ADO.NET Introduction

It is a module of .Net Framework which is used to establish connection between application and data sources. Data sources can be such as SQL Server and XML. ADO.NET consists of classes that can be used to connect, retrieve, insert and delete data.

All the ADO.NET classes are located into **System.Data.dll** and integrated with XML classes located into **System.Xml.dll.**

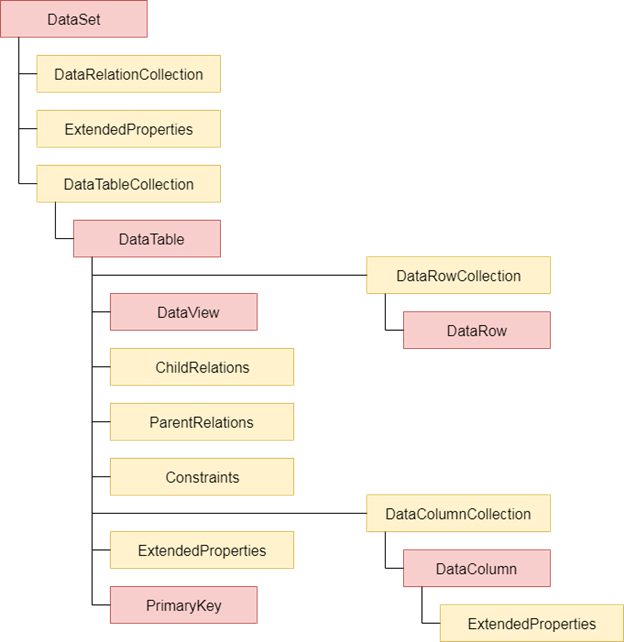
ADO.NET has two main components that are used for accessing and manipulating data are the .NET Framework data provider and the DataSet.

.NET Framework Data Providers

These are the components that are designed for data manipulation and fast access to data. It provides various objects such as **Connection, Command, DataReader and DataAdapter** that are used to perform database operations. We will have a detailed discussion about **Data Providers** in new topic.

The DataSet

It is used to access data independently from any data resource. DataSet contains a collection of one or more DataTable objects of data. The following diagram shows the relationship between .NET Framework data provider and DataSet.



**Fig:** ADO.NET Architecture

**ADO.NET (Active Data Objects for .NET)** is a framework in ASP.NET and C# that provides a set of classes and libraries for data access. It allows you to interact with data sources such as databases, XML files, and web services. ADO.NET provides a structured way to connect to, retrieve, manipulate, and update data from these sources. The architecture of ADO.NET involves several key components:

**Data Providers:**

ADO.NET includes data providers that are specific to different data sources. The two primary data providers are:

**a. SqlConnection/SqlCommand for SQL Server**: Used to connect to Microsoft SQL Server databases.

**b. OleDbConnection/OleDbCommand for OLE DB:** Used to connect to a wide range of data sources via OLE DB providers.

**c. OracleConnection/OracleCommand for Oracle:** Used to connect to Oracle databases.

**d. MySqlConnection/MySqlCommand for MySQL**: Used to connect to MySQL databases.

Each data provider typically includes classes for connections, commands, data readers, data adapters, and more, specific to the target data source.

**Connection:**

The SqlConnection (or similar) class is used to establish a connection to the data source. You specify the connection string, which contains information about the server, database, authentication, and other relevant settings.

**Command:**

The SqlCommand (or similar) class is used to execute SQL or other database commands against the data source. You can use it to execute queries, stored procedures, and other database operations.

**Data Reader:**

The SqlDataReader (or similar) class is used to retrieve data from the database after executing a query. It provides a forward-only, read-only, connected view of the data. Data readers are efficient for reading large datasets.

**Data Adapter:**

Data adapters, such as SqlDataAdapter, OleDbDataAdapter, and MySqlDataAdapter, act as bridges between your application and the database. They are used to fill datasets and update the database with changes made in the application.

**DataSet and DataTable:**

A DataSet is an in-memory representation of data retrieved from a data source. It can contain one or more DataTable objects, which are similar to database tables. Data can be stored, manipulated, and updated in a DataSet before applying changes back to the database using a data adapter.

**Data Binding:**

ADO.NET supports data binding, allowing you to bind data from a DataSet or DataTable to UI controls like DataGrid, GridView, or ListBox, making it easy to display and manipulate data in a web application.

**Transactions:**

ADO.NET provides transaction management through the System.Transactions namespace, allowing you to group multiple database operations into a single transaction to ensure atomicity.

**Error Handling:**

ADO.NET includes exception handling for managing errors that can occur during data access operations.

**Configuration and Connection Pooling:**

You can configure ADO.NET settings in the web.config or app.config files, including connection strings and various data provider-specific settings. ADO.NET also supports connection pooling, which improves performance by reusing database connections.

The architecture of ADO.NET is designed to be flexible and scalable, allowing you to work with various data sources and efficiently manage data access in your ASP.NET applications using C#.

Which one should we use DataReader or DataSet?

We should consider the following points to use DataSet.

* It caches data locally at our application, so we can manipulate it.
* It interacts with data dynamically such as binding to windows forms control.
* It allows performing processing on data without an open connection. It means it can work while connection is **disconnected.**

If we required some other functionality mentioned above, we can use **DataReader** to improve performance of our application.

DataReader does not perform in disconnected mode. It requires DataReader object to be **connected.**

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.

The .NET Framework provides the following data providers that we can use in our application.

|  |  |
| --- | --- |
| **.NET Framework data provider** | **Description** |
| .NET Framework Data Provider for SQL Server | It provides data access for Microsoft SQL Server. It requires the **System.Data.SqlClient**namespace. |
| .NET Framework Data Provider for OLE DB | It is used to connect with OLE DB. It requires the **System.Data.OleDb** namespace. |
| .NET Framework Data Provider for ODBC | It is used to connect to data sources by using ODBC. It requires the **System.Data.Odbc** namespace. |
| .NET Framework Data Provider for Oracle | It is used for Oracle data sources. It uses the **System.Data.OracleClient** namespace. |
| EntityClient Provider | It provides data access for Entity Data Model applications. It requires the **System.Data.EntityClient** namespace. |
| .NET Framework Data Provider for SQL Server Compact 4.0. | It provides data access for Microsoft SQL Server Compact 4.0. It requires the **System.Data.SqlServerCe** namespace. |

.NET Framework Data Providers Objects

Following are the core object of Data Providers.

|  |  |
| --- | --- |
| **Object** | **Description** |
| Connection | It is used to establish a connection to a specific data source. |
| Command | It is used to execute queries to perform database operations. |
| DataReader | It is used to read data from data source. The DbDataReader is a base class for all DataReader objects. |
| DataAdapter | It populates a DataSet and resolves updates with the data source. The base class for all DataAdapter objects is the DbDataAdapter class. |

.NET Framework Data Provider for SQL Server

Data provider for SQL Server is a lightweight component. It provides better performance because it directly access SQL Server without any middle connectivity layer. In early versions, it interacts with ODBC layer before connecting to the SQL Server that created performance issues.

The .NET Framework Data Provider for SQL Server classes is located in the **System.Data.SqlClient** namespace. We can include this namespace in our C# application by using the following syntax.

1. using System.Data.SqlClient;

This namespace contains the following important classes.

|  |  |
| --- | --- |
| **Class** | **Description** |
| SqlConnection | It is used to create SQL Server connection. This class cannot be inherited. |
| SqlCommand | It is used to execute database queries. This class cannot be inherited. |
| SqlDataAdapter | It represents a set of data commands and a database connection that are used to fill the DataSet. This class cannot be inherited. |
| SqlDataReader | It is used to read rows from a SQL Server database. This class cannot be inherited. |
| SqlException | This class is used to throw SQL exceptions. It throws an exception when an error is occurred. This class cannot be inherited. |

.NET Framework Data Provider for Oracle

It is used to connect with Oracle database through Oracle client. The data provider supports Oracle client software version 8.1.7 or a later version. This data provider supports both local and distributed transactions.

Oracle Data Provider classes are located into **System.Data.OracleClient** namespace. We must use both **System.Data.OracleClient** and **System.data** to connect our application with the Oracle database.

1. using System.Data;
2. using System.Data.OracleClient;

Which .NET Framework Data Provider is better

Selection of data provider is depends on the design and data source of our application. Choice of optimum .NET Framework data provider can improve the performance, capability and integrity of our application. The following table demonstrates advantages and disadvantages of data provider.

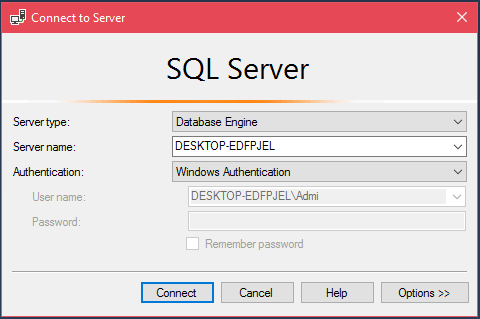
|  |  |
| --- | --- |
| **Data Provider** | **Note** |
| .NET Framework Data Provider for SQL Server | It is good for middle-tier applications, single-tier applications that use Microsoft SQL Server. |
| .NET Framework Data Provider for OLE DB | It is good for single-tier applications that use Microsoft Access databases. |
| .NET Framework Data Provider for ODBC | It is good for middle and single-tier applications that use ODBC data sources. |
| .NET Framework Data Provider for Oracle | It is good for middle and single-tier applications that use Oracle data sources. |
|  |  |

ADO.NET SQL Server Connection

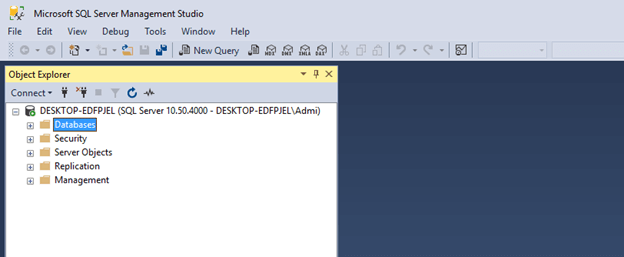
To connect with SQL Server, we must have it installed in our system. We are using Microsoft SQL Server Management Tool to connect with the SQL Server. We can use this tool to handle database. Now, follow the following steps to connect with SQL Server.

1. **Open Microsoft SQL Server Management Tool**

It will prompt for database connection. Provide the server name and authentication.

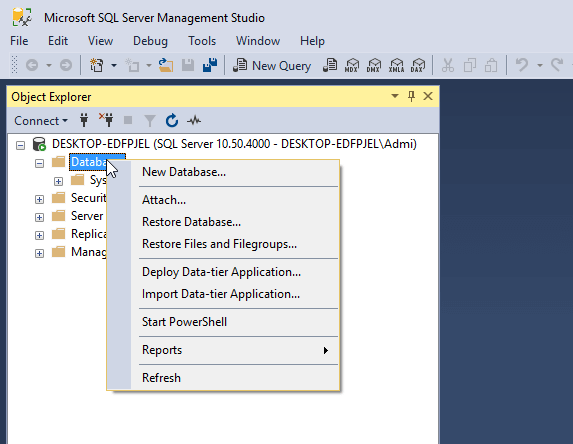


After successful connection, it displays the following window.

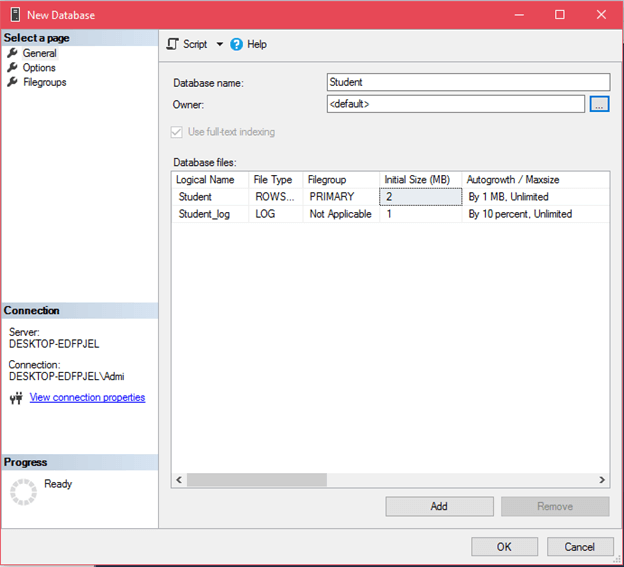


1. **Creating Database**

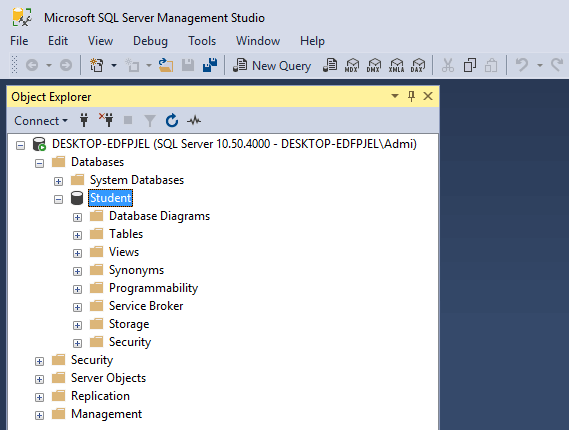
Now, create database by selecting database option then right click on it. It pops up an option menu and provides couple of options.



Click on the **New Database** then it will ask for the database name. Here, we have created a **Student** database.



Click on the Ok button then it will create a database that we can see in the left window of the below screenshot.



1. **Establish connection and create a table**

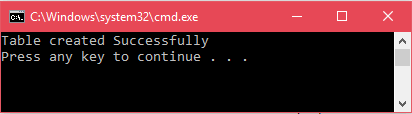
After creating database, now, let's create a table by using the following C# code. In this source code, we are using created **student** database to connect.

In visual studio 2017, we created a .NET console application project that contains the following C# code.

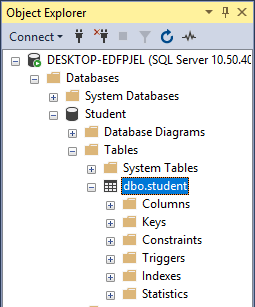
**// Program.cs**

* 1. **using** System;
  2. **using** System.Data.SqlClient;
  3. **namespace** AdoNetConsoleApplication
  4. {
  5. **class** Program
  6. {
  7. **static** **void** Main(**string**[] args)
  8. {
  9. **new** Program().CreateTable();
  10. }
  11. **public** **void** CreateTable()
  12. {
  13. SqlConnection con = **null**;
  14. **try**
  15. {
  16. // Creating Connection
  17. con = **new** SqlConnection("data source=.; database=student; integrated security=SSPI");
  18. // writing sql query
  19. SqlCommand cm = **new** SqlCommand("create table student(id **int** not **null**,
  20. name varchar(100), email varchar(50), join\_date date)", con);
  21. // Opening Connection
  22. con.Open();
  23. // Executing the SQL query
  24. cm.ExecuteNonQuery();
  25. // Displaying a message
  26. Console.WriteLine("Table created Successfully");
  27. }
  28. **catch** (Exception e)
  29. {
  30. Console.WriteLine("OOPs, something went wrong."+e);
  31. }
  32. // Closing the connection
  33. **finally**
  34. {
  35. con.Close();
  36. }
  37. }
  38. }
  39. }

Execute this code using **Ctrl+F5**. After executing, it displays a message to the console as below.



We can see the created table in Microsoft SQL Server Management Studio also. It shows the created table as shown below.



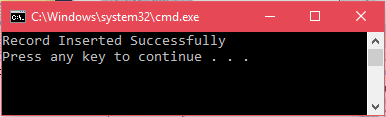
See, we have a table here. Initially, this table is empty so we need to insert data into it.

1. **Insert Data into the Table**

**// Program.cs**

* 1. **using** System;
  2. **using** System.Data.SqlClient;
  3. **namespace** AdoNetConsoleApplication
  4. {
  5. **class** Program
  6. {
  7. **static** **void** Main(**string**[] args)
  8. {
  9. **new** Program().CreateTable();
  10. }
  11. **public** **void** CreateTable()
  12. {
  13. SqlConnection con = **null**;
  14. **try**
  15. {
  16. // Creating Connection
  17. con = **new** SqlConnection("data source=.; database=student; integrated security=SSPI");
  18. // writing sql query
  19. SqlCommand cm = **new** SqlCommand("insert into student
  20. (id, name, email, join\_date)values('101','Ronald Trump','ronald@example.com','1/12/2017')", con);
  21. // Opening Connection
  22. con.Open();
  23. // Executing the SQL query
  24. cm.ExecuteNonQuery();
  25. // Displaying a message
  26. Console.WriteLine("Record Inserted Successfully");
  27. }
  28. **catch** (Exception e)
  29. {
  30. Console.WriteLine("OOPs, something went wrong."+e);
  31. }
  32. // Closing the connection
  33. **finally**
  34. {
  35. con.Close();
  36. }
  37. }
  38. }
  39. }

Execute this code by using **Ctrl+F5** and it will display the following output.



1. **Retrieve Record**

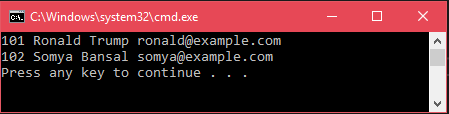
Here, we will retrieve the inserted data. Look at the following C# code.

**// Program.cs**

* 1. **using** System;
  2. **using** System.Data.SqlClient;
  3. **namespace** AdoNetConsoleApplication
  4. {
  5. **class** Program
  6. {
  7. **static** **void** Main(**string**[] args)
  8. {
  9. **new** Program().CreateTable();
  10. }
  11. **public** **void** CreateTable()
  12. {
  13. SqlConnection con = **null**;
  14. **try**
  15. {
  16. // Creating Connection
  17. con = **new** SqlConnection("data source=.; database=student; integrated security=SSPI");
  18. // writing sql query
  19. SqlCommand cm = **new** SqlCommand("Select \* from student", con);
  20. // Opening Connection
  21. con.Open();
  22. // Executing the SQL query
  23. SqlDataReader sdr = cm.ExecuteReader();
  24. // Iterating Data
  25. **while** (sdr.Read())
  26. {
  27. Console.WriteLine(sdr["id"] + " " + sdr["name"]+" "+sdr["email"]); // Displaying Record
  28. }
  29. }
  30. **catch** (Exception e)
  31. {
  32. Console.WriteLine("OOPs, something went wrong.\n"+e);
  33. }
  34. // Closing the connection
  35. **finally**
  36. {
  37. con.Close();
  38. }
  39. }
  40. }
  41. }

Execute this code by **Ctrl+F5** and it will produce the following result. This displays two records, one we inserted manually.

Output:



1. **Deleting Record**

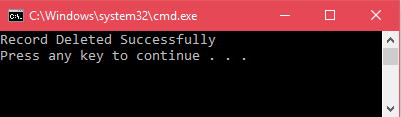
This time **student** table contains two records. The following C# code delete one row from the table.

**// Program.cs**

1. **using** System;
2. **using** System.Data.SqlClient;
3. **namespace** AdoNetConsoleApplication
4. {
5. **class** Program
6. {
7. **static** **void** Main(**string**[] args)
8. {
9. **new** Program().CreateTable();
10. }
11. **public** **void** CreateTable()
12. {
13. SqlConnection con = **null**;
14. **try**
15. {
16. // Creating Connection
17. con = **new** SqlConnection("data source=.; database=student; integrated security=SSPI");
18. // writing sql query
19. SqlCommand cm = **new** SqlCommand("delete from student where id = '101'", con);
20. // Opening Connection
21. con.Open();
22. // Executing the SQL query
23. cm.ExecuteNonQuery();
24. Console.WriteLine("Record Deleted Successfully");
25. }
26. **catch** (Exception e)
27. {
28. Console.WriteLine("OOPs, something went wrong.\n"+e);
29. }
30. // Closing the connection
31. **finally**
32. {
33. con.Close();
34. }
35. }
36. }
37. }

Output:

It displays the following output.



We can verify it by retrieving data back by using SqlDataReader.