

Введение в EntityFramework Core

Введение в EntityFramework Core

Тема лекции

Введение в EntityFramework Core

Цели и задачи

Изучить основы вреймворка EntityFramework Core

Научиться использовать контекст базы данных для формирования запросов.

Введение в EntityFramework Core

Общая информация

Введение в EntityFramework Core

Entity Framework (EF) Core - это облегченная, расширяемая кроссплатформенная версия популярной технологии доступа к данным Entity Framework с открытым исходным кодом.

Введение в EntityFramework Core

EF Core представляет собой **ORM**-инструмент (object-relational mapping - отображение данных на реальные объекты), который:

- ❑ Позволяет разработчикам .NET работать с базой данных с помощью объектов .NET.
- ❑ Устраняет потребность в большей части кода доступа к данным, который обычно необходимо писать.

Введение в EntityFramework Core

EF Core поддерживает работы с разными базами данных. Для этого используются соответствующие поставщики баз данных.

Подробнее о поставщиках БД – см.

<https://docs.microsoft.com/en-us/ef/core/providers/?tabs=dotnet-core-cli>

Введение в EntityFramework Core

В EF Core доступ к данным осуществляется с помощью модели. Модель состоит из классов **сущностей** и объекта **контекста**, который предоставляет взаимодействие с базой данных.

Объект контекста позволяет запрашивать и сохранять данные.

Введение в EntityFramework Core

Базовые классы описаны в библиотеке

Microsoft.EntityFrameworkCore

Вспомогательные классы (например, для создания миграций) описаны в библиотеке

Microsoft.EntityFrameworkCore.Tools

Введение в EntityFramework Core

Поставщики данных для различных СУБД описаны в соответствующих библиотеках, например:

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Sqlite

Pomelo.EntityFrameworkCore.MySql

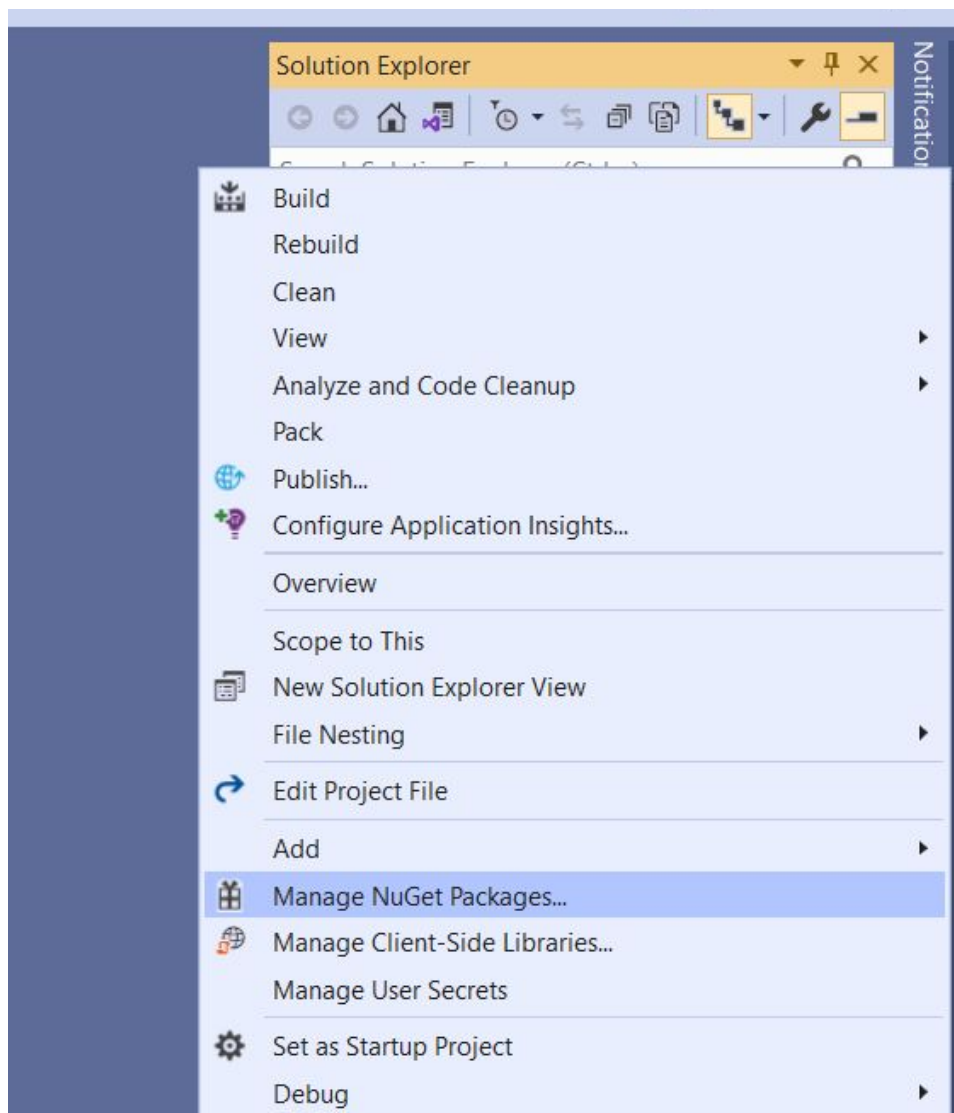
Npgsql.EntityFrameworkCore.PostgreSQL

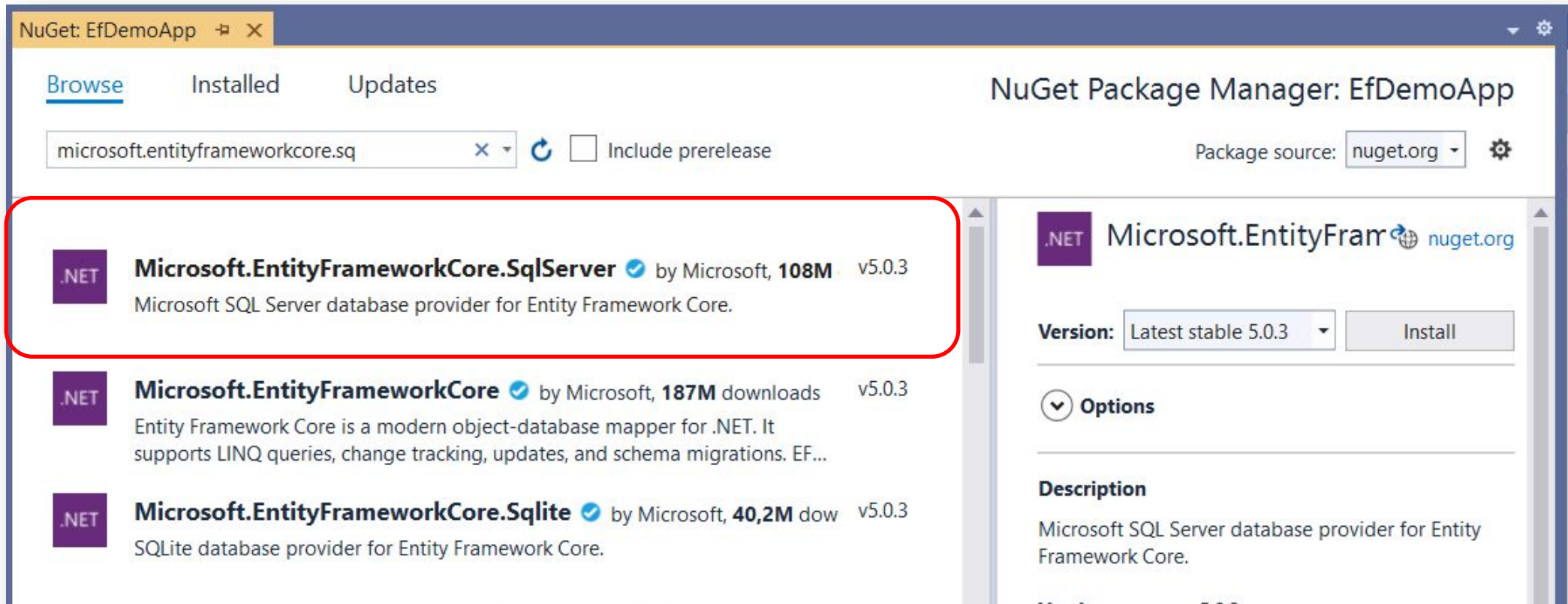
Введение в EntityFramework Core

EF поддерживает следующие подходы к разработке моделей:

- ❑ Создание модели из существующей базы данных.
- ❑ Описание модели. Затем, используя EF Migrations создается база данных из модели. Миграции позволяют развивать базу данных по мере изменения модели.

Введение в EntityFramework Core





Введение в EntityFramework Core

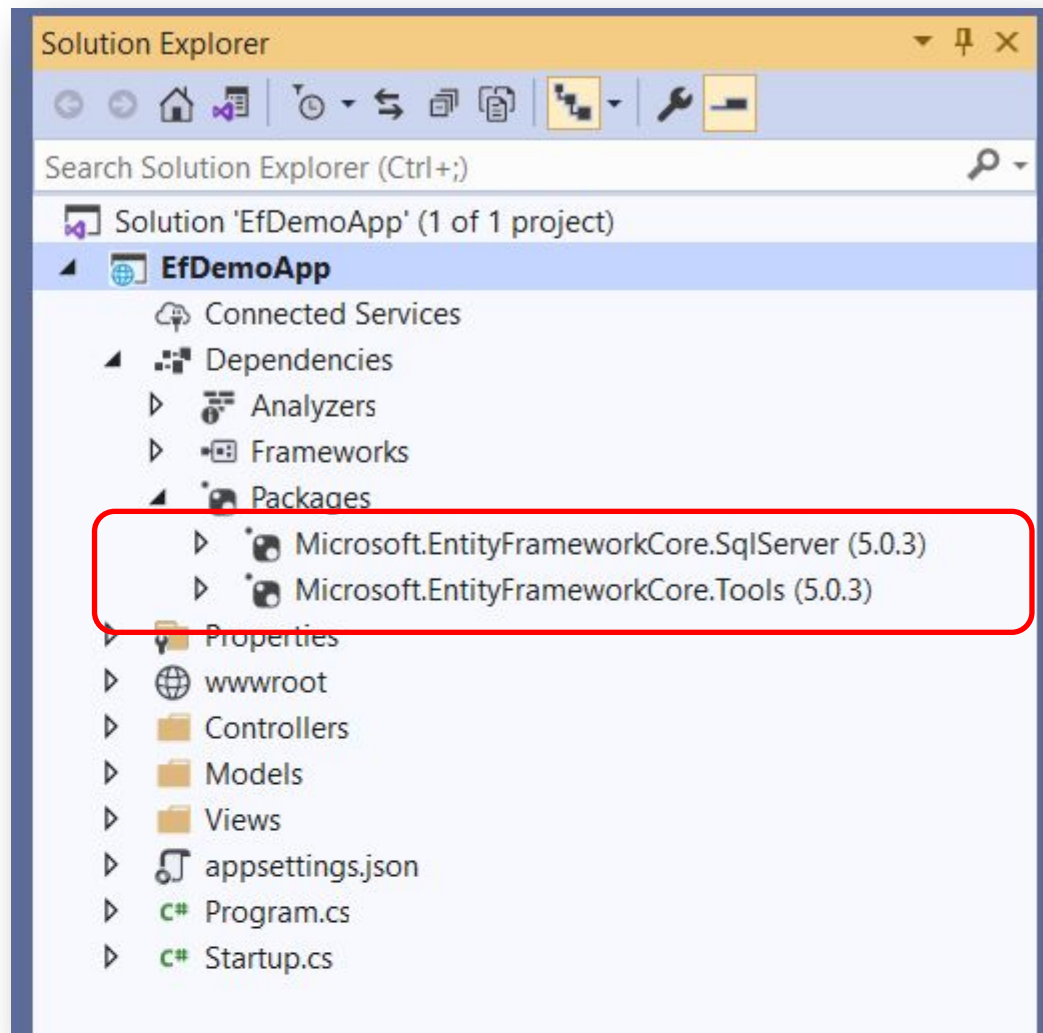


Схема базы данных

Введение в EntityFramework Core

Описание сущностей

Описание сущностей

```
public class Person
{
    public int PersonId { get; set; }
    public string Name { get; set; }
    // НАВИГАЦИОННЫЕ СВОЙСТВА
    public int DepartmentId { get; set; }
    public Department Department { get; set; }
}

public class Department
{
    public int Id { get; set; }
    public string Name { get; set; }
    // НАВИГАЦИОННЫЕ СВОЙСТВА
    public List<Person> Persons { get; set; }
}
```


Введение в EntityFramework Core

Описание контекста базы данных

Контекст базы данных

Экземпляр **DbContext** предоставляет сеанс с базой данных и может использоваться для запроса и сохранения экземпляров сущностей.

DbContext - это комбинация шаблонов **Unit Of Work** и **Repository**.

Контекст базы данных

DbContext отслеживает изменения объектов при:

- получении объектов из запроса
- добавлении/изменении объекта

Привызове метода **SaveChanges** или **SaveChangesAsync** EF Core обнаруживает внесенные изменения и записывает их в базу данных.

Контекст базы данных

```
public class ApplicationDbContext : DbContext
{
    public DbSet<Person> Persons { get; set; }
    public DbSet<Department> Departments { get; set; }
}
```

Контекст базы данных

DbSet/DbSet<TEntity>: представляет набор объектов, которые отображаются на таблицы базы данных

Контекст базы данных

DbSet/DbSet<TEntity>: представляет набор объектов, которые отображаются на таблицы базы данных

DbSet позволяет делать выборку данных с помощью запросов LINQ.

EF Core преобразует эти запросы в запросы к БД, например, в запросы SQL

Введение в EntityFramework Core

Конфигурирование контекста БД

Конфигурирование контекста БД (вариант1)

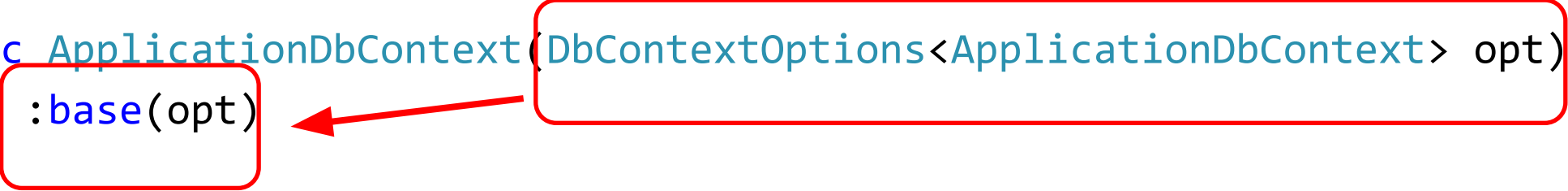
```
public class ApplicationDbContext : DbContext
{
    private string connectionString;
    public ApplicationDbContext(string connStr)
    {
        connectionString = connStr;
    }
    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {
        optionsBuilder.UseSqlServer(connectionString);
    }
    public DbSet<Person> Persons { get; set; }
    public DbSet<Department> Departments { get; set; }
}
```


Конфигурирование контекста БД (Вариант 2)

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> opt)
        :base(opt)
    {

    }

    public DbSet<Person> Persons { get; set; }
    public DbSet<Department> Departments { get; set; }
}
```



The diagram illustrates the relationship between the constructor parameter and the base class constructor call. A red box highlights the parameter `DbContextOptions<ApplicationDbContext> opt` in the constructor signature. Another red box highlights the `:base(opt)` call in the constructor body. A red arrow points from the `opt` parameter to the `opt` argument in the `:base(opt)` call, indicating that the parameter is passed to the base class constructor.

Конфигурирование контекста БД (Вариант 2)

```
var connStr = @"Server=(localdb)\mssqllocaldb; Database = EfDemo;"
              + "Trusted_Connection = True; "
              + "MultipleActiveResultSets = true";

var builder = new DbContextOptionsBuilder<ApplicationDbContext>();
var options = builder.UseSqlServer(connStr).Options;

var context = new ApplicationDbContext(options);
```

Введение в EntityFramework Core

Ключевые поля

Ключевые поля

По умолчанию свойство с именем Id или <имя типа> Id будет настроено как первичный ключ сущности.

```
public class Person
{
    public int PersonId { get; set; }
    . . .
}
```

```
public class Department
{
    public int Id { get; set; }
    . . .
}
```

Настройка ключевых полей в коде (аннотация данных)

```
using System.ComponentModel.DataAnnotations;  
public class Person  
{  
    [Key]  
    public string PassNo { get; set; }  
    . . .  
}
```

Настройка ключевых полей в коде (fluent API, в классе контекста БД)

```
protected override void OnModelCreating(ModelBuilder  
builder)  
{  
    builder.Entity<Person>()  
        .HasKey(p=>p.PassNo);  
}
```

Введение в EntityFramework Core

Отношения между таблицами

Отношения между таблицами (основные термины)

- ❑ **Dependent entity** (Зависимая сущность): это сущность, которая содержит свойства внешнего ключа. Иногда ее называют «child (дочерней)».
- ❑ **Principal entity** (Основная сущность): это сущность, которая содержит свойства первичного / альтернативного ключа. Иногда ее называют «parent (родительской)»
- ❑ **Primary Key** (Главный ключ): свойства, которые однозначно идентифицируют основную сущность.
- ❑ **Foreign Key** (Внешний ключ): свойства в зависимой сущности, которые используются для хранения значений основного ключа для связанной сущности
- ❑ **Navigation Property** (Свойство навигации): свойство, определенное для основной и / или зависимой сущности, которое ссылается на связанную сущность

Отношения между таблицами (основные термины)

- ❑ **Collection Navigation Property** (Свойство навигации коллекции): свойство навигации, которое содержит ссылки на множество связанных сущностей.
- ❑ **Reference navigation property** (Свойство навигации по ссылке): свойство навигации, которое содержит ссылку на один связанный объект.
- ❑ **Inverse navigation property** (Свойство обратной навигации): при обсуждении конкретного свойства навигации этот термин относится к свойству навигации на другом конце отношения.
- ❑ **Self-referencing relationship** (Отношение со ссылками на себя): отношение, в котором типы зависимой и основной сущностей совпадают.

Отношения между таблицами (один-ко-многим)

- ❑ **Department** – основная сущность (Principal)
- ❑ **Person** – зависимая сущность (Dependent)
- ❑ **Department.Id** – основной ключ (Principal key)
– совпадает с главным ключом (Primary key)
- ❑ **Person.DepartmentId** – внешний ключ (Foreign key)
- ❑ **Person.Department** - Свойство навигации по ссылке (Reference navigation property), а также свойство обратной навигации (Inverse navigation property) для Department.Persons
- ❑ **Department.Persons** - Свойство навигации коллекции (Collection Navigation Property)



Отношения между таблицами (один-ко-многим)

Если EF Core обнаруживает навигационные свойства, и имена свойств соответствуют одному из правил:

<имя навигационного свойства>Id


<имя родительской сущности>Id

то связи и вторичные ключи в БД формируются автоматически

Настройка отношений вручную (Аннотация данных)

```
public class Person
{
    . . .
    // НАВИГАЦИОННЫЕ СВОЙСТВА
    public int DepartmentId { get; set; }
    public Department Department { get; set; }
}

public class Department
{
    public int Id { get; set; }
    public string Name { get; set; }
    // НАВИГАЦИОННЫЕ СВОЙСТВА
    [InverseProperty("Department")]
    public List<Person> Persons { get; set; }
}
```



Настройка отношений вручную (fluent API)

```
protected override void OnModelCreating(ModelBuilder
modelBuilder)
{
    modelBuilder.Entity<Person>()
        .HasOne(p => p.Department)
        .WithMany(d => d.Persons);
}
```

Введение в EntityFramework Core

Подключение контекста БД в ASP.Net Core

Подключение контекста БД в ASP.Net Core

файл appsettings.json:

```
"ConnectionStrings": {  
    "Default": "Server=(localdb)\\mssqllocaldb;  
Database = EfDemo; Trusted_Connection = True;  
MultipleActiveResultSets = true"  
},
```

Подключение контекста БД в ASP.Net Core

файл startup.cs:

```
services.AddDbContext<ApplicationDbContext>(opt => {  
    opt.UseSqlServer(Configuration  
        .GetConnectionString("Default"));  
});
```

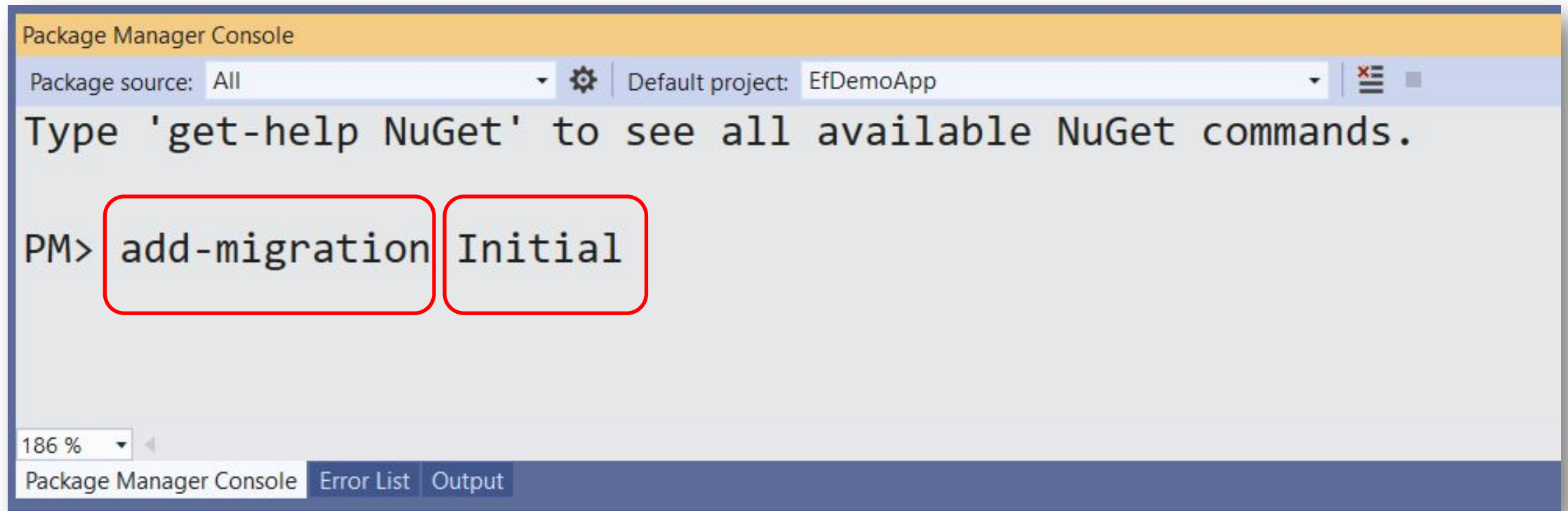

Введение в EntityFramework Core

Migrations

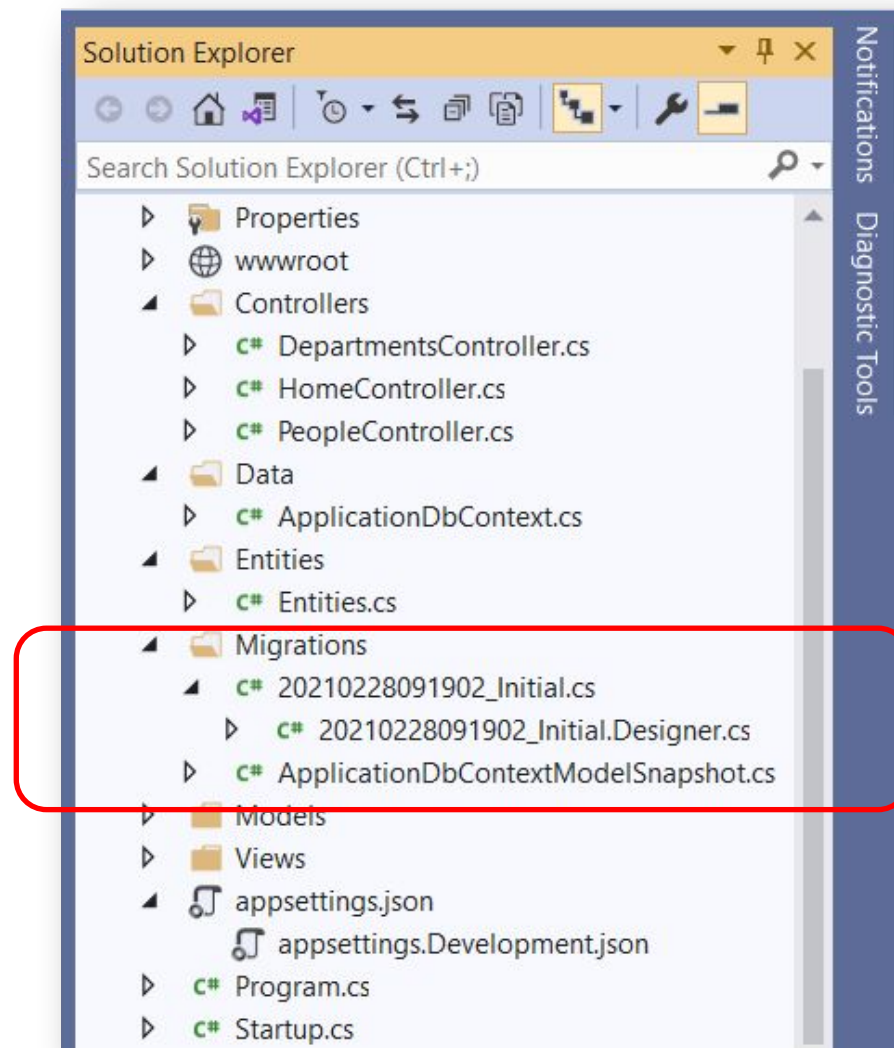
Migrations

Миграции в EF Core предоставляет способ постепенного обновления схемы базы данных, чтобы поддерживать ее синхронизацию с моделью данных приложения, сохраняя при этом существующие данные в базе данных.

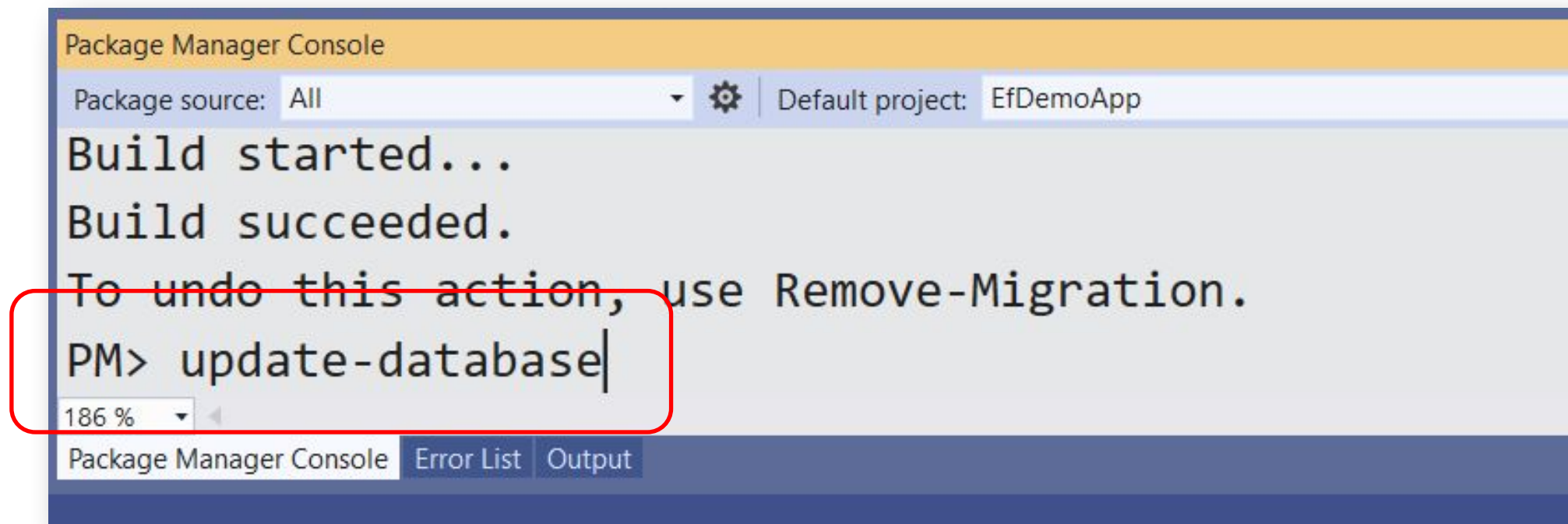
Migrations (добавление миграции)

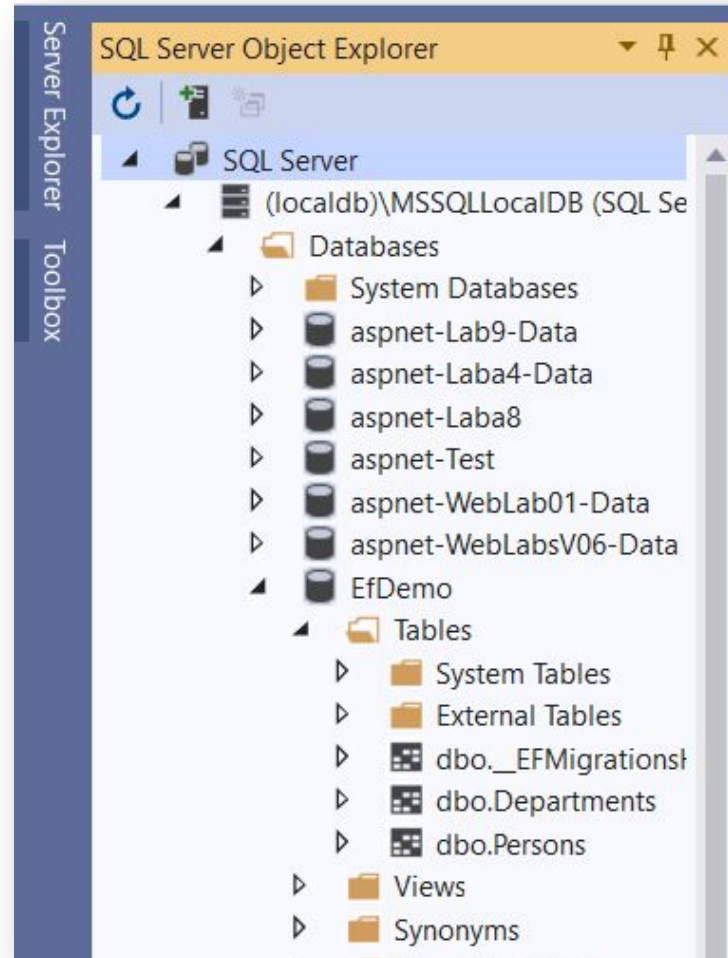


Migrations (добавление миграции)



Применение миграции

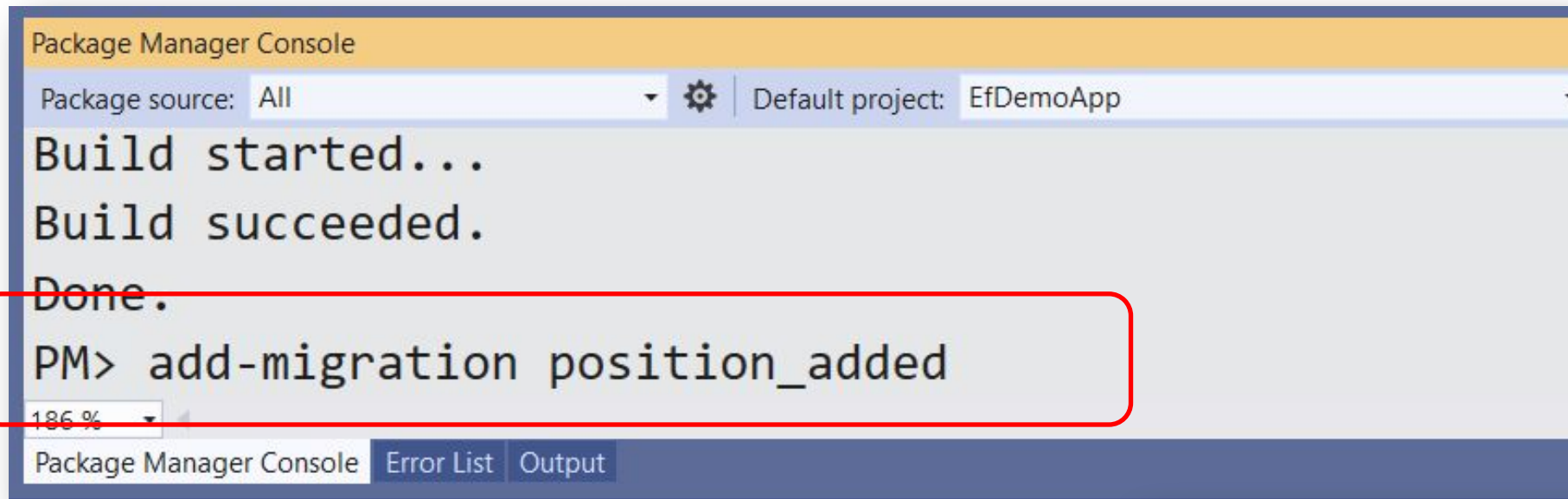




Добавление новой миграции

```
public class Person
{
    public string PersonId { get; set; }
    public string Name { get; set; }
    public string Position { get; set; }
    // НАВИГАЦИОННЫЕ СВОЙСТВА
    public int DepartmentId { get; set; }
    public Department Department { get; set; }
}
```

Добавление новой миграции



Package Manager Console

Package source: All | Default project: EfDemoApp

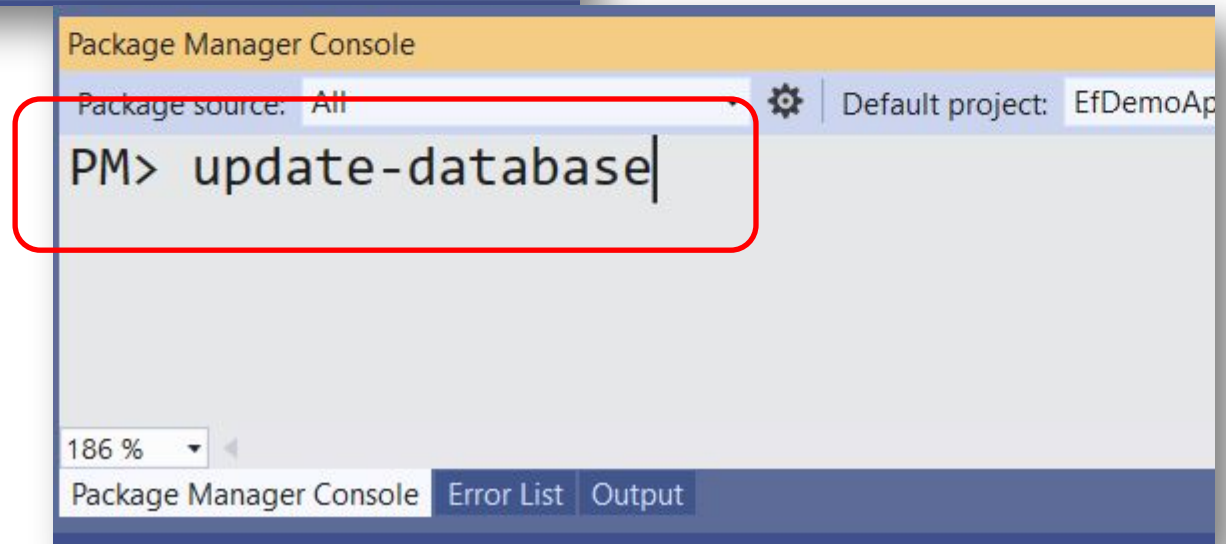
Build started...
Build succeeded.
Done.

PM> add-migration position_added

186 %

Package Manager Console | Error List | Output

This screenshot shows the Package Manager Console window. The top bar is orange and contains the title 'Package Manager Console'. Below it, there are two dropdown menus: 'Package source: All' and 'Default project: EfDemoApp'. The main area is light gray and contains the text 'Build started...', 'Build succeeded.', and 'Done.'. A red rounded rectangle highlights the command 'PM> add-migration position_added' entered in the console. At the bottom, there is a progress bar showing '186 %' and a tab bar with 'Package Manager Console', 'Error List', and 'Output'.



Package Manager Console

Package source: All | Default project: EfDemoApp

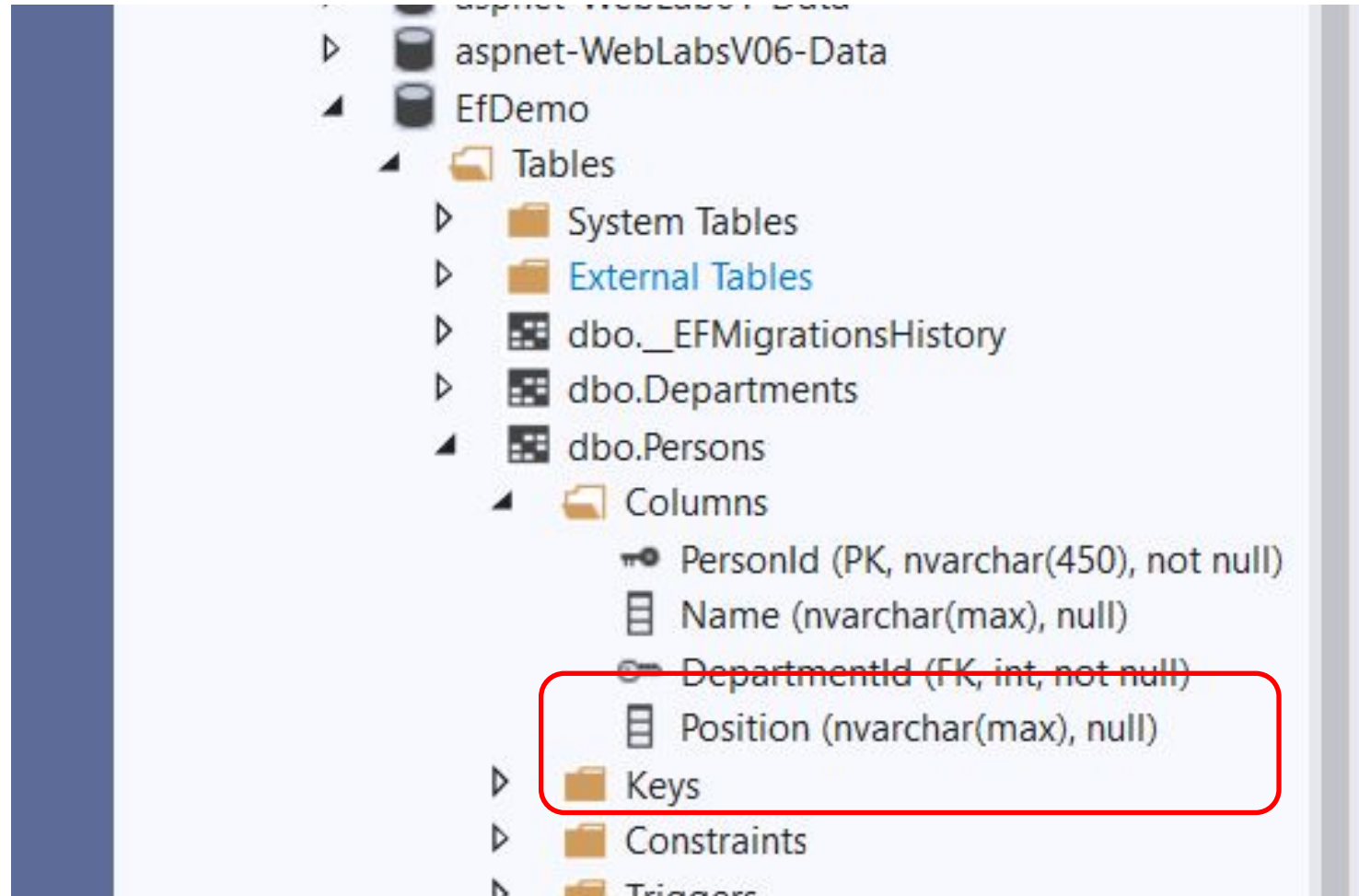
PM> update-database

186 %

Package Manager Console | Error List | Output

This screenshot shows the Package Manager Console window. The top bar is orange and contains the title 'Package Manager Console'. Below it, there are two dropdown menus: 'Package source: All' and 'Default project: EfDemoApp'. The main area is light gray and contains the command 'PM> update-database' entered in the console. A red rounded rectangle highlights this command. At the bottom, there is a progress bar showing '186 %' and a tab bar with 'Package Manager Console', 'Error List', and 'Output'.

Добавление новой миграции



Введение в EntityFramework Core

Запросы к контексту БД

Entity Framework Core использует встроенный в язык запросов (LINQ) для запроса данных из базы данных. LINQ позволяет использовать C # (или любой другой язык .NET) для написания строго типизированных запросов.

EF Core передает запрос LINQ поставщику базы данных. Поставщики баз данных, в свою очередь, переводят его на язык запросов, специфичный для базы данных (например, SQL для реляционной базы данных).

Получение всех данных

```
var context = new ApplicationDbContext(options);  
var persons = context.Persons.ToList();
```

Фильтрация данных

```
var persons = context.Persons  
    .Where(p => p.Name == "Bob");
```

Поиск данных

```
var person = context.Persons  
                .SingleOrDefault(p => p.Name == "Bob");
```

ПОИСК ПО КЛЮЧУ

```
var person = context.Persons.Find(2);
```

Получение связанных данных

```
var persons = context.Persons  
    .Include(p => p.Department);
```