ORM es una forma de mapear una BBDD relacional a clases. Es un reflejo en código.

Trabaja con Code First. Parte de clases y arma una BBDD.

Entity Framework es el ORM por defeco de .Net.

Crea proyecto de web Api con C#.

Crea otro proyecto, biblioteca de clases. Ahí va la DB. En ese proyecto instala entity framework sql server (NuGet). También entity framework tolos.

Cuando se trabaja con entity framework se tiene algo llamado contexto. Concentrador o núcleo de la BBDD.

A esa clase le enchufa el using:

using Microsoft.EntityFrameworkCore;

Y la hace heredera de DbContext.

using Microsoft.EntityFrameworkCore;

namespace BD

{

public class BolsaDeDatosContext:DbContext

{

public BolsaDeDatosContext(DbContextOptions<BolsaDeDatosContext> options) : base(options) { }

}

}

De momento queda así, igual que con Open Bootcamp.

A este proyecto le agrega otra clase “Brand”. Va a ser la tabla.

Le pone estos usings:

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

Con Identity se hace autoincrementable.

public class Brand

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int BrandId { get; set; }

public string Name { get; set; }

}

Otra clase mas, llamada Cerveza.

UTILIZA FOREIGN KEY!!

ACÁ LA PIFIÓ. HAGO AUTOINCREMENTABLA BRANDID Y NO CERVEZAID, QUE ES LA QUE DEBERÍA.

public class Cerveza

{

[Key]

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int BrandId { get; set; }

public string Name { get; set; }

public int CervezaId { get; set; }

[ForeignKey("CervezaId")]

public virtual Brand brand { get; set; }

}

A la clase Brand le agrega una colección de cervezas. Esto se puede porque ya se realizó la foreign key:

public virtual ICollection<Cerveza>cerveza { get; set; }

Agrega esto a la clase de contexto:

public DbSet<Brand>Brands { get; set; }

public DbSet<Cerveza> Cervezas { get; set;}

Añade, que esta es otra alternativa para lo mismo:

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Brand>().ToTable("Cerveza");

}

Hace que el proyecto ppal dependa del de BBDD. Agrega a ese primero, en su Json, el connectionStrings.

Aconseja poner eso, aunque no por qué.

MultipleActiveResultSets=True

En Program Cs se hace algo que antes se hacía en StartApp, para que la conexión llegue al Context, mediante inyección de dependencias a los controladores. Controladores, son la representación de los endpoint.

using BD;

using Microsoft.EntityFrameworkCore;

builder.Services.AddDbContext<BolsaDeDatosContext>(Options =>

{

Options.UseSqlServer(builder.Configuration.GetConnectionString("BolsaDeDatos"));

});

Crea un ámbito utilizando otro using:

using (var scope = app.Services.CreateScope())

{

var context=scope.ServiceProvider.GetRequiredService<BolsaDeDatosContext>();

context.Database.Migrate();

}

Ahí hace la migración, la cual crea la BBDD.

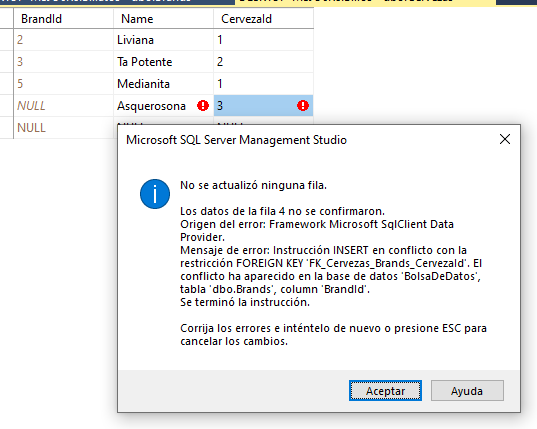
Señala que también hay que instalar el paquete nuget de entity framework tolos.

Va a la consola de administrador de paquetes, selecciona en proyecto determinado “DB” y ejecuta: add-Migration InitDB

Se crea la carpeta y todas sus cosas.

Al ejecutarse la App efectivamente se crea la BBDD.

Luego comenta la parte en la que crea la BBDD para que no borre y cree la BBDD cada vez que se ejecuta.



Para poder agregar una cerveza, primero tiene que haber una marca que la contenga. Cree 2 marcas, por eso las que tienen CervezaId 1 o 2 funcan, pero la que tiene 3 no.

Agrega controlador (Controlador de Api en blanco) al proyecto ppal, llamado BeerController.

public class CervezaController : ControllerBase

{

private BolsaDeDatosContext \_context;

public CervezaController(BolsaDeDatosContext context)

{

}

}

En el constructor recibe al contexto por parámetro. Eso es posible porque antes se hizo inyección de dependencias.

Luego le agrega este “método”:

[HttpGet]

public IEnumerable<Cerveza> Get() => \_context.Cervezas.ToList();

y ejecuta:

