Миграции

Последнее обновление: 27.09.2016

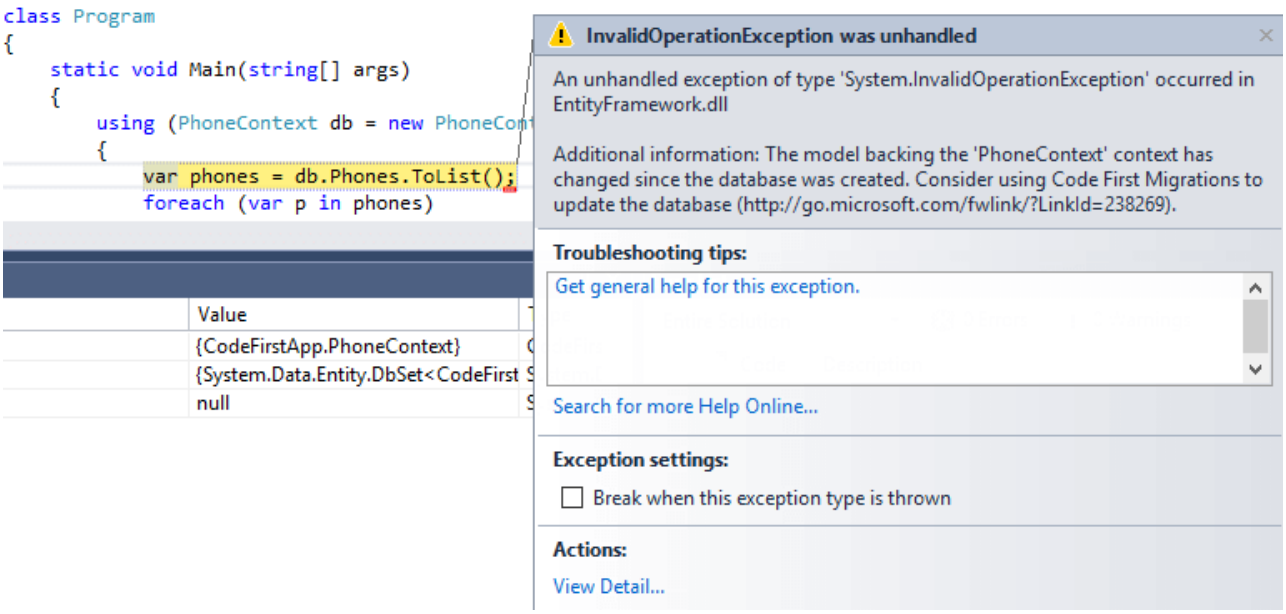
Миграции позоляют вносить изменения в базу данных при изменениях моделей и контекста данных. Так, пусть у нас есть следующая модель Phone и контекст данных:

```
public class PhoneContext : DbContext
{
        public DbSet<Phone> Phones { get; set; }
    public PhoneContext() : base("DefaultConnection")
    { }
}

public class Phone
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Price { get; set; }
}
```

Мы можем использовать этот контекст данных для работы с БД, добавлять и удалять данные. Но в какой-то момент, возможно, нам захочется что-то изменить, например, добавить в модель Phone новое свойство:

```
public class Phone
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Company { get; set; }
    public int Price { get; set; }
}
```

И если мы попытаемся обратиться к базе данных после изменения модели, то мы получим ошибку:



Для решения этой проблемы нам надо изменить базу данных таким образом, чтобы она вновь соответствовала моделям и контексту данных. И для этого мы можем применить миграции.

Для этого в Visual Studio перейдем к окну **Package Manager Console**, которое можно найти внизу VS. Если такого окна нет, то его можно открыть, перейдя к меню **View**->**Other Window**->**Package Manager Console**

Для добавления функционала миграций введем в это окно следующую команду:

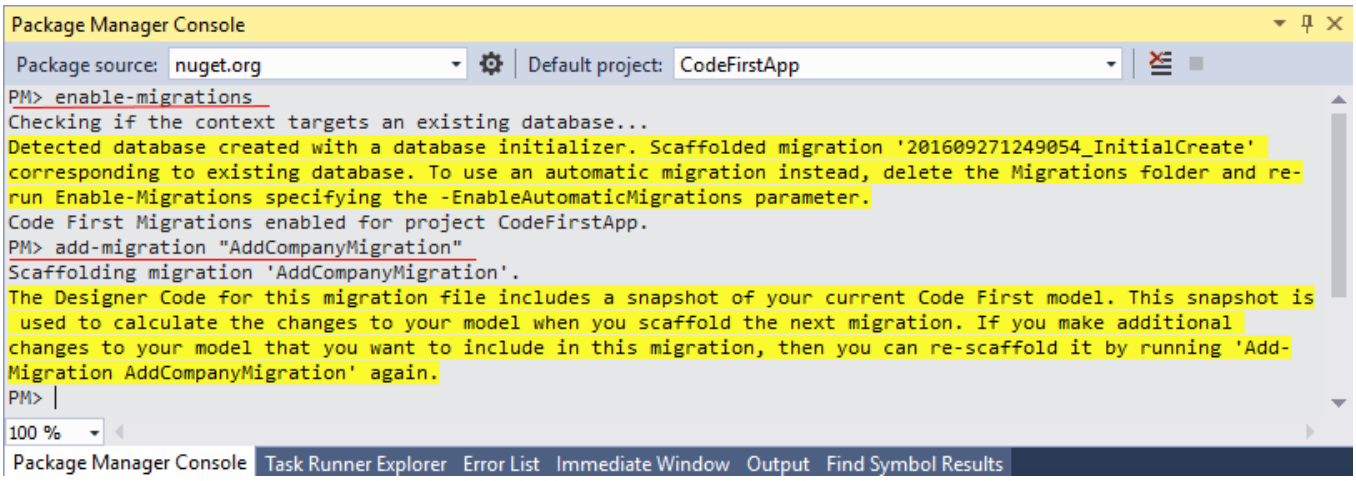
```
enable-migrations
```

После ввода команды нажмем на Enter. И в результате выполнения данной команды в проект будет добавлена папка *Migrations*, в которой будут два файла: *Configration.cs* (содержит базовую конфигурацию миграций) и файл начальной миграции, название которого может отличаться. Файл начальной миграции устанавливает, как база данных определяется на данный момент. То есть в моем случае он будет выглядеть так:

```
namespace CodeFirstApp.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class InitialCreate : DbMigration
    {
        public override void Up()
        {
            CreateTable(
                "dbo.Phones",
                c => new
                    {
                        Id = c.Int(nullable: false, identity: true),
                        Name = c.String(),
                        Price = c.Int(nullable: false),
                    })
                .PrimaryKey(t => t.Id);

        }

        public override void Down()
        {
            DropTable("dbo.Phones");
        }
    }
}
```

Далее выполним в Package Manager Console следующую команду:

```
add-migration "AddCompanyMigration"
```



И после выполнения этой команды в папку Migrations будет добавлена новая миграция:

```
using System;
using System.Data.Entity.Migrations;

public partial class AddCompanyMigration : DbMigration
{
    public override void Up()
    {
        AddColumn("dbo.Phones", "Company", c => c.String());
    }

    public override void Down()
    {
        DropColumn("dbo.Phones", "Company");
    }
}
```

В миграции определяются два метода: `Up()` и `Down()`. В методе Up с помощью вызова метода AddColumn добавляется новый столбец Company в уже имеющуюся таблицу dbo.Phones. Метод Down удаляет столбец на случай, если они существуют. Фактически эти методы равнозначные выражению ALTER в языке SQL, которое меняет структуру базы данных и ее таблиц.

И в завершении чтобы выполнить миграцию, применим этот класс, набрав в той же консоли команду:

```
update-database
```

Эта команда обновит базу данных, добавив в нее новый столбец. Причем данные, которые уже были в таблицы, сохранятся.

Если база данных уже используется в производстве, развернута на сервере, где бы не можем произвести миграции, то мы можем сгенерировать по миграции скрипт. Для этого надо ввести следующую команду:

```
update-database -script
```

В итоге в моем случае будет сгенерирован следующий скрипт SQL:

```
ALTER TABLE [dbo].[Phones] ADD [Company] [nvarchar](max)
INSERT [dbo].[__MigrationHistory]([MigrationId], [ContextKey], [Model], [ProductVersion])
VALUES (N'201609271310287_AddCompanyMigration', N'CodeFirstApp.PhoneContext',
0x1F8B0800000000000400CD57DB6EDB38107D2FB0FF207CC35C38C0E0000 , N'6.1.3-40302')
```

Затем данный скрипт можно выполнить для используемой базы данных.

Методы миграций

Основу миграции составляют ряд методов, которые позволяют удалять, добавлять столбцы и таблицы, изменять настройки столбцов и так далее. Основные методы:

- **CreateTable**: добавляет таблицу
- **DropTable**: удаляет таблицу
- **AddColumn**: добавляет столбец
- **DropColumn**: удаляет столбец
- **AlterColumn**: изменяет настройки столбца
- **AddForeignKey**: добавляет внешний ключ
- **DropForeignKey**: удаляет внешний ключ
- **AddPrimaryKey**: добавляет первичный ключ
- **DropPrimaryKey**: удаляет первичный ключ
- **CreateIndex**: добавляет индекс
- **DropIndex**: удаляет индекс
- **CreateStoredProcedure**: создает хранимую процедуру
- **DropStoredProcedure**: удаляет хранимую процедуру
- **AlterStoredProcedure**: изменяет хранимую процедуру

Далее от этих методов можно строить цепочки дополнительных методов. Например, создание таблицы:

```
CreateTable(
    "dbo.Companies",   // название таблицы
    c => new    // столбцы
    {
        Id = c.Int(nullable: false, identity: true),
        Name = c.String(),
    })
    .PrimaryKey(t => t.Id); // первичный ключ
```

Назад **Содержание** **Вперед**