

Unit 5

Working with Database

Database

- A database is an organized collection of structured information, or data, typically stored electronically in a computer system.
- A database is usually controlled by a database management system (DBMS).
- The main purpose of the database is to operate a large amount of information by storing, retrieving, and managing data.
- Data within the most common types of databases in operation today is typically modeled in rows and columns in a series of tables to make processing and data querying efficient.
- Most databases use Structured Query Language (SQL) for writing and querying data
- There are many databases available like SQL Server, Oracle, MySQL, MongoDB, PostgreSQL, Sybase, Informix, etc.

SQL Server

- SQL Server is a relational database management system, or RDBMS, developed and marketed by Microsoft.
- Similar to other RDBMS software, SQL Server is built on top of SQL, a standard programming language for interacting with the relational databases.
- SQL server is tied to Transact-SQL, or T-SQL, the Microsoft's implementation of SQL that adds a set of proprietary programming constructs.

Download and Setup SQL Server

- Go to URL: <https://www.microsoft.com/en-in/sql-server/sql-server-downloads>
- Download Free Edition of MS SQL Server. Either Developer or Express Edition
- During installation, remember these:
 - In Instance Configuration Screen, choose **Default Instance**
 - In Database Engine Configuration Screen, choose **Mixed Mode(SQL Server authentication and Windows Authentication)** and Enter Password
 - Then follow Next Button.

Database Engine Configuration

Specify Database Engine authentication security mode, administrators and data directories.

- Setup Support Rules
- Installation Type
- Product Key
- License Terms
- Setup Role
- Feature Selection
- Installation Rules
- Instance Configuration
- Disk Space Requirements
- Server Configuration
- Database Engine Configuration**
- Error Reporting
- Installation Configuration Rules
- Ready to Install
- Installation Progress
- Complete

Server Configuration **Data Directories**

Specify the authentication mode and administrators for the Database Engine.

Authentication Mode _____

- Windows authentication mode
- Mixed Mode (SQL Server authentication and Windows authentication)

Specify the password for the SQL Server system administrator (sa) account. _____

Enter password:

Confirm password:

Specify SQL Server administrators _____

--

SQL Server administrators have unrestricted access to the Database Engine.

Install SQL Sever Management Studio

- Get SSMS from this url - <https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-2017>
- Install SSMS
- After install, search Microsoft SQL Server Management Studio and run
- You will see the screen as shown.
- On Authentication, Select SQL Server Authentication. For User name enter sa and for password, enter the one that use provide during installation.



SQL Server

- 1 Server type:
- 2 Server name:
- 3 Authentication:
- 4 User name:
Password:

Database Engine

[Redacted]

Windows Authentication

[Redacted]

[Redacted]

Remember password

Connect Cancel Help Options >>

ADO.NET Basics

- ADO stands for Microsoft ActiveX Data Objects.
- The ADO.NET is one of the Microsoft's data access technology which is used to communicate between the .NET Application (Console, WCF, WPF, Windows, MVC, Web Form, etc.) and data sources such as SQL Server, Oracle, MySQL, XML document, etc.
- 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.

2 Types of Connection Architectures

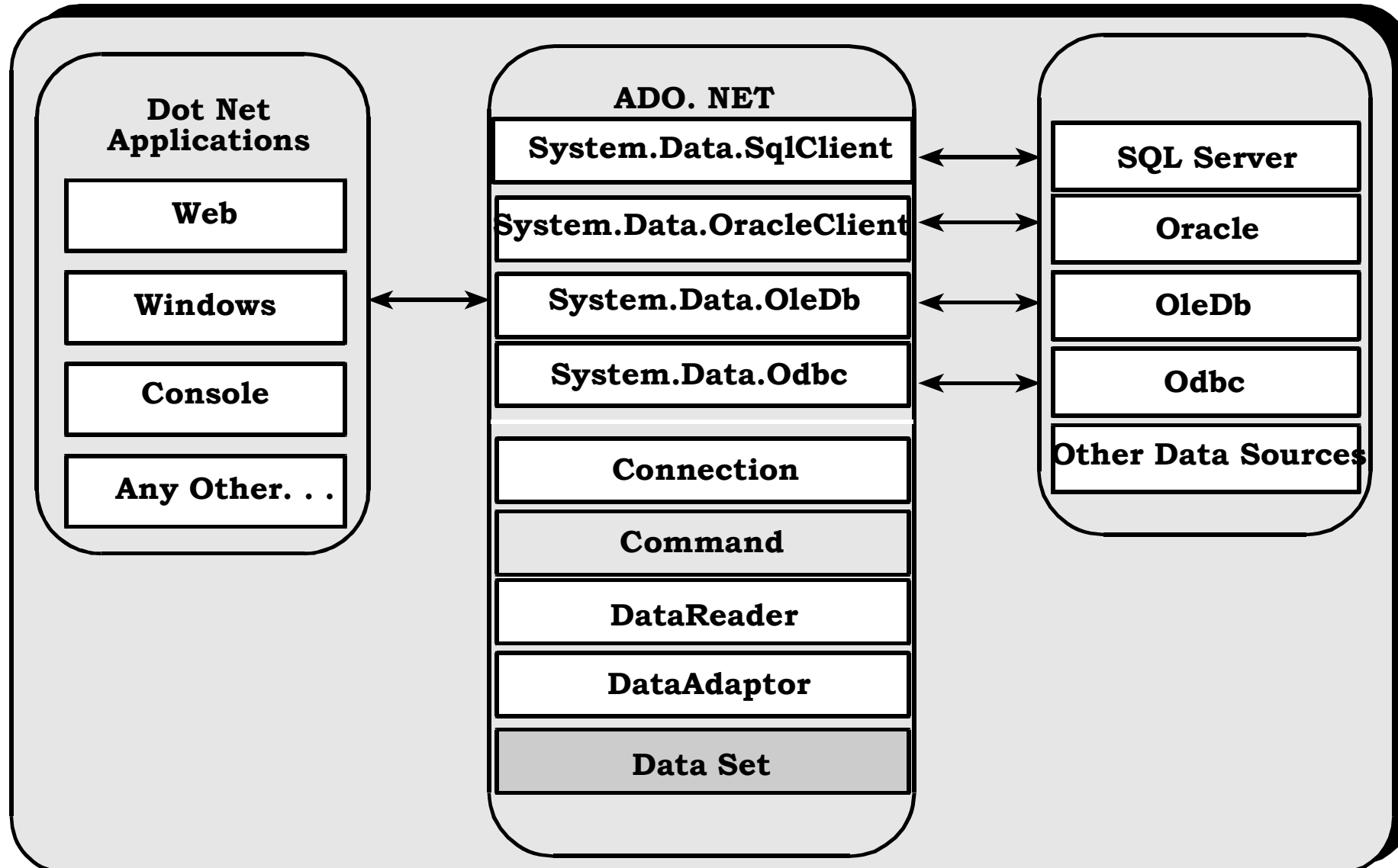
1. **Connected architecture:**

- 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 its class library



Important Classes in ADO.NET

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

Connection Class

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

Important Classes in ADO.NET

Command Class

provides methods for storing and executing SQL statements and Stored Procedures. Various commands that are executed by the Command Class:

a. **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. T
- this method returns a DataReader object that can be used to fill a DataTable object or used directly for printing reports and so forth.

Important Classes in ADO.NET

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.

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.

Important Classes in ADO.NET

DataSet Class

- 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.

Connect to a Database using ADO.NET

- To create a connection, we have to use the connection strings.
- A connection string is required as a parameter to SqlConnection.
- A ConnectionString is a string variable (not case sensitive).
- This contains key and value pairs:Provider, Server, Database, User Id and Password as in the following:

Server="name of the server or IP Address of the server"

Database="name of the database"

UserId="user name who has permission to work with database"

Password="the password of User Id"

- Example - SQL Authentication

```
string constr="server=.;database=db1;user id=sa;password=yourpassword";
```

How to connect, retrieve and display data from a database

1. Create a SqlConnection object using a connection string.
2. Handle exceptions.
3. Open the connection.
4. Create a SqlCommand. To represent a SqlCommand like (select * from studentdetails) and attach the existing connection to it. Specify the type of SqlCommand (Text/StoredProcedure).
5. Execute the command (use ExecuteReader).
6. Get the Result (use SqlDataReader). This is a forwardonly/readonly data object.
7. Process the result
8. Display the result
9. Close the connection

Create a Database in SQL Server

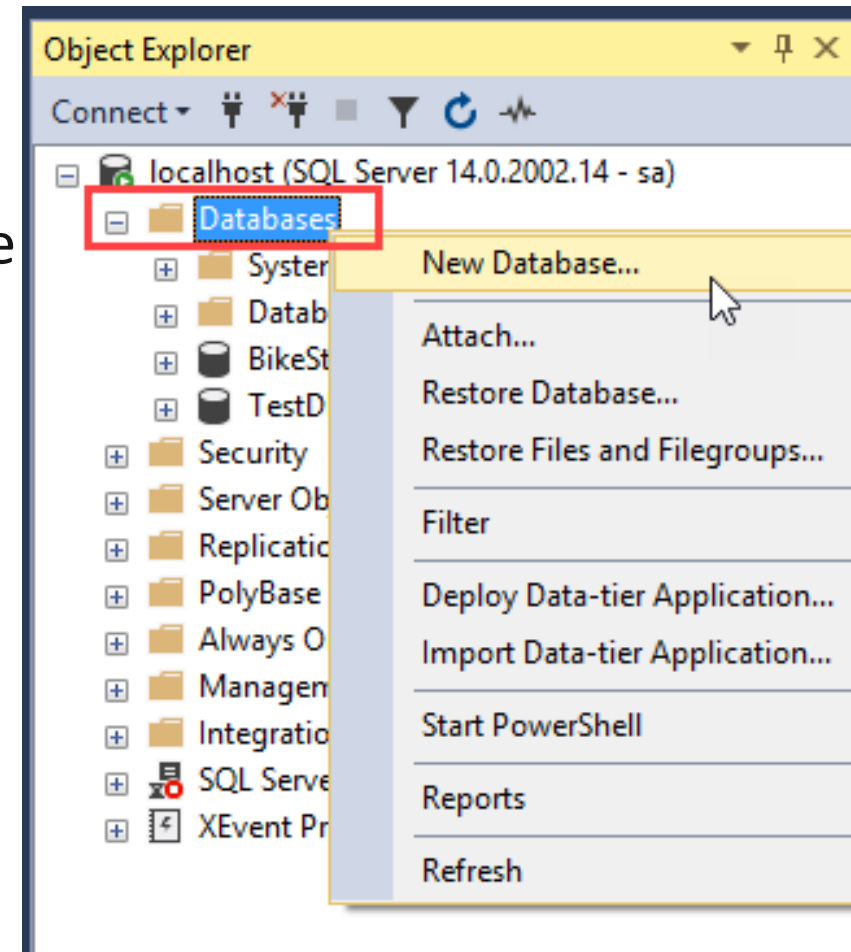
- Create Database using **CREATE DATABASE** statement

Syntax: CREATE DATABASE database_name;

Ex: CREATE DATABASE TestDb;

- Create Database using **Object Explorer**

- Right Click the Database, choose New Database
- Enter name for the database as TestDb
- Then OK



Create a Table in SQL Server

- **Create Table using CREATE TABLE statement**


Syntax: CREATE TABLE <Table_Name>

Ex:

```
CREATE TABLE AddressBook (  
    ID int PRIMARY KEY IDENTITY (1, 1),  
    Name varchar(100),  
    Address varchar(100),  
    Phone varchar(50)  
)
```

- **Create Table using Object Explorer**

- Right Click Table, choose New > Table
- Enter Column Name and DataType
- Set Primary Key and Auto Increment for ID Column
- Save Table with name as AddressBook

	Column Name	Data Type
	ID	int
	Name	varchar(100)
	Address	varchar(100)
	Phone	varchar(50)

Column Properties

Property	Value
> Full-text Specification	No
Has Non-SQL Server Subscriber	No
▼ Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1

EX Showing Connection, Command

```
string name = "Name 1";  
string address = "Address 1";  
string phone = "9801000000";
```

```
string connStr = "Data Source=AM;Initial Catalog=TestDb;User ID=sa;Password=12345";  
SqlConnection conn = new SqlConnection(connStr);
```

```
string sql = "Insert into AddressBook values ('" + name + "', '"  
    + address + "', '" + phone + "')";  
SqlCommand cmd = new SqlCommand(sql, conn);  
conn.Open();  
cmd.ExecuteNonQuery();  
conn.Close();
```

EX – Reading Data with SqlDataAdapter & DataSet

```
string connStr = "Data Source=.;Initial Catalog=TestDb;User ID=sa;Password=123456";
SqlConnection conn = new SqlConnection(connStr);
string sql = "Select * from AddressBook";
SqlDataAdapter da = new SqlDataAdapter(sql, conn);

DataSet ds = new DataSet();
da.Fill(ds);

GridView1.DataSource = ds;
GridView1.DataBind();
```

EX Read Data Using SqlDataReader

```
string connStr = "Data Source=.;Initial Catalog=TestDb;User ID=sa;Password=123456";  
SqlConnection conn = new SqlConnection(connStr);  
string sql = "Select * from AddressBook";  
SqlCommand cmd = new SqlCommand(sql, conn);
```

```
List<MyAddressBook> books = new List<MyAddressBook>();  
conn.Open();  
SqlDataReader dr = cmd.ExecuteReader();
```

```
while(dr.Read())  
{  
    MyAddressBook b = new MyAddressBook();  
    b.ID = (int)dr[0];  
    b.Name = dr["Name"].ToString();  
    b.Address = dr["Address"].ToString();  
    b.Phone = dr["Phone"].ToString();  
    books.Add(b);  
}
```

```
conn.Close();  
GridView2.DataSource = books;  
GridView2.DataBind();
```

```
class MyAddressBook  
{  
    1 reference  
    public int ID { get; set; }  
    1 reference  
    public string Name { get; set; }  
    1 reference  
    public string Address { get; set; }  
    1 reference  
    public string Phone { get; set; }  
}
```

ASP.Net core 3.1 Crud Pperation with ADO.Net

- Ref Link

<https://tutorialshelper.com/asp-net-core-3-1-crud-operation-with-ado-net/>

Entity Framework(EF) Core

- Is a 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.
- Entity Framework is an Object/Relational Mapping (O/RM) framework. It is an enhancement to ADO.NET that gives developers an automated mechanism for accessing & storing the data in the database.
- EF Core is intended to be used with .NET Core applications. However, it can also be used with standard .NET 4.5+ framework based applications.

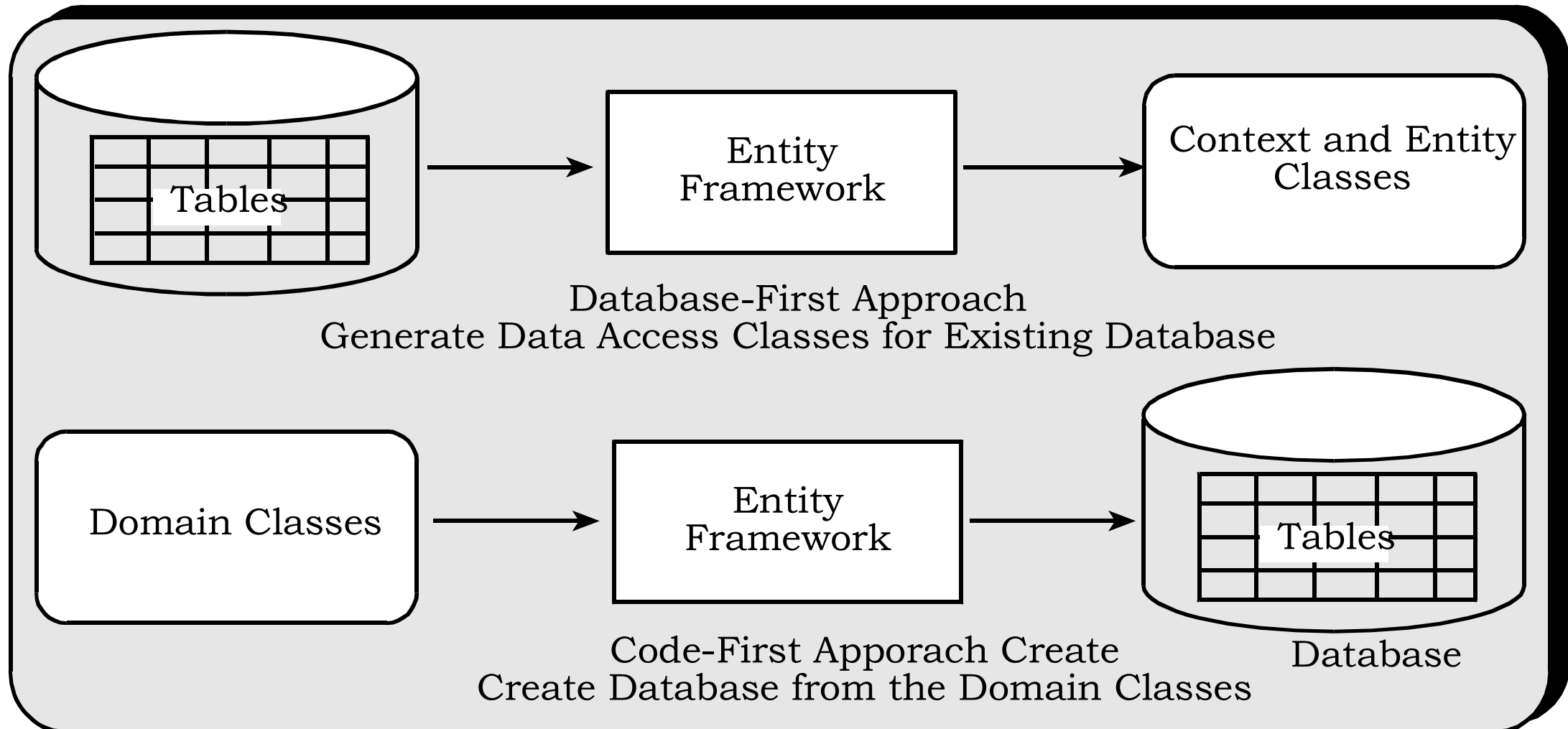
Application Type	ASP.NET Core Applications Web API, Console, etc	.NET 4.5+ Applications Console winForm WPF, ASP.NET	Devices +IoT, Mobile, PC, Xbox, Surface Hub	MobileApplication Android, iOS, Windows
EF Core	EF Core	EF Core	EF Core	EF Core
Framework	.NET Core	.NET 4.56+	UWP	Xamarin
OS	Windows, Mac, Linux	Windows	Windows 10	Mobile

figure showing supported application types, .NET Frameworks and OSs.

EF Core Development Approaches

- EF Core supports two development approaches:
 (1) Code-First (2) Database-First.
- EF Core mainly targets the code-first approach and provides some support for the database-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).
- 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 Development Approaches



EF Core vs EF 6

- Entity Framework Core is the new and improved version of Entity Framework for .NET Core applications.
- EF Core continues to support the following features and concepts, same as EF 6.
 - DbContext & DbSet
 - Data Model
 - Querying using Linq-to-Entities
 - Change Tracking
 - SaveChanges
 - Migrations

EF Core vs EF 6

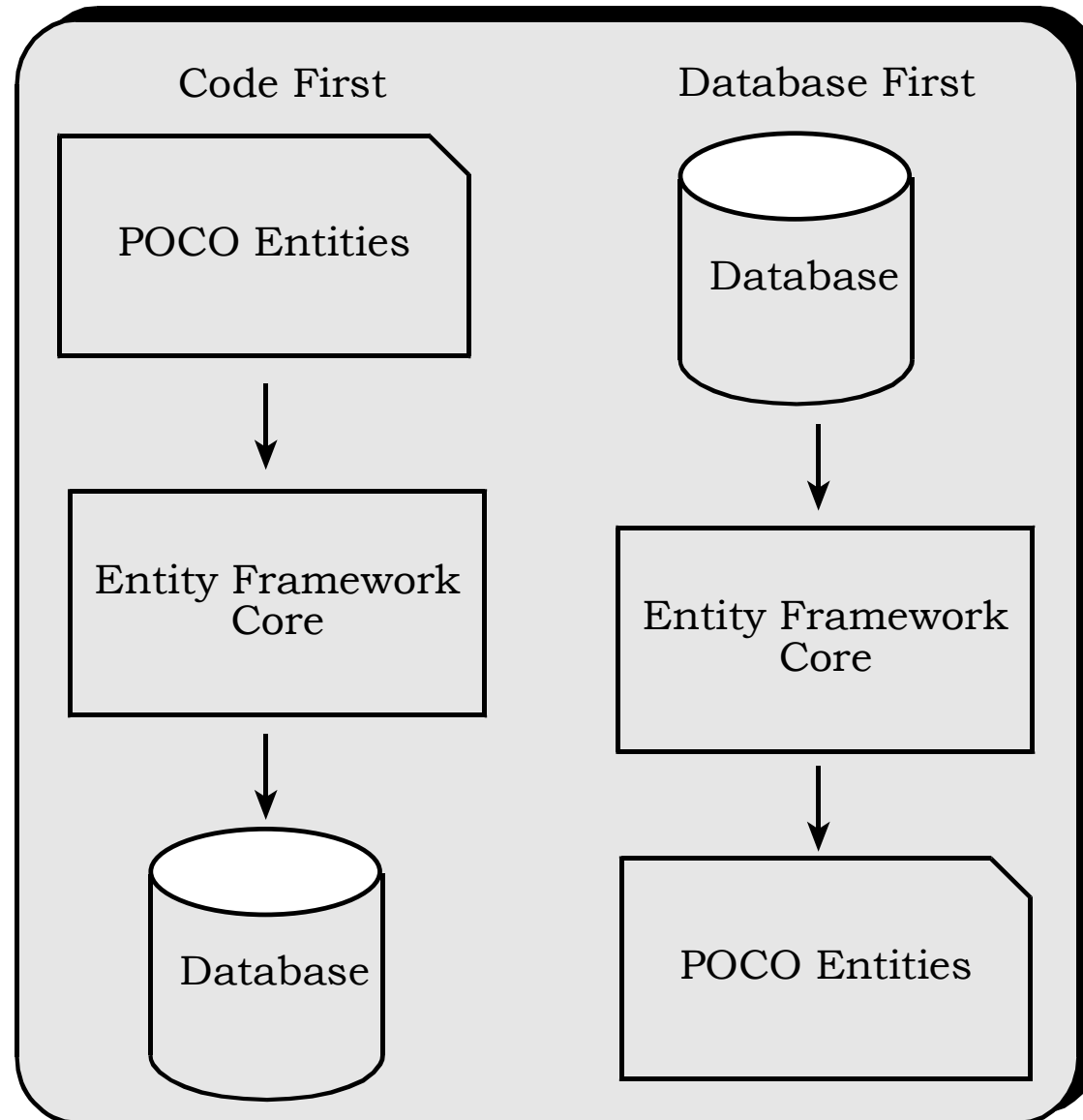
- EF Core includes the following new features which are not supported in EF 6.x:
 1. Easy relationship configuration
 2. Batch INSERT, UPDATE, and DELETE operations
 3. In-memory provider for testing
 4. Support for IoC (Inversion of Control)
 5. Unique constraints
 6. Shadow properties
 7. Alternate keys
 8. Global query filter
 9. Field mapping
 10. DbContext pooling
 11. Better patterns for handling disconnected entity graphs

EF Core Database Providers

- EF Core uses a provider model to access many different databases.
- EF Core includes providers as NuGet packages which you need to install.
- Below table lists database providers and NuGet packages for EF Core.

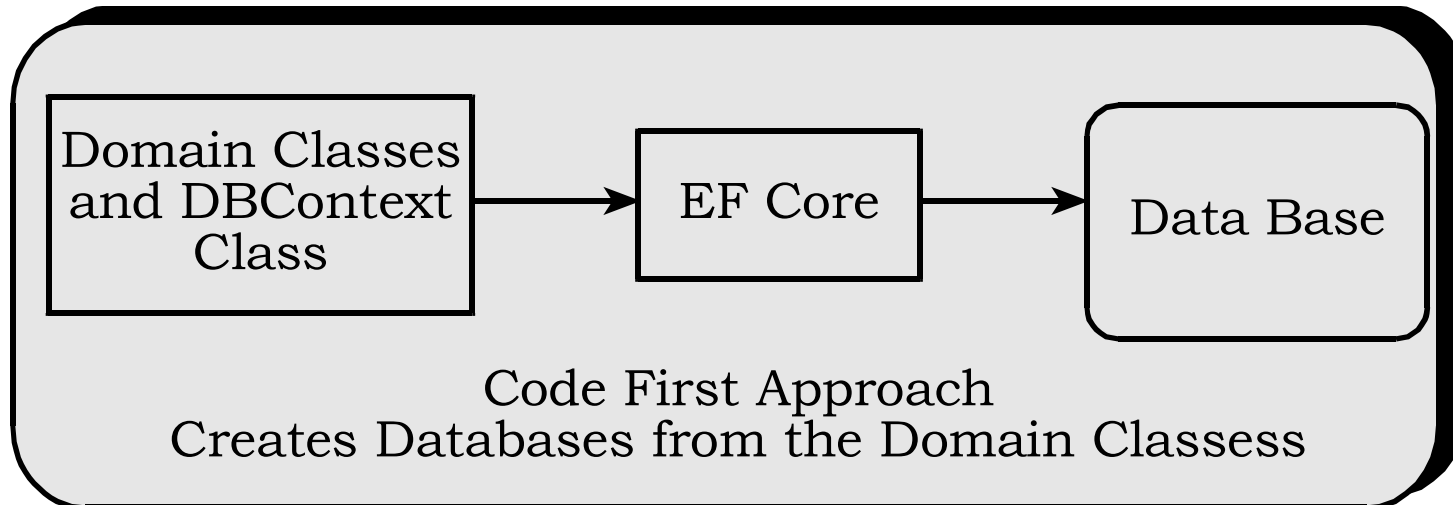
Database	NuGet Package
SQL Server	<u>Microsoft.EntityFrameworkCore.SqlServer</u>
MySQL	<u>MySql.Data.EntityFrameworkCore</u>
PostgreSQL	<u>Npgsql.EntityFrameworkCore.PostgreSQL</u>
SQLite	<u>Microsoft.EntityFrameworkCore.SQLite</u>
SQL Compact	<u>EntityFrameworkCore.SqlServerCompact40</u>
In-memory	<u>Microsoft.EntityFrameworkCore.InMemory</u>

EF Core Development Approaches



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.



EF Core Code First Approach

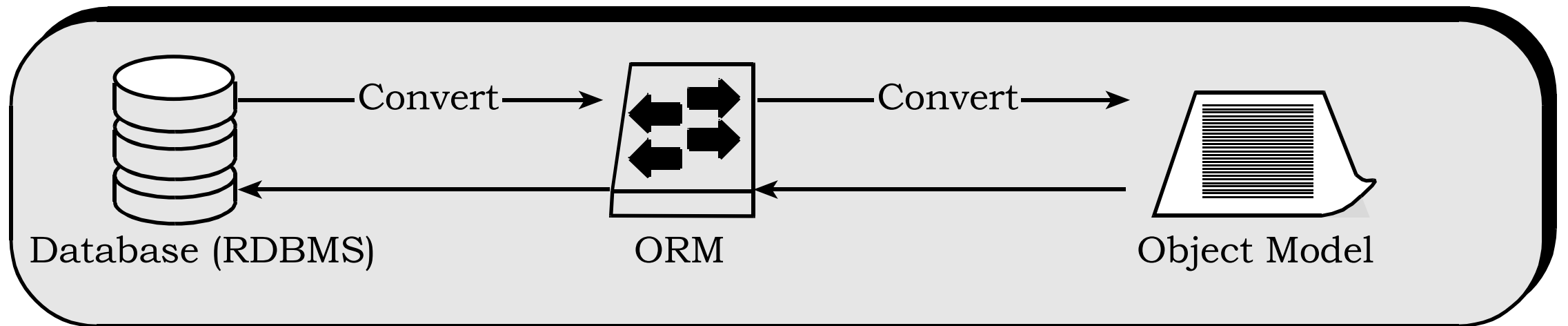
- In the code-first approach, the EF Core creates the database and tables using migration based on the default conventions and configuration. This approach is useful in Domain-Driven Design (DDD).
- Good option if you don't know the whole picture of your database as you can just update your Plain Old Class Object (POCO) entities and let EF sync changes to the database. In other words, you can easily add or remove features defined in your class without worrying about syncing your database using Migrations.
- You don't have to worry about your database as EF will handle the creation for you. In essence, database is just a storage medium with no logic.
- You will have full control over the code. You simply define and create POCO entities and let EF generate the corresponding Database for you. The downside is if you change something in your database manually, you will probably lose them because your code defines the database.
- It's easy to modify and maintain as there will be no auto-generated code.

Object Relational Mappers

- Essential parts of an ASP.NET MVC application is the architectural design. It's the Model-View-Controller (MVC) pattern. It show us the view of the application and the business logic within the application.
 - Model : designed to manage the business logic.
 - View : view that user can see.
 - Controller : manages the interaction between Model and View.
- A one of basic end point of project is the Database. We can prepare the database following many methods. The thing is, we have to access the DB from the next layer (Controller). In that point, object relational mapper(ORM) will come to the battle.

Object Relational Mappers

- An ORM is an application or system that support in the conversion of data within a relational database management system (RDBMS) and the object model that is necessary for use within object-oriented programming.



ADDING EF CORE TO AN APPLICATION

Install Entity Framework Core

- Entity Framework Core can be used with .NET Core or .NET 4.6 based applications. Here, you will learn to install and use Entity Framework Core .NET Core applications
- EF Core is not a part of .NET Core and standard .NET framework. It is available as a NuGet package.
- You need to install NuGet packages for the following two things to use EF Core in your application:
 1. EF Core DB provider
 2. EF Core tools

ADDING EF CORE TO AN APPLICATION

Install EF Core DB Provider

- EF Core allows us to access databases via the provider model. There are different EF Core DB providers available for the different databases. These providers are available as NuGet packages.
- First, install the NuGet package for the provider of database you want to access.
- For, MS SQL Server database,
install `Microsoft.EntityFrameworkCore.SqlServer` NuGet package.
- To Install DB provider NuGet package:
 - Right click on the project in the Solution Explorer in Visual Studio
 - select Manage NuGet Packages.. (or select on the menu: Tools -> NuGet Package Manager -> Manage NuGet Packages For Solution).
 - search for `Microsoft.EntityFrameworkCore.SqlServer` and install

ADDING EF CORE TO AN APPLICATION

Install EF Core Tools

- Along with the DB provider package, you also need to install EF tools to execute EF Core commands. These make it easier to perform several EF Core-related tasks in your project at design time, such as migrations, scaffolding, etc.
- EF Tools are available as NuGet packages.
- To Install EF Core Tools:
 - Right click on the project in the Solution Explorer in Visual Studio
 - select Manage NuGet Packages.. (or select on the menu: Tools -> NuGet Package Manager -> Manage NuGet Packages For Solution).
 - search for Microsoft.EntityFrameworkCore.Tools and install

Data Models

- Entity Framework needs to have a model (Entity Data Model) to communicate with the underlying database. It builds a model based on the shape of your domain classes, the Data Annotations and Fluent API configurations.
- The EF model includes three parts: conceptual model, storage model, and mapping between the conceptual and storage models.
- In the code-first approach, EF builds the conceptual model based on your domain classes (entity classes), the context class and configurations.
- EF Core builds the storage model and mappings based on the provider you use. EF uses this model for CRUD (Create, Read, Update, Delete) operations to the underlying database.

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
 - Caching
 - Transaction management

Create Database From Model Using Entity Framework Core And ASP.NET Core

1. Add these two NuGet packages to the project:

- EntityFrameworkCore.SqlServer
- Microsoft.EntityFrameworkCore.Tools

2. In Models Folder Create a Class with Name as WebUser and add these lines of Codes

```
public class WebUser
{
    [Column("UserID")]
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    0 references
    public int UserID { get; set; }
    [Display(Name = "First Name")]
    [Required]
    [StringLength(25)]
    2 references
    public string FirstName { get; set; }
    [Required]
    [StringLength(25), MinLength(3)]
    [Display(Name = "Last Name")]
    1 reference
    public string LastName { get; set; }
    [EmailAddress]
    1 reference
    public string Email { get; set; }
}
```


Create Database From Model Using Entity Framework Core And ASP.NET Core

3. In Models Folder Create a custom DbContext class named AppDbContext and write the following code.

```
using Microsoft.EntityFrameworkCore;
namespace WebApplicationCoreS1.Models
{
    4 references
    public class AppDbContext : DbContext
    {
        0 references
        public AppDbContext(DbContextOptions <AppDbContext> options) : base(options)
        {
        }
        0 references
        public DbSet<WebUser> WebUsers { get; set; }
    }
}
```

Create Database From Model Using Entity Framework Core And ASP.NET Core

3. Build your project

4. Open the appsettings.json file and Add Database Connection string:

```
"ConnectionStrings": {  
  "DBConnectionString": "Data Source=.; Initial Catalog=DotNetCoreDBS; User Id=sa;  
Password =123456"  
}
```

Create Database From Model Using Entity Framework Core And ASP.NET Core

5. open the Startup class and add this code to the ConfigureServices() method.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
    services.AddDbContext<AppDbContext>(o =>
o.UseSqlServer(Configuration.GetConnectionString("DBConnectionString")));
}
```

The above code uses AddDbContext() method to register AppDbContext. Notice that the database connection string stored in the appsettings.json file is supplied to the UseSqlServer() method.

Create Database From Model Using Entity Framework Core And ASP.NET Core

6. Create Database using EnsureCreated() method

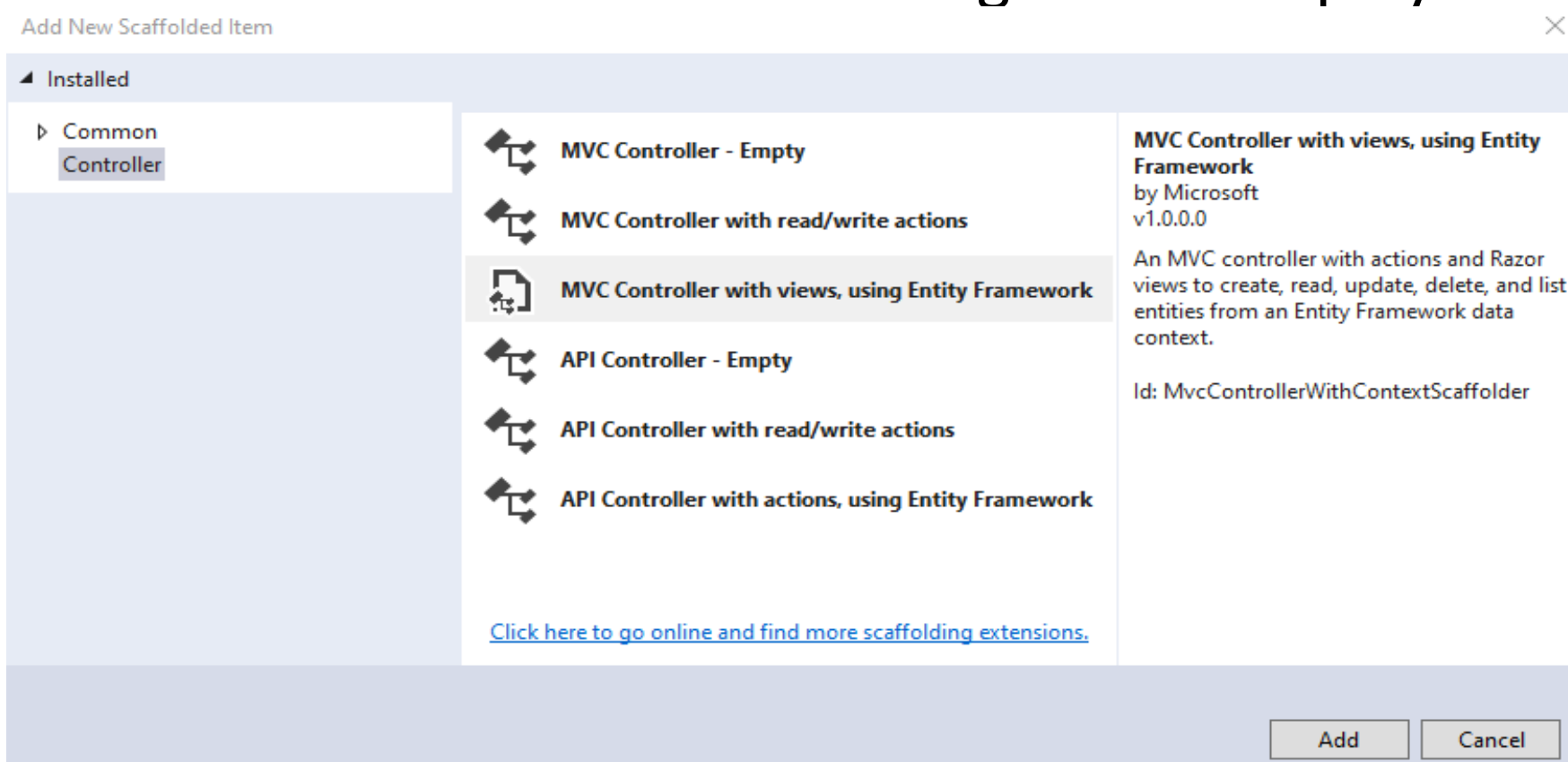
- EF Core model is ready, let's try to create the required database using EnsureCreated() method. This technique is a code based technique and works great for quick and simple database creation scenarios. If database is already exists, then no action is taken, otherwise database is created.
- Add following marked as bold in Configure Method

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env, AppDbContext db)  
{  
    .  
    db.Database.EnsureCreated();  
    app.UseStaticFiles();  
    app.UseRouting();  
    app.UseAuthorization();  
}
```

CRUD Operation Using Entity Framework Core

Create MVC Controller with views, using Entity Framework

- Right-click on the controller folder, select add new item, and then select controller. Then this dialog will be displayed.



CRUD Operation Using Entity Framework Core

Create MVC Controller with views, using Entity Framework

- Enter for Model Class, Data context class, Controller name as shown
- Tick Views as shown

Add MVC Controller with views, using Entity Framework

Model class: WebUser (WebApplicationCoreS1.Models)

Data context class: AppDbContext (WebApplicationCoreS1.Models)

Views:

- Generate views
- Reference script libraries
- Use a layout page:

~/Views/Shared/_Layout.cshtml

(Leave empty if it is set in a Razor _viewstart file)

Controller name: WebUsersController

Add Cancel

CRUD Operation Using Entity Framework Core

- Review your generated code in controller and view pages
- Load your controller in your browser
 - <https://localhost:44347/WebUsers>
 - <https://localhost:44347/WebUsers/Create>
- Click on Edit, Details and Delete

Index

[Create New](#)

First Name	Last Name	Email	
Admin	Lname	info@gmail.com	Edit Details Delete

Create

WebUser

First Name

Last Name

Email

[Create](#)

[Back to List](#)