Chapter -5

ADO.NET is a set of classes (a framework) to interact with data sources such as databases and XML files. ADO is the acronym for ActiveX Data Objects. It allows us to connect to underlying data or databases. It has classes and methods to retrieve and manipulate data.

The following are a few of the .NET applications that use ADO.NET to connect to a database, execute commands and retrieve data from the database.

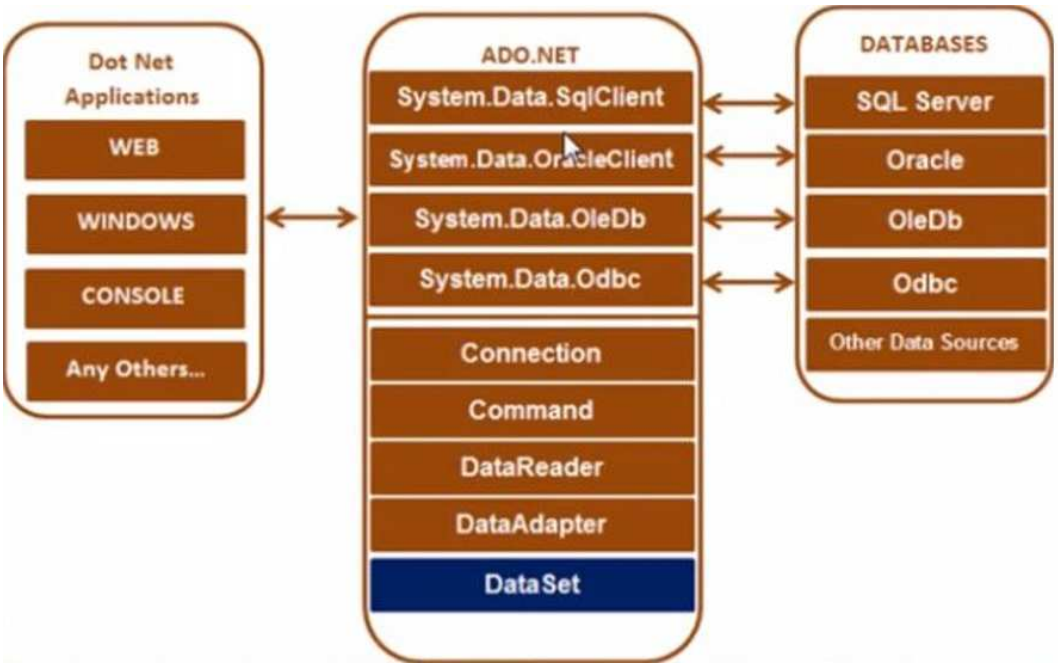
- ASP.NET Web Applications
- Console Applications
- Windows Applications.

Various Connection Architectures

There are following two types of connection architectures:

1. **Connected architecture:** the application remains connected with the database throughout the processing.
2. **Disconnected architecture:** the application automatically connects/disconnects during the processing. The application uses temporary data on the application side called a DataSet.

Understanding ADO.NET and it's class library



In this diagram, we can see that there are various types of applications (Web Application, Console Application, Windows Application and so on) that use ADO.NET to connect to databases (SQL Server, Oracle, OleDb, ODBC, XML files and so on).

Important Classes in ADO.NET

We can also observe various classes in the preceding diagram. They are:

1. Connection Class
2. Command Class
3. DataReader Class
4. DataAdaptor Class
5. DataSet.Class

1. Connection Class

In ADO.NET, we use these connection classes to connect to the database. These connection classes also manage transactions and connection pooling.

ADO.NET connection is an object that provides database connectivity and the entry point to a database. When the connection of an object is instantiated, the constructor takes a connection string that contains the information about the database server, server type, database name, connection type, and database user credentials. Once the connection string is passed and the connection object is created, you can establish a connection with the database. A connection string is usually stored in the web.config file or app.config file of an application.

What namespace or provider is used for connection class?

ADO.NET provides connection to multiple providers. Each provider has a functionality to connect with different database. Here is a list of data providers in ADO.NET and their purpose.

- Data Provider for SQL Server (System.Data.SqlClient).
- Data Provider for MS ACCESS (System.Data.OleDb).
- Data Provider for MYSQL (System.Data.Odbc).
- Data Provider for ORACLE (System.Data.OracleClient).

2. Command Class

The Command class provides methods for storing and executing SQL statements and Stored Procedures. The following are the various commands that are executed by the Command Class.

- **ExecuteReader:** Returns data to the client as rows. This would typically be an SQL select statement or a Stored Procedure that contains one or more select statements. This method returns a DataReader object that can be used to fill a DataTable object or used directly for printing reports and so forth.
- **ExecuteNonQuery:** Executes a command that changes the data in the database, such as an update, delete, or insert statement, or a Stored Procedure that contains one or more of these statements. This method returns an integer that is the number of rows affected by the query.
- **ExecuteScalar:** This method only returns a single value. This kind of query returns a count of rows or a calculated value.
- **ExecuteXMLReader:** (SqlClient classes only) Obtains data from an SQL Server 2000 database using an XML stream. Returns an XML Reader object.

3. DataReader Class

The DataReader is used to retrieve data. It is used in conjunction with the Command class to execute an SQL Select statement and then access the returned rows.

ADO.NET DataReader object is used for accessing data from the data store and is one of the two mechanisms that ADO.NET provides. As we will remember DataReader object provides a read only, forward only, high performance mechanism to retrieve data from a data store as a data stream, while staying connected with the data source. The DataReader is restricted but highly optimized. The .NET framework provides data providers for SQL Server native OLE DB providers and native ODBC drivers,

- SqlDataReader
- OleDbDataReader
- OdbcDataReader

You can use the ADO.NET DataReader to retrieve a read-only, forward-only stream of data from a database. Using the DataReader can increase application performance and reduce system overhead because only one row at a time is ever in memory. After creating an instance of the Command object, you create a DataReader by calling Command.ExecuteReader to retrieve rows from a data source, as shown in the following example

SqlDataReader myReader = myCommand.ExecuteReader();

```
namespace ConsoleApp1
{
   class Program
   {
      static void Main(string[] args)
      {
         Student stu = new Student();
         List<StudentDetails> lst = stu.GetAllStudent();
         foreach (StudentDetails item in lst)
         {
            Console.WriteLine(item.Firstname + "  " + item.Lastname);
         }
         Console.ReadLine();
      }
   }
   public class StudentDetails
   {
      public int Id { get; set; }
      public string Firstname { get; set; }
      public string Lastname { get; set; }
      public string Email { get; set; }
   }
   public class Student
   {
      public List<StudentDetails> GetAllStudent()
      {
         List<StudentDetails> lst = new List<StudentDetails>();

         SqlConnection con = new SqlConnection(@"Data Source=(LocalDB)\MSSQLLocalDB;
Database=SamriddhiDBA; Integrated Security=true;");
         SqlCommand cmd = new SqlCommand("select *from tblStudent", con);

         SqlDataReader dr = null;
         con.Open();
         dr = cmd.ExecuteReader();
         while(dr.Read())
         {
            lst.Add(new StudentDetails() { Id = (int)dr["Id"], Firstname = (string)dr["Firstname"],
Lastname = (string)dr["Lastname"], Email = (string)dr["Email"] });
         }
         con.Close();
         dr.Close();
         return lst;
      }
   }}
```

4. DataAdapter Class

The DataAdapter is used to connect DataSets to databases. The DataAdapter is most useful when using data-bound controls in Windows Forms, but it can also be used to provide an easy way to manage the connection between your application and the underlying database tables, views and Stored Procedures

Example:

```
SqlConnection con = new SqlConnection(“");
SqlCommand cmd = new SqlCommand("select *from tblStudent", con);

DataTable dt = new DataTable();
SqlDataAdapter da = new SqlDataAdapter(cmd);
da.Fill(dt);
```

5. DataSet Class

The DataSet is the heart of ADO.NET. The DataSet is essentially a collection of DataTable objects. In turn each object contains a collection of DataColumn and DataRow objects. The DataSet also contains a Relations collection that can be used to define relations among Data Table Objects.

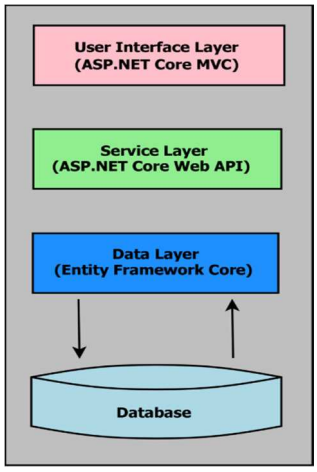
Example:

```
SqlConnection con = new SqlConnection(“");
SqlCommand cmd = new SqlCommand("select *from tblStudent", con);

DataSet ds = new DataSet();
SqlDataAdapter da = new SqlDataAdapter(cmd);
da.Fill(ds);
```

Entity Framework Core

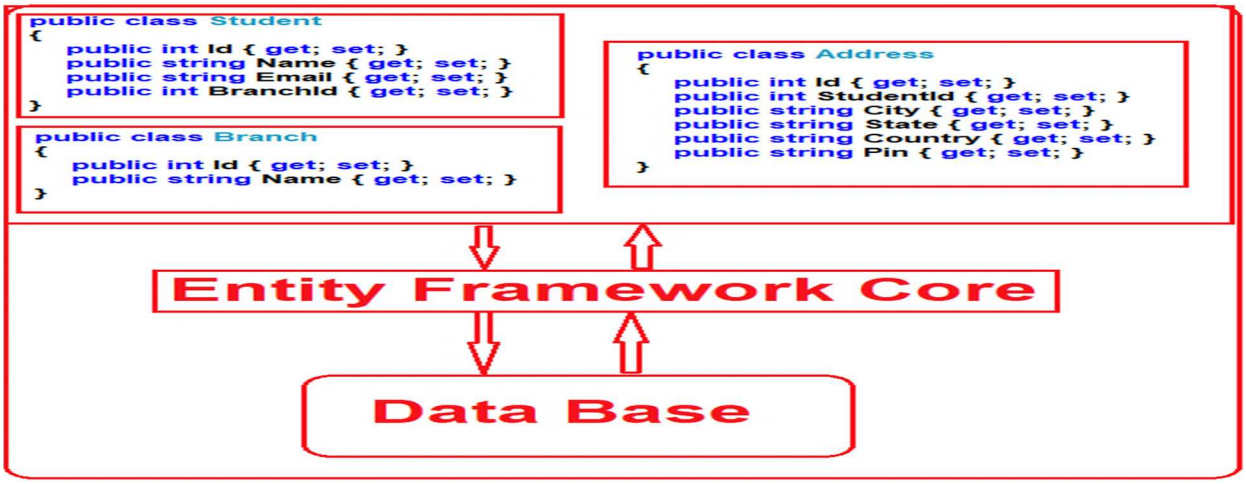
Entity Framework (EF) Core is an ORM (Object-Relational Mapper) Framework for data access in .Net. It was released along with .NET Core and is an extensible, lightweight, Open Source, and cross-platform version of Entity Framework data access technology. It works on multiple operating systems like Windows, Mac, and Linus.



ORM

The term ORM stands for Object-Relational Mapper and it automatically creates classes based on database tables and the vice versa is also true. That is, it can also generate the necessary SQL to create the database based on the classes.

As a developer, we mostly work with the application business objects and the ORM Framework generates the necessary SQL (to perform the CRUD operation) that the underlying database can understand. So, in simple words, we can say that the ORM Framework eliminates the need for most of the data access code that as a developer we generally write. The ORM Framework sits between our application code and the Database.



EF Core Development Approaches:

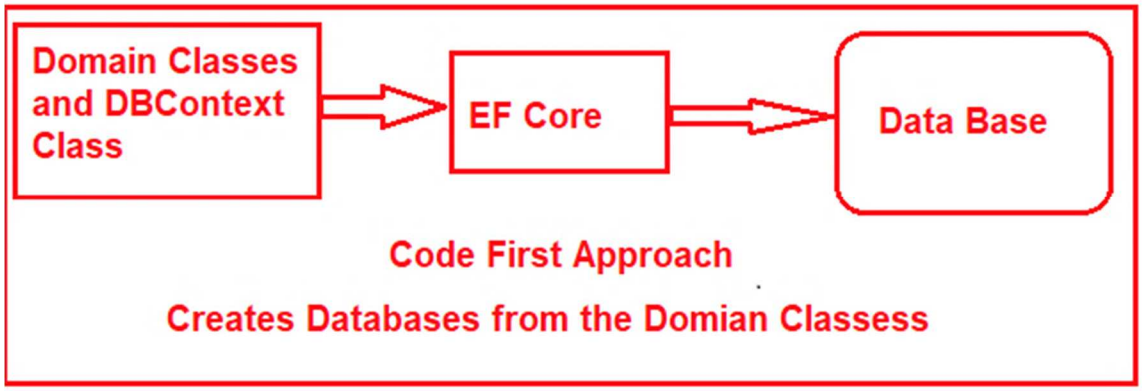
Entity Framework Core supports two development approaches. They are as follows:

1. **Code-First Approach**
2. **Database-First Approach**

EF Core mainly targets the code-first approach and provides little support for the database-first approach at the moment.

EF Core Code First Approach:

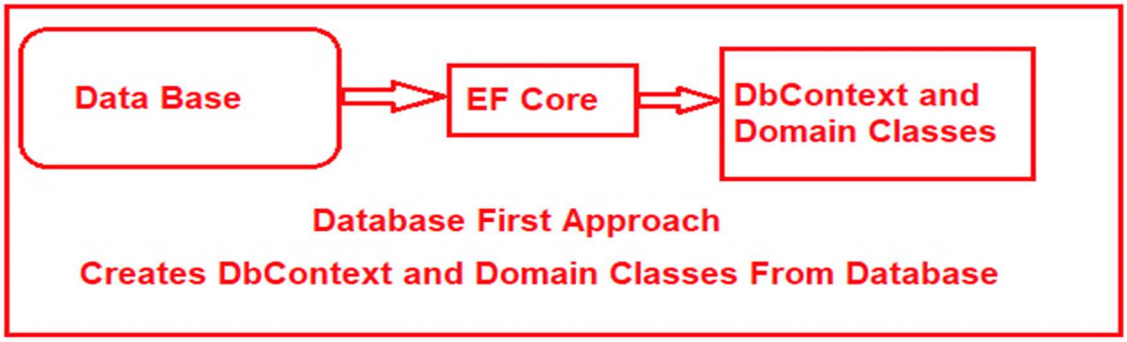
In the EF Core Code First Approach, first, we need to create our application domain classes such as Student, Branch, Address, etc. and a special class that derives from Entity Framework DbContext class. Then based on the application domain classes and DBContext class, the EF Core creates the database and related tables.



Out of the box, in the code-first approach, the EF Core API creates the database and tables using migration based on the default conventions and configuration. If you want then you can also change the default conventions used to create the database and its related tables. This approach is useful in Domain-Driven Design (DDD).

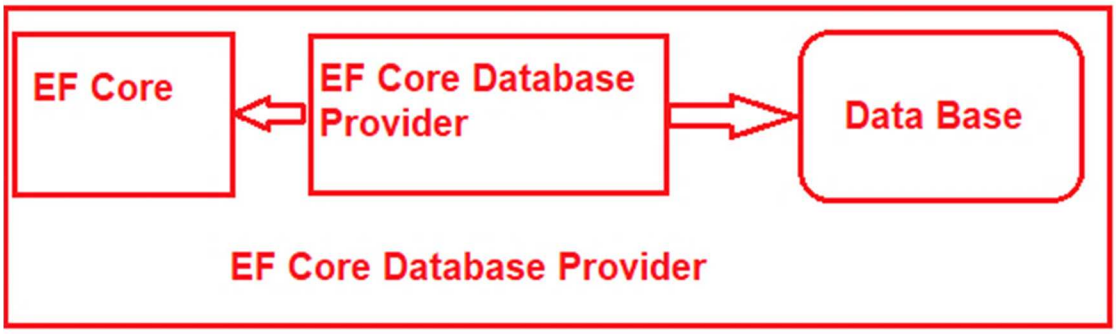
EF Core Database First Approach:

If you have an existing database and database tables are already there, then you need to use the EF Core Database First Approach. In the database-first approach, the EF Core creates the DBContext and Domain classes based on the existing database schema using EF Core Command.



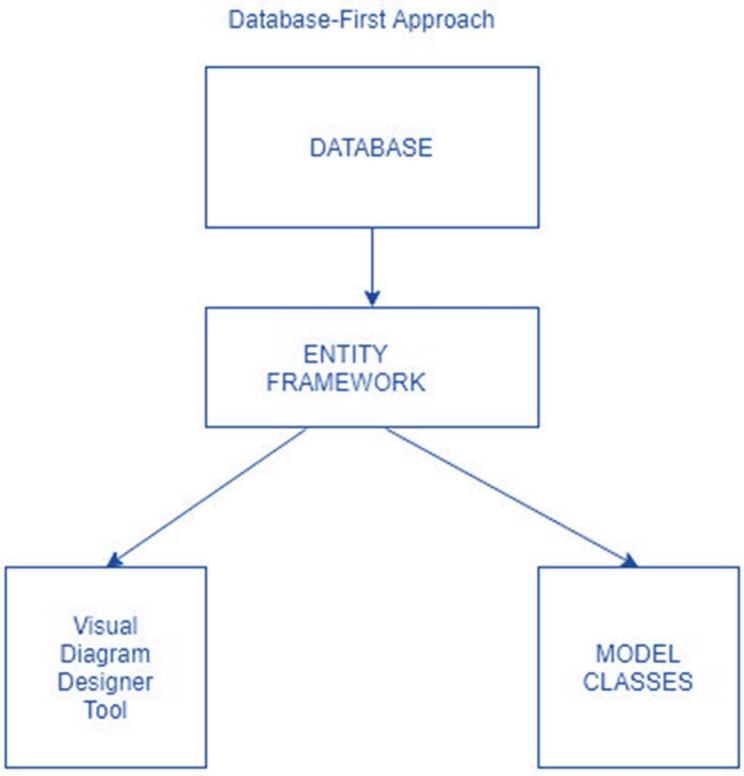
EF Core Database Providers:

The EF Core supports both relational and non-relational databases and this is possible due to database providers. The database providers are available as NuGet packages. The Database Provider sits between the EF Core and the database it supports.



Data Model and Data Context

Data models are **visual representations of an enterprise's data elements and the connections between them**. Entity data model (EDM) refers to a set of concepts that describe data structure, regardless of its stored form. This model uses three key concepts to describe data structure: entity type, association type and property. EDM supports a set of primitive data types that define properties in a conceptual model.



DbContext class is the brain of **Entity Framework Core** and does all the communications with the database.