	Blondesddsl père et fils	24, place Kléber	Strasbourg
BOLID	Bólido Comidas preparadas	C/ Araquil, 67	Madrid
BONAP	Bon app'	12, rue des Bouchers	Marseille
BOTTM	Bottom-Dollar Markets	23 Tsawassen Blvd.	Tsawassen
BSBEV	B's Beverages	Fauntleroy Circus	London
CACTU	Cactus Comidas para llevar	Cerrito 333	Buenos Aires
CENTC	Centro comercial Moctezuma	Sierras de Granada 9993	México D.F.
CHOPS	Chop-suey Chinese	Hauptstr. 29	Bern
COMMI	Comércio Mineiro	Av. dos Lusíadas, 23	Sao Paulo
CONSH	Consolidated Holdings	Berkeley Gardens 12 Brewery	London
DRACD	Drachenblut Delikatessen	Walserweg 21	Aachen
DUMON	Du monde entier	67, rue des Cinquante Otages	Nantes

3:Data in Table 'Orders' in 'Northwind' on '(local)'

OrderID	CustomerID	OrderDate	RequiredDate	ShippedDate	ShipVia
10278	BERGS	8/12/1996	9/9/1996	8/16/1996	2
10279	LEHMS	8/13/1996	9/10/1996	8/16/1996	2
10280	BERGS	8/14/1996	9/11/1996	9/12/1996	1
10281	ROMEY	8/14/1996	8/28/1996	8/21/1996	1
10282	ROMEY	8/15/1996	9/12/1996	8/21/1996	1
10283	LILAS	8/16/1996	9/13/1996	8/23/1996	3
10284	LEHMS	8/19/1996	9/16/1996	8/27/1996	1
10285	QUICK	8/20/1996	9/17/1996	8/26/1996	2
10286	QUICK	8/21/1996	9/18/1996	8/30/1996	3
10287	RICAR	8/22/1996	9/19/1996	8/28/1996	3
10288	REGGC	8/23/1996	9/20/1996	9/3/1996	1
10289	BSBEV	8/26/1996	9/23/1996	8/28/1996	3
10290	COMMI	8/27/1996	9/24/1996	9/3/1996	1
10291	QUEDE	8/27/1996	9/24/1996	9/4/1996	2
10292	TRADH	8/28/1996	9/25/1996	9/2/1996	2

Relationships are implemented by inserting columns with matching values in the two related tables; the CustomerID column is repeated in both tables. The rows with a common value in their CustomerID field are related. In other words, the lines that connect the two tables simply indicate that there are two fields, one on each side of the relationship, with a common value.

These two fields used in a relationship are called *key fields*. The CustomerID field of the Customers table is the *primary key*, because it identifies a single customer. The CustomerID field of the Orders table is the *foreign key* of the relationship. A CustomerID value appears in a single row of the Customers table; it's the table's primary key. However, it may appear in multiple rows of the Orders table, because in this table the CustomerID field is the foreign key. In fact, it will appear in as many rows of the Orders table as there are orders for the specific customer. The operation of matching rows in two tables based on their primary and foreign keys is called a *join*.

The Visual Database Tools

The Server Explorer. The Server Explorer is the Toolbox for database applications, in the sense that it contains all the basic tools for connecting to databases and manipulating their objects.

The Query Builder This is a tool for creating SQL queries. The Query Builder specify the operations to perform on the tables of a database with point-and-click operations. In the background, the Query Builder builds the appropriate SQL statement and executes it against the database.

The Database Designer and Tables Designer These tools allow to work with an entire database or its tables. In the database, we can add new tables, establish relationships between the tables, and so on. When you work with individual tables, you can manipulate the structure of the tables, edit their data, and add constraints.

The Server Explorer

The Server Explorer is a new development tool in Visual Studio .NET or in Visual Studio 2005 that is shared across development languages and projects. With the Server Explorer, you can connect to servers, as well as view and access their resources.

- Database Connections
- Servers

- Crystal Reports
- Event Logs
- Message Queues
- Performance Counters
- Windows Services

Add a data connection

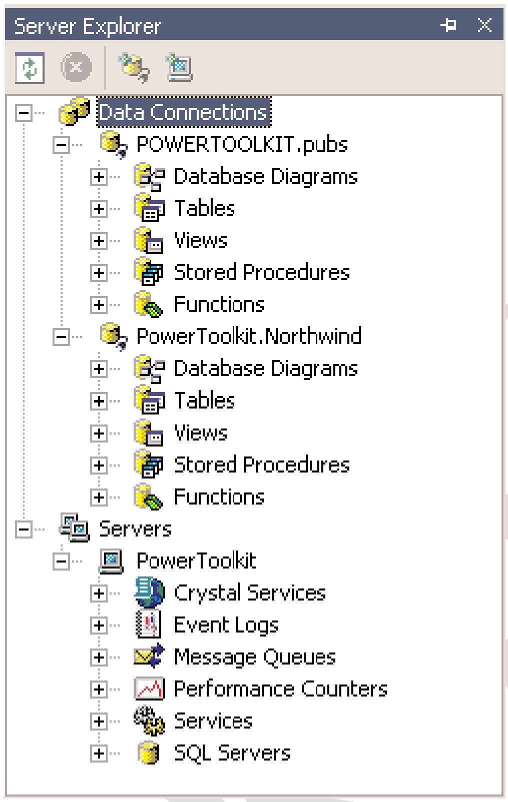
1. In the Server Explorer window, right-click the **Data Connections** node, and then click **Add Connection**.
2. In the **Data Link Properties** dialog box, type or select a server name in the combo box. For example, if SQL Server is installed on the local computer, type **local**. **Note** In Visual Studio 2005, the **Add Connection** dialog box appears.
3. Type the logon information as necessary for your environment.
4. Select a database to use. For example, if the connection is to a SQL Server, you can click **Northwind**.
5. Click **Test Connection** to verify that the data connection is valid. After a few seconds, the following message appears:
   ```
   Test connection succeeded
   ```
   If you encounter an error during this test, check the settings, and make any necessary changes.
6. Click **OK**.

   Notice that the new data connection appears as a child node below the **Data Connections** node.
7. Click to expand the data connection node that you just created.

Add a server

1. In the Server Explorer window, right-click the **Servers** node, and then click **Add Server**.
2. In the **Add Server** dialog box, type a server name that differs from your local server, or type an IP address that differs from your Internet Protocol (IP) address.
3. If you use a different user name to log on to the server, click **Connect using a different user name**, and then type your user name and password.
4. Click **OK**.
   Notice that a new server node appears below the top-level **Servers** node.
5. Click to expand the server node that you just created.

Server Explorer displays the resources that are available to you, such as Crystal Services, Event Logs, Message Queues, Performance Counters, Services and SQL Servers. You receive the same information for your local computer when you click to expand your local server name.



Database Diagrams: A database diagram is a visual representation of a set of related tables, with the relations between the tables. Relations are indicated with line segments between two related tables, and you can quickly learn a lot about the structure of a database by looking at a database diagram.

Tables : We can select a table and edit it, or add a new table to the database. You can edit the table itself (change its design by adding/removing rows or change the data types of one or more columns). Finally, you can view the table's rows and edit them, add new rows, or delete existing rows.

Views This is where you specify the various views you want to use in your applications. Sometimes, the tables are not the most convenient, or even the most expedient, method of looking at your data.
Views are created with SQL SELECT statements. A SELECT statement basically allows you to specify the information you want to retrieve from the database. This information can be stored in a View object, which is just like another table to your application.

Stored Procedures Stored procedures are programs that are stored in the database and perform very specific, and often repeated, tasks. By coding many of the operations you want to perform against the database as stored procedures, you won't have to access the database directly. Moreover, you can call the same stored procedure from several places in your VB code, and you can be sure that the same action is performed every time. Once created, the stored procedure becomes part of the database, and programmers (as well as users) can call it by name, passing the appropriate arguments if necessary. A typical example is a stored procedure for removing orders. The stored procedure must remove the order details first, then remove the order.

Functions The functions of SQL Server are just like the VB functions. They perform specific tasks on the database taking into consideration the arguments passed to the functions when they were called.

Structured Query Language

SQL (Structured Query Language) is a universal language for manipulating tables, and every database management system (DBMS) supports it, so you should invest the time and effort to learn it. You can generate SQL statements with point-and-click operations (the Query Builder is a visual tool for generating SQL statements), but this is no substitute for understanding SQL and writing your own statements.

SQL is a *nonprocedural* language. This means that SQL doesn't provide traditional programming structures like IF statements or loops. Instead, it's a language for specifying the operation you want to perform at an unusually high level. The details of the implementation are left to the DBMS. This is good news for nonprogrammers, but many programmers new to SQL wish it had the structure of a more traditional language. You will get used to SQL and soon be able to combine the best of both worlds: the programming model of VB and the simplicity of SQL.

Example

SELECT StudentName FROM mca

SQL statements are categorized into two major categories, which are actually considered separate languages: the statements for manipulating the data, which form the Data Manipulation Language (DML); and the statements for defining database objects, such as tables or their indexes, which form the Data Definition Language (DDL). The DDL is not of interest to every database developer, and we will not discuss it in this book. The DML is covered in depth, because you'll use these statements to retrieve data, insert new data to the database, and edit or delete existing data.

The statements of the DML part of the SQL language are also known as *queries*, and there are two types of queries: selection queries and action queries. *Selection queries* retrieve information from the database. The queries return a set of rows with identical structure. The columns may come from different tables, but all the rows returned by the query have the same number of columns. *Action queries* modify the database's objects, or create new objects and add them to the database

Selection Queries

We'll start our discussion of SQL with the SELECT statement. Once you learn how to express the criteria for selecting the desired rows with the SELECT statement, you'll be able to apply this information to other data-manipulation statements.
The simplest form of the SELECT statement is

Syntax:

SELECT fields FROM tables

Executing SQL Statements

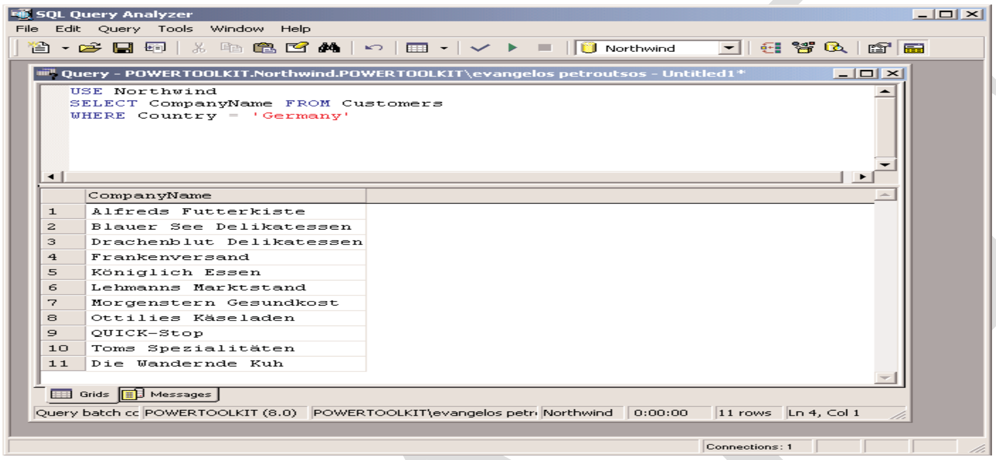
you have two options, the Query Analyzer and the Query Builder. The Query Analyzer executes SQL statements you design. The Query Builder lets you build the statements with visual tools. After a quick overview of the SQL statements, I will describe the Query Builder and show you how to use its interface to build fairly elaborate queries.

Using the Query Analyzer

One of the applications installed with SQL Server is the Query Analyzer.
To start it, select Start -> Programs ➢ SQL Server ➢ Query Analyzer. Initially, its window will be empty.

First, select the desired database's name in the Database drop-down list and then enter the SQL statement you want to execute in the upper pane. The SQL statement will be executed against the selected database when you press Ctrl+E, or click the Run button

USE Northwind
SELECT CompanyName FROM Customers
WHERE Country = 'Germany'


SQL Query Analyzer
File Edit Query Tools Window Help
Northwind
Query - POWERTOOLKIT.Northwind.POWERTOOLKIT\evangelos petroutsos - Untitled1*
Grids
Messages
Connections: 1

```
USE Northwind
SELECT CompanyName FROM Customers
WHERE Country = 'Germany'
```

	CompanyName
1	Alfreds Futterkiste
2	Blauer See Delikatessen
3	Drachenblut Delikatessen
4	Frankenversand
5	Königlich Essen
6	Lehmanns Marktstand
7	Morgenstern Gesundkost
8	Ottilies Käseladen
9	QUICK-Stop
10	Toms Spezialitäten
11	Die Wandernde Kuh

Query batch cc | POWERTOOLKIT (8.0) | POWERTOOLKIT\evangelos petr | Northwind | 0:00:00 | 11 rows | Ln 4, Col 1

Selection Queries
We'll start our discussion of SQL with the SELECT statement. Once you learn how to express the criteria for selecting the desired rows with the SELECT statement, you'll be able to apply this information to other data-manipulation statements.

The simplest form of the SELECT statement is
SELECT fields FROM tablesf

WHERE Clause
The unconditional form of the SELECT statement we used in last few examples is quite trivial. You rarely retrieve data from all rows in a table To restrict the rows returned by the query, use the WHERE clause of the SELECT statement. The most common form of the SELECT statement is the following:

SELECT fields FROM tables WHERE condition.

AS Keyword

By default, each column of a query is labeled after the actual field name in the output. If a table contains two fields named CustLName and CustFName, you can display them with different labels using the AS keyword.

The SELECT statement
SELECT CustLName, CustFName

will produce two columns labeled CustLName and CustFName. The query's output will look much better if you change the labels of these two columns with a statement like the following one:

SELECT CustLName AS [Last Name],
CustFName AS [First Name]

It is also possible to concatenate two fields in the SELECT list with the concatenation operator. Concatenated fields are not labeled automatically, so you must supply your own header for the combined field. The following statement creates a single column for the customer's name and labels it.

Customer Name:
SELECT CustFName + ', ' + CustLName AS [Customer Name]

TOP Keyword

The TOP keyword is used only when the rows are ordered according to some meaningful criteria. Limiting a query's output to the alphabetically top *N* rows isn't very practical. When the rows are sorted according to items sold, revenue generated, and so on, it makes sense to limit the query's output to *N* rows. You'll see many examples of the TOP keyword later in this chapter, after you learn how to order a query's rows

DISTINCT Keyword
The DISTINCT keyword eliminates any duplicates from the cursor retrieved by the SELECT statement. Let's say you want a list of all countries with at least one customer. If you retrieve all country names from the Customers table, you'll end up with many duplicates. To eliminate them, use the DISTINCT keyword, as shown in the following statement:

USE NORTHWIND
SELECT DISTINCT Country FROM Customers

LIKE Operator

The LIKE operator uses pattern-matching characters, like the ones you use to select multiple files in DOS. The LIKE operator recognizes several pattern-matching characters (or *wildcard* characters) to match one or more characters, numeric digits, ranges of letters, and so on;

SQL Wildcard Characters

Wildcard

Character	**Description**
%	Matches any number of characters. The pattern program% will find program, programming, programmer, and so on. The pattern %program% will locate strings that contain the words *program*, *programming*, *nonprogrammer*, and so on.
_ (Underscore character)	Matches any single alphabetic character. The pattern b_y will find *boy* and *bay*, but not *boysenberry*
[ ]	Matches any single character within the brackets. The pattern Santa [YI] nez will find both *Santa Ynez* and *Santa Inez*
[^]	Matches any character not in the brackets. The pattern %q[^u]% will find words that contain the character *q* not followed by *u* (they are misspelled words).
[-]	Matches any one of a range of characters. The characters must be consecutive in the alphabet and specified in ascending order (A to Z, not Z to A). The pattern [a-c]% will find all words that begin with *a*, *b*, or *c* (in lowercase or uppercase).
#	Matches any single numeric character. The pattern D1## will find *D100* and *D139*, but not *D1000* or *D10*

You can use the LIKE operator to retrieve all titles about Windows from the Pubs database, with a statement like the following one.

```
USE PUBS
SELECT titles.title FROM titles WHERE titles.title LIKE ‘%WINDOWS%’
```

Null Values
A very common operation in manipulating and maintaining databases is to locate Null values in fields. The expressions IS NULL and IS NOT NULL find field values that are (or are not) Null. A zero-length string is not the same as a Null field. To locate the rows which have a Null value in their CompanyName column, use the following WHERE clause:

```
WHERE CompanyName IS NULL
```

ORDER Keyword
The rows of a query are not in any particular order. To request that the rows be returned in a specific
Order, use the ORDER BY clause, whose syntax is
ORDER BY col1, col2…

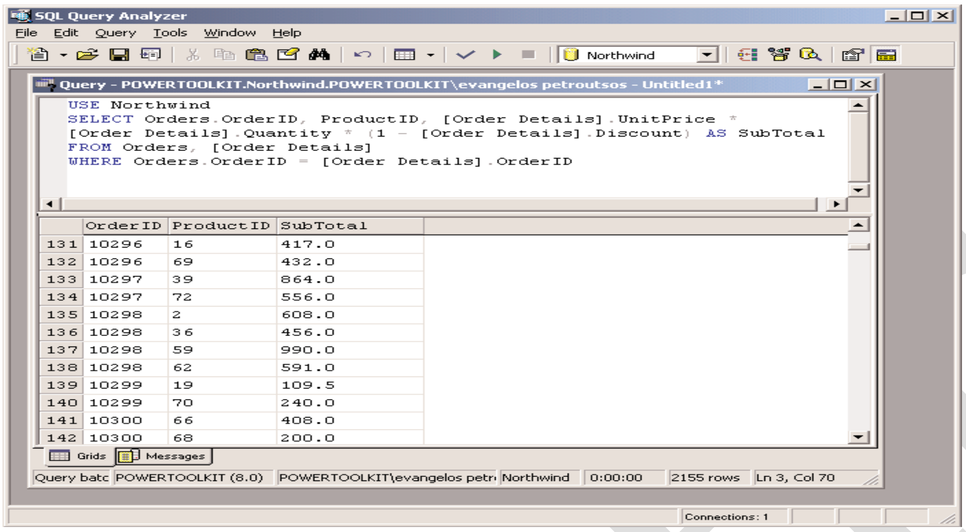
You can specify any number of columns in the ORDER list. The output of the query is ordered according to the values of the first column (col1). If two rows have identical values in this column, then they are sorted according to the second column, and so on. The statement

```
USE NORTHWIND
SELECT Company Name, Contact Name FROM Customers
ORDER BY Country, City
```

Calculated Fields
In addition to column names, you can specify calculated columns in the SELECT statement. as shown in the following statement. The Order Details table contains a row for each invoice line. Invoice #10248, for instance, contains four lines (four items sold), and each detail line appears in a separate row in the Order Details table. Each row holds the number of items sold, the item’s price, and the corresponding discount. To display the line’s subtotal, you must multiply the quantity by the price minus the discount, as shown in the following statement:

```
USE NORTHWIND
SELECT Orders.OrderID, ProductID,
[Order Details].UnitPrice * [Order Details].Quantity *
(1 - [Order Details].Discount) AS SubTotal
```

FROM Orders, [Order Details]
WHERE Orders.OrderID = [Order Details].OrderID


SQL Query Analyzer
File Edit Query Tools Window Help
Northwind
Query - POWERTOOLKIT.Northwind.POWERTOOLKIT\evangelos petroutsos - Untitled1*
Grids
Messages

```
USE Northwind
SELECT Orders.OrderID, ProductID, [Order Details].UnitPrice *
[Order Details].Quantity * (1 - [Order Details].Discount) AS SubTotal
FROM Orders, [Order Details]
WHERE Orders.OrderID = [Order Details].OrderID
```

```
     OrderID  ProductID  SubTotal
131  10296    16         417.0
132  10296    69         432.0
133  10297    39         864.0
134  10297    72         556.0
135  10298    2          608.0
136  10298    36         456.0
137  10298    59         990.0
138  10298    62         591.0
139  10299    19         109.5
140  10299    70         240.0
141  10300    66         408.0
142  10300    68         200.0
```

```
Query batc | POWERTOOLKIT (8.0) | POWERTOOLKIT\evangelos petr | Northwind | 0:00:00 | 2155 rows | Ln 3, Col 70
Connections: 1
```

Totaling and Counting
SQL supports some aggregate functions, which act on selected fields of all the rows returned by the query. The aggregate functions, listed in Table 20.2, perform basic calculations like summing, counting, and averaging numeric values. Aggregate functions accept field names as arguments, and they return a single value, which is the sum of all values.

SQL's Aggregate Functions

Function	**Returns**
COUNT()	The number (count) of values in a specified column
SUM()	The sum of values in a specified column
AVG()	The average of the values in a specified column
MIN()	The smallest value in a specified column
MAX()	The largest value in a specified column

SQL Joins
Joins specify how you connect multiple tables in a query, and there are four types of joins:

- Left outer, or left join
- Right outer, or right join
- Full outer, or full join
- Inner join

A *join* operation combines all the rows of one table with the rows of another table. Joins are usually followed by a condition, which determines which records in either side of the join will appear in the result.

Left Joins
This join displays all the records in the left table and only those records of the table on the right that match certain user-supplied criteria.

This join has the following syntax:

FROM (primary table) LEFT JOIN (secondary table) ON (primary table).(field)
(comparison) (secondary table).(field).

The following statement will retrieve all the titles from the Pubs database along with their publisher. If some titles have no publisher, they will be included to the result:

```
USE PUBS

SELECT title, pub_name
FROM titles LEFT JOIN publishers
ON titles.pub_id = publishers.pub_id
```

Right Joins
This join is similar to the left outer join, except that all rows in the table on the right are displayed and only the matching rows from the left table are displayed.
This join has the following syntax:
FROM (secondary table) RIGHT JOIN (primary table) ON (secondary table).(field)
(comparison) (primary table).(field)

This join has the following example:

```
USE PUBS
SELECT title, pub_name
FROM titles RIGHT JOIN publishers
ON titles.pub_id = publishers.pub_id
```

Full Joins
The full join returns all the rows of the two tables, regardless of whether there are matching rows or not. In effect, it's a combination of left and right joins. To retrieve all the titles and all publishers, and match publishers to their titles, use the following join:

```
USE PUBS
SELECT title, pub_name
FROM titles FULL JOIN publishers
ON titles.pub_id = publishers.pub_id
```

Inner Joins

This join returns the matching rows of both tables, similar to the WHERE clause, and has the following syntax:

FROM (primary table) INNER JOIN (secondary table) ON (primary table).(field) (comparison) (secondary table).(field)

the following example:

```
USE PUBS
SELECT titles.title, publishers.pub_name FROM titles, publishers
WHERE titles.pub_id = publishers.pub_id
```

Grouping Rows

Sometimes you need to group the results of a query, so that you can calculate subtotals. Let's say you need not only the total revenues generated by a single product, but a list of all products and the revenues they generated. The example of the previous section "Totaling and Counting" calculates the total revenue generated by a single product. If you omit the WHERE clause, it will calculate the total revenue generated by all products. It is possible to use the SUM() function to break the calculations at each new product ID as demonstrated in the following statement. To do so, you must group the product IDs together with the GROUP BY clause.

```
USE NORTHWIND
SELECT ProductID,
SUM(Quantity * UnitPrice *(1 - Discount)) AS [Total Revenues]
FROM [Order Details]
GROUP BY ProductID
ORDER BY ProductID
```

Limiting Groups with HAVING

The HAVING clause limits the groups that will appear in the cursor. In a way, it is similar to the WHERE clause, but the HAVING clause allows you to use aggregate functions. The following statement will return the IDs of the products whose sales exceed 1,000 units:

```
USE NORTHWIND
```

```
SELECT ProductID, SUM(Quantity)
FROM [Order Details]
GROUP BY  ProductID
HAVING SUM(Quantity) > 1000
```

the Order Details table with a WHERE clause:

```
USE               NORTHWIND
SELECT        Products.ProductName,
              [Order Details].ProductID,
              SUM(Quantity) AS [Items Sold]
FROM          Products, [Order Details]
WHERE         [Order Details].ProductID = Products.ProductID
GROUP BY      [Order Details].ProductID, Products.ProductName
HAVING        SUM(Quantity) > 1000
ORDER BY      Products.ProductName
```

IN and NOT IN Keywords
The IN and NOT IN keywords are used in a WHERE clause to specify a list of values that a column must match (or not match). They are more of a shorthand notation for multiple OR operators. The following is statement that retrieves the names of the customers in all German-speaking countries:

```
USE NORTHWIND
SELECT CompanyName
FROM Customers
WHERE Country IN (‘Germany’, ‘Austria’, ‘Switzerland’)
```

The BETWEEN Keyword
The BETWEEN keyword lets you specify a range of values and limit the selection to the rows that have a specific column in this range. The BETWEEN keyword is a shorthand notation for an expression like column >= minValue AND column <= maxValue.

The following statement example:

```
USE NORTHWIND
SELECT OrderID, OrderDate, CompanyName
FROM Orders, Customers
WHERE Orders.CustomerID = Customers.CustomerID AND
(OrderDate BETWEEN ‘1/1/1997’ AND ‘1/1/1998’)
```

Action Queries
These queries are called *action queries*, and they're quite simple compared to the selection queries. There are three types of actions you can perform against a database: insertions of new rows, deletions of existing rows, and updates (edits) of existing rows. For each type of action there's a SQL statement, appropriately named INSERT, DELETE, and UPDATE. Their syntax is very simple, and the only complication is how you specify the affected rows (for deletions and updates).

They return the number of rows affected, but you disable this feature by calling the statement:
SET NOCOUNT ON

Deleting Rows
The DELETE statement deletes one or more rows from a table, and its syntax is:
DELETE table_name WHERE criteria

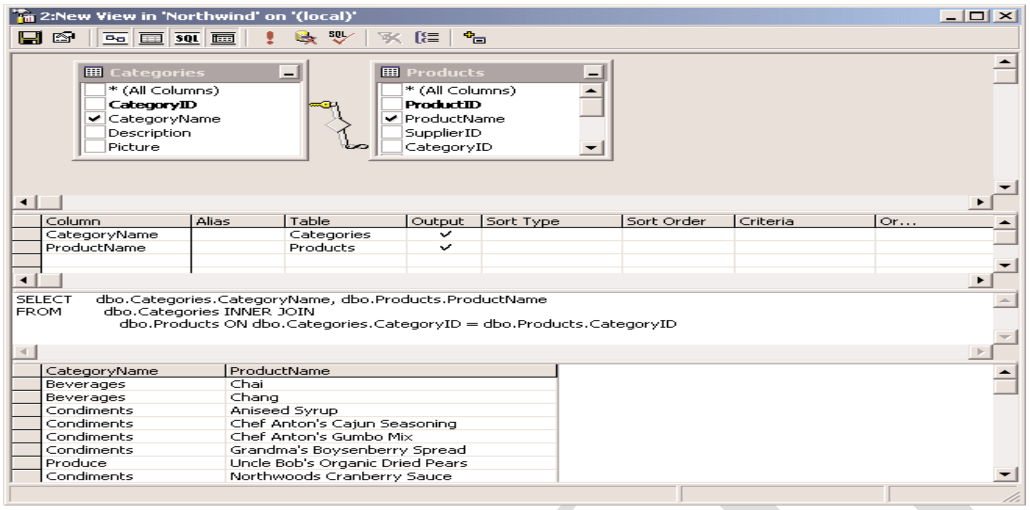
Inserting New Rows
The syntax of the INSERT statement is:
INSERT table_name (column_names) VALUES (values)

Example
INSERT Customers (CustomerID, CompanyName) VALUES ('FRYOG', 'Fruit & Yogurt')

Editing Existing Rows
The UPDATE statement edits a row's fields, and its syntax is
UPDATE table_name SET field1 = value1, field2 = value2, … WHERE criteria
Example
UPDATE Customers SET Country='United Kingdom'
WHERE Country = 'UK'

The Query Builder
The Query Builder is a visual tool for building SQL statements. It's a highly useful tool that generates SQL statements.

Views are based on SQL statements, and you will see the Query Builder with the statement that implements the view you selected.
You can also create new queries by creating a new view. A view is the result of a query: it's a virtual table that consists of columns from one or more tables selected with a SQL SELECT statement.



Column	Alias	Table	Output	Sort Type	Sort Order	Criteria	Or...
CategoryName		Categories	✓				
ProductName		Products	✓				

```
SELECT   dbo.Categories.CategoryName, dbo.Products.ProductName
FROM      dbo.Categories INNER JOIN
            dbo.Products ON dbo.Categories.CategoryID = dbo.Products.CategoryID
```

CategoryName	ProductName
Beverages	Chai
Beverages	Chang
Condiments	Aniseed Syrup
Condiments	Chef Anton's Cajun Seasoning
Condiments	Chef Anton's Gumbo Mix
Condiments	Grandma's Boysenberry Spread
Produce	Uncle Bob's Organic Dried Pears
Condiments	Northwoods Cranberry Sauce

Diagram Pane

To select a table, right-click anywhere on the Diagram pane and you will see the Add Table dialog box. Select as many tables as you need and then close the Add Table dialog box. The little shape in the middle of the line indicates the type of join that must performed on the two tables, and it can take several shapes. To change the type of the relation, you can right-click the shape and select one of the options in the context menu. The diamond-shaped icon indicates an inner join, which requires that only rows with matching primary and foreign keys will be retrieved.By default, the Query Builder treats all joins as inner joins, but you can change the type of the join.

The first step in building a query is the selection of the fields that will be included in the result. Select the fields you want to include in your query by checking the box in front of their names, in the corresponding tables. As you select and deselect fields, their names appear in the Grid pane.

Grid Pane

The Grid pane contains the selected fields. Some fields may not be part of the output—you may use them only for selection purposes—but their names will appear on this pane. To exclude them from the output, clear the box in the Output column.
The Alias column contains a name for the field. By default, the column's name is the alias. This is the heading of each column in the output, and you can change the default name to any string that suits you.

SQL Pane

As you build the statement with point-and-click operations, the Query Builder generates the SQL statement that must be executed against the database to retrieve the specified data. The statement that retrieves product names along with their categories is shown next:

```
SELECT dbo.Products.ProductName, dbo.Categories.CategoryName
FROM dbo.Categories INNER JOIN dbo.Products
ON dbo.Categories.CategoryID = dbo.Products.CategoryID
```

Results Pane

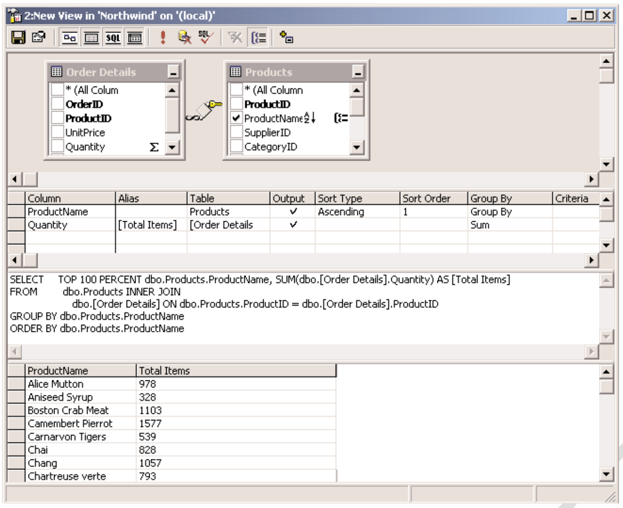
To execute a query, right-click somewhere on the SQL pane and select Run from the context menu. The Query Builder will execute the statement it generated and will display the results in the Results pane at the bottom of the window. The heading of each column is the column's name, unless you've specified an alias for the column.

SQL at Work: Calculating Sums

we'll build a query that retrieves all the products, along with the quantities sold. The names of the products will come from the Products table, while the quantities must be retrieved from the Order Details table. Because the same product appears in multiple rows of the tables (each product appears in multiple invoices with different quantities), we must sum the quantities of all rows that refer to the same product.

Create a new view in the Server Explorer to start the Query Builder, right-click the upper pane, and select Add Table. On the Add Table dialog box, select the tables Products and Order Details, then close the dialog box. The two tables will appear on the Diagram pane with a line connecting them. This is their relation. Now check the fields you want to include in the query: Select the field ProductName in the Products table and the field Quantity in the Order Details table. Expand the options in the Sort Type box in the ProductName row and select Ascending. The Query Builder will generate the following SQL statement:

```
SELECT dbo.Products.ProductName, dbo.[Order Details].Quantity
FROM dbo.Products INNER JOIN dbo.[Order Details]
ON dbo.Products.ProductID = dbo.[Order Details].ProductID
ORDER BY dbo.Products.ProductName
```



Column	Alias	Table	Output	Sort Type	Sort Order	Group By	Criteria
ProductName		Products	✓	Ascending	1	Group By	
Quantity	[Total Items]	[Order Details	✓			Sum	

```
SELECT    TOP 100 PERCENT dbo.Products.ProductName, SUM(dbo.[Order Details].Quantity) AS [Total Items]
FROM      dbo.Products INNER JOIN
            dbo.[Order Details] ON dbo.Products.ProductID = dbo.[Order Details].ProductID
GROUP BY dbo.Products.ProductName
ORDER BY dbo.Products.ProductName
```

ProductName	Total Items
Alice Mutton	978
Aniseed Syrup	328
Boston Crab Meat	1103
Camembert Pierrot	1577
Carnarvon Tigers	539
Chai	828
Chang	1057
Chartreuse verte	793

Stored Procedures

Stored procedures are short programs that are executed on the server and perform very specific tasks. Any action you perform against the database frequently should be coded as a stored procedure, so that you can call it from within any application or from different parts of the same application. A stored procedure that retrieves customers by name is a typical example, and you'll call this stored procedure from many different placed in your application.

Stored procedures isolate programmers from the database and minimize the risk of impairing the database's integrity. When all programmers access the same stored procedure to add a new invoice to the database. They simply call the stored procedure passing the invoice's fields as arguments.

Another advantage of using stored procedures is that they're compiled by SQL Server and they're executed faster. Stored procedures contain traditional programming statements to validate arguments, use default argument values, and so on. The language you use to write stored procedure is called T-SQL, and it's a superset of SQL.

```
CREATE PROCEDURE dbo.StoredProcedure1
/*
(
@parameter1 datatype = default value,
@parameter2 datatype OUTPUT
)
```

```
*/
AS
/* SET NOCOUNT ON */
RETURN
```

ADO .NET

Data access is making the application interact with a database, where all the data is stored. Different applications have different requirements for database access. VB .NET uses ADO .NET (Active X Data Object) as it's data access and manipulation protocol which also enables us to work with data on the Internet.

Evolution of ADO.NET
The first data access model, DAO (data access model) was created for local databases with the built-in Jet engine which had performance and functionality issues. Next came RDO (Remote Data Object) and ADO (Active Data Object) which were designed for Client Server architectures but soon ADO took over RDO. ADO was a good architecture but as the language changes so is the technology. With ADO, all the data is contained in a recordset object which had problems when implemented on the network and penetrating firewalls.
ADO was a connected data access, which means that when a connection to the database is established the connection remains open until the application is closed. Leaving the connection open for the lifetime of the application raises concerns about database security and network traffic. Also, as databases are becoming increasingly important and as they are serving more people, a connected data access model makes us think about its productivity.

Example : an application with connected data access may do well when connected to two clients, the same may do poorly when connected to 10 and might be unusable when connected to 100 or more. Also, open database connections use system resources to a maximum extent making the system performance less effective.

ADO.NET

When an application interacts with the database, the connection is opened to serve the request of the application and is closed as soon as the request is completed. Likewise, if a database is Updated, the connection is opened long enough to complete the Update operation and is closed. By keeping connections open for only a minimum period of time, ADO .NET conserves system resources and provides maximum security for databases and also has less impact on system performance. Also, ADO .NET when interacting with the database uses XML and converts all the data into XML format for database related operations making them more efficient.

The ADO.NET Data Architecture

Data Access in ADO.NET relies on two components: DataSet and Data Provider.

DataSet

The dataset is a disconnected, in-memory representation of data. It can be considered as a local copy of the relevant portions of the database. The DataSet is persisted in memory and the data in it can be manipulated and updated independent of the database. When the use of this DataSet is finished, changes can be made back to the central database for updating. The data in DataSet can be loaded from any valid data source like Microsoft SQL server database, an Oracle database or from a Microsoft Access database.

Data Provider

The Data Provider is responsible for providing and maintaining the connection to the database. A DataProvider is a set of related components that work together to provide data in an efficient and performance driven manner. The .NET Framework currently comes with two DataProviders: the SQL Data Provider which is designed only to work with Microsoft's SQL Server 7.0 or later and the OleDb DataProvider which allows us to connect to other types of databases like Access and Oracle. Each DataProvider consists of the following component classes:

1. The Connection object which provides a connection to the database
2. The Command object which is used to execute a command
3. The DataReader object which provides a forward-only, read only, connected recordset
4. The DataAdapter object which populates a disconnected DataSet with data and performs update

Data access with ADO.NET can be summarized as follows:

1. A connection object establishes the connection for the application with the database.
2. The command object provides direct execution of the command to the database. If the command returns more than a single value, the command object returns a DataReader to provide the data. Alternatively, the DataAdapter can be used to fill the Dataset object. The database can be updated using the command object or the DataAdapter.

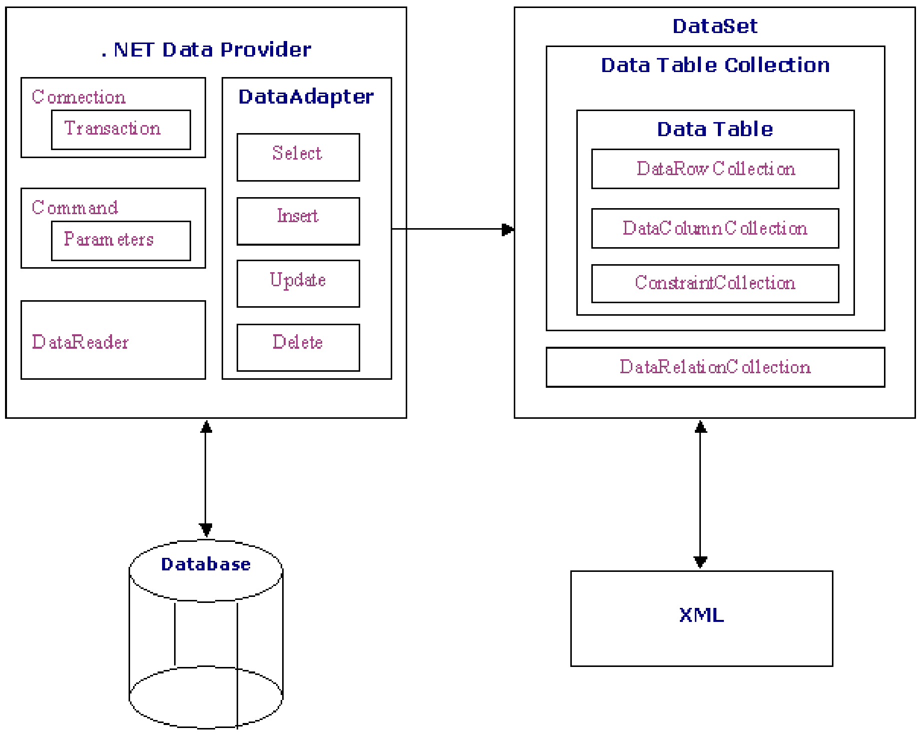
Component classes that make up the Data Providers

The Connection Object

The Connection object creates the connection to the database. Microsoft Visual Studio .NET provides two types of Connection classes: the SqlConnection object, which is designed specifically to connect to Microsoft SQL Server 7.0 or later, and the OleDbConnection object, which can provide connections to a wide range of database types like Microsoft Access and Oracle. The Connection object contains all of the information required to open a connection to the database.

The Command Object

The Command object is represented by two corresponding classes: SqlCommand and OleDbCommand. Command objects are used to execute commands to a database across a data connection. The Command objects can be used to execute stored procedures on the database, SQL commands, or return complete tables directly. Command objects provide three methods that are used to execute commands on the database:



ADO .NET Data Architecture

1. ExecuteNonQuery: Executes commands that have no return values such as INSERT, UPDATE or DELETE
2. ExecuteScalar: Returns a single value from a database query
3.
4. ExecuteReader: Returns a result set by way of a DataReader object

The DataReader Object

The DataReader object provides a forward-only, read-only, connected stream recordset from a database. Unlike other components of the Data Provider, DataReader objects cannot be directly instantiated. Rather, the DataReader is returned as the result of the Command object's ExecuteReader method. The SqlCommand.ExecuteReader method returns a SqlDataReader object, and the OleDbCommand.ExecuteReader method returns an OleDbDataReader object. The DataReader can provide rows of data directly to application logic when you do not need to keep the data cached in memory. Because only one row is in memory at a time, the DataReader provides the lowest overhead in terms of system performance but requires the exclusive use of an open Connection object for the lifetime of the DataReader.

The DataAdapter Object

The DataAdapter is the class at the core of ADO .NET's disconnected data access. It is essentially the middleman facilitating all communication between the database and a DataSet. The DataAdapter is used either to fill a DataTable or DataSet with data from the database with it's Fill method. After the memory-resident data has been manipulated, the DataAdapter can commit the changes to the database by calling the Update method. The DataAdapter provides four properties that represent database commands:

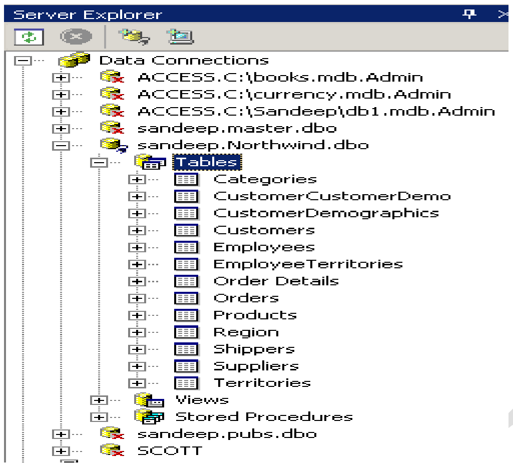
1. SelectCommand
2. InsertCommand
3. DeleteCommand
4. UpdateCommand

When the Update method is called, changes in the DataSet are copied back to the database and the appropriate InsertCommand, DeleteCommand, or UpdateCommand is executed.

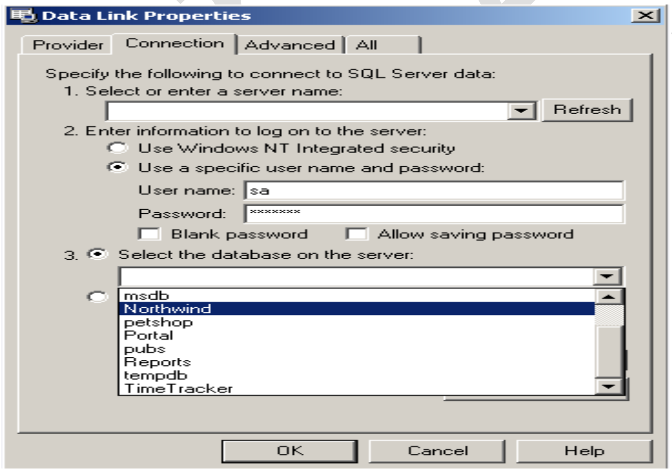
Data Access with Server Explorer

Visual Basic allows us to work with databases in two ways, visually and code. In Visual Basic, Server Explorer allows us to work with connections across different data sources visually. Lets see how we can do that with Server Explorer. Server Explorer can be viewed by selecting View-

>Server Explorer from the main menu or by pressing Ctrl+Alt+S on the keyboard. The window that is displayed is the Server Explorer which lets us create and examine data connections. The Image below displays the Server Explorer.

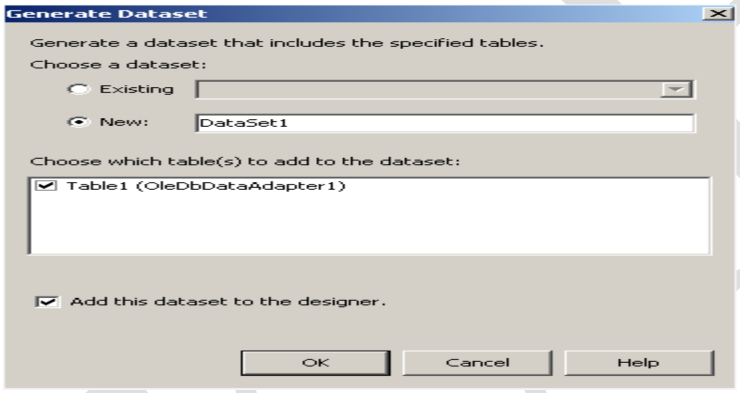


First, we need to establish a connection to this database. To do that, right-click on the Data Connections icon in Server Explorer and select Add Connection item. Doing that opens the Data Link Properties dialog which allows you to enter the name of the server you want to work along with login name and password. The Data Link properties window can be viewed in the Image below.



Specify the following to connect to SQL Server data:
1. Select or enter a server name:
2. Enter information to log on to the server:
( ) Use Windows NT Integrated security
(●) Use a specific user name and password:
User name: sa
Password: *******
[ ] Blank password [ ] Allow saving password
3. (●) Select the database on the server:
msdb
Northwind
petshop
Portal
pubs
Reports
tempdb
TimeTracker

Since we are working with a database already on the server, select the option "select the database on the server". Selecting that lists the available databases on the server, select Northwind database from the list. Once you finish selecting the database, click on the Test Connection tab to test the connection. If the connection is successful, the message "Test Connection Succeeded" is displayed. When connection to the database is set, click OK and close the Data Link Properties. Closing the data link properties adds a new Northwind database connection to the Server Explorer and this connection which we created just now is part of the whole Visual Basic environment which can be accessed even when working with other applications. When you expand the connection node ("+" sign), it displays the Tables, Views and Stored Procedures in that Northwind sample database. Expanding the Tables node will display all the tables available in the database. In this example we will work with Customers table to display its data. Now drag Customers table onto the form from the Server Explorer. Doing that creates SQLConnection1 and SQLDataAdapter1 objects which are the data connection and data adapter objects used to work with data. They are displayed on the component tray. Now we need to generate the dataset that holds data from the data adapter. To do that select Data->Generate DataSet from the main menu or right-click SQLDataAdapter1 object and select generate DataSet menu.


Generate Dataset

Generate a dataset that includes the specified tables.

Choose a dataset:

( ) Existing

(•) New: DataSet1

Choose which table(s) to add to the dataset:

[✓] Table1 (OleDbDataAdapter1)

[✓] Add this dataset to the designer.

Once the dialogbox is displayed, select the radio button with New option to create a new dataset. Make sure Customers table is checked and click OK. Clicking OK adds a dataset, DataSet11 to the component tray and that's the dataset with which we will work. Now, drag a DataGrid from toolbox. We will display Customers table in this data grid. Set the data grid's DataSource property to DataSet11 and it's DataMember property to Customers. Next, we need to fill the dataset with data from the data adapter. The following code does that:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)_
Handles MyBase.Load
DataSet11.Clear()
SqlDataAdapter1.Fill(DataSet11)
```

'filling the dataset with the data adapter's fill method
End Sub

Once the application is executed, Customers table is displayed in the data grid. That's one of the simplest ways of displaying data using the Server Explorer window.

Creating a DataSet

To create a new database application, start a new project as usual. When the project's form appears, open the Server Explorer and expand one of the databases. Select the Northwind database and expand its icon to see the objects of the database. In the Tables section, select the Customers table, drag it with the mouse, and drop it on the form. VB will add two new objects in the Components tray: *SqlConnection1* and *SqlDataAdapter1*.

The first object, *SqlConnection1*, is the application's connection to the database. This object contains all the information needed to connect to the database. If you look at its properties, you will see that its ConnectionString property is:

data source=PowerToolkit;initial catalog=Northwind;integrated security=SSPI;
persist security info=False;workstation id=POWERTOOLKIT;packet size=4096

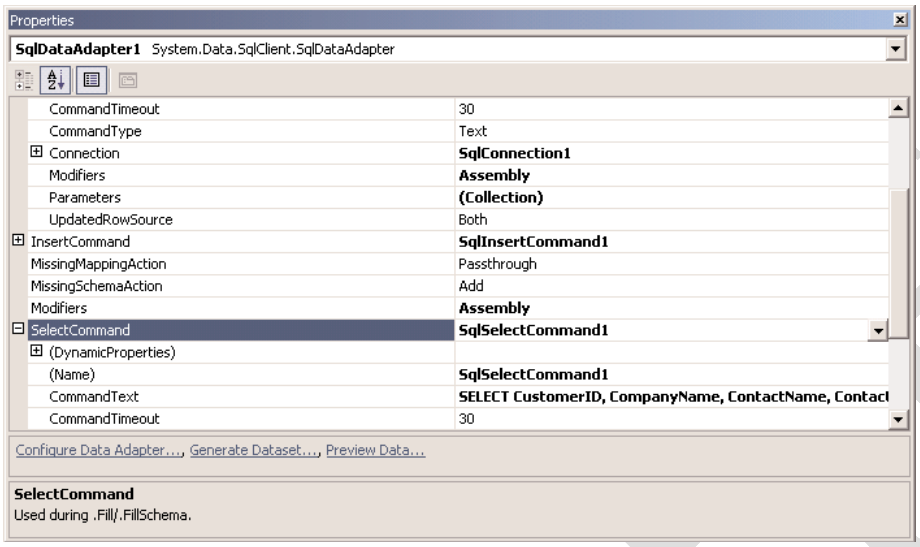
SqlDataAdapter1 is the channel between your application and the database. The DataSet doesn't know anything about the database—it's not its job to know about the database. The application can request the data through the DataAdapter object, process them and then rely on the DataAdapter to update the database.

The SelectCommand object has a property called CommandText, which is a SELECT SQL statement:
SELECT CustomerID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, Country, Phone, Fax FROM dbo.Customers
This statement was generated automatically when you dropped the Customers table on the form. VB picked up the information from the table's structure in the database and create a SELECT statement to retrieve all the columns of all rows.

If you select the SelectCommand item in the Properties window and then click the button with the ellipsis that appears next to the item's setting, the Query Builder window will pop up and you can edit the SELECT statement. You can also edit the SELECT statement by selecting the s*qlDataAdapter1* object on the designer and clicking the Configure Data Adapter command at the bottom of the Properties window.

The *SqlDataAdapter1* also has an InsertCommand property, which is shown next:

INSERT INTO dbo.Customers(CustomerID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, Country, Phone, Fax)



Properties

SqlDataAdapter1 System.Data.SqlClient.SqlDataAdapter

CommandTimeout	30
CommandType	Text
⊞ Connection	SqlConnection1
Modifiers	Assembly
Parameters	(Collection)
UpdatedRowSource	Both
⊞ InsertCommand	SqlInsertCommand1
MissingMappingAction	Passthrough
MissingSchemaAction	Add
Modifiers	Assembly
⊟ SelectCommand	SqlSelectCommand1
⊞ (DynamicProperties)	
(Name)	SqlSelectCommand1
CommandText	SELECT CustomerID, CompanyName, ContactName, Contact
CommandTimeout	30

Configure Data Adapter..., Generate Dataset..., Preview Data...

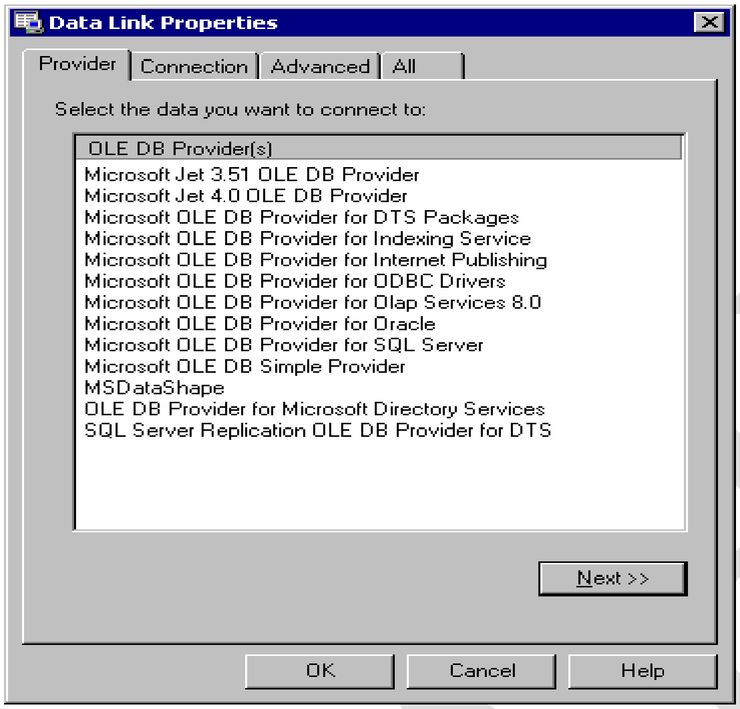
SelectCommand
Used during .Fill/.FillSchema.

VALUES (@CustomerID, @CompanyName, @ContactName, @ContactTitle, @Address, @City, @Region, @PostalCode, @Country, @Phone, @Fax);
SELECT CustomerID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, Country, Phone, Fax FROM dbo.Customers
WHERE (CustomerID = @Select_CustomerID)

Any string that starts with the @ symbol is a variable. The DataAdapter sets the values of all these variables to the values of the new row to be inserted and then executes the InsertCommand against the database. The INSERT statement will add a new row to the Customers table. The SELECT statement following the INSERT statement selects the newly added row from the table and returns it to the application. There are two more commands in the SqlDataAdapter object, UpdateCommand and DeleteCommand.

Microsoft Access and Oracle Database

The process is same when working with Oracle or MS Access but with some minor changes. When working with Oracle you need to select Microsoft OLE DB Provider for Oracle from the Provider tab in the DataLink dialog. You need to enter the appropriate Username and password. The Data Link Properties window can be viewed in the Image below.



Select the data you want to connect to:

OLE DB Provider(s)
Microsoft Jet 3.51 OLE DB Provider
Microsoft Jet 4.0 OLE DB Provider
Microsoft OLE DB Provider for DTS Packages
Microsoft OLE DB Provider for Indexing Service
Microsoft OLE DB Provider for Internet Publishing
Microsoft OLE DB Provider for ODBC Drivers
Microsoft OLE DB Provider for Olap Services 8.0
Microsoft OLE DB Provider for Oracle
Microsoft OLE DB Provider for SQL Server
Microsoft OLE DB Simple Provider
MSDataShape
OLE DB Provider for Microsoft Directory Services
SQL Server Replication OLE DB Provider for DTS

When working with MS Access you need to select Microsoft Jet 4.0 OLE DB provider from the Provider tab in DataLink properties.
Using OleDb Provider

The Objects of the OleDb provider with which we work are:

1. The OleDbConnection Class : The OleDbConnection class represents a connection to OleDb data source. OleDb connections are used to connect to most databases.
2. The OleDbCommand Class: The OleDbCommand class represents a SQL statement or stored procedure that is executed in a database by an OLEDB provider.
3. The OleDbDataAdapter Class : The OleDbDataAdapter class acts as a middleman between the datasets and OleDb data source. We use the Select, Insert, Delete and Update commands for loading and updating the data.
4. The OleDbDataReader Class :The OleDbDataReader class creates a data reader for use with an OleDb data provider. It is used to read a row of data from the database. The data is read as forward-only stream which means that data is read sequentially, one row after another not allowing you to choose a row you want or going backwards.

The Command Objects
Each DataAdapter has four command objects, which provide the information needed to interact

with the database: DeleteCommand, InsertCommand, UpdateCommand, and SelectCommand.

The Command and DataReader Objects

The DataReader is an object that lets you iterate through the rows retrieved by a query. It's faster than storing all the rows to a DataSet, but you can't move back and forth in the rows.

Once the Command object has been set up, you can execute it by calling one of the following methods:

ExecuteReader Executes the command and returns a DataReader object, which you can use to read the results, one row at a time.

ExecuteXMLReader Executes the command and returns a XMLDataReader object, which you can use to read the results, one row at a time.

ExecuteScalar Executes the command, returns the first column of the first row in the result, and ignores all other rows.

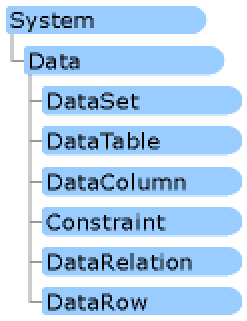
ExecuteNonQuery Executes a SQL command against the database and returns the number of rows affected. Use this method to execute a command that updates the database.

```
Private Sub Button1_Click(ByVal sender As System.Object,_
ByVal e As System.EventArgs) Handles Button1.Click
SqlConnection1.Open()
Dim SQLReader As System.Data.SqlClient.SqlDataReader
SQLReader = SqlCommand1.ExecuteReader()
While SQLReader.Read
ListBox1.Items.Add(SQLReader.Item("CategoryID") & vbTab & _
SQLReader.Item("CategoryName"))
End While
SqlConnection1.Close()
End Sub
```

Dataset

Datasets store data in a disconnected cache. The structure of a dataset is similar to that of a relational database; it exposes a hierarchical object model of tables, rows, and columns. In addition, it contains constraints and relationships defined for the dataset

Dataset Namespace



The fundamental parts of a dataset are exposed to you through standard programming constructs such as properties and collections. For example:

The DataSet class includes the Tables collection of data tables and the Relations collection of DataRelation objects.

The DataTable class includes the Rows collection of table rows, the Columns collection of data columns, and the ChildRelations and ParentRelations collections of data relations.

The DataRow class includes the RowState property, whose values indicate whether and how the row has been changed since the data table was first loaded from the database. Possible values for the RowState property include Deleted, Modified, New, and Unchanged

Transactions

A *transaction* is a series of actions that must either succeed, or fail, as a whole.

The following pseudo-code is the skeleton of a transaction:

```
Begin Transaction
Try
{ statements to complete transaction }
Commit Transaction
Catch Exception
Rollback Transaction
End Try
```

A DataRow's States
In addition to versions, rows have states, too; a row can be in one of the following states:

Added The row has been added to the DataTable, but it hasn't been accepted yet (rows are accepted after they're written to the database as well).

Deleted The row has been deleted. However, it remains in the DataSet marked as Deleted, so that the Update method can delete the matching row of the underlying table.

Detached The row has been created but it has not been added to a DataTable yet. A row is in this state while you set its fields and before you actually add it to a table.

Modified The row has been modified, but it hasn't been accepted yet.

Unchanged The row hasn't been changed yet.

```
Public Function UpdateDataSource(ByVal dataSet As EditProducts.DSProducts) _
As System.Int32
Me.OleDbConnection1.Open()
Dim UpdatedRows As System.Data.DataSet
Dim InsertedRows As System.Data.DataSet
Dim DeletedRows As System.Data.DataSet
Dim AffectedRows As Integer = 0
UpdatedRows = DataSet.GetChanges(System.Data.DataRowState.Modified)
InsertedRows = DataSet.GetChanges(System.Data.DataRowState.Added)
DeletedRows = DataSet.GetChanges(System.Data.DataRowState.Deleted)
Try
If (Not (UpdatedRows) Is Nothing) Then
AffectedRows = OleDbDataAdapter1.Update(UpdatedRows)
AffectedRows = (AffectedRows + OleDbDataAdapter2.Update(UpdatedRows))
AffectedRows = (AffectedRows + OleDbDataAdapter3.Update(UpdatedRows))
End If
If (Not (InsertedRows) Is Nothing) Then
AffectedRows = (AffectedRows + OleDbDataAdapter1.Update(InsertedRows))
AffectedRows = (AffectedRows + OleDbDataAdapter2.Update(InsertedRows))
AffectedRows = (AffectedRows + OleDbDataAdapter3.Update(InsertedRows))
End If
If (Not (DeletedRows) Is Nothing) Then
AffectedRows = (AffectedRows + OleDbDataAdapter1.Update(DeletedRows))
AffectedRows = (AffectedRows + OleDbDataAdapter2.Update(DeletedRows))
AffectedRows = (AffectedRows + OleDbDataAdapter3.Update(DeletedRows))
End If
Catch updateException As System.Exception
Throw updateException
```

```
Finally
Me.OleDbConnection1.Close()
End Try
End Function
```

Coding

Public Class Form1

DECLERATION

```
   Dim con As ADODB.Connection
   Dim cmd As ADODB.Command
   Dim str, cnstr, sql As String
   Dim cn As OleDb.OleDbConnection
```

FORM LOAD

```
   Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
   System.EventArgs) Handles MyBase.Load
      con = New ADODB.Connection
      con.Open("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\new
      folder\employee.mdb")
   End Sub
```

ADDING A RECORD

```
   Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
   System.EventArgs) Handles Button1.Click
      str = "insert into table1 values('" + TextBox1.Text + "','" +
      TextBox2.Text + "','" + TextBox3.Text + "','" + TextBox4.Text +
      "','" + TextBox5.Text + "','" + TextBox6.Text + "','" +
      TextBox7.Text + "')"
      cmd = New ADODB.Command
      cmd.ActiveConnection = con
      cmd.CommandText = str
      cmd.Execute(MsgBox("Add"))
      cmd.Cancel()

   End Sub
```

DELETING A RECORD

```
   Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
   System.EventArgs) Handles Button2.Click
```

```
        str = "delete * from table1 where empno=" + ComboBox1.Text + ""
        cmd = New ADODB.Command
        cmd.ActiveConnection = con
        cmd.CommandText = str
        cmd.Execute()
        MsgBox("Delete")
        cmd.Cancel()
    End Sub
```

ADDING ITEMS IN COMBO BOX

```
Private Sub ComboBox1_GotFocus(ByVal sender As Object, ByVal e As
    System.EventArgs) Handles ComboBox1.GotFocus
        ComboBox1.Items.Clear()
        cnstr = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\new
        folder\employee.mdb"
        cn = New OleDb.OleDbConnection(cnstr)
        cn.Open()
        sql = "select empno from table1"
        Dim ocmd As New OleDb.OleDbCommand(sql, cn)
        Dim odatareader As OleDb.OleDbDataReader = ocmd.ExecuteReader
        While odatareader.Read
            ComboBox1.Items.Add(odatareader.GetValue(0).ToString())
        End While
        odatareader.Close()
        cn.Close()
    End Sub
```

SELECTING ITEMS IN COMBO BOX & DISPLAYING IN TEXT BOX

```
Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ComboBox1.SelectedIndexChanged
cnstr = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\new folder\employee.mdb"
        cn = New OleDb.OleDbConnection(cnstr)
        cn.Open()
        sql = "select * from table1 where empno=" & CInt(ComboBox1.SelectedItem) & ""
        Dim ocmd As New OleDb.OleDbCommand(sql, cn)
        Dim odatareader As OleDb.OleDbDataReader = ocmd.ExecuteReader
        While odatareader.Read
            TextBox1.Text = odatareader.GetValue(0).ToString
            TextBox2.Text = odatareader.GetValue(1).ToString
            TextBox3.Text = odatareader.GetValue(2).ToString
            TextBox4.Text = odatareader.GetValue(3).ToString
            TextBox5.Text = odatareader.GetValue(4).ToString
            TextBox6.Text = odatareader.GetValue(5).ToString
            TextBox7.Text = odatareader.GetValue(6).ToString
        End While
        odatareader.Close()
        cn.Close()
    End Sub
```

UPDATING A RECORD

```
  Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Button3.Click
        str = "delete * from table1 where empno=" + ComboBox1.Text + ""
        cmd = New ADODB.Command
        cmd.ActiveConnection = con
        cmd.CommandText = str
        cmd.Execute()
        cmd.Cancel()
        str = "insert into table1 values('" + TextBox1.Text + "','" + TextBox2.Text + "','" +
TextBox3.Text + "','" + TextBox4.Text + "','" + TextBox5.Text + "','" + TextBox6.Text + "','" +
TextBox7.Text + "')"
        cmd = New ADODB.Command
        cmd.ActiveConnection = con
        cmd.CommandText = str
        cmd.Execute()
        MsgBox("Update")
        cmd.Cancel()
    End Sub
End Class
```