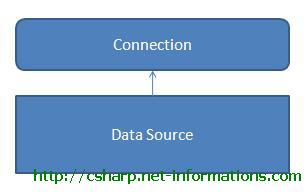
**C# ADO.NET Connection**

The **Connection** Object is a part of **ADO.NET** Data Provider and it is a unique session with the Data Source. The Connection Object is Handling the part of physical communication between the C# application and the Data Source.

The **Connection** Object connect to the specified Data Source and open a connection between the C# application and the Data Source, depends on the parameter specified in the **Connection String** . When the connection is established , SQL Commands will execute with the help of the Connection Object and retrieve or manipulate data in the Data Source.

Once the Database activity is over , **Connection** should be closed and release the Data Source resources .



In C# the type of the Connection is depend on which Data Source system you are working with. The following are the commonly used Connections from the C# applications.

**C# SQl Server Connection**

The SqlConnection Object is Handling the part of physical communication between the C# application and the **SQL Server Database** . An instance of the SqlConnection class in C# is supported the Data Provider for SQL Server Database. The SqlConnection instance takes Connection String as argument and pass the value to the Constructor statement.

**connetionString="Data Source=ServerName;**

**Initial Catalog=DatabaseName;User ID=UserName;Password=Password"**

**cnn = new SqlConnection(connetionString);**

When the connection is established , SQL Commands will execute with the help of the Connection Object and retrieve or manipulate the data in the database. Once the Database activities is over , Connection should be closed and release the Data Source resources .

**cnn.Close();**

The Close() method in SqlConnection Class is used to close the Database Connection. The Close method rolls back any pending transactions and releases the Connection from the SQL Server Database.

# C# ODBC Connection

An instance of the **OdbcConnection Class** in C# is supported the ODBC Data Provider. The OdbcConnection instance takes Connection String as argument and pass the value to the Constructor statement. When the connection is established between C# application and the Data Source the SQL Commands will execute with the help of the **Connection Object** and retrieve or manipulate data in the database.

**connetionString = "Driver={Microsoft Access Driver (\*.mdb)};**

**DBQ=yourdatabasename.mdb;";**

**cnn = new OdbcConnection(connetionString);**

Once the Database activities is over you should be closed the Connection and release the Data Source resources . The Close() method in **OdbcConnection Class** is used to close the Database Connection.

**cnn.Close();**

The Close method rolls back any pending transactions and releases the Connection from the Database connected by the **ODBC Data Provider** .

# C# ODBC Connection

An instance of the **OdbcConnection Class** in C# is supported the ODBC Data Provider. The OdbcConnection instance takes Connection String as argument and pass the value to the Constructor statement. When the connection is established between C# application and the Data Source the SQL Commands will execute with the help of the **Connection Object** and retrieve or manipulate data in the database.

**connetionString = "Driver={Microsoft Access Driver (\*.mdb)};**

**DBQ=yourdatabasename.mdb;";**

**cnn = new OdbcConnection(connetionString);**

Once the Database activities is over you should be closed the Connection and release the Data Source resources . The Close() method in **OdbcConnection Class** is used to close the Database Connection.

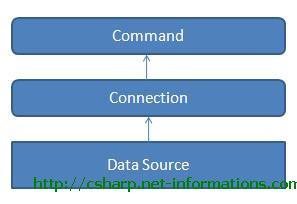
**cnn.Close();**

The Close method rolls back any pending transactions and releases the Connection from the Database connected by the **ODBC Data Provider** .

**C# ADO.NET Command**

The **Command Object** in ADO.NET executes SQL statements and Stored Procedures against the data source specified in the C# Connection Object. The Command Object requires an instance of a C# **Connection Object** for executing the SQL statements .

In order to retrieve a resultset or execute an SQL statement against a Data Source , first you have to create a **Connection Object** and open a connection to the Data Source specified in the connection string. Next step is to assign the open connection to the connection property of the **Command Object** . Then the Command Object can execute the SQL statements. After the execution of the SQl statement, the Command Object will return a result set . We can retrieve the result set using a **Data Reader** .



The **Command Object** has a property called **CommandText** , which contains a String value that represents the command that will be executed against the Data Source. When the **CommandType** property is set to StoredProcedure, the CommandText property should be set to the name of the stored procedure.

Click the following links to see some important built in methods uses in the Command Object to execute the SQL statements.

# C# ADO.NET SqlCommand - ExecuteNonQuery

The **ExecuteNonQuery()** is one of the most frequently used method in SqlCommand Object, and is used for executing statements that do not return result sets (ie. statements like insert data , update data etc.) .

**Command.ExecuteNonQuery();**

The ExecuteNonQuery() performs Data Definition tasks as well as Data Manipulation tasks also. The Data Definition tasks like creating Stored Procedures ,Views etc. perform by the **ExecuteNonQuery()** . Also Data Manipulation tasks like Insert , Update , Delete etc. also perform by the ExecuteNonQuery() of SqlCommand Object.

The following C# example shows how to use the method ExecuteNonQuery() through SqlCommand Object.

# C# ADO.NET SqlCommand - ExecuteScalar

The **ExecuteScalar()** in C# SqlCommand Object is using for retrieve a single value from Database after the execution of the SQL Statement. The ExecuteScalar() executes SQL statements as well as Stored Procedure and returned a scalar value on first column of first row in the returned Result Set.

If the **Result Set** contains more than one columns or rows , it will take only the value of first column of the first row, and all other values will ignore. If the Result Set is empty it will return a **NULL** reference.

It is very useful to use with aggregate functions like **Count(\*)** or **Sum()** etc. When compare to ExecuteReader() , ExecuteScalar() uses fewer System resources.

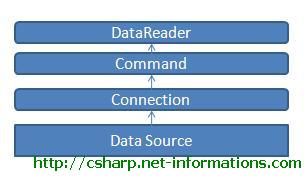
# C# ADO.NET SqlCommand - ExecuteReader

The **ExecuteReader()** in C# SqlCommand Object sends the SQL statements to the Connection Object and populate a SqlDataReader Object based on the SQL statement. When the ExecuteReader method in SqlCommand Object execute , it will instantiate a **SqlClient.SqlDataReader** Object.

The SqlDataReader Object is a stream-based , forward-only, read-only retrieval of query results from the Data Source, which do not update the data it contains. The **SqlDataReader** cannot be created directly from code, they can created only by calling the ExecuteReader method of a C# Command Object.

**C# ADO.NET DataReader**

**DataReader Object** in ADO.NET is a stream-based , forward-only, read-only retrieval of query results from the Data Sources , which do not update the data. The DataReader cannot be created directly from code, they can created only by calling the **ExecuteReader** method of a Command Object.



**SqlDataReader sqlReader = sqlCmd.ExecuteReader();**

The DataReader Object provides a connection oriented data access to the Data Sources. A **Connection Object** can contain only one DataReader at a time and the connection in the DataReader remains open, also it cannot be used for any other purpose while data is being accessed. When we started to read from a DataReader it should always be open and positioned prior to the first record. The **Read()** method in the DataReader is used to read the rows from DataReader and it always moves forward to a new valid row, if any row exist .

**DataReader.Raed();**

Usually we are using two types of DataReader in ADO.NET. They are **SqlDataReader** and the **OleDbDataReader** . The System.Data.SqlClient and System.Data.OleDb are containing these DataReaders respectively. From the following link you can see in details about these classes in C#.

# C# Multiple Result Sets

The **DataReader Object** is a stream-based , forward-only, read-only retrieval of query results from the Data Source, which do not update the data in the Data Source. When the **ExecuteReader** method in SqlCommand Object execute , it will instantiate a SqlClient.SqlDataReader Object in C#.

**SqlDataReader sqlReader = sqlCmd.ExecuteReader();**

In some situations we need to execute multiple SQL statements with the Command Object. In these types of situations the SqlDataReader returns multiple ResultSets also. In order to retrieving multiple ResultSets from SqlDataReader we use the **NextResult()** method of the SqlDataReader.

**sqlReader.NextResult();**

In the following source code you can see how to get multiple result sets from SqlDataReader() .

# C# Table Schema from SqlDataReader

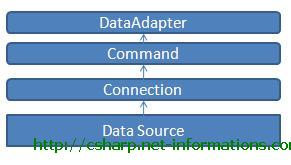
The **SqlDataReader** Object is a stream-based , forward-only, read-only retrieval of query results from the Data Source, which do not update the data. When the **ExecuteReader** method in C# SqlCommand Object execute , it will instantiate a SqlClient.SqlDataReader Object.

**SqlDataReader sqlReader = sqlCmd.ExecuteReader();**

While **SqlDataReader** is open, you can retrieve the schema information about the current result set using the GetSchemaTable method of SqlDataReader. **GetSchemaTable** returns a DataTable object, which populated with rows and columns that contain the schema information for the current result set.

**C# ADO.NET DataAdapter**

**DataAdapter** is a part of the ADO.NET Data Provider. DataAdapter provides the communication between the **Dataset** and the Datasource. We can use the DataAdapter in combination with the DataSet Object. DataAdapter provides this combination by mapping Fill method, which changes the data in the DataSet to match the data in the data source, and Update, which changes the data in the data source to match the data in the DataSet. That is, these two objects combine to enable both data access and data manipulation capabilities.



The **DataAdapter** can perform Select , Insert , Update and Delete SQL operations in the Data Source. The Insert , Update and Delete SQL operations , we are using the continuation of the Select command perform by the DataAdapter. The **SelectCommand** property of the DataAdapter is a Command Object that retrieves data from the data source. The **InsertCommand** , **UpdateCommand** , and **DeleteCommand** properties of the DataAdapter are Command objects that manage updates to the data in the data source according to modifications made to the data in the DataSet. From the following links describes how to use SqlDataAdapter and **OleDbDataAdapter** in detail.

# ExecuteReader and ExecuteNonQuery

**ExecuteReader :** ExecuteReader used for getting the query results as a DataReader object. It is readonly forward only retrieval of records and it uses select command to read through the table from the first to the last.

**SqlDataReader reader;**

**reader = Command.ExecuteReader();**

**while (reader.Read()) {**

**MessageBox.Show(reader.Item(0));**

**}**

**reader.Close();**

**ExecuteNonQuery :** ExecuteNonQuery used for executing queries that does not return any data. It is used to execute the sql statements like update, insert, delete etc. ExecuteNonQuery executes the command and returns the number of rows affected.

**int retValue = 0;**

**Command = new SqlCommand(Sql, Connection);**

**retValue = Command.ExecuteNonQuery();**