ORM i migracije

Pristup podacima iz programskog koda 25/26

Ishod učenja 5 - ORM i migracije

Code first pristup

→ Za razliku od database first pristupa, u code first pristupu prvo definiramo razrede (eng. *class*) koje će biti mapirane na tablice u relacijskoj bazi podataka

→ Često korišten pristup u praksi

→ Podržan od strane većine ORM alata (?)

Migracije

→ Skup promjena na shemi baze podataka

→ Potrebno je pratiti koje su promjene izvršene, kako bi shema uvijek bila ažurna

→ Često ju je moguće generirati automatski, putem definicija klasa i promjena

Entity Framework + migracije

→ Napraviti entitetske razrede, definirati poveznice po potrebi

→ Koristiti atribute ili `OnModelCreating` metodu u `DbContext` razredu

Korisni atributi

Dodaju korisne informacije putem deklarativnog pristupa, za stupce/članove:

→ `[Key]`

→ `[DatabaseGenerated(DatabaseGeneratedOption.Identity)]`

→ `[Column]`

→ `[Required]`

Za klase:

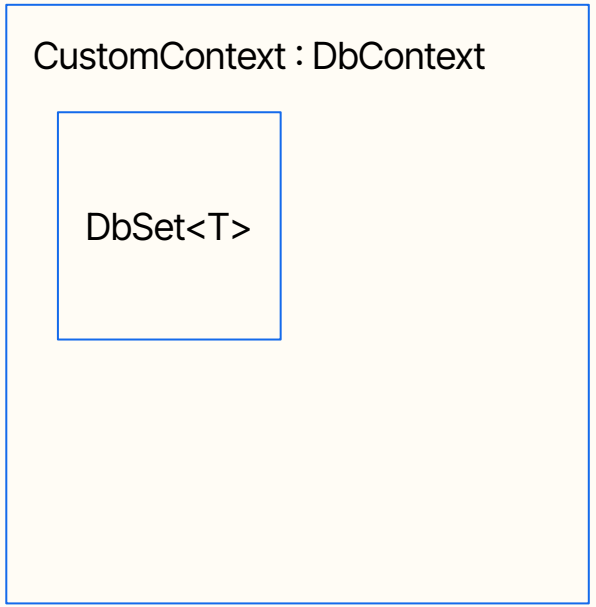
→ `[Table]`

https://learn.microsoft.com/en-us/dotnet/csharp/advanced-topics/reflection-and-attributes/attribute-tutorial

Setup

→ Nakon definicije razreda, potrebno je dodati instance `DBSet<T>` gdje je T entitetski razred

→ U `DbContext` potrebno je dodati definiciju spajanja na bazu (npr. `UseNpgsql(<cs>)`)



Prva migracija

Potrebno je napraviti migraciju:

```
dotnet ef migrations add <migration_name>
```

Potom je primijeniti:

```
dotnet ef database update
```

(pripazite da je instaliran `dotnet-ef` alat)

EF Migration History

\d "__EFMigrationsHistory"

```
Table "public.__EFMigrationsHistory"
 Column         | Type                         |  Nullable   | Default
----------------+------------------------------+-------------+---------
MigrationId     | character varying(150)   |  not null   |
ProductVersion | character varying(32)    |  not null   |
```

EF Migration tracking

Koristeći `__EFMigrationHistory` tablicu i `ModelSnapshot` klasu generiranu s migracijom, EF prati stanje trenutne sheme i promjene

Ovisno o tim promjenama, generira se migracija kako bi dovela shemu baze podataka u traženo stanje, zadano definicijom klasa - *code first*

Model snapshot

```
[DbContext(typeof(CustomDbContext))]
0 references
partial class CustomDbContextModelSnapshot : ModelSnapshot
{
    0 references
    protected override void BuildModel(ModelBuilder modelBuilder)
    {
        #pragma warning disable 612, 618
        modelBuilder
            .HasAnnotation("ProductVersion", "8.0.0")
            .HasAnnotation("Relational:MaxIdentifierLength", 63);

        NpgsqlModelBuilderExtensions.UseIdentityByDefaultColumns(modelBuilder);

        modelBuilder.Entity("CodeFirstExample.Data.Patient", b =>
            {
                b.Property<int>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("integer")
                    .HasColumnName("id");

                NpgsqlPropertyBuilderExtensions.UseIdentityByDefaultColumn(b.Property<int>("Id"));

                b.Property<DateTime>("DateOfBirth")
                    .HasColumnType("timestamp with time zone")
                    .HasColumnName("date_of_birth");

                b.Property<string>("FirstName")
                    .IsRequired()
                    .HasColumnType("text")
                    .HasColumnName("first_name");

                b.Property<string>("LastName")
                    .IsRequired()
                    .HasColumnType("text")
                    .HasColumnName("last_name");

                b.Property<char>("Sex")
                    .HasColumnType("character(1)")
                    .HasColumnName("sex");

                b.HasKey("Id");

                b.ToTable("patient");
            });
        #pragma warning restore 612, 618
    }
}
```

Migracijska klasa

→ Definira `Up` i `Down` proceduru

→ Parcijalna klasa, ostatak se nalazi u `*.Designer.cs` datoteci

```
/// <inheritdoc />
1 reference
public partial class InitialMigration : Migration
{
    /// <inheritdoc />
    0 references
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "patient",
            columns: table => new
            {
                id = table.Column<int>(type: "integer", nullable: false)
                    .Annotation("Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                first_name = table.Column<string>(type: "text", nullable: false),
                last_name = table.Column<string>(type: "text", nullable: false),
                sex = table.Column<char>(type: "character(1)", nullable: false),
                date_of_birth = table.Column<DateTime>(type: "timestamp with time zone", nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_patient", x => x.id);
            });
    }

    /// <inheritdoc />
    0 references
    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "patient");
    }
}
```

Primjer - patient manager

Napravite jednostavnu aplikaciju konzolnu aplikaciju koja vrši CRUD operacije nad pacijentom

- definiran s imenom, prezimenom, godinom rođenja i spolom

Upotrijebite *code first* pristup i usporedite ga s *database first* pristupom!

Primjer - nastavak

→ Proširite aplikativno rješenje da sprema podatke o pacijentovim oboljenjima koji imaju datum dijagnoze i mogući datum ozdravljenja

→ Stvorite i primijenite nove migracije !

Isprobati

→ LINQ upiti + sintaksa