ADO.NET in c#

**Introduction about in ADO.NET** **:-** ADO.NET provides consistent access to data sources such as Microsoft SQL Server and XML, as well as to data sources.

Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, manipulate, and update the data that they contain.

🡺 ADO.NET includes .NET Framework data providers for connecting to a database, executing commands, and retrieving results.

🡺 The ADO.NET API is designed so it can be used from all programming languages that target the .NET Framework, such as Visual Basic, C#, J# and Visual C++.

🡺ADO.NET provides functionality to developers writing managed code similar to the functionality provided to native component object model (COM) developers by ActiveX Data Objects (ADO)

ADO.NET provides the following two models for accessing data from a Data Source:

1. Connection-Oriented Architecture
2. Disconnection Oriented Architecture

**1. Connection-Oriented Architecture**

In this case, we require a continuous connection with the Data Source for accessing data in it.

Here the “DataReader” class holds the data on client machines.

**2. Disconnection Oriented Architecture**

In this case, we do not require a continuous connection with the Data Source for accessing data.

Here the “DataSet” class holds the data in the client machines.

## Connected and Disconnected Data : -

*As mentioned earlier, ADO.NET supports two different programming environments: connected and disconnected.*

Connected : -

The connected environment provides forward-only, read-only access to data in the data source and the ability to execute commands against the data source. The connected classes provide a common way to work with connected data regardless of the underlying data source. They include Connection, Command, DataReader, Transaction, ParameterCollection, and Parameter classes.

# Disconnected :-

The Discoonected environment allows data retrieved from the data source to be manipulated and later reconciled with the data source. The disconnected classes provide a common way to work with disconnected data regardless of the underlying data source. They include the Dataset,DataTable,DataColumn,

DataRow,Constraint,DataRealtionship And Dataview Classesss

Finally, ADO.NET introduces the connected DataAdapter class to bridge the data source and disconnected classes by way of the connected classes. The DataAdapter is an abstraction of the connected classes that simplifies filling the disconnected DataSet or DataTable classes with data from the data source and updating the data source to reflect any changes made to the disconnected data.

Diagram

Description automatically generated

### Connected Classes :-

The following classes are used by ADO.NET to communicate directly with the data source:

**Connection :-**

Maintains information required to connect to the data source through a connection string. The connection string contains information such as the name of the data source and its location, and authorization credentials and settings. The Connection class has methods to open and close the connection, for transactions to be initiated on the connection, as well as control other properties of the connection.

**Command :-**

Executes SQL statements or stored procedures against the data source. The command class has a ParameterCollection object containing Parameter objects that allow parameterized SQL statements and stored procedures to be used against the data source.

**DataReader :-**

Provides connected forward-only, read-only access to the data source. It is optimized for speed. The DataReader is instantiated through a Command object.

**Parameter :-**

Allows parameters for both parameterized queries and stored procedures to be defined and set to appropriate values. The Parameter class is accessed through the ParametersCollection object within a Command object. It supports input and output parameters as well as return values from stored procedures.

**Transaction :-**

Allows transactions to be created on a connection so that multiple changes to data in a data source are treated as a single unit of work and either all committed or cancelled.

**DataAdapter:-**

Bridges the data source and the disconnected DataSet or DataTable classes. The DataAdapter wraps the connected classes to provide this functionality. It provides a method to retrieve data into a disconnected object and a method to reconcile modified data in the disconnected object with the data source. The CommandBuilder class can generate the logic to reconcile changes in simple situations; custom logic can be supplied to deal with complex situations and optimize performance.

### **Disconnected Classes :-**

The following ADO.NET classes allow data to be retrieved from the data set, examined and modified offline, and reconciled with the data source through the DataAdapter

**DataSet :-**

Provides a consistent way to deal with disconnected data completely independently of the data source. The DataSet is essentially an in-memory relational database, serving as a container for the DataTable, DataColumn, DataRow, Constraint, and DataRelation objects.

The XML format serializes and transports a DataSet. A DataSet can be accessed and manipulated either as XML or through the methods and properties of the DataSet interchangeably; the XmlDataDocument class represents and synchronizes the relational data within a DataSet object with the XML Document Object Model (DOM).

**DataTable** :-

Allows disconnected data to be examined and modified through a collection of DataColumn and DataRow classes. The DataTable allows constraints such as foreign keys and unique constraints to be defined using the Constraint class.

**DataColumn :-**

Corresponds to a column in a table. The DataColumn class stores metadata about the structure of the column that, together with constraints, defines the schema of the table. The DataColumn can also create expression columns based on other columns in the table.

**DataRow :-**

Corresponds to a row in a table and can examine and update data in the DataTable. The DataTable exposes DataRow objects through the DataRowCollection object it contains. The DataRow caches changes made to data contained in its columns, storing both original and current values. This allows changes to be cancelled or to be later reconciled with the data source.

**Constraint :-**

Allows constraints to be placed on data stored within a DataTable. Unique and foreign key constraints can be created to maintain data integrity.

**DataRelation :-**

Provides a way to indicate a relationship between different DataTable objects within a DataSet. The DataRelation relates columns in the parent and child tables allowing navigation between the parent and child tables and referential integrity to be enforced through cascading updates and deletes.

**DataView :-**

Allows data, once retrieved into a DataSet or DataTable, to be viewed in different ways. It allows data to be sorted based on column values and for a subset of the data to be filtered so that only rows matching specified criteria are displayed.

**Namespace :-**  Each provider exists in a namespace within the System.Data namespace, and consists of a number of classes.

* System.Data.SqlClient

**Disconnected Data access :-**

ADO.NET Components :-

There are two components of ADO.NET that you can use to access and manipulate data :-

* .NET Framework data providers
* The Dataset

Graphical user interface, diagram

Description automatically generated

# s 1) ADO.NET Framework Data Providers :-

# Data provider is used to connect to the database, execute commands and retrieve the record. It is lightweight component with better performance. It also allows us to place the data into DataSet to use it further in our application

* .NET Framework Data Provider for **SQL Server** :- It provides data access for Microsoft SQL Server. It requires the **System.Data.SqlClient**namespace.
* The NET Framework Data Providers are components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data.
* The Connection object provides connectivity to a data source.
* The Command object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information.
* The **DataReader** provides a high-performance stream of data from the data source. Finally, the **DataAdapter** provides the bridge between the DataSet object and the data source.
* The **DataAdapter** uses Command objects to execute SQL commands at the data source to both load the **DataSet** with data, and reconcile changes made to the data in the **DataSet** back to the data source.

1. **The Connection object :-**

Listed below are the common connection object methods we could work with :-

* **Open** - Opens the connection to our database
* **Close** - Closes the database connection
* **Dispose** - Releases the resources on the connection object. Used to force garbage collecting, ensuring no resources are being held after our connection is used. Incidentally, by using the Dispose method you automatically call the Close method as well.
* **State** - Tells you what type of connection state your object is in, often used to check whether your connection is still using any resources.

**Ex**. if (ConnectionObject.State == ConnectionState.Open)

1. **The Command Object :-**

* **ExecuteReader**  **: -**Simply executes the SQL query against the database, using the Read() method to traverse through data.
* **🡺** 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.
* **ExecuteNonQuery:-** Used whenever you work with SQL stored procedures with parameters.
* **🡺**  Use this operation **to execute any arbitrary SQL statements in SQL Server if you do not want any result set to be returned**. You can use this operation to create database objects or change data in a database by executing UPDATE, INSERT, or DELETE statements
* **ExecuteScalar :-** Returns a lightning fast single value as an object from your database
* **🡺** Use the ExecuteScalar method **to retrieve a single value (for example, an aggregate value) from a database**. This requires less code than using the ExecuteReader method, and then performing the operations that you need to generate the single value using the data returned by a SqlDataReader.

**Ex :-**object val = Command.ExecuteScalar();

1. **The DataReader Object :-**

* **Read :-**  Moves the record pointer to the first row, which allows the data to be read by column name or index position
* **Has Rows :-**  HasRows checks if any data exists, and is used instead of the Read method.
* **is Closed :-** A method that can determine if the DataReader is closed.
* **Close :-** Closes the DataReader.

1. **The DataAdapter**

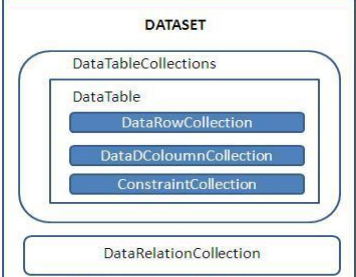
**🡪** Using an adapter, you can read, add, update, and delete records in a data source. To allow you to specify how each of these operations should occur, an adapter supports the following four properties:

* **SelectCommand :-** Reference to a command (SQL statement or stored procedure name) that retrieves rows from the data store
* **InsertCommand :-** Reference to a command for inserting rows into the data store.
* **UpdateCommand :-** Reference to a command for modifying rows in the data store.
* **DeleteCommand :-** Reference to a command for deleting rows from the data store.

**2) The DataSet :-**

🡪The ADO.NET DataSet contains DataTableCollection and their DataRelationCollection . It represents a collection of data retrieved from the Data Source.

* The DataSet contains a collection of one or more DataTable objects made up of rows and columns of data, as well as primary key, foreign key, constraint, and relation information about the data in the DataTable objects

****

**We Have to Give Connection with DataBase :-**

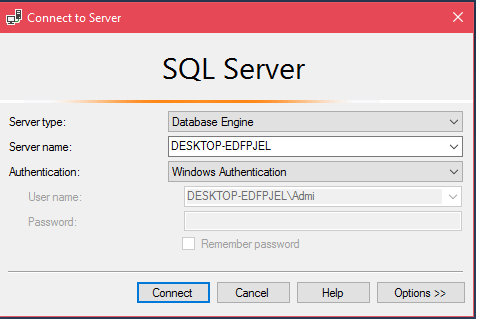
♦ The connection string can be put into the web.config file as shown below:-

**Graphical user interface, text

Description automatically generated**

# ADO.NET SQL Server Connection :-

# To connect with SQL Server, we must have it installed in our system. We are using Microsoft SQL Server Management Tool to connect with the SQL Server. We can use this tool to handle database. Now, follow the following steps to connect with SQL Server.

* Open Microsoft SQL Server Management Tool
* 
* You Have To Pass Your Database Creditals in the Connection strings in Web Cofig

# ADO.NET SqlConnection Class

# It is used to establish an open connection to the SQL Server database. It is a sealed class so that cannot be inherited. SqlConnection class uses SqlDataAdapter and SqlCommand classes together to increase performance when connecting to a Microsoft SQL Server database.

# Connection does not close explicitly even it goes out of scope. Therefore, you must explicitly close the connection by calling Close() method.

## **SqlConnection Signature :- System.Data.Common.DbConnection**

# Now, let's create an example that establishes a connection to the SQL Server. We have created a ****Student**** database and will use it to connect. Look at the following C# code

# Example :-  (SqlConnection connection = new SqlConnection(connectionString))

# **Using** block is used to close the connection automatically. We don't need to call close () method explicitly, ****using**** block do this for ours implicitly when the code exits the block.

# Graphical user interface, text, application Description automatically generated

# Text Description automatically generated

# ADO.NET SQLCommand class :-

# This class is used to store and execute SQL statement for SQL Server database. It is a sealed class so that cannot be inherited.

# SqlCommand(String, SqlConnection) :- It is used to initialize a new instance of the SqlCommand class. It takes two parameters, first is query string and second is connection string.

##### **Understanding the ADO.NET SqlCommand Object in C#:-**

Text

Description automatically generated with medium confidence

# ADO.NET SQLDataReader Class :- This class is used to read data from SQL Server database. It reads data in forward-only stream of rows from a SQL Server database. it is sealed class so that cannot be inherited. It inherits DbDataReader class and implements IDisposable interface.

# 🡪 To create a SqlDataReader instance, we must call the ExecuteReader method of the SqlCommand object.

# Diagram Description automatically generated with medium confidence

# ADO.NET DataSet :-

# The DataSet represents a subset of the database in memory. That means the ADO.NET DataSet is a collection of data tables that contains the relational data in memory in tabular format.

# ADO.NET DataAdapter :- The DataAdapter works as a bridge between a DataSet and a data source to retrieve data. DataAdapter is a class that represents a set of SQL commands and a database connection. It can be used to fill the DataSet and update the data source.

# Update(DataSet) :- It is used to call the respective INSERT, UPDATE, or DELETE statements.

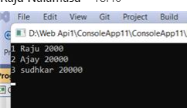
# Graphical user interface, text Description automatically generated

🡺 **Bring The Students data by using Ado.net with DbConnection :-**

 SqlConnection con1 = new SqlConnection("integrated security=yes;database=banking;data source=LAPTOP-A9L4R34G\\SQLEXPRESS");-🡪Connection With Database  
            string query1 = "select \* from emp"; --🡪 select statement   
            SqlCommand cmd1 = new SqlCommand(query1, con1);  
            try  
            {  
                con1.Open();  
                SqlDataReader sdr = cmd1.ExecuteReader();  
                while (sdr.Read())  
                {  
                    Console.WriteLine(sdr["eno"] + " " + sdr["ename"] + " " + sdr["salary"]); // Displaying Record    
                }  
                con1.Close();

           }  
            catch (Exception e)  
            {  
                Console.WriteLine("Error Generated. Details: " + e.ToString());

           }



🡺 Inserting the data student table :-

 SqlConnection con = new SqlConnection("integrated security=yes;database=banking;data source=LAPTOP-A9L4R34G\\SQLEXPRESS");  
            string query = "insert into Emp(eno,ename,salary) VALUES(@eno,@ename,@salary)";  
            SqlCommand cmd = new SqlCommand(query, con);  
           // Pass values to Parameters  
            cmd.Parameters.AddWithValue("@eno", "1");  
            cmd.Parameters.AddWithValue("@ename", "Raju");  
            cmd.Parameters.AddWithValue("@salary", "2000");  
            try  
            {  
                con.Open();  
                cmd.ExecuteNonQuery();  
                Console.WriteLine("Records Inserted Successfully");  
                con.Close();  
            }  
            catch (Exception e)  
            {  
                Console.WriteLine("Error Generated. Details: " + e.ToString());  
               
            }

 Graphical user interface, text, application

Description automatically generated Graphical user interface, application, table

Description automatically generated

* Update the student Details in the table :-

   SqlConnection cn = new SqlConnection("integrated security=yes;database=banking;data source=LAPTOP-A9L4R34G\\SQLEXPRESS");  
            string quer = "UPDATE Emp SET ename='Raju' ,salary='3000' WHERE eno = 2";  
            SqlCommand cm = new SqlCommand(quer, cn);  
            try  
            {  
                cn.Open();  
                cm.ExecuteScalar();  
                Console.WriteLine("Update Inserted Successfully");  
                cn.Close();  
            }  
            catch (Exception e)  
            {  
                Console.WriteLine("Error Generated. Details: " + e.ToString());  
            }  
            Console.ReadLine();

Graphical user interface, text, application

Description automatically generated Graphical user interface, application, table

Description automatically generated 🡺 **Updated Name and salary records**

🡺 **Deleting the one record** :-

SqlConnection con2 = new SqlConnection("integrated security=yes;database=banking;data source=LAPTOP-A9L4R34G\\SQLEXPRESS");  
            string query2 = "delete from emp where eno = '1'";  
            SqlCommand cmd2 = new SqlCommand(query2, con2);

           try  
            {  
                con2.Open();

               cmd2.ExecuteNonQuery();  
                Console.WriteLine("Record Deleted Successfully");  
                con2.Close();

           }  
            catch (Exception e)  
            {  
                Console.WriteLine("Error Generated. Details: " + e.ToString());

           }

Text

Description automatically generated Table

Description automatically generated

🡺**Deleted one record**