**Общие сведения**

добавляем зависимости

1) EF Core под целевую СУБД - EntityFrameworkCore.SqlServer

2) Для создания БД из модели (Code first) - EntityFrameworkCore.Tools

3) Создаем сущности модели

public class User

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

}

4) Создаем контекст, в котором определяем настройку подключения к БД (переопределение OnConfiguring)

public class AppContext : DbContext

{

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer(@"Server=(localdb)\MSSQLLocalDB; Database=TestDB; Trusted\_Connection=True");

}

}

5) генерируем БД на основе модели:

для этого нужно зайти в Package Manager Console:

> Add-Migration Initial

6) выполняем Update-Database (создание базы)

Для создания модели из БД (database first)

1. EntityFrameworkCore.Design
2. заходим в PM и вводим команду

Scaffold-DbContext "Server=(localdb)\MSSQLLocalDB; Database=TestDB; Trusted\_Connection=true;" Microsoft.EntityFrameworkCore.SqlServer

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

Добавление информации

using (var db = new TestDBContext())

{

var u1 = new User() { Name = "", Age = 23 };

var u2 = new User() { Name = "", Age = 34 };

db.Users.Add(u1);

db.Users.Add(u2);

db.SaveChanges();

}

Настройка сущностей осуществляется через DataAnnotation или FluentAPI

Для того чтобы выполнить сопоставление модели и БД нужно выполнить миграцию.

Миграции позволяют вносить изменения в БД при изменениях моделей и контекста данных.

Для создания миграции в окне Packege Manager Console вводится следующая команда:

1. **Add-Migration название\_миграции**

Название миграции представляет произвольное название, главное чтобы все миграции в проекте имели разные названия.

После создания миграции ее нужно ВЫПОЛНИТЬ с помощью следующей команды:

1. **Update-Database**

**Определение моделей**

Включение сущностей в модель:

1. По умолчанию все типы сущностей, для которых определены в контексте данных наборы **DbSet**, включаются в модель и в дальнейшем сопоставляются с таблицами в базе данных.
2. Включения сущности в модель представляет вызов **Entity()**

protected override void OnModelCreating(ModelBuilder mb)

{

Mb.Entity<Имя\_сущности>();

}

**Исключение из модели**

Применение Fluent API заключается в вызове метода **Ignore();**

protected override void OnModelCreating(ModelBuilder mb)

{

Mb.Ignore<Имя\_сущности>(); // Таблица не будет создана

}

Аннотации данных предлагают установку над классом атрибута **[NotMapped]**

**Также можно ограничить сопоставление свойств**

protected override void OnModelCreating(ModelBuilder mb)

{

Mb.Entity<Имя\_сущности>().Ignore(p=>p.Имя\_игнорируемого\_свойства);

}

Или просто пометить атрибутом [NotMapped]

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

[Table] – имя таблицы в БД

[Column] – имя стобца в БД

.HasName(“ИмяСтобцавБД”)

[Key] – первичный ключ

.HasKey(u=>u.\_имя\_стобца\_) // Fluent API add primary key

// пример //

protected override void OnModelCreating(ModelBulider modelBuilder)

{

modelBuilder<User>().HasKey(u=>u.Number);

}

Для использования альтернативного первичного ключа:

.HasAlternateKey(u=>u.sth);

Или составной ключ:

.HasAlternateKey(u=> new {u.Passport, u.PhoneNumber});

Установка индекса:

.HasIndex(u=>u.Passport).IsUnique();

Чтобы избежать случая, когда столбцы содержат тип NULL нужно использовать:

[Required] | .IsRequired();

Ограничение длины типа:

[MaxLenght(50)] [MinLenght(50)] | .HasMaxLenght(50) .HasMinLenght(50)

Для непосредственного использования конкретного типа данных для стоблца БД:

[Column(TypeName = “varchar(100)”)] | .HasColumnType(“varchar(100)”);