Az ORM rendszerek (Object-Relational Mapping rendszerek) olyan szoftveres eszközök, amelyek lehetővé teszik az objektumorientált programozási nyelvek és a relációs adatbázisok közötti könnyebb kommunikációt. Az ORM technológia az objektumorientált programokban használt osztályok (modellek) és a relációs adatbázisban található táblák közötti megfeleltetést (mappinget) végzi.

Mit csinál egy ORM rendszer?

Az ORM rendszerek automatikusan kezelik az adatbázis és az alkalmazás közötti adatleképezést. Ez azt jelenti, hogy:

Az adatbázis tábláit az ORM egyenértékű osztályokká alakítja a programkódban.

A táblák sorai az osztályok objektumaiként jelennek meg.

Az oszlopok a programbeli osztályok tulajdonságainak felelnek meg.

Az ORM rendszerek automatikusan végrehajtják az adatbázis-műveleteket (lekérdezés, beszúrás, módosítás, törlés) anélkül, hogy explicit SQL-parancsokat kellene írni.

==========

Az Entity Framework (EF) a Microsoft által fejlesztett Object-Relational Mapping (ORM) eszköz, amely megkönnyíti a relációs adatbázisok és a C# vagy .NET alapú alkalmazások közötti kommunikációt. Az Entity Framework az adatokkal végzett műveleteket (lekérdezés, módosítás, törlés stb.) egyszerűsíti azáltal, hogy a relációs adatbázisok tábláit és kapcsolatait objektumokká alakítja a kód szintjén.

Mi az Entity Framework?

Az Entity Framework egy ORM-eszköz, amely lehetővé teszi, hogy a relációs adatbázisokkal való munka során elkerüljük a nyers SQL parancsok írását. Ehelyett a .NET osztályokat használhatjuk az adatok kezelésére. Az EF az osztályok tulajdonságait automatikusan leképezi az adatbázis tábláira és mezőire.

==========

Mire jó az Entity Framework?

1. Adatbázisműveletek egyszerűsítése:

Az EF segítségével kevesebb kódot kell írni az adatbázis-műveletekhez, mivel az SQL parancsok helyett LINQ és C# objektumokat használhatunk.

Példa nélkül: Ahelyett, hogy SQL-lekérdezést írnánk, használhatunk LINQ-alapú lekérdezéseket, amelyek olvashatóbbak és könnyebben karbantarthatók.

2. Adatbázis és alkalmazás közötti kapcsolat kezelése:

Az EF automatikusan kezeli az adatbázis és az objektumok közötti kapcsolatokat (például idegen kulcsokat és relációkat).

3. Platformfüggetlenség biztosítása:

Az EF különböző adatbázis-kezelő rendszerekkel működik, például SQL Server, MariaDB, MySQL, PostgreSQL stb.

4. Adatmodellezés:

Az EF lehetővé teszi, hogy az adatmodellünk az alkalmazásunk logikai szerkezete legyen. A modellek C# osztályokként kezelhetők, így könnyebb dolgozni velük a kódban.

5. Karbantarthatóság és olvashatóság:

Az EF-nek köszönhetően az adatbázisműveletek kódja tisztább és érthetőbb, mivel az SQL-lekérdezések el vannak rejtve az absztrakció mögött.

==========

Az Entity Framework típusai

1. Code First:

Először az adatmodellt hozod létre a kódban, és az EF automatikusan generálja az adatbázist a modellek alapján.

2. Database First:

Már meglévő adatbázis esetén az EF automatikusan generálja a szükséges kódot (modelleket és DbContext osztályt).

3. Model First:

Először egy vizuális adatmodellt hozol létre az EF Designer segítségével, amely alapján generálódik az adatbázis.

==========

C# Entity Framework használata meglévő MariaDB adatbázissal

1. Új C# Projekt Létrehozása

- Hozz létre egy új konzolos vagy ASP.NET projektet a Visual Studio-ban vagy más fejlesztői környezetben.

- Győződj meg róla, hogy a projekt .NET 5.0 vagy újabb verzióra van beállítva, amely támogatja az Entity Framework Core-t.

2. MariaDB Connector és Entity Framework Telepítése

- Nyisd meg a NuGet Package Manager-t vagy használj parancssort.

- Telepítsd az alábbi csomagokat:

  Install-Package Pomelo.EntityFrameworkCore.MySql

  Install-Package Microsoft.EntityFrameworkCore.Tools

- A Pomelo.EntityFrameworkCore.MySql a MariaDB-hez optimalizált EF Core szolgáltatás.

3. Kapcsolódás a MariaDB Adatbázishoz

- Hozz létre egy Connection Stringet az appsettings.json fájlban vagy a kódon belül:

```
{
  "ConnectionStrings": {
    "MariaDBConnection":
"Server=localhost;Database=adatbazis_nev;User=root;Password=jelszo;"
  }
}
```

- Alternatív megoldás: Helyezd el közvetlenül a DbContext konfigurációjában.

4. DbContext Osztály Létrehozása

- Hozz létre egy osztályt, amely az DbContext osztályból származik:

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

    // DbSet példányok a meglévő táblákhoz
    public DbSet<User> Users { get; set; }
    public DbSet<Product> Products { get; set; }
}
```

- A meglévő adatbázis-táblákhoz modelleket kell definiálni, ha még nincsenek.

5. Modellek Generálása (Scaffolding)

- Futtass egy parancsot, amely automatikusan létrehozza a meglévő adatbázistáblákhoz a modelleket és a DbContext-et:

dotnet ef dbcontext scaffold "Server=localhost;Database=adatbazis_nev;User=root;Password=jelszo;" Pomelo.EntityFrameworkCore.MySql -o Models

- A parancs az összes meglévő tábla alapján létrehozza a modelleket és a DbContext osztályt a Models mappában.

- Ha csak bizonyos táblákat szeretnél generálni:

dotnet ef dbcontext scaffold "ConnectionString" Pomelo.EntityFrameworkCore.MySql -o Models --table TablaNev1 --table TablaNev2

6. DbContext Konfigurálása

- A Program.cs (vagy Startup.cs) fájlban állítsd be a DbContext-et:

```
var builder = WebApplication.CreateBuilder(args);


builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseMySql(
        builder.Configuration.GetConnectionString("MariaDBConnection"),
        new MySqlServerVersion(new Version(8, 2, 12)) // MariaDB verzió
    ));
```

7. Adatbázisműveletek Tesztelése

- Injektáld a DbContext-et a kódba:

```
using (var context = new AppDbContext(options))
{
    var users = context.Users.ToList(); // Adatok lekérdezése
}
```

8. További Konfigurációk és Finomhangolások

- Adatannotációk: A modellek tulajdonságait annotálhatod (pl. kulcs, egyedi mezők, hossz).

```
public class User
{
    [Key]
    public int Id { get; set; }

    [Required]
    [MaxLength(50)]
    public string Name { get; set; }
}
```

- Fluent API használata: Haladó konfigurációk a OnModelCreating metódusban.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<User>().HasKey(u => u.Id);
}
```

9. Migrációk Kezelése

- A meglévő adatbázis használata miatt a migrációk nem szükségesek az induláskor. Azonban új táblák és mezők hozzáadásakor a következő parancsokat használhatod:

- Új migráció létrehozása:

```
dotnet ef migrations add InitialMigration
```

- Migráció alkalmazása:

```
dotnet ef database update
```

10. Példa Futtatására

Egy egyszerű adatlekérdezés:

```
using (var context = new AppDbContext(options))
{
   var users = context.Users.Where(u => u.Name.Contains("John")).ToList();
   foreach (var user in users)
   {
      Console.WriteLine(user.Name);
   }
}
```