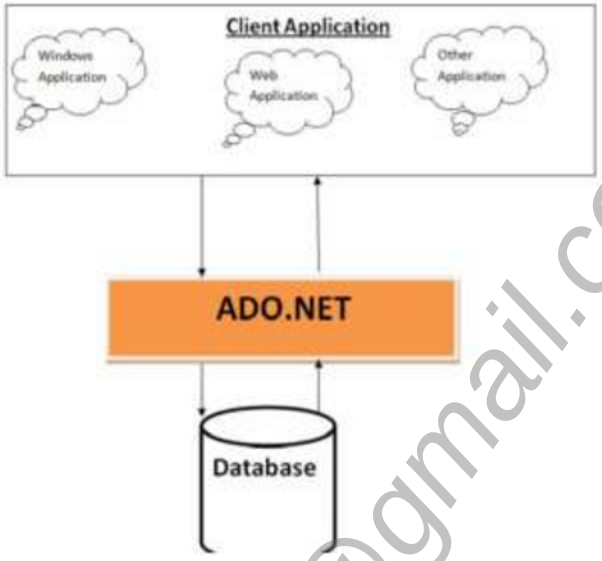
# Unit 5
# Working with Database

## ADO.NET

- ADO.NET which stands for ActiveX Data Object, is a data access technology from the Microsoft .NET Framework.
- The ASP.NET CORE framework defines a number of namespaces to interact with a Relational Database System like Microsoft SQL Server, Oracle, MySQL, etc. Collectively, these namespaces are known as ADO.NET.
- ADO.NET provides a bridge between the front end controls and the back end database as shown in the figure below




- It is a part of the base class library that is included with the Microsoft .NET Framework.
- It is commonly used by programmers to access and modify data stored in relational database systems, though it can also access data in non-relational data sources
- ADO.NET is commonly used by programmers used to establish connection between application and data sources, access and perform insertion, retrieving, modification and deletion of data stored in relational database system such as SQL Server.
- We can use ADO.NET in 2 connectional manners, also known as environments, as below
  1. Connected Environment
     - A Connected Environment means the application remains connected with the database throughout the whole length of the operation. Here you typically interact with the database using connection, command, and data reader objects.
     - Main Steps to access Database
       a. Establish a database connection
       b. Retrieve, insert, update or delete data by using Execute Command and DataReader
       c. Close the connection

  2. Disconnected Environment
     - A Disconnected Environment allows data retrieved from the data source to be manipulated and later reconciled with the database. Here you use DataTables and DataSets, which are client-side copy of external data, to traverse and manipulate the contents.
     - Steps to access Database
       a. Establish a connection to the database
       b. Create a DataSet/DataTable
       c. Retrieve, insert, update or delete data by updating DataSet/DataTable using DataAdapter
       d. Close the connection

## ADO.NET Object Model

- In the ADO.NET object model, the data residing in a database is retrieved through a data provider. The data provider is the set of components including the Connection, Command, DataReader, and DataAdapter objects.
- An application can access data either through a DataSet or through a DataReader object.
- SO, a data provider is used for connecting to a database, retrieving data, storing data in a dataset, reading the retrieved data, and updating the database.

- The following figure shows the ADO.NET Object model.

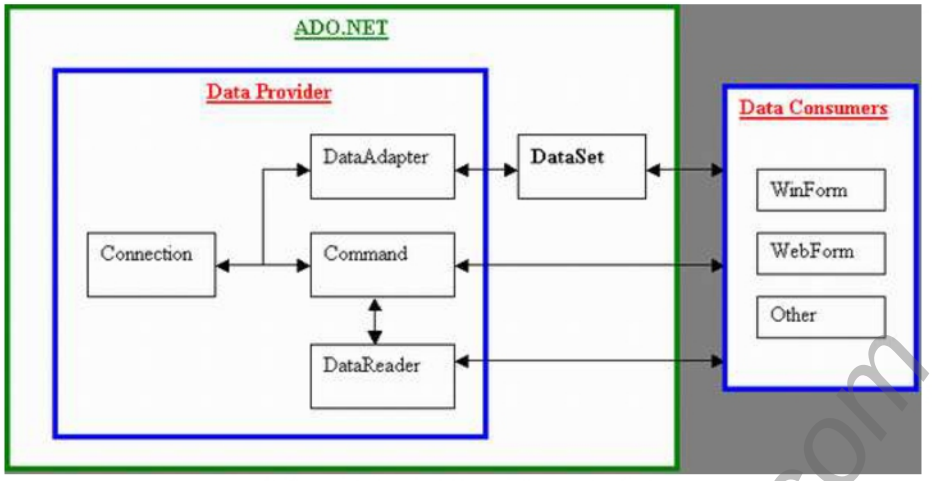



Fig: ADO.NET Object Model

- The four components of data provider are
  1. Connection: This component is used to establish a connection with a data source such as database.
  2. Command: The command class provides methods for executing SQL statements and Stored procedures. This component is used to retrieve, insert, delete or modify data in a data source. Some commands that are executed by command class includes
     a. ExecuteReader():Executes (runs) a SQL query and returns the data provider's SqlDataReader object, which provides forward-only, read-only access for the result of the query. It returns data to the client as rows.
     b. ExecuteNonQuery():Executes a SQL nonquery (e.g., insert, update, delete, or create table).
     c. ExecuteScalar(): ExecuteScalar method is used to execute SQL Commands or store procedure, and after executing returns a single value(example record counts) from the database. It also returns the first column of the first row in the result set from a database.
  3. DataReader: This component is used to retrieve data from the data source in a read only and forward only mode. It is used in conjuction with the command class to execute SQL select statement and then access the returned rows.
  4. DataAdapter: It is used to connect DataSet to Database. This component is used to transfer data to and from the database.
- The dataset is a memory based relational representation of data i.e. collection of DataTable object. A dataset is a part of **disconnected environment**. A dataset is disconnected cached set of records that are retrieved from the database. A dataset contains a collection of one or more DataTable objects made up of rows and columns of data.

**Creating and Managing Connections in ADO.NET**

- The connection component of a dataprovider establishes a connection with a data base. To connect to a Microsoft SQL Server, you use the SQL connection class.
- The following are the commonly used properties and Methods of the SqlConnection class.
  1. ConnectionString: provides information, such as database name and, user credentials for database access and so on.
  2. Open(): Opens the connection for accessing the database.
  3. Close(): Closes the connection to the database.

Example:

```
using System.Data.SqlClient;
public void DatabaseOperation()
{
        SqlConnection cn = new SqlConnection();
        cn.ConnectionString = "Data source=LAPTOP\QC6CBB7\SQLEXPRESS; Initial Catalog=Sample;
        User Id=sa; Password=faculty";
```

```
            cn.Open();
                //Perform Some Database Operation
            cn.Close()
    }
```

In the above code,

- Data Source: Specifies the provider name or your server name.
- Initial Catalog: Specifies the name of the database.
- User Id and Password: Provide the username and password of your database server.

**Programming Examples**

1. **A database db_Mac has a table called tbl_student having attributes sid, name roll, address and phone_number. Using ADO.net, write console application to**

   **a) Input the record of student from user and insert new record**

```
using System;
using System.Data.SqlClient;
namespace DemoADO
{
    class Program
    {
        static void Main(string[] args)
        {
            SqlConnection cn = new SqlConnection();
            cn.ConnectionString = "Data Source=.;Initial Catalog=db_Mac1;User
                                                        ID=sa;Password=*******";
            Console.WriteLine("Enter Id");
            int id = Convert.ToInt16( Console.ReadLine());
            Console.WriteLine("Enter Name");
            String nm = Console.ReadLine();
            Console.WriteLine("Enter Roll");
            int roll =Convert.ToInt16( Console.ReadLine());
            Console.WriteLine("Enter Address");
            String add = Console.ReadLine();
            Console.WriteLine("Enter Phone Number");
            String pn = Console.ReadLine();
            try
            {
                cn.Open();
                string sql = "insert into tbl_student values
                            ("+id+",'"+nm+"',"+roll+",'"+add+"','"+pn+"') ";
                SqlCommand cmd = new SqlCommand(sql, cn);
                cmd.ExecuteNonQuery();
                Console.WriteLine("One Record Inserted Successfully");
                Console.ReadKey();
            }
            catch (Exception f)
            {
                Console.Write(f.ToString());
            }
            finally
            {
                cn.Close();
            }
        }
    }
}
```

   **b. Display all the records**

```
using System;
using System.Data.SqlClient;
namespace DemoADO
{
    class Program
    {
```

```
        static void Main(string[] args)
        {
            SqlConnection cn = new SqlConnection();
            cn.ConnectionString = "Data Source=LAPTOP-CQC6CBB7\\SQLEXPRESS;Initial
                                   Catalog=db_Mac1;User ID=sa;Password=11111111";
            try
            {
                cn.Open();
                string sql = "select * from tbl_student";
                SqlCommand cmd = new SqlCommand(sql,cn);
                SqlDataReader dr = cmd.ExecuteReader();
                while (dr.Read())
                {
                    string id = dr[0].ToString(); //dr["Id"].ToString();
                    string name = dr[1].ToString();
                    string roll = dr[2].ToString();
                    string add = dr[3].ToString();
                    string pn = dr[4].ToString();
                    string msg = id + "\t" + name + "\t" + roll + "\t" + add + "\t" + pn;
                    Console.WriteLine(msg);
                }
                Console.ReadKey();
            }
            catch (Exception f)
            {
                Console.Write(f.ToString());
            }
            finally
            {
                cn.Close();
            }
        }
    }
}
```

**c. Delete record whose id is entered by user.**

```
using System;
using System.Data.SqlClient;
namespace DemoADO
{
    class Program
    {
        static void Main(string[] args)
        {
            SqlConnection cn = new SqlConnection();
            cn.ConnectionString = "Data Source=LAPTOP-CQC6CBB7\\SQLEXPRESS;Initial
                                   Catalog=db_Mac1;User ID=sa;Password=1111";
            Console.WriteLine("Enter id to be deleted");
            int id = Convert.ToInt16(Console.ReadLine());
            try
            {
                cn.Open();
                string sql = "delete from tbl_student where id="+id+"";
                SqlCommand cmd = new SqlCommand(sql, cn);
                cmd.ExecuteNonQuery();
                Console.WriteLine("Record Deleted Successfully");
                Console.ReadKey();
            }
            catch (Exception f)
            {
                Console.Write(f.ToString());
            }
            finally
            {
                cn.Close();
```

```
            }
        }
    }
}
```

**d. Update the name to "Hari" whose id is 5**

```
using System;
using System.Data.SqlClient;
namespace DemoADO
{
    class Program
    {
        static void Main(string[] args)
        {
            SqlConnection cn = new SqlConnection();
            cn.ConnectionString = "Data Source=LAPTOP-CQC6CBB7\\SQLEXPRESS;Initial
                                    Catalog=db_Mac1;User ID=sa;Password=1111";
            try
            {
                cn.Open();
                string sql = "update tbl_student set Name='Hari' where id=1";
                SqlCommand cmd = new SqlCommand(sql, cn);
                cmd.ExecuteNonQuery();
                Console.WriteLine("Record Updated Successfully");
                Console.ReadKey();
            }
            catch (Exception f)
            {
                Console.Write(f.ToString());
            }
            finally
            {
                cn.Close();
            }
        }
    }
}
```

*Note: In case of Console Application(.net Core), we have to install the SqlClient package first.*

**e. Count the total number of records stored in the table.**

```
using System;
using System.Data.SqlClient;
namespace DemoADO
{
    class Program
    {
        static void Main(string[] args)
        {
            SqlConnection cn = new SqlConnection();
            cn.ConnectionString = "Data Source=LAPTOP-CQC6CBB7\\SQLEXPRESS;Initial
            Catalog=db_Mac1;User ID=sa;Password=1111";
            try
            {
                cn.Open();
                string sql = "select count(*) from tbl_student";
                SqlCommand cmd = new SqlCommand(sql, cn);
                object c = cmd.ExecuteScalar();
                Console.WriteLine("Total number of records=" + c);
                Console.ReadKey();
            }
            catch (Exception f)
            {
                Console.Write(f.ToString());
            }
```

```
            finally
            {
                cn.Close();
            }
        }
    }
}
```

2. **A database db_Mac has a table called tbl_login having attributes sid, uname and password. Using ADO.net write a console program to input username and password and then check if the user is authentic user or not and display suitable message.**

   i. **Working in disconnected environment**

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace DemoADO
{
    class Program
    {
        static void Main(string[] args)
        {
            SqlConnection cn = new SqlConnection();
            cn.ConnectionString = "Data Source=LAPTOP-CQC6CBB7\\SQLEXPRESS;Initial
            Catalog=db_Mac1;User ID=sa;Password=1111";
            Console.WriteLine("Enter username and Password");
            string uname = Console.ReadLine();
            string pwd = Console.ReadLine();
            try
            {

                cn.Open();
                string sql = "select * from tbl_login where uname='" + uname + "' and
                password ='"+pwd+"'";
                SqlDataAdapter da = new SqlDataAdapter(sql, cn);
                DataTable dt = new DataTable();//Also we can use Dataset instead of
                                                                        DataTable
                da.Fill(dt);
                if (dt.Rows.Count==1)
                {
                    Console.WriteLine("Valid Login");
                }
                else
                {
                    Console.WriteLine("Invalid Login");
                }
                Console.ReadKey();
            }
            catch (Exception f)
            {
                Console.Write(f.ToString());
            }
            finally
            {
                cn.Close();
            }
        }
    }
}
```

   ii. **Working in connected environment**

```
using System;
using System.Data.SqlClient;
namespace DemoADO
{
    class Program
    {
        static void Main(string[] args)
```

```
        {
            SqlConnection cn = new SqlConnection();
            cn.ConnectionString = "Data Source=LAPTOP-CQC6CBB7\\SQLEXPRESS;Initial
                                   Catalog=db_Mac1;User ID=sa;Password=1111";
            Console.WriteLine("Enter username and Password");
            string uname = Console.ReadLine();
            string pwd = Console.ReadLine();

            try
            {
                cn.Open();
                string sql = "select * from tbl_login where uname='"+uname+"' and
                              password='"+pwd+"'";
                SqlCommand cmd = new SqlCommand(sql, cn);
                SqlDataReader dr = cmd.ExecuteReader();
                if (dr.Read())
                {
                    Console.WriteLine("Valid Login");
                }
                else
                {
                    Console.WriteLine("Invalid Login");
                }
                Console.ReadKey();
            }
            catch (Exception f)
            {
                Console.Write(f.ToString());
            }
            finally
            {
                cn.Close();
            }
        }
    }
}
```

3. **A database db_Mac has a table called tbl_studnet having attributes Id, name roll, address and phone_number. Using ADO.net and ASP.net core MVC write a program**
   a) **To insert a record of students upon button click**

**Modal Class: StudentModel.cs**

```
namespace AdoNetDemo.Models
{
    public class StudentModel
    {
        public int Id { get; set; }
        public string name { get; set; }
        public int roll { get; set; }
        public string address { get; set; }
        public string phonenumber { get; set; }
    }
}
```

**Controller Class: HomeController.cs**

```
using System;
using Microsoft.AspNetCore.Mvc;
using AdoNetDemo.Models;
using System.Data.SqlClient;
namespace AdoNetDemo.Controllers
{
    public class HomeController : Controller
```

```
    {
        public IActionResult insertstudent()
        {
            return View(null);
        }



        [HttpPost]
        public IActionResult addnewuser(StudentModel s)
        {
            SqlConnection cn = new SqlConnection();
            cn.ConnectionString = "Data Source=LAPTOP-CQC6CBB7\\SQLEXPRESS;Initial Catalog =
                                   db_Mac1; User ID = sa; Password =1111";
            try
            {
                cn.Open();
                string sql = "insert into tbl_student values ("+s.Id+", '"+s.name+"',
                              "+s.roll+", '"+s.address+"', '"+s.phonenumber+"') ";
                SqlCommand cmd = new SqlCommand(sql, cn);
                cmd.ExecuteNonQuery();
                return Content("One Recorded Inserted Successfully");
            }
            catch (Exception f)
            {
                return Content(f.ToString());
            }
            finally
            {
                cn.Close();
            }
        }
    }
}
```

**View:insertstudent.cshtml**

```
@model AdoNetDemo.Models.StudentModel
<html>
<body>
    <form method="post" action="addnewuser">
        <label asp-for="Id"></label>
        <input asp-for="Id" />
        <br />
        <label asp-for="name"></label>
        <input asp-for="name" />
        <br />
        <label asp-for="roll"></label>
        <input asp-for="roll" />
        <br />
        <label asp-for="address"></label>
        <input asp-for="address" />
        <br />
        <label asp-for="phonenumber"></label>
        <input asp-for="phonenumber" />
        <br />
        <input type="submit" value="Save" />
    </form>
</body>
</html>
```

**b) To read all the records from the student records and display them in html table.**

**Modal Class: StudentModel.cs**

```
namespace AdoNetDemo.Models
{
    public class StudentModel
    {
        public int Id { get; set; }
        public string name { get; set; }
        public int roll { get; set; }
        public string address { get; set; }
        public string phonenumber { get; set; }
    }
}
```

**Controller Class: HomeController.cs**

```
using AdoNetDemo.Models;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
namespace AdoNetDemo.Controllers{
    public class HomeController : Controller
    {
        public IActionResult showallrecord()
        {
            List<StudentModel> entirestudents = new List<StudentModel>();
            SqlConnection cn = new SqlConnection();
            cn.ConnectionString = "Data Source=LAPTOP-CQC6CBB7\\SQLEXPRESS;Initial Catalog =
              db_Mac1; User ID = sa; Password = 1111";
            try
            {
                cn.Open();
                string sql = "select * from tbl_student";
                SqlCommand cmd = new SqlCommand(sql, cn);
                SqlDataReader dr = cmd.ExecuteReader();
                while (dr.Read())
                {
                    StudentModel s = new StudentModel();
                    s.Id = Convert.ToInt16(dr[0]);
                    s.name = dr[1].ToString();
                    s.roll = Convert.ToInt16( dr[2]);
                    s.address = dr[3].ToString();
                    s.phonenumber = dr[4].ToString();
                    entirestudents.Add(s);
                }
                return View(entirestudents);
            }
            catch (Exception f)
            {
                return Content(f.ToString());
            }
            finally
            {
                cn.Close();
            }
        }
    }
}
```

**View:showallrecord.cshtml**

```
@model IEnumerable<AdoNetDemo.Models.StudentModel>
<html>
<body>
<table>
    <thead>
        <tr>
            <th>
```

```
                @Html.DisplayNameFor(model => model.Id)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.name)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.roll)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.address)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.phonenumber)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model)
{
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Id)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.name)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.roll)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.address)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.phonenumber)
            </td>
        </tr>
}
    </tbody>
</table>
</body>
</html>
```

4. **A database db_Mac has a table called tbl_login having attributes Id, uname and password. Using ADO.net and ASP.net core MVC, write a program to check the user is authentic user or not and finally display suitable message by redirecting to new page.**

**Model:LoginModel.cs**

```
using System.ComponentModel.DataAnnotations;
namespace AdoNetDemo.Models
{
    public class LoginModel
    {
        public int Id { get; set; }
        public string uname { get; set; }
        [DataType(DataType.Password)]
        public string password { get; set; }
    }
}
```

**Controller: HomeController.cs**

```
using AdoNetDemo.Models;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Data.SqlClient;
namespace AdoNetDemo.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult showloginpage()
        {
            return View(null);
        }
        public IActionResult verifylogin(LoginModel l)
        {
            SqlConnection cn = new SqlConnection();
            cn.ConnectionString = "Data Source=LAPTOP-CQC6CBB7\\SQLEXPRESS;Initial Catalog =
                                    db_Mac1; User ID = sa; Password = 1111";
            try
            {
                cn.Open();
                string sql = "select * from tbl_login where uname='" + l.uname + "' and
                password ='"+l.password+"'";
                SqlCommand cmd = new SqlCommand(sql, cn);
                SqlDataReader dr = cmd.ExecuteReader();
                if (dr.Read())
                {
                    ViewBag.message = "Welcome to our site!!!";
                }
                else
                {
                    ViewBag.message = "Invalid Login!!!";
                }
                return View();
            }
            catch (Exception f)
            {
                return Content(f.ToString());
            }
            finally
            {
                cn.Close();
            }
        }
    }
}
```

**View: showloginpage.cshtml**

```
@model AdoNetDemo.Models.LoginModel
    <html>
    <body>
      <form method="post" action="verifylogin">
          <label asp-for="uname"></label>
          <input asp-for="uname" />
          <br />
          <label asp-for="password"></label>
          <input asp-for="password" />
          <br />
          <input type="submit" value="Create" />
      </form>
    </body>
</html>
```

**View: verifylogin.cshtml**

```
<html>
<body>
    @ViewBag.message;
</body>
</html>
```

## Object-Relational Mapper (ORM)

- When we work with an object-oriented system, there is a mismatch between the object model and the relational database. RDBMSs represent data in a tabular format whereas object-oriented languages, such as C# represent it as an interconnected graph of objects.

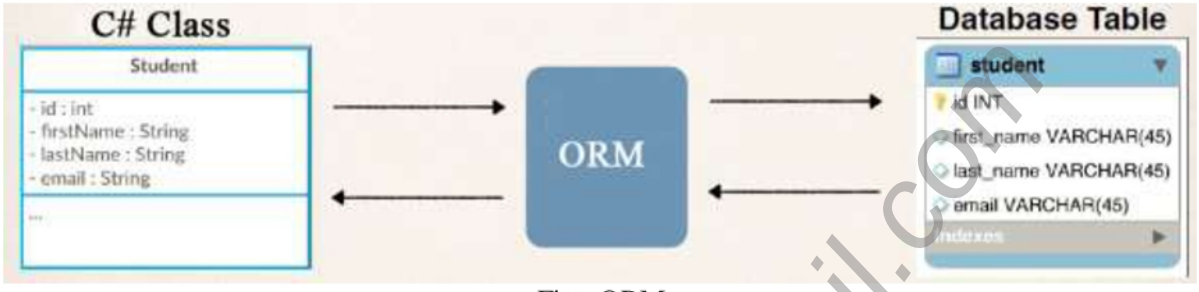



Fig: ORM

- ORM stands for object relational mapping, where objects are used for connecting the programming language on to the database systems, with the facility to work SQL along with the object oriented programming concepts.
- An ORM is an application or system or programming technique that supports the conversion of data with a relational database management system and the object model that is necessary for use within object oriented programming.
- There are many types of ORM languages like Django ORM for Python, Hibernate ORM for java, Dapper ORM for c# etc.

**Advantage**

- No need to learn a database query language.
- It propagates the idea of data abstraction thus improving data security.
- Instead of storing big procedures in pl/SQL in the backend, these can be saved in the frontend. It improves flexibility to make changes.
- If there are many relations like 1: m, n: m and lesser 1:1 in database structure then ORM works well.
- It reduces the hassles for coders by reducing the database query part handling.
- Queries via ORM can be written irrespective of whatever database one is using in the back end. This provides a lot of flexibility to the coder. This is one of the biggest advantages offered by ORMs.
- This are available for any object-oriented language so it is not only specific to one language.

**Disadvantages**

- Performance: We can probably get more performant queries by writing them yourself.
- Complexity: There is overhead involved in learning how to use any given ORM.
- Time Consuming: The initial configuration of an ORM can be a headache.

## Entity Framework (EF) Core

- Entity Framework is an Object/Relational Mapping (O/RM) framework that is intended to be used in .NET Core application.
- Entity Framework Core is the new version of Entity Framework after EF 6.x.
- It is open-source, lightweight, extensible and a cross-platform version of Entity Framework data access technology.
- It is an enhancement to ADO.NET that gives developers an automated mechanism for accessing & storing the data in the database.
- EF Core can serve as an object-relational mapper (O/RM), which:
    - Enables .NET developers to work with a database using .NET objects.
    - Eliminates the need for most of the data-access code that typically needs to be written.
- EF Core supports two development approaches
    1. Code-First

- In the code-first approach, EF Core API creates the database and tables using migration based on the conventions and configuration provided in your domain classes. This approach is useful **in Domain Driven Design (DDD).**


2. Database-First.
   - In the database-first approach, EF Core API creates the domain and context classes based on your existing database using EF Core commands. This has limited support in EF Core as it does not support visual designer or wizard.


- EF Core mainly targets the **code-first approach** and provides little support for the database-first approach because the visual designer or wizard for DB model is not supported as of EF Core 2.0.

For detail, Visit

Entity Framework Core Tutorials (entityframeworktutorial.net)

**Data Provider**

- Entity Framework Core uses a DataProvider plugin libraries to access different databases.
- There are different EF Core DB providers available for the different databases. These providers are available as NuGet packages.
- The following table lists some database providers and NuGet packages for EF Core.

| Database | NuGet Package |
|---|---|
| SQL Server | Microsoft.EntityFrameworkCore.SqlServer |
| MySQL | MySql.Data.EntityFrameworkCore |

- Select on the menu: Tools -> NuGet Package Manager -> Package Manager Console and execute the following command to install the SQL Server provider package:

**PM> Install-Package Microsoft.EntityFrameworkCore.SqlServer**

**Data Context**

- The DbContext class is an integral part of Entity Framework.
- An instance of DbContext represents a session with the database which can be used to query and save instances of your entities to a database.
- DbContext is a combination of the Unit Of Work and Repository patterns.
- DbContext in EF Core allows us to perform following tasks:
  - Manage database connection
  - Configure model & relationship
  - Querying database
  - Saving data to the database
  - Configure change tracking
  - Transaction management

- To use DbContext in our application, we need to create the class that derives from DbContext, also known as context class. This context class typically includes DbSet<TEntity> properties for each entity in the model. Consider the following example of context class in EF Core.

```
public class MacContext:DbContext
{
    public MacContext()
    {
    }
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(@"Server=LAPTOP-
            CQC6CBB7\SQLEXPRESS;Database=db_Mac;Trusted_Connection=True;");
    }



    //entities
    public DbSet<StudentModel> tbl_student { get; set; }
    public DbSet<LoginModel> tbl_login { get; set; }
}
```

    - In the above example, the MacContext class is derived from the DbContext class and contains the DbSet<TEntity> properties of StudentModel and LoginModel type.
    - It also overrides the OnConfiguring and OnModelCreating methods.
    - We must create an instance of MacContext to connect to the database and save or retrieve StudentModel or LoginModel data.
    - The OnConfiguring() method allows us to select and configure the data source to be used with a context using DbContextOptionsBuilder.

- Some frequently used DbContext Methods are
    1. Add():Adds a new entity to DbContext with Added state and starts tracking it. This new entity data will be inserted into the database when SaveChanges() is called.
    2. AddRange():Adds a collection of new entities to DbContext with Added state and starts tracking it. This new entity data will be inserted into the database when SaveChanges() is called.
    3. Update():Attaches disconnected entity with Modified state and start tracking it. The data will be saved when SaveChagnes() is called.
    4. UpdateRange():Attaches a collection of disconnected entities with Modified state and start tracking it. The data will be saved when SaveChagnes() is called.
    5. Remove():Sets Deleted state to the specified entity which will delete the data when SaveChanges() is called.
    6. RemoveRange():Sets Deleted state to a collection of entities which will delete the data in a single DB round trip when SaveChanges() is called.
    7. SaveChanges():Execute INSERT, UPDATE or DELETE command to the database for the entities with Added, Modified or Deleted state.

**Data Model**

- With EF Core, data access is performed using a model.
- A model is made up of entity classes and a context object that represents a session with the database. The context object allows querying and saving data. For more information, see Creating a Model.
- EF supports the following model development approaches:
    1. **Generate a model from an existing database.**
        - In Visual Studio, select menu Tools -> NuGet Package Manger -> Package Manger Console and run the following command:

```
PM> Scaffold-DbContext "Server=.\SQLExpress;Database=db_Mac;Trusted_Connection=True;"
        Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

*Note: Microsoft.EntityFrameworkCore package must be installed first for this command to be executed*

        - The above Scaffold-DbContext command creates entity classes for each table in the db_Mac database and context class (by deriving DbContext) with Fluent API configurations for all the entities in the Models folder.

- In the above command, the first parameter is a connection string which includes three parts: DB Server, database name and security info.
- Here, Server=.\SQLExpress; refers to local SQLEXPRESS database server.
- Database=db_Mac; specifies the database name "db_Mac" for which we are going to create classes.
- Trusted_Connection=True; specifies the Windows authentication. It will use Windows credentials to connect to the SQL Server.
- The second parameter is the provider name. We use provider for the SQL Server, so it is Microsoft.EntityFrameworkCore.SqlServer. The -OutputDir parameter specifies the directory where we want to generate all the classes which is the Models folder in this case.

2. **Hand code or manually coding a model to match the database.**
   **Once a model is created, use EF Migrations to create a database from the model. Migrations allow evolving the database as the model changes.**
   - EF Core includes different migration commands to create or update the database based on the model (entities and context).
   - We can execute the migration command using NuGet Package Manger Console.

     PM> add-migration CreateMacDB

     This will create a new folder named **Migrations** in the project and necessary migration classes.

     to remove migration use command

     PM> `remove-migration`

     Note: To performing migration operation there must be primary key field in each model to be mapped. The primary key is the field that starts with entity name and ends with Id i.e. <entityname>Id, such that entityname is optional. If we have to specify fields that doesn't follow this format as primary key then we have to override the onModelCreating() method as below

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
        modelBuilder.Entity<Student>().HasKey(x => x.Id);
        base.OnModelCreating(modelBuilder);
}
```

     Or, use the [Key] Attribute for the properties to be used as key attribute in the model.

   - After creating a migration, we still need to create the database using the **update-database** command in the Package Manager Console, as below.

     PM> update-database

     This will create the database with the name and location specified in the connection string in the UseSqlServer() method. It creates a table for each DbSet property (tbl_student and tbl_course).

**Programming Examples**

1. **A database db_Mac has a table called tbl_studnet having attributes Id(Identity), name roll, address and phone_number. Using entity framework, Asp.net Core MVC, write a program**
   a. **To insert a record of students upon button click**

**Model: StudentModel.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace EntityTest.Models
{
    public class StudentModel
    {
```

```
            public int Id { get; set; }
            public string name { get; set; }
            public int roll { get; set; }
            public string address { get; set; }
            public string phonenumber { get; set; }

    }
}
```

**Database Context: MacContext.cs**

```
using Microsoft.EntityFrameworkCore;
namespace EntityTest.Models
{
    public class MacContext:DbContext
    {

`       public MacContext()
        {
        }
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Server=LAPTOP-
                    CQC6CBB7\SQLEXPRESS;Database=db_Mac1;Trusted_Connection=True;");
        }
        public DbSet<StudentModel> tbl_student { get; set; }
    }
}
```

**Controller: HomeController.cs**

```
using System;
using Microsoft.AspNetCore.Mvc;
using EntityTest.Models;
namespace EntityTest.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult insertstudent()
        {
            return View(null);
        }
        [HttpPost]
        public IActionResult addnewuser(StudentModel s)
        {
            using (MacContext ent = new MacContext())
            {
                try
                {
                    ent.tbl_student.Add(s);
                    ent.SaveChanges();
                    return Content("One Recorded Inserted Successfully");
                }
                catch (Exception f)
                {
                    return Content(f.ToString());
                }
            };
        }
    }
}
```

**View: insertstudent.cshtml**

```
@model EntityTest.Models.StudentModel
<html>
<body>
    <form method="post" action="addnewuser">
        <label asp-for="name"></label>
        <input asp-for="name" />
        <br />
        <label asp-for="roll"></label>
        <input asp-for="roll" />
        <br />
        <label asp-for="address"></label>
        <input asp-for="address" />
        <br />
        <label asp-for="phonenumber"></label>
        <input asp-for="phonenumber" />
        <br />
        <input type="submit" value="Save" />
    </form>
</body>
</html>
```

**b. To read all the records from the student records and display them in html table.**

**Modal Class: StudentModel.cs**

```
namespace AdoNetDemo.Models
{
    public class StudentModel
    {
        public int Id { get; set; }
        public string name { get; set; }
        public int roll { get; set; }
        public string address { get; set; }
        public string phonenumber { get; set; }
    }
}
```

**Database Context: MacContext.cs**

```
using Microsoft.EntityFrameworkCore;
namespace EntityTest.Models
{
    public class MacContext:DbContext
    {

`       public MacContext()
        {
        }
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Server=LAPTOP-
                    CQC6CBB7\SQLEXPRESS;Database=db_Mac1;Trusted_Connection=True;");
        }
        public DbSet<StudentModel> tbl_student { get; set; }
    }
}
```

**Controller Class: HomeController.cs**

```
using System;
using Microsoft.AspNetCore.Mvc;
using EntityTest.Models;
using System.Collections.Generic;
using System.Linq;
namespace EntityTest.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult showallrecord()
        {
            using (MacContext ent = new MacContext())
            {
                try
                {
                    List<StudentModel> entire_std = new List<StudentModel>();
                    entire_std = ent.tbl_student.ToList();
                    return View(entire_std);
                }
                catch (Exception f)
                {
                    return Content(f.ToString());
                }
            };
        }
    }
}
```

**View:showallrecord.cshtml**

```
@model IEnumerable<EntityTest.Models.StudentModel>
<html>
<body>
    <table>
        <thead>
            <tr>
                <th>
                    @Html.DisplayNameFor(model => model.Id)
                </th>
                <th>
                    @Html.DisplayNameFor(model => model.name)
                </th>
                <th>
                    @Html.DisplayNameFor(model => model.roll)
                </th>
                <th>
                    @Html.DisplayNameFor(model => model.address)
                </th>
                <th>
                    @Html.DisplayNameFor(model => model.phonenumber)
                </th>
                <th></th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Model)
            {
                <tr>
                    <td>
                        @Html.DisplayFor(modelItem => item.Id)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.name)
                    </td>
                    <td>
```

```
                    @Html.DisplayFor(modelItem => item.roll)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.address)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.phonenumber)
                </td>
            </tr>
        }
    </tbody>
</table>
</body>
</html>
```

**c. To read all the records from the student whose id is greater than 2 and are from pokhara records and display them in html table.**

**Modal Class: StudentModel.cs**

```
namespace AdoNetDemo.Models
{
    public class StudentModel
    {
        public int Id { get; set; }
        public string name { get; set; }
        public int roll { get; set; }
        public string address { get; set; }
        public string phonenumber { get; set; }
    }
}
```

**Database Context: MacContext.cs**

```
using Microsoft.EntityFrameworkCore;
namespace EntityTest.Models
{
    public class MacContext:DbContext
    {

`       public MacContext()
        {
        }
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Server=LAPTOP-
                    CQC6CBB7\SQLEXPRESS;Database=db_Mac1;Trusted_Connection=True;");
        }
        public DbSet<StudentModel> tbl_student { get; set; }
    }
}
```

**Controller Class: HomeController.cs**

```
using System;
using Microsoft.AspNetCore.Mvc;
using EntityTest.Models;
using System.Collections.Generic;
using System.Linq;
namespace EntityTest.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult showallrecord()
        {
            using (MacContext ent = new MacContext())
```

```
            {
                try
                {
                    List<StudentModel> entire_std = new List<StudentModel>();
                           entire_std = ent.tbl_student.Where (a=>a.Id>2 &&
                           a.address=="Pokhara").ToList();
                      return View(entire_std);
                }
                catch (Exception f)
                {
                    return Content(f.ToString());
                }
            };
        }
    }
}
```

**View:showallrecord.cshtml**

```
@model IEnumerable<EntityTest.Models.StudentModel>
<html>
<body>
    <table>
        <thead>
            <tr>
                <th>
                    @Html.DisplayNameFor(model => model.Id)
                </th>
                <th>
                    @Html.DisplayNameFor(model => model.name)
                </th>
                <th>
                    @Html.DisplayNameFor(model => model.roll)
                </th>
                <th>
                    @Html.DisplayNameFor(model => model.address)
                </th>
                <th>
                    @Html.DisplayNameFor(model => model.phonenumber)
                </th>
                <th></th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Model)
            {
                <tr>
                    <td>
                        @Html.DisplayFor(modelItem => item.Id)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.name)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.roll)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.address)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.phonenumber)
                    </td>
                </tr>
            }
        </tbody>
    </table>
```

```
</body>
</html>
```

### d. Delete entire records

**Modal Class: StudentModel.cs**

```
namespace AdoNetDemo.Models
{
    public class StudentModel
    {
        public int Id { get; set; }
        public string name { get; set; }
        public int roll { get; set; }
        public string address { get; set; }
        public string phonenumber { get; set; }
    }
}
```

**Database Context: MacContext.cs**

```
using Microsoft.EntityFrameworkCore;
namespace EntityTest.Models
{
    public class MacContext:DbContext
    {

`       public MacContext()
        {
        }



        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Server=LAPTOP-
                    CQC6CBB7\SQLEXPRESS;Database=db_Mac1;Trusted_Connection=True;");
        }
        public DbSet<StudentModel> tbl_student { get; set; }
    }
}
```

**Controller Class: HomeController.cs**

```
using System;
using Microsoft.AspNetCore.Mvc;
using EntityTest.Models;
using System.Collections.Generic;
using System.Linq;
namespace EntityTest.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult deleteallrecord()
        {
            using (MacContext ent = new MacContext())
            {
                try
                {
                    ent.tbl_student.RemoveRange(ent.tbl_student.ToList());
                     ent.SaveChanges();
                    return Content("Successfully Done");
                }
                catch (Exception f)
                {
                    return Content(f.ToString());
```

```
                    }
                };
            }
        }
    }
```

**e. Delete record whose id is 1**

**Modal Class: StudentModel.cs**

```
namespace AdoNetDemo.Models
{
    public class StudentModel
    {
        public int Id { get; set; }
        public string name { get; set; }
        public int roll { get; set; }
        public string address { get; set; }
        public string phonenumber { get; set; }
    }
}
```

**Database Context: MacContext.cs**

```
using Microsoft.EntityFrameworkCore;
namespace EntityTest.Models
{
    public class MacContext:DbContext
    {

`       public MacContext()
        {
        }


        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Server=LAPTOP-
                    CQC6CBB7\SQLEXPRESS;Database=db_Mac1;Trusted_Connection=True;");
        }
        public DbSet<StudentModel> tbl_student { get; set; }
    }
}
```

**Controller Class: HomeController.cs**

```
using System;
using Microsoft.AspNetCore.Mvc;
using EntityTest.Models;
using System.Collections.Generic;
using System.Linq;
namespace EntityTest.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult showallrecord()
        {
            using (MacContext ent = new MacContext())
            {
                try
                {
                    ent.tbl_student.Remove(ent.tbl_student.Where
                    (a=>a.Id==1).FirstOrDefault());
                     ent.SaveChanges();
                    return Content("Successfully Done");
                }
                catch (Exception f)
```

```
                {
                    return Content(f.ToString());
                }
            };
        }
    }
}
```

### f. Update the address of all students to Pokhara

**Modal Class: StudentModel.cs**

```
namespace AdoNetDemo.Models
{
    public class StudentModel
    {
        public int Id { get; set; }
        public string name { get; set; }
        public int roll { get; set; }
        public string address { get; set; }
        public string phonenumber { get; set; }
    }
}
```

**Database Context: MacContext.cs**

```
using Microsoft.EntityFrameworkCore;
namespace EntityTest.Models
{
    public class MacContext:DbContext
    {

`       public MacContext()
        {
        }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Server=LAPTOP-
                    CQC6CBB7\SQLEXPRESS;Database=db_Mac1;Trusted_Connection=True;");
        }
        public DbSet<StudentModel> tbl_student { get; set; }
    }
}
```

**Controller Class: HomeController.cs**

```
using System;
using Microsoft.AspNetCore.Mvc;
using EntityTest.Models;
using System.Collections.Generic;
using System.Linq;
namespace EntityTest.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult updateallrecord()
        {
            using (MacContext ent = new MacContext())
            {
                try
                {
                    List<StudentModel> entirestd = new List<StudentModel>();
                    entirestd = ent.tbl_student.ToList();
                    entirestd.ForEach(a => a.address = "Pokhara");
```

```
                    //or
                    //foreach (var a in entirestd)
                    //{
                    //    a.address = "Pokhara";
                    //}

                    ent.tbl_student.UpdateRange(entirestd);
                    ent.SaveChanges();
                    return Content("Successfully Done");
                }
                catch (Exception f)
                {
                    return Content(f.ToString());
                }
            };
        }
    }
}
```

**g. Update the address of all students to Pokhara whose id is greater than 5**

**Modal Class: StudentModel.cs**

```
namespace AdoNetDemo.Models
{
    public class StudentModel
    {
        public int Id { get; set; }
        public string name { get; set; }
        public int roll { get; set; }
        public string address { get; set; }
        public string phonenumber { get; set; }
    }
}
```

**Database Context: MacContext.cs**

```
using Microsoft.EntityFrameworkCore;
namespace EntityTest.Models
{
    public class MacContext:DbContext
    {

`       public MacContext()
        {
        }
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Server=LAPTOP-
                CQC6CBB7\SQLEXPRESS;Database=db_Mac1;Trusted_Connection=True;");
        }
        public DbSet<StudentModel> tbl_student { get; set; }
    }
}
```

**Controller Class: HomeController.cs**

```
using System;
using Microsoft.AspNetCore.Mvc;
using EntityTest.Models;
using System.Collections.Generic;
using System.Linq;
```

```
namespace EntityTest.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult updateallrecord()
        {
            using (MacContext ent = new MacContext())
            {
                try
                {
                    List<StudentModel> entirestd = new List<StudentModel>();
                     entirestd = ent.tbl_student.Where (a=>a.Id>5).ToList();
                     entirestd.ForEach(a => a.address = "Pokhara");

                     //or
                    //foreach (var a in entirestd)
                    //{
                    //    a.address = "Pokhara";
                    //}

                    ent.tbl_student.UpdateRange(entirestd);
                    ent.SaveChanges();
                    return Content("Successfully Done");
                }
                catch (Exception f)
                {
                    return Content(f.ToString());
                }
            };
        }
    }
}
```