Entity Framework Core

What is Entity framework

→It is Microsoft's ORM (Object-Relational Mapper) for .NET applications, which simplifies working with relational databases. An ORM allows developers to use object-oriented code to interact with database structures, like tables, without needing to write SQL queries directly.

Step to interact with Application to database with the help of Entity framework core

Step1:

Make Model Class : User

```
   public int Id { get; set; }
 public string? Name { get; set; }
 public string? Address { get; set; }
 public string? Phone { get; set; }
 public string? Email { get; set; }
```

You can specified the column name and datatype etc.,

```
public int Id { get; set; }
[Column("Username",TypeName ="varchar(100)")]
 public string? Name { get; set; }
[Column("Useraddress",TypeName ="varchar(100)")]
 public string? Address { get; set; }
[Column("Userphone",TypeName ="int")]
 public string? Phone { get; set; }
[Column("Useremail ,TypeName ="varchar(100)")]
 public string? Email { get; set; }
```

Step2:

Make DbContext : UserDbContext

```
 public class UserDbContext : DbContext {
 public UserDbContext (DbContextOptions<UserDbContext> options) :
base(options) { }

     public DbSet<User> Users { get; set; }
 }
```

What is DbContext

`DbContext` is the primary class that acts as a bridge between my .NET application and the database. It is used to configure and manage the database connection and provides methods to interact with the data stored in the database.

1. Database Connection

- `DbContext` manages the connection to the database. It uses a connection string provided in the configuration (like `appsettings.json`) to establish this connection.

2. Database Sessions

- Each `DbContext` instance represents a session with the database. This session is responsible for tracking changes to objects, which is essential for updating the database with any modifications made in the application.

3. Querying Data (Read)

- Through `DbContext`, you can query data from the database using **LINQ (Language Integrated Query)**.
- You define **DbSet properties** within your `DbContext` class to represent each table. These `DbSet`s allow you to query and fetch data directly from the database.

4. Saving Changes (Create, Update, Delete)

- `DbContext` can track changes made to objects and can save those changes back to the database. When you add, update, or delete an entity, the changes are recorded by the `DbContext`, which can then be saved to the database with `SaveChanges()` or `SaveChangesAsync()`.

5. Managing Transactions

- `DbContext` also manages transactions, ensuring that operations either fully succeed or fail, thus keeping data consistent. EF Core can use an implicit transaction for each `SaveChanges` call, but you can also manage transactions explicitly if needed.

6. Configuring Relationships and Constraints

- Using `DbContext`, you can configure relationships between tables (like one-to-many or many-to-many relationships), constraints, indexes, and other database rules in the database schema.

`What is this (DbContextOptions<UserDbContext> options) : base(options)`

`DbContextOptions<UserDbContext>`:

- This is a configuration object that carries information about how the `DbContext` should connect to the database, which database provider to use (like SQL Server, SQLite, etc.), and other options like logging, lazy loading, etc.
- `UserDbContext` in `DbContextOptions<UserDbContext>` specifies the type of context. Here, `UserDbContext` is the name of the `DbContext` class, which means that the options are specifically configured for this context.
- **`options` parameter**:
- The `options` parameter is an instance of `DbContextOptions<UserDbContext>` passed in when the `UserDbContext` is created. This parameter provides `UserDbContext` with the necessary information on how to connect to and interact with the database.
- The options are usually configured in the `Startup` class or `Program.cs` using dependency injection, allowing the database configuration to be managed from a central place in the application.
- **`: base(options)`**:
- This part of the constructor syntax is calling the base class constructor of `DbContext` with `options` as a parameter.
- `DbContext` is the parent class of `UserDbContext`, so using `: base(options)` ensures that the base class `DbContext` is properly configured with the options required for database connectivity.

What is this `public DbSet<User> Users { get; set; }`

DbSet<User> represents the User table in the database.

- **`Users` property** allows access to the data in this table.
- This property enables CRUD operations on the `User` table via Entity Framework Core without directly writing SQL queries.

Step 3:
Setup in appsettings.json

```
 "ConnectionStrings": {
   "EmployeePortal":
"Server=localhost;port=3306;Database=users;username=root;password
=admin@123 ",
   "SqlServerUserPortal": "Server=DESKTOP-K7PDDFH\\
SQLEXPRESS03;Database=UserPortal;User Id=sa;Password=admin@123;"
 },
```

Step 4:
Registering Connection String in Program.cs File.

```
builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseMySql(builder.Configuration.GetConnectionString("Emplo
yeePortal"),
    new MySqlServerVersion(new Version(8, 0, 21))));
```

Step 5:
Add a migration and run the migration

- Add-Migration Initial
- Update-database

Step 6:
Use in Controller :

```
private readonly Application2DbContext dbContext;
```

```
public UserController(Application2DbContext dbContext)
{
    this.dbContext = dbContext;
}
```