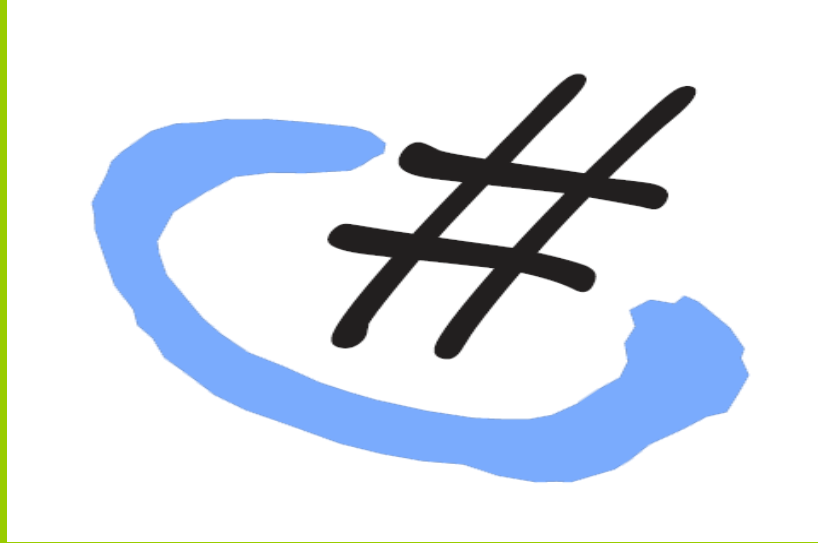


Think About
ADO.NET

Demo Version



By: Ahmed Rabie El Bohoty

About The Author

Name	Ahmed Rabie El Bohoty
Country	Egypt
City	Mansoura
Specialist	Computer Science
Job	Software Engineering
Certifications	Certified from Sun and Microsoft
Contact	ahmed_bohoty@hotmail.com

Index

Chapter 1: Database Overview

Chapter 2: Data Retrievals

Chapter 3: Play with ADO.NET

Chapter 4: What is the Islam?

قال الله تعالى : ﴿ وَإِذْ قَالَ اللَّهُ يَا عِيسَى ابْنَ مَرْيَمَ ءَأَنْتَ قُلْتَ لِلنَّاسِ اتَّخِذُونِي وَأُمِّيَ إِلَهَيْنِ مِنْ دُونِ اللَّهِ قَالَ سُبْحَانَكَ مَا يَكُونُ لِي أَنْ أَقُولَ مَا لَيْسَ لِي بِحَقٍّ إِنْ كُنْتُ قُلْتُهُ فَقَدْ عَلِمْتَهُ تَعْلَمُ مَا فِي نَفْسِي وَلَا أَعْلَمُ مَا فِي نَفْسِكَ إِنَّكَ أَنْتَ عَلَّامُ الْغُيُوبِ ۖ مَا قُلْتُ لَهُمْ إِلَّا مَا أَمَرْتَنِي بِهِ أَنْ أَعْبُدُوا اللَّهَ رَبِّي وَرَبَّكُمْ وَكُنْتُ عَلَيْهِمْ شَهِيدًا مَا دُمْتُ فِيهِمْ فَلَمَّا تَوَفَّيْتَنِي كُنْتُ أَنْتَ الرَّقِيبَ عَلَيْهِمْ وَأَنْتَ عَلَى كُلِّ شَيْءٍ شَهِيدٌ ۖ إِنْ تَعَذَّبْهُمْ فَإِنَّهُمْ عِبَادُكَ وَإِنْ تَغْفِرْ لَهُمْ فَإِنَّكَ أَنْتَ الْعَزِيزُ الْحَكِيمُ ۝ المائدة: 116 - 118.

ISLAM and the AIM of LIFE

What is your purpose in life? What is the rationale behind our life? Why do we live in this life? These questions frequently intrigue people who try to find accurate answers.

People provide different answers to these questions. Some people believe the purpose of life is to accumulate wealth. But one may wonder: What is the purpose of life after one has collected colossal amounts of money? What then? What will the purpose be once money is gathered? If the purpose of life is to gain money, there will be no purpose after becoming wealthy. And in fact, here lies the problem of some disbelievers or misbelievers at some stage of their life, when collecting money is the target of their life. When they have collected the money they dreamt of, their life loses its purpose. They suffer from the panic of nothingness and they live in tension and restlessness.

Note: To Read More go to the last chapter in book

Some Islamic Site

www.islam-guide.com/
www.islamonline.net
www.islamway.com
www.islamhouse.com
www.islamdoor.com
www.islamqa.com

Chapter 1

Database Overview

Overview

- ✓ A Database is an organized collection of information that is divided into tables. Each table is further divided into rows and columns; these columns store the actual information.
- ✓ You access a database using Structured Query Language (SQL), which is a standard language supported by most database software including SQL Server, Access, and Oracle.

Query operations are divided into three different categories:-

- Data Definition Language (DDL)
 - ⇒ Commands are used to create and manage the objects in a database.
 - ⇒ Example: create, modify, and drop databases, tables, indexes, views, stored procedures
- Data Control Language (DCL)
 - ⇒ Statements control the security permissions for users and database objects.
 - ⇒ Specific user or users who belong to a database role or Windows user group.
- Data Manipulation Language (DML)
 - ⇒ Statements used to work with data.
 - ⇒ Retrieve data (select), insert rows into a table, modify values, and delete rows.

What's New in SQL Server 2005?

Will be discussed in the next module and incorporation of the .NET Framework with SQL Server.

T-SQL and the .NET Framework

Allows developers to use programming languages to write stored procedures and functions that access and manipulate data with object-oriented code, rather than SQL statements

The uses of Transact-SQL (Examples)

- Graphical user interface (GUI).
- Web pages that extract data from SQL Server databases.

- Data warehouses in which data is extracted from online transaction processing (OLTP) systems and summarized for decision-support analysis.

Data Manipulation Language (DML)

By using DML statements, you can change data or retrieve information.

DML statements include:

- SELECT.
- INSERT.
- UPDATE.
- DELETE.

Example:

```
use master
SELECT optname,install_failures
FROM MSreplication_options GO
```

Data Definition Language (DDL)

- Creating Objects
 - Syntax: Create <Object type> <Object Name>
- Altering Objects
 - Syntax: Alter <Object Type> <Object Name>
- Dropping Objects
 - Syntax: Drop <Object Type> <Object Name>

Create:

- Used to create a new object
- Example: Table, View, Procedure, Trigger, and Function

```
CREATE TABLE Appointment ( AppointmentID Int , Description
VarChar(50) , StartDateTime DateTime , EndDateTime DateTime ,
Resource VarChar(50) Null )
```

Alter:

- Used to modify the structure of an existing object.

```
ALTER TABLE Appointment ADD LeadTime SmallInt Null
```

DROP:

- Used to delete an existing object.
- You may not be able to drop a table if it contains data participating in a relationship or if another object depends on the object you intend to drop.

```
DROP TABLE Appointment
```

Data Control Language (DCL)

- GRANT → command gives permission set to a user or role.
- DENY → command explicitly restricts a permission set.
- REVOKE → command is used to remove a permission set on an object.

Examples

This statement grants SELECT permission to the user `guest` on the Product table:

```
GRANT SELECT ON Appointment TO guest
```

```
GRANT SELECT, INSERT, UPDATE ON Product TO Paul
```

Transact-SQL Syntax Elements

- Batch Directives
- Comments
- Identifiers
- Types of Data
- Variables
- Functions
- Operator
- Expressions
- Reserved Keywords

Batch Directives

► Go

⇒ send the current batch of Transact-SQL statements to SQL Server

► Exec

⇒ used to execute a user-defined function, system procedure, user-defined stored procedure

Comments

⇒ In-line Comments → Used to disable lines of a statement.

Example

This example uses an in-line comment to explain what a calculation is doing.

```
USE northwind
SELECT productname
, (unitsinstock - unitsonorder) -- Calculates inventory
, supplierid
FROM products
GO
```

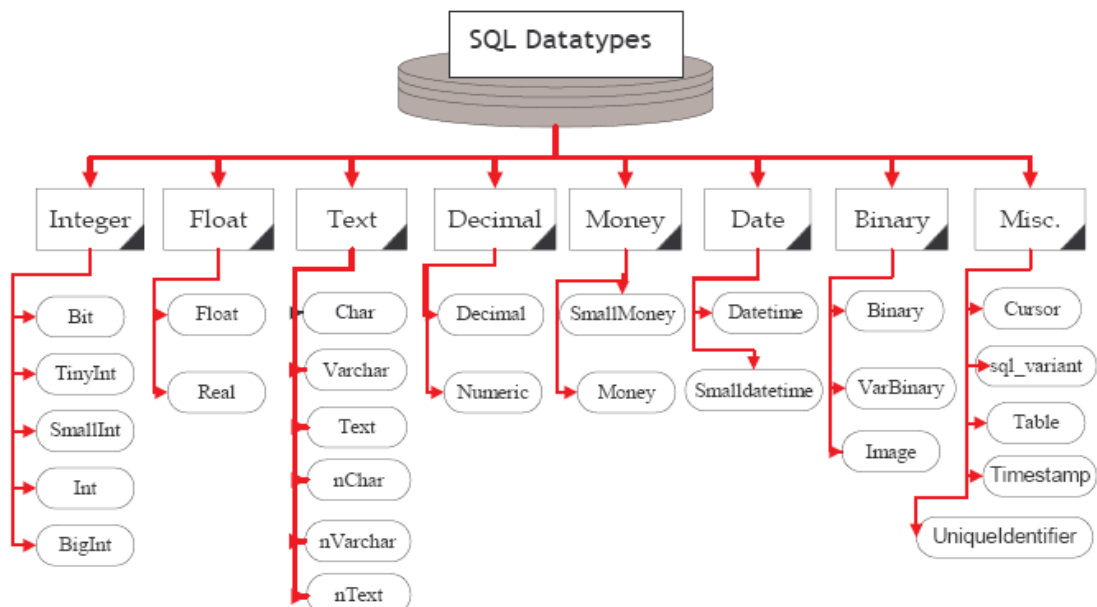
⇒ Block Comments → Create multiple line blocks of comments.

Example

This example shows a comment header that spans several lines.

```
/*
and displays the unit price, the unit This code retrieves all rows of the products table
price increased by 10 percent, and the name of the product.
*/
```

- Identifiers
 - ⇒ Standard Identifiers
 - ⇒ Delimited Identifiers
- Types of Data → define the data values allowed for each column in a database table and related Metadata.



- Variables: language elements with assigned values. You can use local variables in Transact SQL.
- So what is the next we must ask here a critical question, what is the difference between the Varchar and nVarchar?
 - ⇒ nVarchar → for Unicode.
 - ⇒ Varchar → for English where ASCII.

Syntax

```
DECLARE {@local_variable data_type} [,...n] SET @local_variable_name
= expression
```

Example:

```
declare @str nchar(11)
set @str='ali'
SELECT @str = string FROM Table_1 where id=4
SELECT @str AS string
Go
```

- Functions → Transact-SQL provides many functions that return information.

⇒ Aggregate Functions: Operate on the collection of values and return a single summarizing value.

Example:

```
SELECT AVG(id) AS Avgid FROM Table_1 GO
```

⇒ Scalar Functions → return a single value operating from none or many single scalar values.

Example:

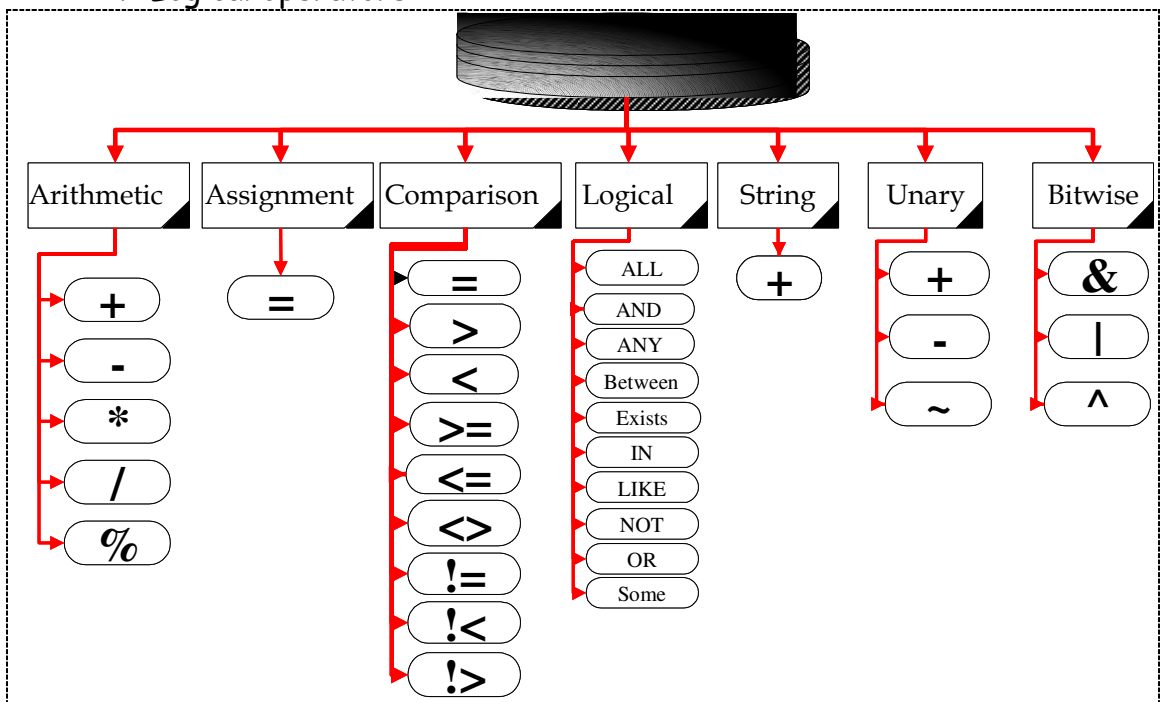
```
SELECT DB_NAME() AS 'database'
```

⇒ Rowset Functions → Can be used like table references in a Transact-SQL statement.

```
SELECT * FROM OPENQUERY(OracleSvr, 'SELECT name, id FROM owner.titles')
```

- Operator

- ⇒ Arithmetic operators
- ⇒ Comparison operators
- ⇒ Logical operators



- Expressions

⇒ Expressions are a combination of symbols and operators that evaluate to a single data value. They can be simple such as a constant, variable, column or scalar value.

Example:

```
SELECT string, (coulm1* colum2) as Total FROM Table_1 where (coulm1*  
colum2)> 10000
```

- Reserved Keywords

- ⇒ You cannot include reserved keywords in a Transact-SQL.
- ⇒ If an object name matches a keyword, whenever you refer to the object you must enclose it within delimiting identifiers, such as quotation marks or brackets [].

Chapter 2

Data Retrieval

Outlines

- The SELECT Statement
- SQL Server 2005 Schemas
- Filtering Rows
- Sorting Displaying the TOP Results
- n Rows in a Result Set
- Performance Considerations

General Idea

SQL queries are used to reach into the database and pull out useful information.

The SELECT Statement

Clause	Explanation
SELECT	indicating that you want to return columns
FROM	Followed by a table or view name, or multiple tables with join expressions
WHERE	Followed by filtering criteria
ORDER BY	Followed by a list of columns for sorting

Examples

```
SELECT Name, StandardCost, Color FROM Product
```

```
SELECT * FROM Product
```

```
SELECT CustomerID, SalesPersonID FROM Customer
```

Take Care!!!

```
SELECT CustomerID, SalesPersonID, PurchaseOrderNumber FROM Customer
INNER JOIN SalesOrderHeader
ON Customer.CustomerID = SalesOrderHeader.CustomerID
```

Now, when you execute this query, what happens? You get an error that looks like this:

- 10 -

أشهد ان لا اله الا الله وان محمد رسول الله
ربي اغفر لي ولوالدي وللمسلمين والمسلمات .. اللهم اجعل القرآن ربيع قلوبنا ونور صدورنا

Server: Msg 209, Level 16, State 1, Line 1 Ambiguous column name 'CustomerID'. Server: Msg 209, Level 16, State 1, Line 1 Ambiguous column name 'SalesPersonID'.

So it must be →

```
SELECT Customer.CustomerID , SalesOrderHeader.SalesPersonID ,  
SalesOrderHeader.PurchaseOrderNumber FROM Customer INNER JOIN  
SalesOrderHeader ON Customer.CustomerID = SalesOrderHeader.CustomerID
```

SQL Server 2005 Schemas

The Schema object represents an ownership context for a Microsoft SQL Server database object.

Query without Schema referred

```
SELECT * FROM Product
```

Query with Schema referred

because the Product table is in the Production schema, you must use this syntax:

```
SELECT * FROM Production.Product
```

Column Aliasing

Examples of these three techniques:

```
SELECT FirstName + ' ' + LastName AS FullName FROM Contact
```

```
SELECT FirstName + ' ' + LastName FullName FROM Contact
```

```
SELECT FullName = FirstName + ' ' + LastName FROM Contact
```

► Produces a single column

```
SELECT FirstName + ' ' + LastName AS EmployeeName FROM Employee
```

	EmployeeName
1	Terri Duffy
2	Jian Shuo Wang
3	Michael Sullivan
4	Sharon Salavaria
5	Gail Erickson
6	Jossef Goldberg
7	Ovidiu Crăciun
8	Janice Galvin
9	Thierry D'Hers

```
SELECT FirstName, LastName, BirthDate, DateDiff (Day, BirthDate,
GetDate ())/365 As Age FROM Employee
```

	FirstName	Lastname	BirthDate	Age
1	Terri	Duffy	9/1/1961 12:00:00 AM	42
2	Jian Shuo	Wang	12/13/1964 12:00:00 AM	39
3	Michael	Sullivan	7/17/1969 12:00:00 AM	35
4	Sharon	Salavaria	6/3/1951 12:00:00 AM	53
5	Gail	Erickson	10/29/1942 12:00:00 AM	61

```
SELECT Name, ListPrice, 'Mountain Bike' AS SubCategoryName FROM
Product WHERE ProductSubCategoryID = 1
```

Δ In this example we make retrievals and aliasing specific result with aliasing column

	Name	ListPrice	SubCategoryName
1	Mountain-500 Black, 40	539.9900	Mountain Bike
2	Mountain-500 Black, 42	539.9900	Mountain Bike
3	Mountain-500 Black, 44	539.9900	Mountain Bike
4	Mountain-500 Black, 48	539.9900	Mountain Bike

Filtering Rows

The WHERE Clause

```
SELECT * FROM Product
WHERE Color = 'Black'
```

```
SELECT Name, StandardCost, Color FROM Product
WHERE Color = 'Black'
```

Example:

```
SELECT Name, ListPrice FROM Product WHERE ListPrice < 5.00
```

Using Operators

Comparison Operators

= (Equals)	<> (Not Equal To)
> (Greater Than)	!= (Not Equal To)
< (Less Than)	!< (Not Less Than)
>= (Greater Than or Equal To)	!> (Not Greater Than)
<= (Less Than or Equal To)	

Logical Operators

ALL	IN
AND	LIKE
ANY	NOT
BETWEEN	OR
EXISTS	SOME

Arithmetic Operators

+	(Add)	/	(Divide)
-	(Subtract)	%	(Modulo)
*	(Multiply)		

Examples:

```
SELECT ProductID, Name, ListPrice FROM Product WHERE
ProductSubCategoryID = 1 OR ListPrice < 1000
```

```
SELECT ProductID, Name, ListPrice FROM Product WHERE NOT
ProductSubCategoryID = 2
```

```
SELECT ProductID, Name, StandardCost FROM Product
WHERE StandardCost IS NULL
```

```
SELECT FirstName, LastName, BirthDate FROM Employee
WHERE BirthDate >= '1-1-62' AND BirthDate <= '12-31-85'
```

```
SELECT FirstName, LastName, BirthDate FROM Employee
WHERE BirthDate BETWEEN '1-1-62' AND '12-31-85'
```

```
SELECT Name FROM StateProvince WHERE StateProvinceCode
IN ('WA', 'OR', 'CA', 'ID', 'NV')
```

```
SELECT Name, ProductNumber, ListPrice, ProductSubCategoryID FROM
Product WHERE ProductSubCategoryID = 1 OR ProductSubCategoryID = 2
AND ListPrice > 500 AND ListPrice < 1000
```

Sorting Results

Ex: `SELECT FirstName, LastName FROM Employee
ORDER BY LastName`

Ex: `SELECT FirstName, LastName FROM Employee
ORDER BY LastName ASC`

Ex: `SELECT FirstName, LastName FROM Employee
ORDER BY LastName, FirstName`

Notes:

- Ascending order
- Descending order
- Default is → ASC

```
SELECT FirstName, LastName FROM Employee ORDER BY LastName, FirstName
```

```
SELECT FirstName, LastName, BirthDate FROM Employee ORDER BY  
BirthDate DESC
```

Eliminating Duplicate Rows

- Use the DISTINCT clause to remove duplicate rows in the result set.
- This example retrieves all rows from the suppliers table but displays each country name only once. USE adventureworks `SELECT DISTINCT City FROM Person.Address`

Ex: `SELECT DISTINCT City FROM Person.Address`

Displaying the TOP n Rows in Results Set

- Use TOP n to specify that you want SQL Server to return only a specific number of rows in the results

Ex: `SELECT TOP (5) OrderQty, SalesOrderID
FROM Sales.SalesOrderDetail`



Performance Considerations

- Use positive rather than negative search conditions. Negative search conditions such as NOT BETWEEN, NOT IN, and IS NOT NULL. May slow data retrieval because all rows are evaluated.
- Avoid using the LIKE search condition if you can write a more specific query. Data retrieval may be slower when you use the LIKE search condition.
- Use exact matches or ranges as search conditions when possible. Again, specific queries perform better.
- Data retrieval may decrease if you use the ORDER BY clause because SQL Server must determine and sort the result set before it returns the first row.

Chapter 3

Play with ADO.NET

What is ADO.NET?

1. ADO.NET is the new database technology of the .NET (Dot Net) platform, and it builds on Microsoft ActiveX® Data Objects (ADO).
2. ADO.NET is an integral part of the .NET Compact Framework, providing access to relational data, XML documents, and application data.
3. ADO.NET defines DataSet and DataTable objects which are optimized for moving disconnected sets of data across intranets and Internets, including through firewalls. It also includes the traditional Connection and Command objects, as well as an object called a DataReader that resembles a forward-only, read-only ADO record set. If you create a new application, your application requires some form of data access most of the time.

The ADO.NET classes are found in System.Data.dll (Root Class) and are integrated with the XML classes in System.Xml.dll.

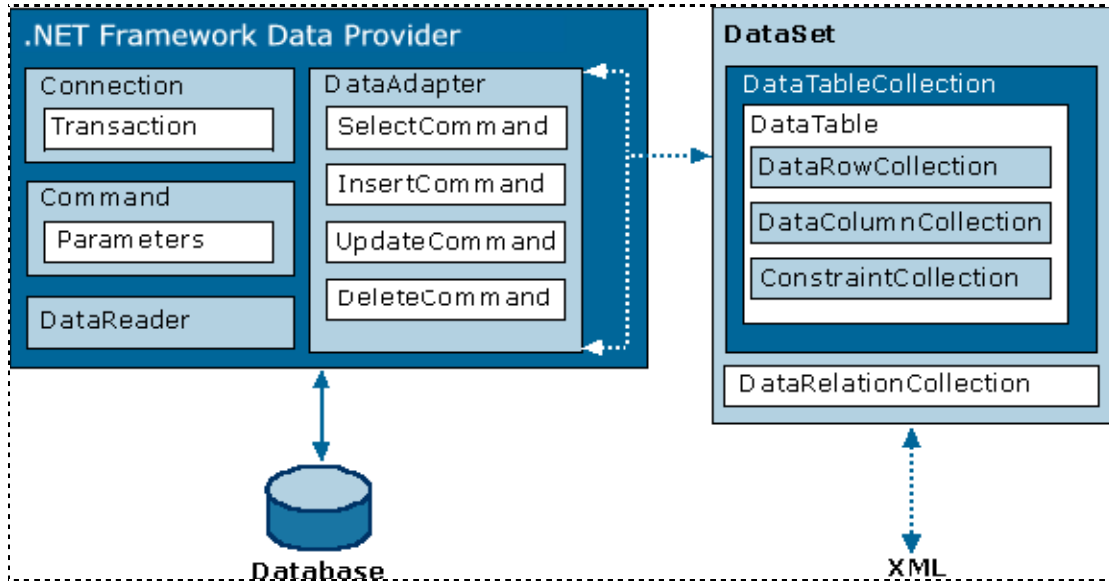
DataSet object represents a disconnected cache of data which is made up of DataTables and DataRelations that represent the result of the command.

When you retrieve data, you use an object known as a data reader. When you work with disconnected data, the data is cached locally in a relational data structure, either a data table or a dataset.

ADO.NET and the .NET Base Class Library

Namespace	Description
System.Data	Classes, interfaces, delegates, and enumerations that define and partially implement the ADO.NET architecture
System.Data.Common	Classes shared by .NET Framework data providers
System.Data.Design	Classes that can be used to generate a custom-typed dataset
System.Data.Odbc	The .NET Framework data provider for ODBC
System.Data.OleDb	The .NET Framework data provider for OLE DB
System.Data.Sql	Classes that support SQL Server-specific functionality
System.Data.OracleClient	The .NET Framework data provider for Oracle
System.Data.SqlClient	The .NET Framework data provider for SQL Server
System.Data.SqlServerCe	The .NET Compact Framework data provider for SQL Server Mobile
System.Data.SqlTypes	Classes for native SQL Server data types
Microsoft.SqlServer.Server	Components for integrating SQL Server and the CLR

The ADO.NET Object Model (Architecture)



☒ ADO.NET allows data manipulation to be done using either a connected or a disconnected model. The following table describes how each method works

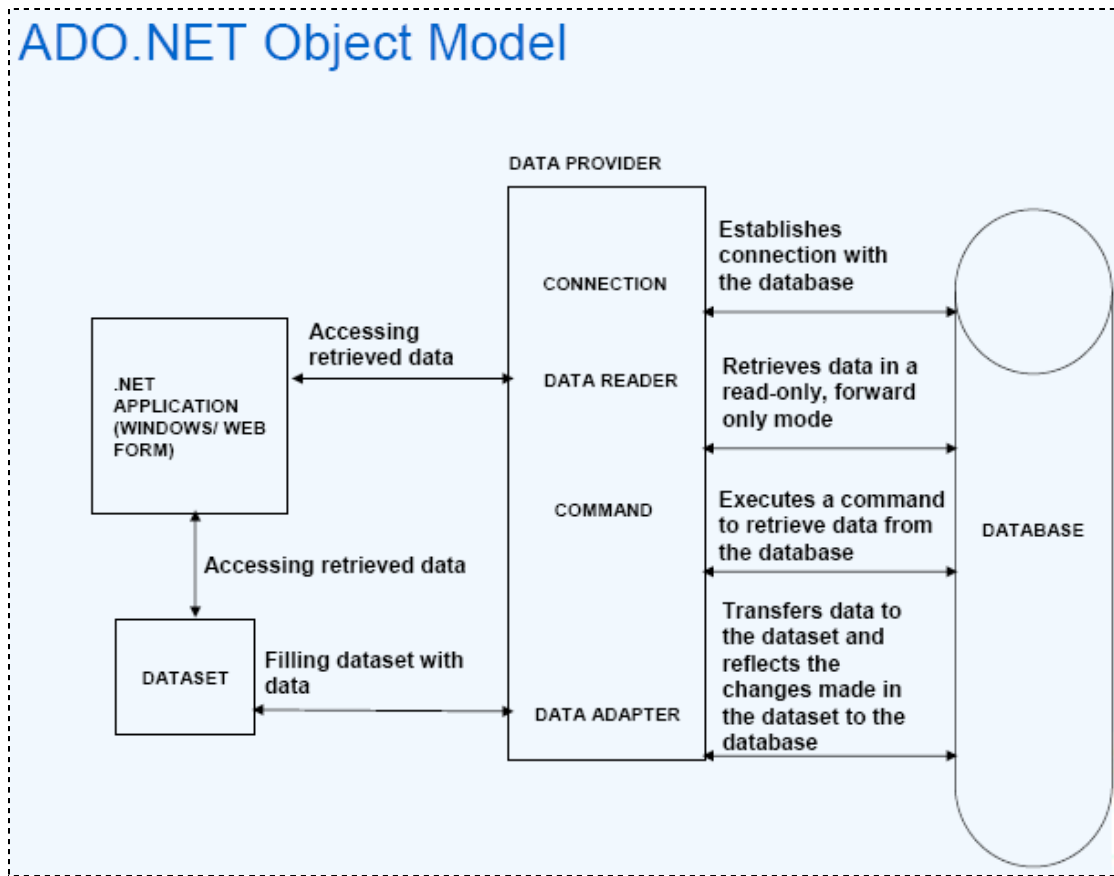
Connected Model	Disconnected Model
1. The application establishes a connection with the data source. 2. The application reads data and executes commands to manipulate data on the data source. 3. The connection remains open until closed by either the client or the server.	1. The application establishes a connection with the data source. 2. Requested data is transferred to the application and stored in a memory cache on the client. 3. The connection is closed. 4. The application modifies or otherwise works with the data. 5. To synchronize changes back to the data source, the connection is reestablished, the data transferred, and the connection is immediately closed. The disconnected model helps conserve server resources and helps the scalability of the application.

The following table lists the components used in each data access model.

Component	Description	Applicable Data Access Model
Data source	The data source is the database on the server, but can also be many other kinds of data repositories such as an XML file.	Connected Disconnected
Connection	The Connection object includes information necessary to establish a connection to the data source. Connections are data source specific. For example, use a SqlConnection to make a	Connected Disconnected

	connection to a SQL database.	
Command	A Command object executes read and write operations on the data source. Command objects are data source specific and are associated with a Connection object.	Connected Disconnected
DataReader	The DataReader represents a set of read-only, forward-only rows from a database. Use a DataReader when you need fast access and need only to read through the data from beginning to end one time. The DataReader is data source specific.	Connected
DataAdapter	A DataAdapter manages the process of communicating with the data source. Commands are associated with a DataAdapter. When a command is executed, the DataAdapter automatically opens and closes the connection with the data source. The DataAdapter is data source specific.	Disconnected
DataSet	The DataSet is the client-side, in-memory copy of the data. DataAdapters fill the DataSet tables, rows, and columns using data from the data source. DataSets are source neutral, meaning they can contain data from different kinds of data sources. DataSets also have multiple tables. A single DataSet can therefore be used to hold data from multiple different sources. You can also use the DataSet to establish relationships between tables.	Disconnected

Summarize to ADO.NET Process



Golden Steps to deal with any database

- Connect to database
- Access Data
 - Your command (insert, update, delete or delete)
 - Determine the mechanism of retrieving data (dataset or DataReader)
- Close connection

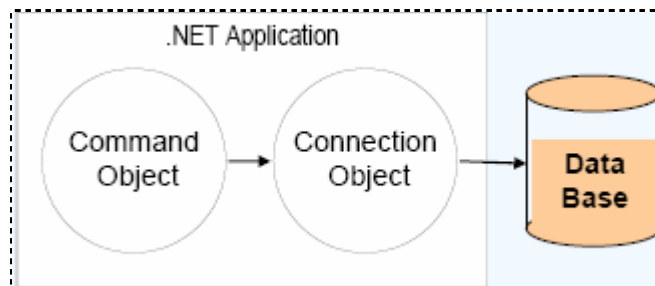


SqlCommand class

Used to interact with database using SQL Queries or stored procedure. This class cannot be inherited. For example, you can do select, insert, modify, and delete commands on rows of data in a data base table

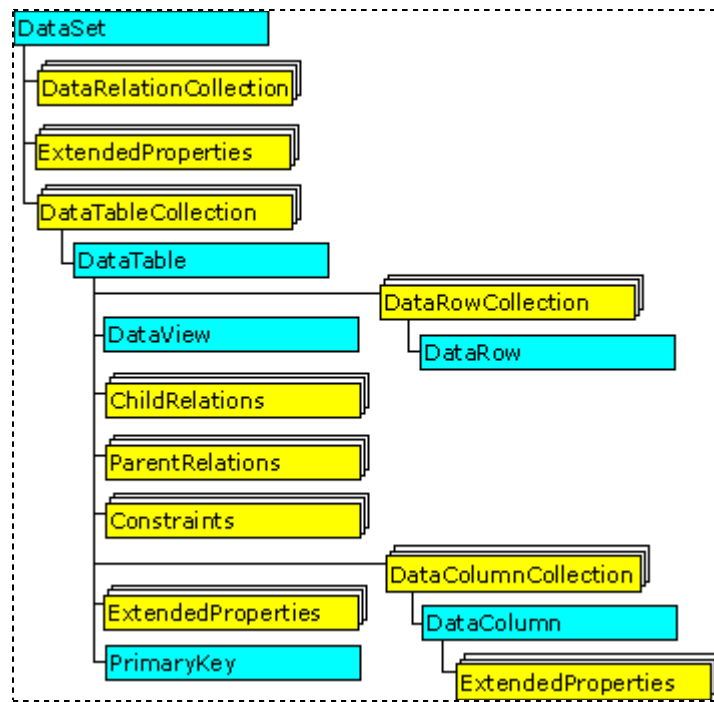
Important method in SqlCommand

ExecuteReader	Executes commands that return rows. For increased performance, ExecuteReader invokes commands using the Transact-SQL sp_executesql system stored procedure. Therefore, ExecuteReader might not have the effect that you want if used to execute commands such as Transact-SQL SET statements.
ExecuteNonQuery	Executes commands such as Transact-SQL INSERT, DELETE, UPDATE, and SET statements.
ExecuteScalar	Retrieves a single value (for example, an aggregate value) from a database.



ADO.NET Dataset

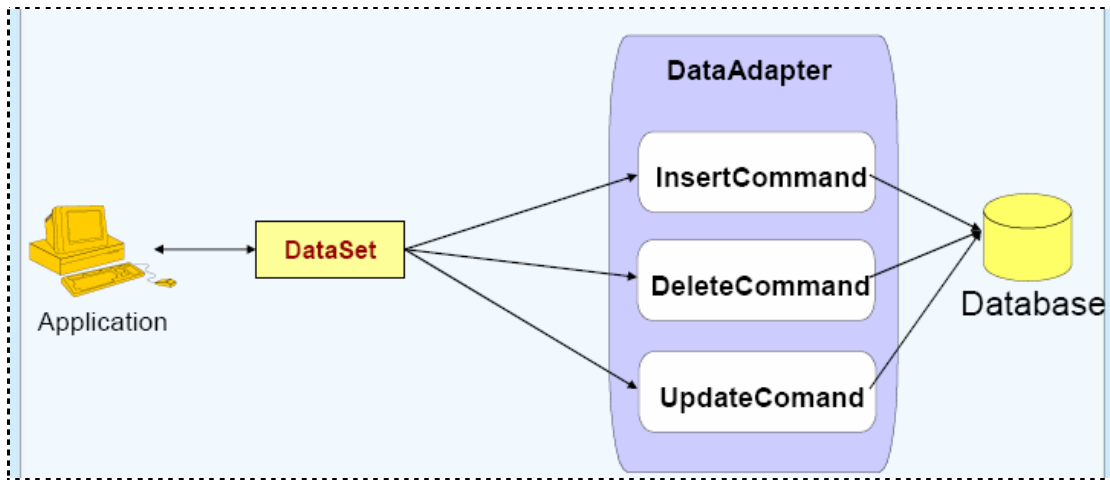
Represent disconnected model. Essentially this class is an in-memory database. It can contain tables made up of multiple columns of varying data types. Each table can contain multiple rows, and those rows can be related to each other through foreign keys as well as complex relationships that enforce parent/child data constraints. Data Tables can assign new, unique, numeric identifiers to rows as they are added to the table.



DataAdapter

The DataAdapter class functions very much like an electrical plug. An electrical plug connects an appliance to a power source. The DataAdapter connects a DataSet (or DataTable) to a data source. This "plug" has four prongs, one for each type of connection that can take place:

- **InsertCommand** This command is executed when an item in an associated DataTable is ready to be inserted into the data source.
- **DeleteCommand** This command is executed when an item in an associated DataTable is ready to be deleted from the data source.
- **UpdateCommand** This command is executed to commit pending changes to an item in a DataTable.
- **SelectCommand** This command is executed to populate the DataTable or DataSet with the information retrieved from the data source.



"DataAdapter like a bridge that used to pass command through it"

Example 1.1

[Test Your Connection]

```
using System.Data.SqlClient;
using System.Data;
class Program
{
    static void Main(string[] args)
    {
        SqlConnection testConnection = new SqlConnection("Your path of
        connection String here");

        try
        {
            testConnection.Open();
            if (testConnection.State == ConnectionState.Open)
            {
                Console.WriteLine("Successfully opened a connection");
            }
        }
        catch (Exception)
        {
            if (testConnection.State != ConnectionState.Open)
            {
                Console.WriteLine("Failed to open a connection");
            }
        }
        finally
        {
            // Closing a connection ensures connection pooling.
            if (testConnection.State == ConnectionState.Open)
            {
                testConnection.Close();
            }
            testConnection.Dispose();
        }
    }
}
```

Generating Provider-Specific Connection Strings

There are many parameters that could be specified in a connection string. Each of those parameter names needs to be spelled properly with a valid value in order for us to establish a connection to the database successfully. Not only could it get difficult to remember each parameter name and its spellings properly ADO.NET 2.0 tries to address this problem by providing a `DbConnectionStringBuilder` class. The `DbConnectionStringBuilder` object strongly types various connection string constituent values in order to avoid trivial programming errors as well as to make the connection string information more manageable.

The `DbConnectionStringBuilder` work like this scenario

The database I am interested in connecting to . . .

. . . Is on my local machine.

. . . Has the name "Test".

. . . Will allow me to connect using Windows authentication.

. . . etc.

Example 1.2

[Test Connection establish or not]

```
static void Main(string[] args)
{
    SqlConnectionStringBuilder connstrBuilder = new
    SqlConnectionStringBuilder();
    connstrBuilder.DataSource = "(local)";
    connstrBuilder.InitialCatalog = "Test";
    connstrBuilder.IntegratedSecurity = true;

    using (SqlConnection testConnection =
    new SqlConnection(connstrBuilder.ToString()))
    {
        try
        {
            testConnection.Open();
            if (testConnection.State == ConnectionState.Open)
            {
                Console.WriteLine("Connection successfully opened");
                Console.WriteLine("Connection string used: " +
                testConnection.ConnectionString);
            }
        }
        catch (Exception)
        {
            if (testConnection.State != ConnectionState.Open)
            {
                Console.WriteLine("Connection open failed");
                Console.WriteLine("Connection string used: "
                + testConnection.ConnectionString);
            }
        }
    }
}
```

```
    }  
    }  
    // Automatic dispose call on conn ensures connection is closed.  
    Console.WriteLine("Press any key to continue ..");  
    Console.Read();  
}
```

How to write your connection String to be shared in app.config

Steps

- ✓ Add Reference: System. Configuration
- ✓ Make using to your Reference
- ✓ Add app.config file to your project.

Note:

In scope of configuration write your connectionStrings and in scope of connectionStrings write the tag that hold setting of your connectionStrings.

Name	Functionality
Name	Set name to your connectionStrings to deal with it when you call the connection string.
connectionString	The path of your connectionString
providerName	The type of your database such as SQL Server , Oracle , etc.

In app.config write:

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <connectionStrings>  
    <add name="ConnString" connectionString="Data  
Source=.\SQLEXPRESS;AttachDbFilename=C:\Documents and  
Settings\Administrator\My Documents\AhmedRabie.mdf;Integrated  
Security=True;Connect Timeout=30;User Instance=True"  
providerName="System.Data.SqlClient"/>  
  </connectionStrings>  
</configuration>
```

Code:

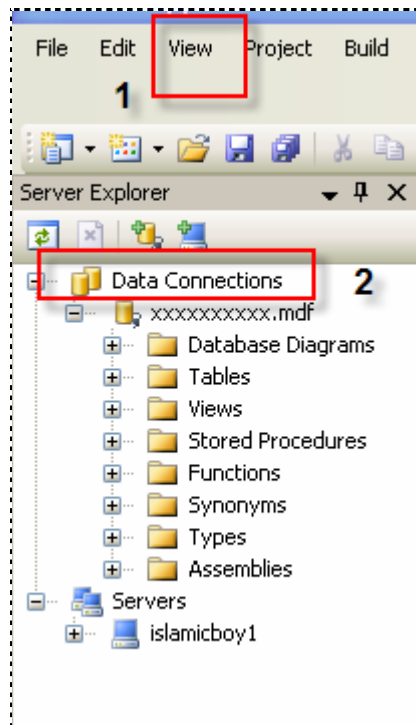
```
static void Main(string[] args)  
{  
    SqlConnection con = new  
    SqlConnection(ConfigurationManager.ConnectionStrings["ConnString"].Co  
nnectionString);  
    con.Open();  
    if (con.State == ConnectionState.Open)  
    {
```

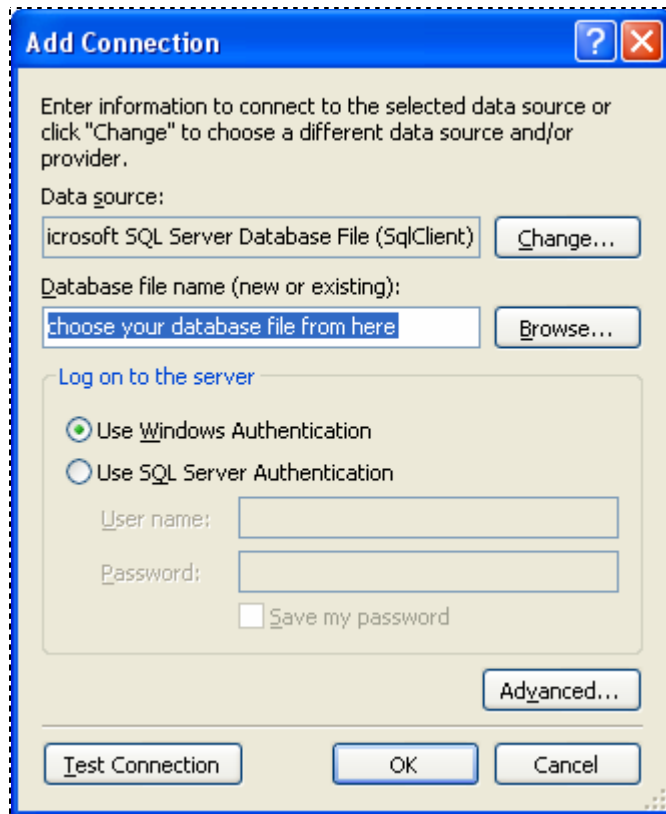
```
        Console.WriteLine("open");  
    }  
    else  
    {  
        Console.WriteLine("not");  
    }  
    con.Close();  
}
```

How to get the path of your Connection String

There are two way to get the path of your connection string

1. to attach or create your database in your application (SQL EXPRESS)
 - From view>Server Explorer>Data Connections>Click Right>Add Connection.



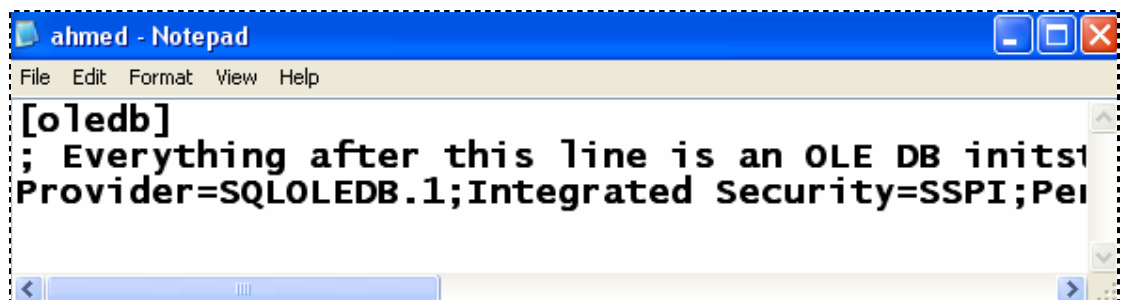


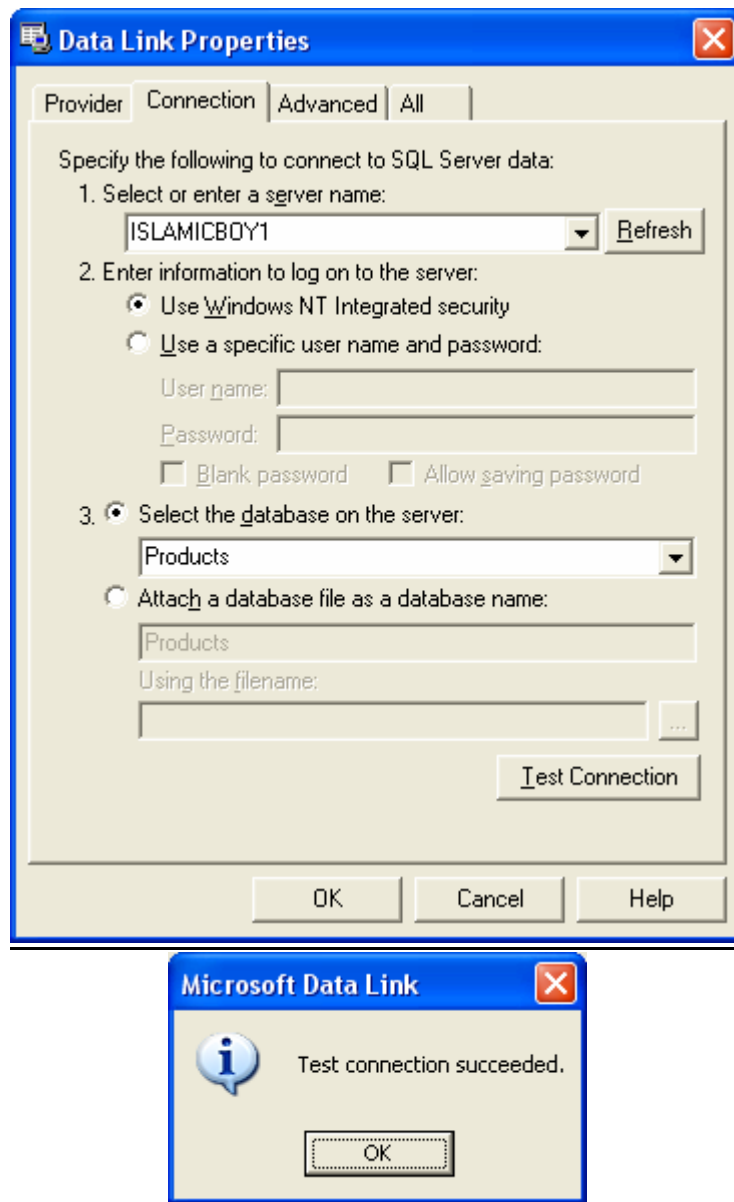
2. to use database from SQL SERVER

- The Microsoft universal data link (.udl) file offers a convenient, alternative method for creating and remembering complex connection strings.

Steps:

- Create a new text file on your hard disk. Name it myfile.udl.
- Double-click myfile.udl to bring up the Data Link Properties
- Open the Provider tab and choose the Microsoft OLE DB Provider for SQL SERVER
- In connection tab write the name of server and then select your database file that you want to work with it.
- Click ok to submit your work.
- To view your connection string, open any notepad file and drag and drop to myfile.udl on it and it will appear like this





Connection Pooling

It's important to realize that the majority of the time the user might hold an open connection and not actively use it because he is busy with other parts of the application. The application could essentially "time slice" his expensive resource—an open connection—and pool it between multiple users.

Using connection pooling with ADO.NET is really simple because you don't have to do anything to use connection pooling with the default settings; instead, you have to turn it off explicitly should you decide not to use it. For instance, for `SqlConnection`, if you don't wish to pool your connections, you simply add the following key-value pair to your connection string:

```
Pooling=false;
SqlConnection testConnection =
new SqlConnection("Data Source=(local);Initial Catalog=Test;" +
"Integrated Security=SSPI;Pooling=false");
```

Example 1.3

[Demonstrating Connection Pooling in C#]

```
SqlConnection testConnection =
new SqlConnection("Data Source=(local);Initial
Catalog=Test;Integrated Security=SSPI;");
    long startTicks = DateTime.Now.Ticks;
    for (int i = 1; i <= 100; i++)
    {
        testConnection.Open();
        testConnection.Close();
    }
    long endTicks = DateTime.Now.Ticks;
    Console.WriteLine("Time taken : " + (endTicks -
startTicks) + " ticks.");
    testConnection.Dispose();
```

The following example will show how to get the list of ADO.NET data providers registered in the machine and application configuration file

Example 1.4

[Display Framework data providers]

```
using System;
using System.Data;
using System.Data.Common;

class MainClass
{
    public static void Main(string[] args)
    {
        using (DataTable providers =
DbProviderFactories.GetFactoryClasses())
        {
            Console.WriteLine("Available Framework data providers:");
            foreach (DataRow prov in providers.Rows)
            {
                Console.WriteLine("Name:{0}", prov["Name"]);
                Console.WriteLine("Description:{0}", prov["Description"]);
                Console.WriteLine("Invariant Name:{0}", prov["InvariantName"]);
            }
        }
    }
}
```

The following example show how to calculate number of rows in your table

Example 1.5

[Do a row count using SqlCommand]

```
using System;
using System.Data;
using System.Data.SqlClient;

class CommandScalar
{
    static void Main()
    {
        SqlConnection conn = new
SqlConnection("server=(local)\\SQLEXPRESS;database=MyDatabase;Integra
ted Security=SSPI;");

        string sql = @"select count(*) from employee";

        SqlCommand cmd = new SqlCommand(sql, conn);
        Console.WriteLine("Command created and connected.");

        try
        {
            conn.Open();
Console.WriteLine("Number of Employees is {0}", cmd.ExecuteScalar());
        }
        catch (SqlException ex)
        {
            Console.WriteLine(ex.ToString());
        }
        finally
        {
            conn.Close();
            Console.WriteLine("Connection Closed.");
        }
    }
}
```

Example 1.6

[Display your data in cmd using DataSet]

```
SqlConnection sqlCon = new SqlConnection();
sqlCon.ConnectionString =
"Server=.;database=MyDataBase;uid=sa;pwd=sa ";
SqlCommand sqlCmd = new SqlCommand();
sqlCmd.CommandText = "select * from studentInfo";
sqlCmd.Connection = sqlCon;
SqlDataAdapter adapetr = new SqlDataAdapter(sqlCmd);
DataSet daSet = new DataSet();
adapetr.Fill(daSet);
sqlCon.Close();
DataTable table = daSet.Tables[0];
foreach (DataRow row in table.Rows)
{
Console.WriteLine(row[0]+"\\t"+row[1]+"\\t"+row[2]+"\\n");
}
}
```

Quick Notes

► Note that it's no important to open connection when you use dataset to retrieve data from your table in database but its necessary to open it when you manipulate your data with some operation like update or delete. The explanation of this due to the natural of dataset that take copy from tables and host it in memory of client and then you determine the table that you want to display, in other hand when you make any change you need to make open because you want to enter to specific cell in table and change it but when you make retrieve you take your data as it.

► If you observe the previous code you will ask why I close connection before display data, the answer to this question because you fill your data from source to the memory of client so you already take copy from your data and you don't need to make connection continues and when you make changes in data you will make re-connection to server to commit changes.

► what is the difference between DataSet and DataTable?

DataTable	Represent one table in memory
DataSet	Represent more than table in memory

So if you have one table in your database you can use DataTable instead of DataSet.

Example 1.7

[Display your Data in DataGridView]

```
SqlConnection sqlCon = new
SqlConnection ("Server=.;database=MyDataBase;uid=sa;pwd=sa");

SqlCommand sqlCmd = new SqlCommand();
sqlCmd.CommandText = "select * from StudentInfo";
sqlCmd.Connection = sqlCon;
sqlCon.Open();
SqlDataAdapter adapter = new SqlDataAdapter(sqlCmd);
DataSet dSet = new DataSet();
adapter.Fill(dSet, "StudentInfo");
dataGridView1.DataSource = dSet;
dataGridView1.DataMember = "StudentInfo";
sqlCon.Close();
```

to fill yor data in DataAdapter you will meet three shape of code that do the same purpose

```
adapter.Fill(dSet, "StudentInfo");
dataGridView1.DataSource = dSet;
```

```
dataGridView1.DataMember = "StudentInfo";
```

OR

```
adapter.Fill(dSet)  
dataGridView1.DataSource = dSet.Tables[0];
```

OR

```
adapter.Fill(dSet)  
DataTable dt = dSet.Tables[0];  
dataGridView1.DataSource = dt;
```

Example 1.8

[display some element in Combox]

```
SqlConnection sqlCon2 = new SqlConnection();  
sqlCon2.ConnectionString = "Your Path here";  
SqlCommand sqlCmd2 = new SqlCommand();  
sqlCmd2.CommandText = "select * from StudentInfo";  
sqlCmd2.Connection = sqlCon2;  
sqlCon2.Open();  
SqlDataAdapter adapter = new SqlDataAdapter(sqlCmd2);  
DataSet ds = new DataSet()  
adapter.Fill(ds);  
comboBox1.DataSource = ds.Tables[0]; // My dataSet  
comboBox1.ValueMember = "name"; // This is the actual  
column name that holds the values I want
```

in the previous example we bind combox using the following code:

```
comboBox1.DataSource = ds.Tables[0];  
comboBox1.ValueMember = "name";
```

we can do the same task using the following code also:

```
DataTable table = ds.Tables[0];  
dataGridView1.DataSource = table;  
  
comboBox1.Items.Clear();  
foreach (DataRow row in table.Rows)  
{  
    comboBox1.Items.Add(row[1]);  
}  
  
sqlCon.Close();
```

SqlDataReader

Provides a way of reading a forward-only stream of rows from a SQL Server database. This class cannot be inherited. To create a SqlDataReader, you must call the ExecuteReader method of the SqlCommand object, instead of directly using a constructor.

Example 1.9

[Display your Data in Console]

```
SqlConnection sqlCon = new SqlConnection();
    sqlCon.ConnectionString = "Your connection String";
    sqlCon.Open();
    SqlCommand sqlCmd = new SqlCommand();
    sqlCmd.CommandText = "select * from studentInfo";
    sqlCmd.Connection = sqlCon;
    SqlDataReader sqlReader = sqlCmd.ExecuteReader();
    Console.WriteLine("ID" + "\t" + "Name" + "\t" + "Age" + "\n");
    while (sqlReader.Read())
    {
        Console.WriteLine(sqlReader[0] + "\t" + sqlReader[1] + "\t" +
            sqlReader[2] + "\n");
    }
    sqlCon.Close();
```

Example 1.10

DataReader Example (display some element in Combox)

```
SqlConnection sqlCon2 = new SqlConnection();
    sqlCon2.ConnectionString = @"Data
Source=.\SQLEXPRESS;AttachDbFilename=D:\Documents and
Settings\ahmed\My Documents\Ahmed.mdf;Integrated
Security=True;Connect Timeout=30;User Instance=True";
    SqlCommand sqlCmd2 = new SqlCommand();
    sqlCmd2.CommandText = "select * from studentInfo";
    sqlCmd2.Connection = sqlCon2;
    sqlCon2.Open();
    SqlDataReader reader = sqlCmd2.ExecuteReader();
    while (reader.Read())
    {
        //or //comboBox1.Items.Add(reader[1]);
        comboBox1.Items.Add(reader.GetString(1));
    }
    comboBox1.Text = comboBox1.Items[0].ToString();
```

While we're using an instance of `SQLDataReader`, we're talking to it through an `IDataReader` interface. Talking to objects through interfaces makes our code more portable—for instance, it could more easily be converted to work with a data provider other than SQL Server—so use `IDataReader` unless you need additional functionality that the basic `IDataReader` interface doesn't support.

Example 1.10

```
static void Main()
{
    string connectionString = "you connection string";
    string query = "SELECT * FROM Products";
    using (SqlConnection connection =
        new SqlConnection(connectionString))
```

```
using (SqlCommand command = new SqlCommand(query, connection))
{
    connection.Open();
    IDataReader dr =
    command.ExecuteReader(CommandBehavior.CloseConnection);
    while (dr.Read())
    {
        comboBox1.Items.Add(reader.GetString(1));
    }
}
}
```

Fill dataGridView Using DataReader

One of the most questions asked to me, how to fill dataGridView Using DataReader, if you note the behavior of DataReader you will note that when you read using DataReader you read row by row so you make looping to read all records if you want to display all record or any element from record, in other hand dataGridView need to take all data of your records in one package, so the solution is to make centralize store by make class that contains properties and set value for each properties when you read the value of each record and store each object from class that you create in ArrayList so the ArrayList will represent your store.

Steps:-

1. create class that contains properties

```
public class MyDetails
{
    private int age;

    public int Age
    {
        get { return age; }
        set { age = value; }
    }

    private string name;

    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    private int id;

    public int Id
    {
        get { return id; }
        set { id = value; }
    }
}
```

2. create ArrayList to represent your store

```
ArrayList sequence = new ArrayList();
```

3. when you retrieve do the following

```
while (reader.Read())
{
    MyDetails m = new MyDetails();
    m.Id = (int)reader[0];
    m.Name = reader[1].ToString();
    m.Age = (int)reader[2];
    sequence.Add(m);
}
dataGridView1.DataSource = sequence;
```

So the final code

```
SqlConnection sqlCon = null;
try
{
    sqlCon = new SqlConnection();
    sqlCon.ConnectionString = "Your Connection String";
    SqlCommand cmd = new SqlCommand();
    cmd.Connection = sqlCon;
    cmd.CommandText = "SELECT * FROM StudentInfo";
    sqlCon.Open();
    SqlDataReader reader = cmd.ExecuteReader();
    while (reader.Read())
    {
        MyDetails m = new MyDetails();
        m.Id = (int)reader[0];
        m.Name = reader[1].ToString();
        m.Age = (int)reader[2];
        sequence.Add(m);
    }
    dataGridView1.DataSource = sequence;
}
finally
{
    sqlCon.Close();
}
```

Data Binding

Data binding is the process of binding the retrieval data to control on windows forms to be displayed in a customize format.

Data binding is of two types

- ▶ Simple Data Binding
- ▶ Complex Data Binding

Simple Binding

- ControlObject.DataBindings.Add("Property", DataSource, "Data member");
- Binding TextBox with au_id column from DataSet
`textBox1.DataBindings.Add( "Text", myDataSet1.Tables[0], "Au_id");`
- Binding TextBox with string array called myData
`textBox1.DataBindings.Add( "Text", myData, "");`
- By default, the first record from data source is bound.
- To bind, the other records,
`this.BindingContext[ DataSource ].Position = the position of the records`

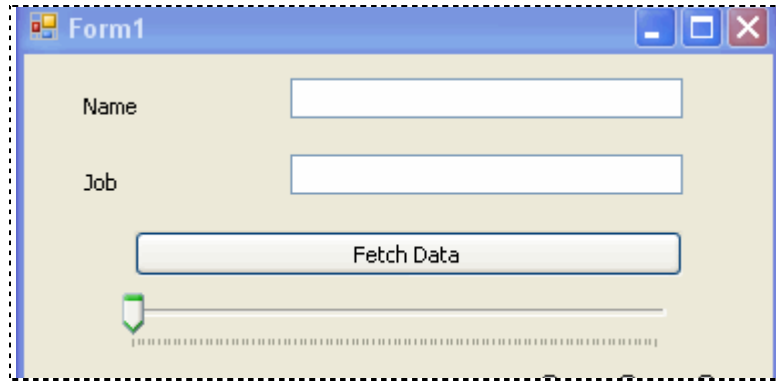
Example

- `this.BindingContext[ myDataSet1.Tables[0] ].Position = 10;`
 - `this.BindingContext[ myData ].Position = 10;`
-

Complex Binding Example with ComboBox

```
comboBox1.DataSource = dataSet11.Tables["Authors"];  
comboBox1.DisplayMember = "au_fname";  
comboBox1.ValueMember = "au_id";
```

Example: Scrolling Data Binding



```
public partial class Form1 : Form
{
    ArrayList sequence = new ArrayList();

    public Form1()
    {

    }

    private void trackBar1_Scroll(object sender, EventArgs e)
    {
        this.BindingContext[ds, "StudentInfo"].Position =
            trackBar1.Value;
    }

    private DataSet CreateDataSet()
    {
        string customers = "SELECT * FROM StudentInfo";
        DataSet ds = new DataSet();

        using (SqlConnection con = new SqlConnection(@"your connection"))
        {
            {
                SqlDataAdapter da = new SqlDataAdapter(customers, con);
                da.Fill(ds, "StudentInfo");
            }

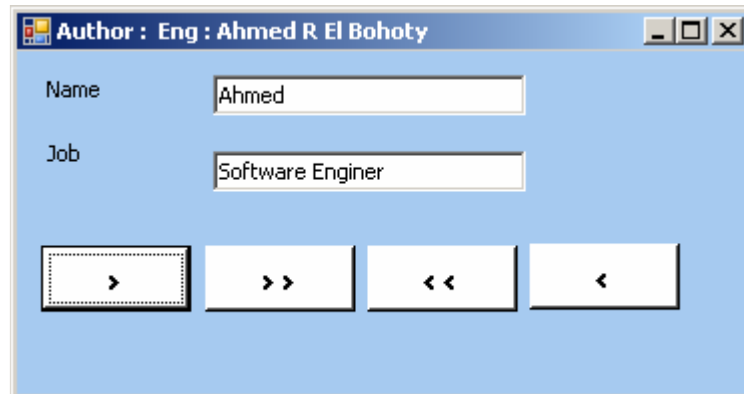
            return ds;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            button1.Enabled = false;
            ds = CreateDataSet();

            textBox1.DataBindings.Add("Text", ds, "StudentInfo.name");
            textBox2.DataBindings.Add("Text", ds, "StudentInfo.job");

            trackBar1.Minimum = 0;
            trackBar1.Maximum = this.BindingContext[ds, "StudentInfo"].Count - 1;
        }
    }
}
```

Navigation between record using data binding



Steps

1. Add 2 TextBox
2. Add 2 Lable
3. Add 4 Buttons

```
private void btnFetchData_Click(object sender, EventArgs e)
{
    ds = FillData();
    txtName.DataBindings.Add("Text", ds, "emp.Name");
    txJob.DataBindings.Add("Text", ds, "emp.Job");
}

private void button1_Click(object sender, EventArgs e)
{
    this.BindingContext[ds, "emp"].Position = 0;
}

private void button2_Click(object sender, EventArgs e)
{
    this.BindingContext[ds, "emp"].Position -= 1; ;
}

private void button3_Click(object sender, EventArgs e)
{
    this.BindingContext[ds, "emp"].Position += 1;
}

private void button4_Click(object sender, EventArgs e)
{
    this.BindingContext[ds, "emp"].Position =
        this.BindingContext[ds, "emp"].Count - 1;
}
```

How to Make Arabic Column in DataGridView

Because we can not name any column to be Arabic in any RDBMS so some people make applications that have Arabic Interface therefore, DataGridView Can Solve This Problem by Give Arabic Name to Your Column When You Display Your Data.



There are two ways to make Arabic column in DataGridView, the first way is so easy it depends on changing the Header Text of DataGridView as the following:

```
dataGridView1.Columns[0].HeaderText = "م";  
dataGridView1.Columns[1].HeaderText = "الاسم";  
dataGridView1.Columns[2].HeaderText = "العمر";
```

Note: to complete your code set the property of RightToLeft

```
dataGridView1.RightToLeft = true;
```

The second way as the following:

DataGridViewTextBoxColumn: used to host cells that enable displaying and editing of text strings.

Steps

1. Create instance from DataGridViewTextBoxColumn class
2. Add cells you created to be hosted in dataGridView1

```
dataGridView1.Columns.AddRange(new DataGridViewTextBoxColumn[] {  
    idDataGridViewTextBoxColumn,  
    nameDataGridViewTextBoxColumn,  
    ageDataGridViewTextBoxColumn});
```

3. Ok, after you add cells to dataGridView you want to make link between the cell which you write on it Arabic text and the column

```
idDataGridViewTextBoxColumn.DataPropertyName = "id";  
idDataGridViewTextBoxColumn.HeaderText = "م";
```

Example

```
DataGridViewTextBoxColumn idDataGridViewTextBoxColumn;  
DataGridViewTextBoxColumn nameDataGridViewTextBoxColumn;  
DataGridViewTextBoxColumn ageDataGridViewTextBoxColumn;  
  
private void Form1_Load(object sender, EventArgs e)  
{  
  
    idDataGridViewTextBoxColumn = new DataGridViewTextBoxColumn();  
    nameDataGridViewTextBoxColumn = new DataGridViewTextBoxColumn();  
    ageDataGridViewTextBoxColumn = new DataGridViewTextBoxColumn();  
  
    dataGridView1.Columns.AddRange(new DataGridViewTextBoxColumn[] {  
        idDataGridViewTextBoxColumn,  
        nameDataGridViewTextBoxColumn,  
        ageDataGridViewTextBoxColumn});  
  
    using (SqlConnection sqlCon = new SqlConnection(  
        @"Your Connection String"))  
    {  
        SqlCommand cmd = sqlCon.CreateCommand();  
        cmd.CommandText = "Select * from StudentInfo";  
        SqlDataAdapter adapter = new SqlDataAdapter();  
        adapter.SelectCommand = cmd;  
        DataSet dset = new DataSet();  
        adapter.Fill(dset);  
  
        idDataGridViewTextBoxColumn.DataPropertyName = "id";  
        idDataGridViewTextBoxColumn.HeaderText = "م";  
        idDataGridViewTextBoxColumn.Name =  
        "idDataGridViewTextBoxColumn";  
        //  
        // nameDataGridViewTextBoxColumn  
        //  
        nameDataGridViewTextBoxColumn.DataPropertyName = "name";  
        nameDataGridViewTextBoxColumn.HeaderText = "الاسم";
```

```
nameDataGridViewTextBoxColumn.Name =  
"nameDataGridViewTextBoxColumn";  
//  
// ageDataGridViewTextBoxColumn  
//  
ageDataGridViewTextBoxColumn.DataPropertyName = "age";  
ageDataGridViewTextBoxColumn.HeaderText = "العمر";  
ageDataGridViewTextBoxColumn.Name =  
"ageDataGridViewTextBoxColumn";  
  
dataGridView1.DataSource = dset.Tables[0];  
}  
}
```

Using Stored Procedures in ADO.NET

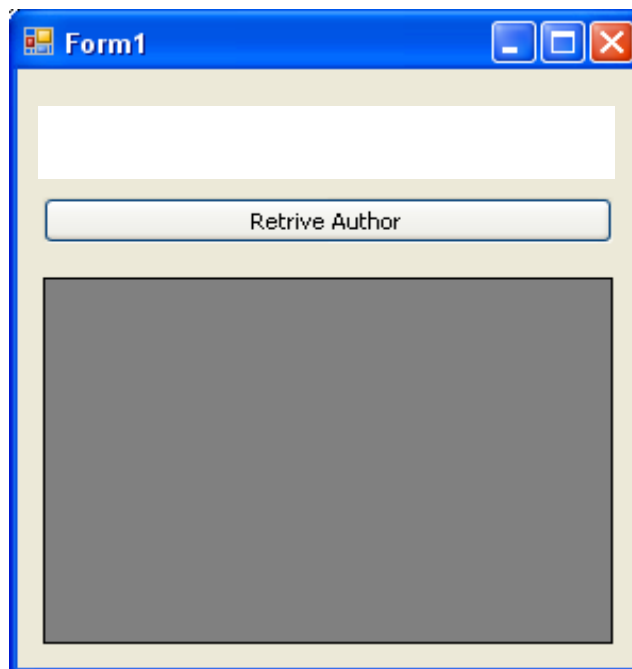
A stored procedure is a module of code that allows you to reuse a desired piece of functionality and call that functionality by name. Stored procedures are SQL statements that allow you to perform a task repeatedly. You can create a procedure once and reuse it any number of times in your program

SQL injection attack

- ▶ To solve the problem of SQL injection you have two choices
 - **First:** use a stored procedure
 - **Second:** To se SQL Parameterise

If you use a stored procedure that uses a parameter, it provides a level of protection against SQL injection attacks in a Web application. A SQL injection attack uses T-SQL code entered by a user to access unauthorized information. This can make SQL Server more vulnerable to future attack, as a hacker gathers information about the structures used in SQL Server

To be More Sure about the problem of SQL injections see how I will drop the table by pass drop statement in the textbox.



Steps

Add Text Box

Add Button

Add DataGridView

Code Steps

In Run Time add to the textbox the following Statement

```
Ahmed Rabie'; DROP TABLE BOOKS --
```

And this you code:

```
string sqlcmd = "SELECT * FROM BOOKS WHERE Authors = '" +  
textBox1.Text + "'";  
SqlConnection con = new SqlConnection();  
con.ConnectionString = @"Data  
Source=.\SQLEXPRESS;AttachDbFilename=C:\Documents and  
Settings\Administrator\My Documents\AhmedRabie.mdf;Integrated  
Security=True;Connect Timeout=30;User Instance=True";  
SqlCommand cmd = new SqlCommand();  
cmd.Connection = con;  
cmd.CommandText = sqlcmd;  
SqlDataAdapter adapter = new SqlDataAdapter();  
adapter.SelectCommand = cmd;  
DataSet dst = new DataSet();  
adapter.Fill(dst, "BOOKS");  
dataGridView1.DataSource = dst;  
dataGridView1.DataMember = "BOOKS";
```

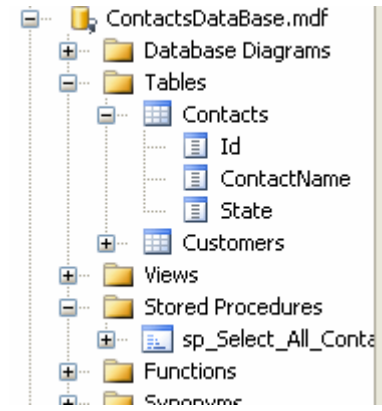
When you make refresh to your database you will note that your table is removed from your database.

Work with Stored Procedure divided into 3 types of work

- Executing a Stored Procedure with No Parameters
⇒ Ex.: Produces a list of the names of employees in the Northwind database. It requires no input and doesn't need to set a return value.
- Executing a Stored Procedure with input Parameters
⇒ Create a stored procedure that produces a list of orders for a given employee. You'll pass the employee ID to the stored procedure for use in a query.
- Executing a Stored Procedure with Output Parameters
⇒ Output parameters are usually used to pass values between stored procedures, but sometimes they need to be accessed from C#.

Executing a Stored Procedure with No Parameters

► Example:



```
Create procedure sp_Select_All_Contacts
as
select
id, ContactName, State
from
Contacts
```

Code:

```
SqlConnection con = new SqlConnection();
con.ConnectionString = @"Your Connection String";
SqlCommand cmd = new SqlCommand();
cmd.CommandType = CommandType.StoredProcedure;
cmd.CommandText = "sp_Select_All_Contacts";
cmd.Connection = con;
SqlDataAdapter ad = new SqlDataAdapter(cmd);
DataSet ds = new DataSet();
ad.Fill(ds);
dataGridView1.DataSource = ds.Tables[0];
```

Executing a Stored Procedure with Input Parameters

► Example:

```
create procedure sp_Select_By_ContactId
@contactId int
as
Select id, ContactName, State
from
Contacts
where id = @contactId
```

Code:

```
SqlConnection con = new SqlConnection();
con.ConnectionString = @"Your Connection String Here";
SqlCommand cmd = new SqlCommand();
cmd.CommandType = CommandType.StoredProcedure;
cmd.CommandText = "sp_Select_By_ContactId";
cmd.Connection = con;

// create input parameter
SqlParameter inparm =
cmd.Parameters.Add("@Contactid", SqlDbType.Int);
inparm.Direction = ParameterDirection.Input;
inparm.Value = 2;

SqlDataAdapter ad = new SqlDataAdapter(cmd);
DataSet ds = new DataSet();
ad.Fill(ds);
dataGridView1.DataSource = ds.Tables[0];
```

Executing a Stored Procedure with Output Parameters

► Example:

```
create procedure sp_Orders_By_EmployeeId2
@employeeid int,
@ordercount int = 0 output
as
select orderid, customerid
from orders
where employeeid = @employeeid;
select @ordercount = count(*)
from orders
where employeeid = @employeeid
return @ordercount
```

To test your stored procedure

```
Declare @return_value int,
@ordercount int
Execute @return_value=sp_Orders_By_EmployeeId2
@employeeid=2,
@ordercount=@ordercount output
Select @ordercount as '@ordercount'
Select 'Return value' =@return_value
```

Parameters to Commands

As we take about the problem of SQL Injection and the problem of hacking, adding Parameters to Commands is another solution to avoid SQL Injection.

When working with data, you'll often want to filter results based on some criteria. Typically, this is done by accepting input from a user and using that input to form a SQL query. For example, a sales person may need to see all orders between specific dates. Another query might be to filter customers by city.

As you know, the SQL query assigned to a SqlCommand object is simply a string. So, if you want to filter a query, you could build the string dynamically, but you wouldn't want to. Here is a bad example of filtering a query.

```
// don't ever do this!  
SqlCommand cmd = new SqlCommand (  
    "select * from Customers where city = '" + inputCity + "'";
```

Don't ever build a query this way! The input variable, inputCity, is typically retrieved from a TextBox control on either a Windows form or a Web Page. Anything placed into that TextBox control will be put into inputCity and added to your SQL string. This situation invites a hacker to replace that string with something malicious. In the worst case, you could give full control of your computer away.

Instead of dynamically building a string, as shown in the bad example above, use parameters. Anything placed into a parameter will be treated as field data, not part of the SQL statement, which makes your application much more secure.

Using parameterized queries is a three step process:

1. Construct the SqlCommand command string with parameters.
2. Declare a SqlParameter object, assigning values as appropriate.
3. Assign the SqlParameter object to the SqlCommand object's Parameters property.

Example

```
class ParamDemo
{
    static void Main()
    {
        // conn and reader declared outside try
        // block for visibility in finally block
        SqlConnection conn = null;
        SqlDataReader reader = null;

        string inputName = "ahmed";

        try
        {
            // instantiate and open connection
            conn = new
                SqlConnection(@"your connection String here");
            conn.Open();

            // 1. declare command object with parameter
            SqlCommand cmd = new SqlCommand(
                "select * from StudentInfo where name = @Name", conn);

            // 2. define parameters used in command object
            SqlParameter param = new SqlParameter();
            param.ParameterName = "@Name";
            param.Value = inputName;

            // 3. add new parameter to command object
            cmd.Parameters.Add(param);

            // get data stream
            reader = cmd.ExecuteReader();

            // write each record
            while (reader.Read())
            {
                Console.WriteLine("{0}, {1}", reader["name"], reader["age"]);
            }
        }
        finally
        {
            // close reader
            if (reader != null)
            {
                reader.Close();
            }

            // close connection
            if (conn != null)
            {
                conn.Close();
            }
        }
    }
}
```

Example:

This example show to make update, delete and insert using perimetries

	id	name	age
▶	1	Ahmed	20
	2	44	2
	3	xx	40
	4	Ahmed	50
	5	5	5
	6	3	3
	7	7	7
*			

Form1

Three empty text boxes for input.

Buttons: Add, Delete, Update.

Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace WindowsApplication9
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        SqlDataAdapter sqlDa;
        SqlConnection con = null;
        DataSet dSet;

        private void Form1_Load(object sender, EventArgs e)
        {
            // instantiate and open connection
            con = new
                SqlConnection(@"Your connection String Here");
        }
    }
}
```

```
//Connecting database
con.Open();
//create sql adapter for the "StudentInfo" table

sqlDa = new SqlDataAdapter("select * from StudentInfo", con);
//create dataset instance

dSet = new DataSet();
//fill the dataset

sqlDa.Fill(dSet, "StudentInfo");
//bind the data grid with the data set

dataGridView1.DataSource = dSet.Tables["StudentInfo"];

//build select command

SqlCommand selCmd = new SqlCommand("select * from StudentInfo", con);
sqlDa.SelectCommand = selCmd;

}

private void button1_Click(object sender, EventArgs e)
{
    SqlCommand insCmd = new SqlCommand(
        "insert into StudentInfo (id,Name, Age) values(@Id,@Name, @Age)",
        con);

    insCmd.Parameters.Add("@id", SqlDbType.Int, 4, "id").Value =
        textBox3.Text;

    insCmd.Parameters.Add("@Name", SqlDbType.NChar, 10, "Name").Value =
        textBox1.Text;

    insCmd.Parameters.Add("@Age", SqlDbType.Int, 4, "Age").Value =
        textBox1.Text; ;

    insCmd.ExecuteNonQuery();
}

private void buttonDelete_Click(object sender, EventArgs e)
{
    try
    {
        SqlCommand delCmd = new SqlCommand(
            "delete from StudentInfo where id=@id", con);
        delCmd.Parameters.Add("@id", SqlDbType.Int, 4, "No").Value =
            textBox3.Text;

        sqlDa.DeleteCommand = delCmd;
        delCmd.ExecuteNonQuery();
        MessageBox.Show("Delete Occur");
    }
    catch (SqlException)
    {
    }
}

private void buttonUpdate_Click(object sender, EventArgs e)
{
}
```

```
//build update command
SqlCommand upCmd = new SqlCommand(
    "update StudentInfo set Name=@Name, Age=@Age where id=@id", con);

upCmd.Parameters.Add("@Name", SqlDbType.NChar, 10, "Name").Value =
textBox2.Text;

upCmd.Parameters.Add("@Age", SqlDbType.Int, 4,
    "Age").Value=textBox1.Text;

upCmd.Parameters.Add("@id", SqlDbType.Int, 4,
    "id").Value=textBox1.Text;

        sqlDa.UpdateCommand = upCmd;
        upCmd.ExecuteScalar();
    }
}
```

Searching and Filtering

DataView

- The DataView class to view only specific rows in a DataTable object using a filter
- You can also sort the rows viewed by a DataView.
- You can add, modify, and remove rows from a DataView, and those changes will also be applied to the underlying DataTable that the DataView reads from

Example

```
class Program
{
    static void Main(string[] args)
    {
        SqlConnection con = new SqlConnection();
        con.ConnectionString = @"Your connection String";
        SqlCommand cmd = con.CreateCommand();
        cmd.Connection = con;
        cmd.CommandText = "Select * from StudentInfo";
        cmd.CommandType = CommandType.Text;
        SqlDataAdapter adapter = new SqlDataAdapter();
        adapter.SelectCommand = cmd;
        DataSet dset = new DataSet();
        adapter.Fill(dset);
        con.Close();
        DataTable dtable = dset.Tables[0];

        string filterExpression = "name='ahmed'";
        string sortExpression = "id DESC, name DESC";

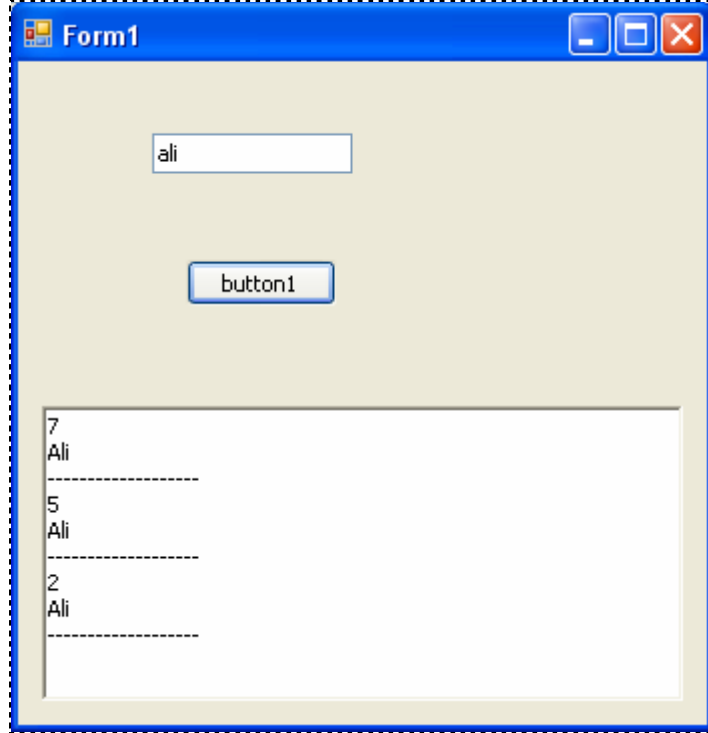
        DataViewRowState rowStateFilter =
        DataViewRowState.OriginalRows;

        DataView StudentDV = new DataView();
        StudentDV.Table = dtable;
        StudentDV.RowFilter = filterExpression;
        StudentDV.Sort = sortExpression;
        StudentDV.RowStateFilter = rowStateFilter;

        //display data that hosted in dataView
        foreach (DataRowView myDataViewRow in StudentDV)
        {
            for (int count = 0; count
<StudentDV.Table.Columns.Count-1; count++)
            {
                Console.WriteLine(myDataViewRow[count]);
            }
            Console.WriteLine("-----");
        }
    }
}
```

Example

Like the previous example but show result in win control.



```
private void button1_Click(object sender, EventArgs e)
{
    SqlConnection con = new SqlConnection();
    con.ConnectionString = @"Your connection String";
    SqlCommand cmd = con.CreateCommand();
    cmd.Connection = con;
    cmd.CommandText = "Select * from StudentInfo";
    cmd.CommandType = CommandType.Text;
    SqlDataAdapter adapter = new SqlDataAdapter();
    adapter.SelectCommand = cmd;
    DataSet dset = new DataSet();
    adapter.Fill(dset);
    con.Close();
    DataTable dtable = dset.Tables[0];

    //string filterExpression = "name='ahmed'";
    string filterExpression = "name=" + "'" + textBox1.Text + "'";
    string sortExpression = "id DESC, name DESC";

    DataRowState rowStateFilter = DataRowState.OriginalRows;

    DataView StudentDV = new DataView();
    StudentDV.Table = dtable;
    StudentDV.RowFilter = filterExpression;
    StudentDV.Sort = sortExpression;
    StudentDV.RowStateFilter = rowStateFilter;
}
```

```
//display data that hosted in dataView
foreach (DataRowView myDataRowView in StudentDV)
{
    for (int count = 0; count < StudentDV.Table.Columns.Count - 1; count++)
    {
        richTextBox1.Text += myDataRowView[count]+"\n";
    }
    richTextBox1.Text += "-----" + "\n";
}
}
```

Using the Default Sort Algorithm

If you want to sort the DataRowView objects in your DataView based on the primary key of your DataTable, you can use a shortcut. Instead of setting of your DataView to true. The Sort property of your DataView, you set the PrimaryKey property of your DataTable and then set the ApplyDefaultSort property

Change this in previous code to the following:-

```
dtable.PrimaryKey =
new DataColumn[]
{
    dtable.Columns["id"]
};
```

The next example sets the ApplyDefaultSort property of customersDV to true:

```
StudentDV.ApplyDefaultSort = true;
```

Table used through all examples

	id	name	age
▶	1	Ahmed	20
	2	Ali	2
	3	Ahmed	40
	4	Ahmed	50
	5	Ali	5
	6	Ahmed	3
	7	Ali	7
*	NULL	NULL	NULL

Example 3:

```
class Program
{
    static void Main(string[] args)
    {
        SqlConnection con = new SqlConnection();
        con.ConnectionString = @"Your Connection String";
        SqlCommand cmd = con.CreateCommand();
        cmd.Connection = con;
        cmd.CommandText = "Select * from StudentInfo";
        cmd.CommandType = CommandType.Text;
        SqlDataAdapter adapter = new SqlDataAdapter();
        adapter.SelectCommand = cmd;
        DataSet dset = new DataSet();
        adapter.Fill(dset);
        con.Close();
        DataTable dtable = dset.Tables[0];
        dtable.PrimaryKey = new DataColumn[] {
            dtable.Columns["id"]};

        string filterExpression = "name='ahmed'";

        DataRowState rowStateFilter =
            DataRowState.OriginalRows;

        DataView StudentDV = new DataView();
        StudentDV.Table = dtable;
        StudentDV.RowFilter = filterExpression;

        StudentDV.ApplyDefaultSort = true;

        StudentDV.RowStateFilter = rowStateFilter;

        //display data that hosted in dataView
        foreach (DataRowView myDataViewRow in StudentDV)
        {
            for (int count = 0; count <
StudentDV.Table.Columns.Count - 1; count++)
            {
                Console.WriteLine(myDataViewRow[count]);
            }
            Console.WriteLine("-----");
        }
    }
}
```

Performing Advanced Filtering

The RowFilter property of a DataView is similar to a WHERE clause in a SELECT statement. You can therefore use very powerful filter expressions in your DataView. For example, you can use AND, OR, NOT, IN, LIKE, comparison operators, arithmetic operators, wildcard characters (* and %), and aggregate functions.

Here's a simple example that uses the LIKE operator and the percent (%) wildcard character to filter rows with a CustomerName that starts with Fr:

```
String filterExpression = "CompanyName LIKE 'Fr%'";
customersDV.RowFilter = filterExpression;
```

Finding the Index of a DataRowView Using the Find () Method

- ▶ The Find () method returns the index of the DataRowView with the specified primary key in your DataView.
- ▶ The int returned by this method is the index of the DataRowView if found; otherwise -1 is returned.

To find the correct index, you must first set the Sort property of your .DataView to sort on the primary key

```
string sortExpression = "ID";
customersDV.Sort = sortExpression;
```

Then

```
int index = StudentDV.Find(4);
Console.WriteLine(index);
```

Finding DataRowView Objects Using the FindRows () Method

Goal: use the FindRows() method of StudentDV to find the DataRowView whose ID is 4

```
DataRowView[] StudentDRVs = StudentDV.FindRows(4);
foreach (DataRowView myDataRowView in StudentDRVs)
{
    for (int count = 0; count < StudentDV.Table.Columns.Count; count++)
```

```
{  
    Console.WriteLine(myDataRowView[count]);  
}  
Console.WriteLine("");  
}
```

Modifying, and Removing DataRowView Objects from a DataView

► Adding a DataRowView to a DataView

```
DataRowView studentDRV = studentDV.AddNew();  
studentDRV ["ID"] = 8;  
studentDRV ["Name"] = "Hossam";  
studentDRV ["age"] = 25;  
studentDRV.EndEdit();
```

Notice the use of the EndEdit () method of the studentDRV DataRowView to end the editing

► Modifying an Existing DataRowView

▼ To begin modifying an existing DataRowView in a DataView, you call the **BeginEdit ()** method of the DataRowView in your DataView.

```
studentDV [0].BeginEdit ();
```

You can then modify a DataColumn in the underlying DataRow through the DataRowView. The following example sets the CompanyName DataColumn to Widgets Inc.:

```
studentDV [0] ["Name"] = "Osama";
```

Once you've finished making your modifications, you call the EndEdit () method to make your modifications permanent in the underlying DataTable. For example:

```
studentDV [0].EndEdit ();
```

► Removing an Existing DataRowView

▼ To remove an existing DataRowView from a DataView, you can call the Delete () method of either the DataView or the DataRowView.

```
studentDV.Delete (1);
```

▼ to make committed to the removed row in DataTable use AcceptChanges () method of your DataTable. For example:

```
dTable.AcceptChanges ();
```

Note

RejectChanges () method of a DataTable to undo the deletions

Example: AddModifyAndRemoveDataRowViews

```
using System;
using System.Data;
using System.Data.SqlClient;

class AddModifyAndRemoveDataRowViews
{
    public static void DisplayDataRow(
        DataRow myDataRow,
        DataTable myDataTable
    )
    {
        Console.WriteLine("\nIn DisplayDataRow()");
        foreach (DataColumn myDataColumn in myDataTable.Columns)
        {
            Console.WriteLine(myDataColumn + " = " +
                myDataRow[myDataColumn]);
        }
    }

    public static void Main()
    {
        SqlConnection mySqlConnection =
            new SqlConnection(
                @"Data Source=.\SQLEXPRESS;AttachDbFilename=C:\Documents
and Settings\Administrator\My Documents\xxxxxxxxxx.mdf;Integrated
Security=True;Connect Timeout=30;User Instance=True"
            );
        SqlCommand mySqlCommand = mySqlConnection.CreateCommand();
        mySqlCommand.CommandText =
            "SELECT id, name, age " +
            "FROM studentinfo";
        SqlDataAdapter mySqlDataAdapter = new SqlDataAdapter();
        mySqlDataAdapter.SelectCommand = mySqlCommand;
        DataSet myDataSet = new DataSet();
        mySqlConnection.Open();
        mySqlDataAdapter.Fill(myDataSet, "studentinfo");
        mySqlConnection.Close();
        DataTable studentinfoDT = myDataSet.Tables["studentinfo"];

        // set up the filter expression
        string filterExpression = "name = 'ahmed'";

        // create a DataView object named studentDV
        DataView studentDV = new DataView();
        studentDV.Table = studentinfoDT;
        studentDV.RowFilter = filterExpression;

        // add a new DataRowView (adds a DataRow to the DataTable)
```

```
Console.WriteLine("\nCalling studentDV.AddNew()");
DataRowView customerDRV = studentDV.AddNew();
customerDRV["id"] = 8;
customerDRV["name"] = "J7 Company";
customerDRV["age"] = 10;
Console.WriteLine("customerDRV[\" id\"] = " +
    customerDRV["id"]);
Console.WriteLine("customerDRV[\" name\"] = " +
    customerDRV["name"]);
Console.WriteLine("customerDRV[\" age\"] = " +
    customerDRV["age"]);
Console.WriteLine("customerDRV.IsNew = " +
customerDRV.IsNew);
Console.WriteLine("customerDRV.IsEdit = " +
customerDRV.IsEdit);
customerDRV.EndEdit();
// get and display the underlying DataRow
DataRow customerDR = customerDRV.Row;
DisplayDataRow(customerDR, studentinfoDT);

// modify the name of customerDRV
Console.WriteLine("\nSetting studentDV[0][\" name\"] to
Widgets Inc.");
studentDV[0].BeginEdit();
studentDV[0]["name"] = "Widgets Inc.";
Console.WriteLine("studentDV[0][\" id\"] = " +
    studentDV[0]["id"]);
Console.WriteLine("studentDV[0][\" name\"] = " +
    studentDV[0]["name"]);
Console.WriteLine("studentDV[0].IsNew = " +
studentDV[0].IsNew);
Console.WriteLine("studentDV[0].IsEdit = " +
studentDV[0].IsEdit);
studentDV[0].EndEdit();

// display the underlying DataRow
DisplayDataRow(studentDV[0].Row, studentinfoDT);

// remove the second DataRowView from studentDV
Console.WriteLine("\nstudentDV[1][\" id\"] = " +
    studentDV[1]["id"]);
Console.WriteLine("\nCalling studentDV.Delete(1)");
studentDV.Delete(1);
Console.WriteLine("studentDV[1].IsNew = " +
studentDV[1].IsNew);
Console.WriteLine("studentDV[1].IsEdit = " +
studentDV[1].IsEdit);

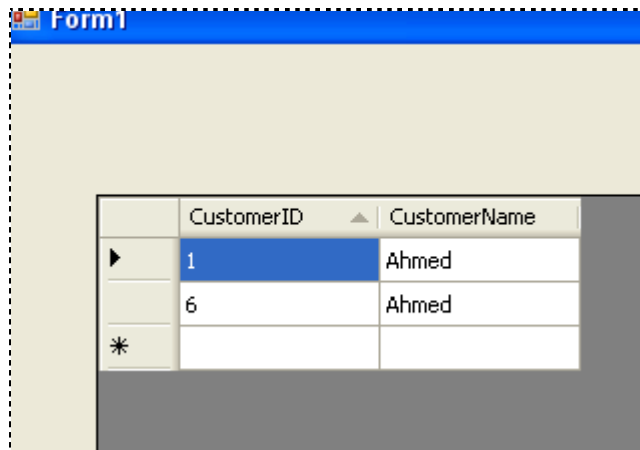
// remove the third DataRowView from studentDV
Console.WriteLine("\nstudentDV[2][\" id\"] = " +
    studentDV[2]["id"]);
Console.WriteLine("\nCalling studentDV[2].Delete()");
studentDV[2].Delete();

// call the AcceptChanges() method of studentinfoDT to
// make the deletes permanent in studentinfoDT
studentinfoDT.AcceptChanges();

// display the rows in the studentDV DataView object
Console.WriteLine("\nDataRowView objects in studentDV:\n");
```

```
foreach (DataRowView myDataRowView in studentDV)
{
    for (int count = 0; count <
studentDV.Table.Columns.Count; count++)
    {
        Console.WriteLine(myDataRowView[count]);
    }
    Console.WriteLine("");
}
}
```

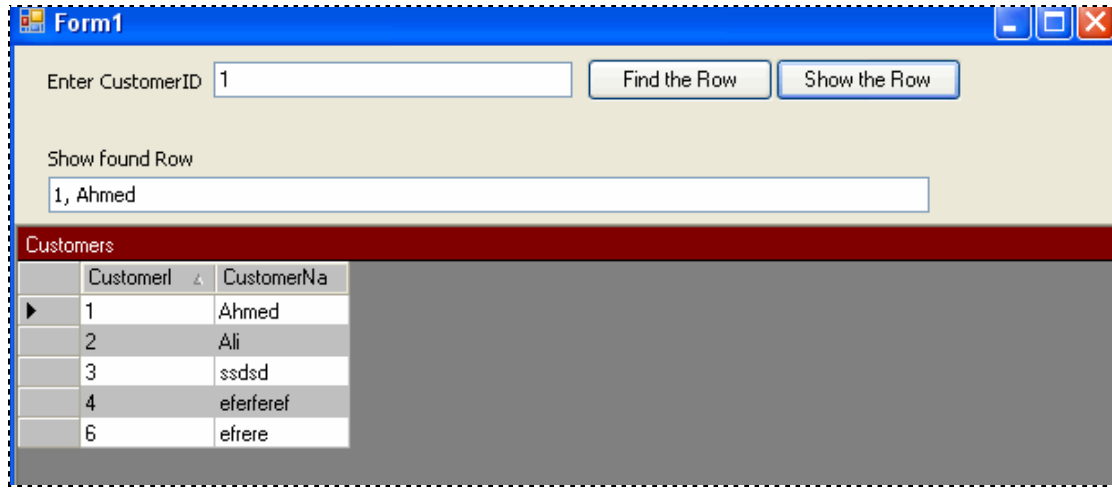
Example to display the filtering data in dataGridView



```
namespace WindowsApplication14
{
    public partial class Form1 : Form
    {
        DataSet ds;
        SqlConnection cn;
        SqlCommand cmd;
        SqlDataAdapter da;
        public Form1()
        {
            InitializeComponent();
            // Connect to Database and create Controls
            cn = new SqlConnection(@"Your Connection String Here");
            cmd = new SqlCommand("Select * from customers", cn);
            cmd.CommandType = CommandType.Text;
            da = new SqlDataAdapter(cmd);
            ds = new DataSet();
            da.Fill(ds);
            RunQuery();
        }
        public void RunQuery()
        {
            DataTable dtable = ds.Tables[0];
            string filterExpression = "CustomerName='ahmed'";
            dtable.PrimaryKey = new DataColumn[] {
                dtable.Columns["CustomerID"]};
            DataView customerDv = new DataView(dtable);
            customerDv.RowFilter = filterExpression;
        }
    }
}
```

```
customerDv.ApplyDefaultSort = true;
dataGridView1.DataSource = customerDv;
    }
}
}
```

Example



CustomerID	CustomerName
1	Ahmed
2	Ali
3	ssdsd
4	eferferf
6	efrere

Code

```
namespace WindowsSortView
{
    public partial class Form1 : Form
    {
        private SqlConnection cn;
        private SqlCommand cmd;
        private SqlDataAdapter da;
        private DataSet ds;

        public Form1()
        {
            InitializeComponent();
            // Connect to Database and create Controls
            cn = new SqlConnection(@"Data
Source=.\SQLEXPRESS;AttachDbFilename=C:\Documents and
Settings\Administrator\My Documents\ContactsDataBase.mdf;Integrated
Security=True;Connect Timeout=30;User Instance=True");
            cmd = new SqlCommand("Select * from customers", cn);
            cmd.CommandType = CommandType.Text;
            da = new SqlDataAdapter(cmd);
            ds = new DataSet();
            RunQuery();
        }

        #region Fill DataGrid with Data

        private void RunQuery()
        {
            try
            {
```

```
        DataColumn[] dcPk = new DataColumn[1];

        da.Fill(ds, "Customers");

        // Set Primary Key
        dcPk[0] =
ds.Tables["Customers"].Columns["CustomerID"];
        ds.Tables["Customers"].PrimaryKey = dcPk;

        // Set Default Sort
        ds.Tables[0].DefaultView.Sort = "CustomerID";
        DataGrid.DataSource = ds.Tables[0];
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());
        Debug.WriteLine(ex.ToString());
    }
}
#endregion

private void btnFind_Click(object sender, System.EventArgs e)
{
    try
    {
        int intRow;

        // Finds the row specified in txtFindArg
        intRow = ds.Tables[0].DefaultView.Find(txtFindArg.Text);
        Debug.WriteLine(intRow);
        if (intRow == -1)
        {
            MessageBox.Show("No PK matches " + txtFindArg.Text);
        }
        else
        {
            // Jump to the Row and select it
            DataGrid.CurrentRowIndex = intRow;
            DataGrid.Select(intRow);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.ToString());
    }
}

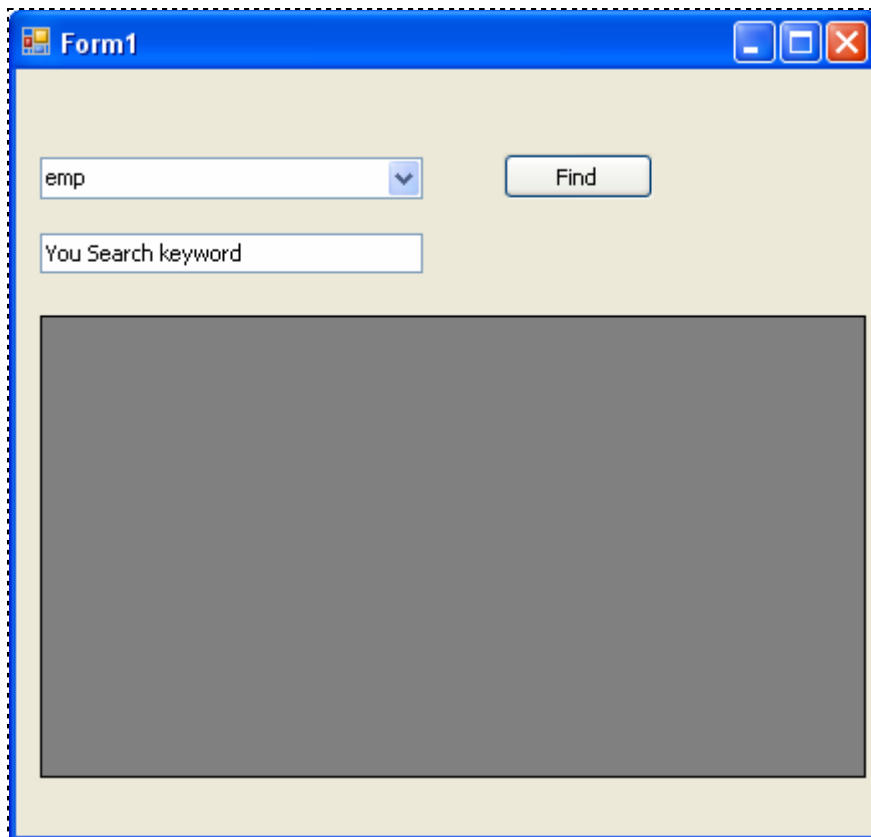
#region data need to access specific records
private void btnFindRow_Click(object sender, System.EventArgs e)
{
    try
    {
        DataRow drFound;

        // Find the Row specified in txtFindArg
    }
}
```

```
drFound = ds.Tables[0].Rows.Find(txtFindArg.Text);  
if (drFound == null)  
{  
    MessageBox.Show("No PK matches " + txtFindArg.Text);  
}  
else  
{  
    txtFoundRow.Text = drFound[0].ToString() + ", " +  
        drFound[1].ToString();  
    //drFound[2].ToString();  
}  
}  
catch (Exception ex)  
{  
    MessageBox.Show(ex.ToString());  
    Debug.WriteLine(ex.ToString());  
}  
}  
#endregion  
}  
}
```

Example: Search project

Project scenario: → we have 2 table and user choose the table to search on it



Code

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    #region Form1_Load
    private void Form1_Load(object sender, EventArgs e)
    {
        comboBox1.Items.Add("emp");
        comboBox1.Items.Add("users");
    }
    #endregion

    #region Method FillData return value type Dataset
    private DataSet FillData(string TableName)
    {
        SqlConnection con = new SqlConnection(@"Your con String");
        string SQL = string.Format("select * from {0}", TableName);
        SqlCommand cmd = new SqlCommand(SQL, con);
        SqlDataAdapter ad = new SqlDataAdapter(cmd);
        DataSet ds = new DataSet();
        ad.Fill(ds);
        return ds;
    }
    #endregion

    #region btnFind_Click
    private void btnFind_Click(object sender, EventArgs e)
    {
        switch (comboBox1.Text)
        {
            case "emp":
                DataSet ds = FillData(comboBox1.Text);
                //int x = Convert.ToInt32(textBox1.Text);
                string FilterExpression =
                string.Format("name='{0}'", textBox1.Text);
                string SortExpression = "id desc";

                DataView empDv = new DataView();
                empDv.Table = ds.Tables[0];
                empDv.RowFilter = FilterExpression;
                empDv.Sort = SortExpression;

                dataGridView1.DataSource = empDv;
                break;

            case "Users":
                DataSet ds = FillData(comboBox1.Text);
                //int x = Convert.ToInt32(textBox1.Text);
                string FilterExpression =
                string.Format("UserName='{0}'", textBox1.Text);
                string SortExpression = "Password desc";

                DataView empDv = new DataView();
                empDv.Table = ds.Tables[0];
                empDv.RowFilter = FilterExpression;
```

```
empDv.Sort = SortExpression;

dataGridView1.DataSource = empDv;
}

}
#endregion

private void comboBox1_SelectedIndexChanged(object sender,
EventArgs e)
{
    DataSet ds = FillData(comboBox1.Text);

    DataView empDv = new DataView();
    empDv.Table = ds.Tables[0];

    dataGridView1.DataSource = empDv;
}
}
```

Chapter 4

What is the Islam?

ISLAM and the AIM of LIFE

What is your purpose in life? What is the rationale behind our life? Why do we live in this life? These questions frequently intrigue people who try to find accurate answers.

People provide different answers to these questions. Some people believe the purpose of life is to accumulate wealth. But one may wonder: What is the purpose of life after one has collected colossal amounts of money? What then? What will the purpose be once money is gathered? If the purpose of life is to gain money, there will be no purpose after becoming wealthy. And in fact, here lies the problem of some disbelievers or misbelievers at some stage of their life, when collecting money is the target of their life. When they have collected the money they dreamt of, their life loses its purpose. They suffer from the panic of nothingness and they live in tension and restlessness.

Can Wealth Be an Aim?

We often hear of a millionaire committing suicide, sometimes, not the millionaire himself but his wife, son, or daughter. The question that poses itself is: Can wealth bring happiness to one's life? In most cases the answer is NO. Is the purpose of collecting wealth a standing purpose? As we know, the five-year old child does not look for wealth: a toy for him is equal to a million dollars. The eighteen-year old adolescent does not dream of wealth because he is busy with more important things. The ninety-year old man does not care about money; he is worried more about his health. This proves that wealth cannot be a standing purpose in all the stages of the individual's life.

Wealth can do little to bring happiness to a disbeliever, because he/she is not sure about his fate. A disbeliever does not know the purpose of life. And if he has a purpose, this purpose is doomed to be temporary or self destructive.

What is the use of wealth to a disbeliever if he feels scared of the end and skeptical of everything. A disbeliever may gain a lot of money, but will surely lose himself.

Worshipping Allah as an Aim

On the contrary, faith in Allah gives the believer the purpose of life that he needs. In Islam, the purpose of life is to worship Allah. The term "Worship" covers all acts of obedience to Allah.

The Islamic purpose of life is a standing purpose. The true Muslim sticks to this purpose throughout all the stages of his life, whether he is a child, adolescent, adult, or an old man.

Worshipping Allah makes life purposeful and meaningful, especially within the framework of Islam. According to Islam this worldly life is just a short stage of our life. Then there is the other life. The boundary between the first and second life is the death stage, which is a transitory stage to the second life. The type of life in the second stage a person deserves depends on his deeds in the first life. At the end of the death stage comes the day of judgment. On this day, Allah rewards or punishes people according to their deeds in the first life.

The First Life as an Examination

So, Islam looks at the first life as an examination of man. The death stage is similar to a rest period after the test, i. e. after the first life. The Day of Judgment is similar to the day of announcing the results of the examinees. The second life is the time when each examinee enjoys or suffers from the outcome of his behavior during the test period.

In Islam, the line of life is clear, simple, and logical: the first life, death, the Day of Judgment, and then the second life. With this clear line of life, the Muslim has a clear purpose in life. The Muslim knows he is created by Allah. Muslims know they are going to spend some years in this first life, during which they have to obey God, because God will question them and hold them responsible for their public or private deeds, because Allah knows about all the deeds of all people. The Muslim knows that his deeds in the first life will determine the type of second life they will live in. The Muslim knows that this first life is a very short one, one hundred years, more or less, whereas the second life is an eternal one.

The Eternity of the Second Life

The concept of the eternity of the second life has a tremendous effect on a Muslims during their first life, because Muslims believe that their first life determines the shape of their second life. In addition, this determines the shape of their second life and this determination will be through the Judgment of Allah, the All just and Almighty.

With this belief in the second life and the Day of Judgment, the Muslim's life becomes purposeful and meaningful. Moreover, the Muslim's standing purpose is to go to Paradise in the second life.

In other words, the Muslim's permanent purpose is to obey Allah, to submit to Allah, to carry out His orders, and to keep in continuous contact with Him through prayers (five times a day), through fasting (one month a year), through charity (as often as possible), and through pilgrimage (once in one's life).

The Need for a Permanent Purpose

Disbelievers have purposes in their lives such as collecting money and property, indulging in sex, eating, and dancing. But all these purposes are transient and passing ones. All these purposes come and go, go up and down. Money comes and goes. Health comes and goes. Sexual activities cannot continue forever. All these lusts for money, food and sex cannot answer the individual's questions: so what? Then What?

However, Islam saves Muslims from the trouble of asking the question, because Islam makes it clear, from the very beginning, that the permanent purpose of the Muslim in this life is to obey Allah in order to go to Paradise in the second life.

We should know that the only way for our salvation in this life and in the hereafter is to know our Lord who created us, believe in Him, and worship Him alone.

We should also know our Prophet whom Allah had sent to all mankind, believe in Him and follow Him. We should, know the religion of truth which our Lord has commanded us to believe in, and practice it ...

Those in search of truth

Who have an open mind and heart,

Islamic Education Foundation

Welcome You.

Objectives: -

To Convey the message of Islam

To Educate Muslims about Islam

To keep in close contact with new Muslims.

Activities:

Offering Courses and presenting lectures about Islam in several languages.

Teaching Islam and Arabic.

Teaching new Muslims to receive the Holy Quran.

Helping Non- Muslims embrace Islam and complete the required procedures