Хранимые функции

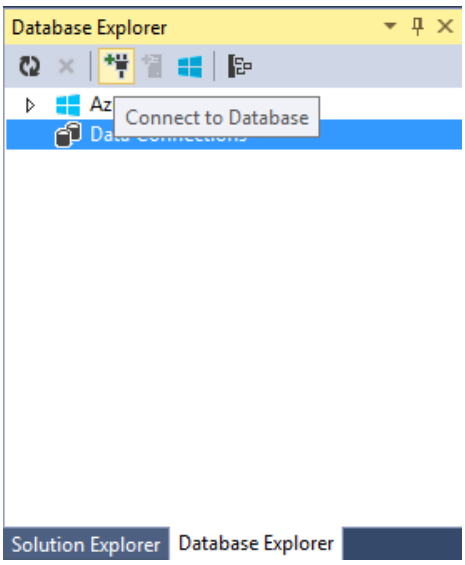
Последнее обновление: 31.10.2015

Особое внимание при работе с sql-запросами представляют хранимые функции и процедуры. Рассмотрим вызов хранимой функции в приложении на C#.

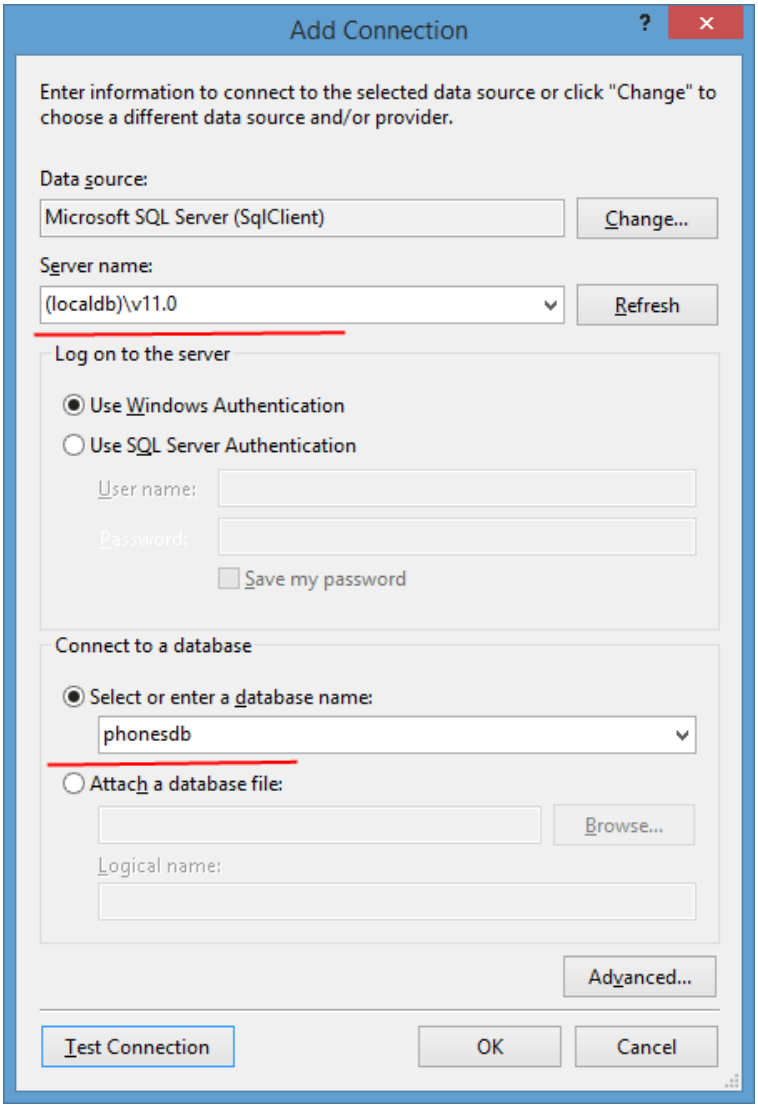
Вначале создадим функцию. Пусть наша база данных описывается следующим контекстом данных и моделями:

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

public class Company
{
    public int Id { get; set; }
    public string Name { get; set; }
    public ICollection<Phone> Phones { get; set; }
    public Company()
    {
        Phones = new List<Phone>();
    }
}

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

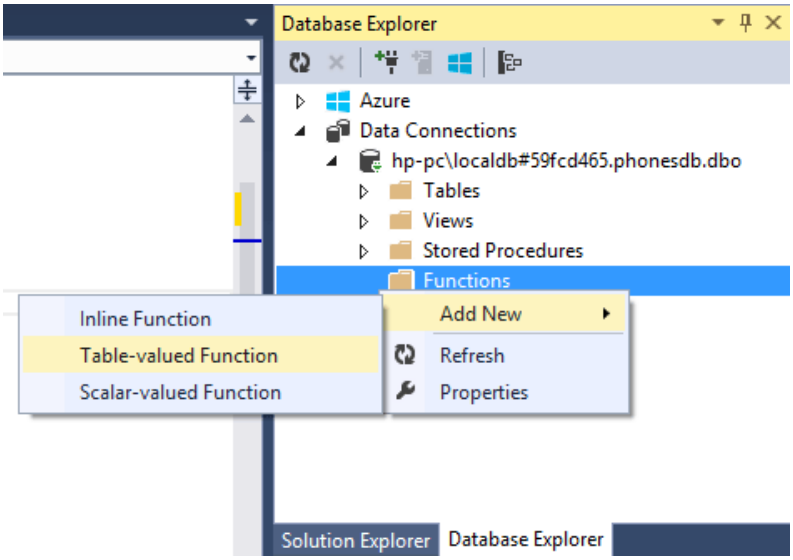
Теперь в Visual Studio в окне Database Explorer откроем базу данных. Для этого нажмем в окне Database Explorer на кнопку **Connect to Database**:



Выберем нужную базу данных. В моем случае это база данных phonesdb:



После открытия базы данных в окне Database Explorer найдем узел Functions и нажмем на него правой кнопкой мыши. В появившемся контекстном меню выберем **Add New -> Table-valued Function**:



После этого Visual Studio генерирует и октрывает файл скрипта со следующим содержимым:

```
CREATE FUNCTION [dbo].[Function]
(
	@param1 int,
	@param2 char(5)
)
RETURNS @returntable TABLE
(
	c1 int,
	c2 char(5)
)
AS
BEGIN
	INSERT @returntable
	SELECT @param1, @param2
	RETURN
END
```

Изменим скрипт следующим образом:

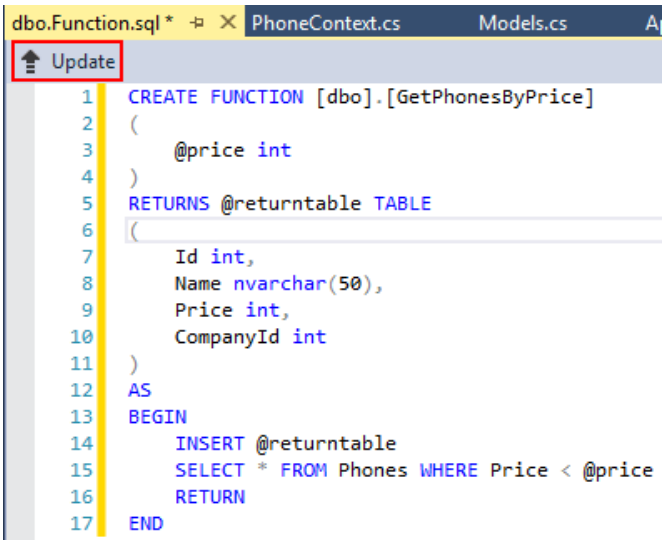
```
CREATE FUNCTION [dbo].[GetPhonesByPrice]
(
	@price int
)
RETURNS @returntable TABLE
(
	Id int,
	Name nvarchar(50),
	Price int,
	CompanyId int
)
AS
BEGIN
	INSERT @returntable
	SELECT * FROM Phones WHERE Price < @price
	RETURN
END
```

С помощью выражения `CREATE FUNCTION [dbo].[GetPhonesByPrice]` создается новая функция GetPhonesByPrice. Далее после названия определяется список параметров. Пусть наша функция принимает только один параметр `@price`, который имеет тип `int`, то есть целочисленное значение.

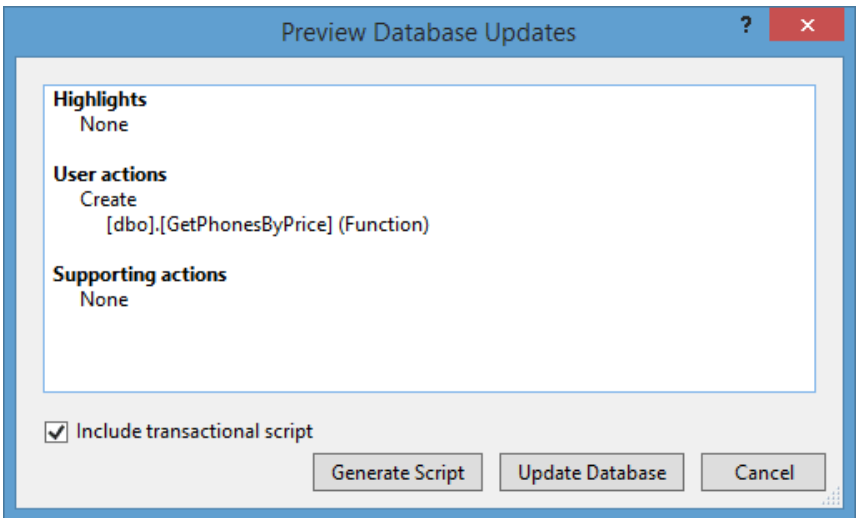
Затем идет определение возвращаемого объекта-таблицы в выражении `RETURNS @returntable TABLE(...)`. В скобках идет перечисление столбцов возвращаемой таблицы. В данном случае они совпадают с определением таблицы Phones. То есть таблица будет содержать объекты класса Phone.

Между выражениями BEGIN и END идет собственно выполнение функции. В данном случае с помощью оператора WHERE функция будет находить все строки, у которых столбец Price содержит меньшее значение, чем в параметре @price.

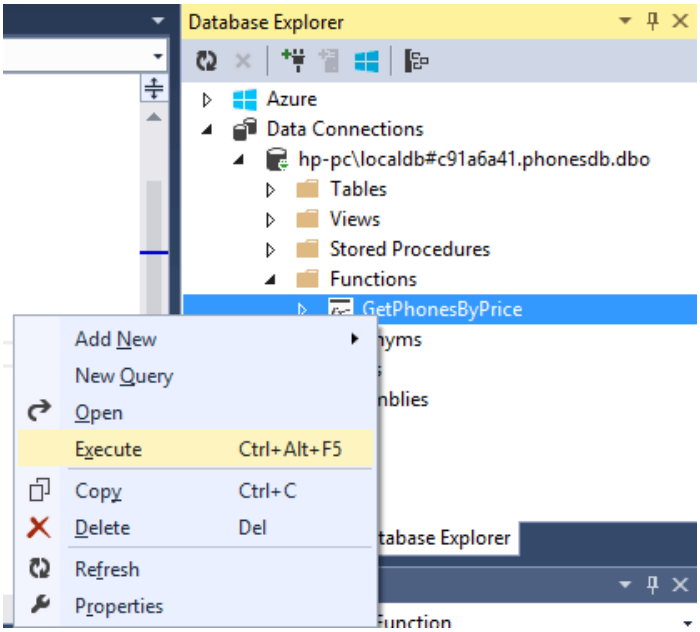
Теперь добавим функцию в базу данных. Для этого нажмем на кнопку Update:



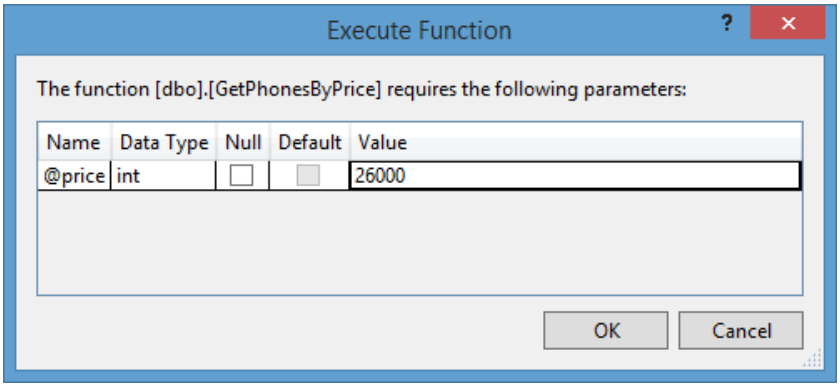
Затем в появившемся окне нажмем на кнопку **Update Database**



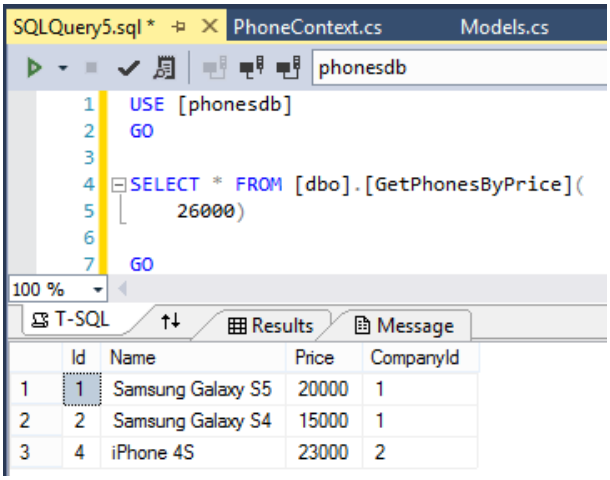
После этого в окне Database Explorer в узле Functions появится подузел добавленной функции. И мы ее можем уже использовать. Но перед обращением к ней из кода c# мы ее протестируем, чтобы убедиться, что она работает как надо. Для этого нажмем на функцию правой кнопкой мыши и в появившемся меню выберем пункт **Execute**:



После этого откроется окно для установки входных параметров функции. Введем в поле Value какое-нибудь число, которое будет передаваться в функцию в качестве параметра:



И Visual Studio сгенерирует и сразу же выполнит скрипт с функцией и переданным в нее параметром:



Как видно, я получил ожидаемые результаты, значит, функция работает правильно.

Теперь обратимся к ней из кода C#:

```
using(PhoneContext db = new PhoneContext())
{
    System.Data.SqlClient.SqlParameter param = new System.Data.SqlClient.SqlParameter("@price", 26000);
    var phones = db.Database.SqlQuery<Phone>("SELECT * FROM GetPhonesByPrice (@price)",param);
    foreach (var phone in phones)
```

```
        Console.WriteLine(phone.Name);
}
```

В этом случае я получу те же результаты, что и при выполнении скрипта выше.

Теперь похожим образом создадим новую функцию, которая будет вычислять сумму с учетом скидки. Код функции:

```
CREATE FUNCTION [dbo].[GetPriceWithDiscount]
(
        @discount int
)
RETURNS @returntable TABLE
(
        Name nvarchar(50),
        Price decimal(8,3)
)
AS
BEGIN
        INSERT @returntable
        SELECT Name, Price - Price * @discount / 100
        FROM Phones
        RETURN
END
```

Функция принимает в качестве параметра процент скидки, например, 10 %. И на выходе она возвращает таблицу из двух полей - названия модели и цены с учетом скидки.

Так как функция фактически будет возвращать новый объект с двумя свойствами - Name и Price, при этом Price уже имеет тип decimal, то нам нужен соответствующий класс C#. Добавим в проект следующий класс:

```
public class DiscountPhone
{
    public string Name { get; set; }
    public decimal Price { get; set; }
}
```

И теперь результат функции мы можем получить следующим образом:

```
using(PhoneContext db = new PhoneContext())
{
        // скидка - 15%
    System.Data.SqlClient.SqlParameter param = new System.Data.SqlClient.SqlParameter("@discount", 15);
    var phones = db.Database.SqlQuery<DiscountPhone>("SELECT * FROM GetPriceWithDiscount (@discount)",param);
    foreach (var p in phones)
        Console.WriteLine("{0} - {1}", p.Name, p.Price);
}
```

Назад **Содержание** **Вперед**