Оглавление

[WPF, язык разметки XAML 2](#_Toc519554159)

[Установка стартового окна 2](#_Toc519554160)

[Фоновая картинка объекта 2](#_Toc519554161)

[Считывание данных с формы, преобразование типов 3](#_Toc519554162)

[Задание текста по умолчанию (Placeholder) 3](#_Toc519554163)

[Работа с базами данных средствами ADO .NET 4](#_Toc519554164)

[Общие сведения о работе с БД средствами ADO .NET 4](#_Toc519554165)

[Общие правила связки элементов управления и источников данных 6](#_Toc519554166)

[Вывод данных в DataGrid (ADO. NET) 6](#_Toc519554167)

[Настройка и работа с ComboBox/ListBox (ADO. NET) 6](#_Toc519554168)

[Добавление данных (без биндинга) 7](#_Toc519554169)

[Редактирование данных (биндинг) (ADO. NET) 8](#_Toc519554170)

[C# 11](#_Toc519554171)

[Глобальные переменные (Email, Role) 11](#_Toc519554172)

[Язык запросов LINQ 11](#_Toc519554173)

[Лямбда - выражения 11](#_Toc519554174)

[Работа с БД (создание, импорт данных, SQL) 12](#_Toc519554175)

[SQL Server Management Studio (SSMS) 12](#_Toc519554176)

[Создание базы данных из SQL скрипта 13](#_Toc519554177)

[Создание таблицы в SSMS согласно ER диаграмме и словарю данных 15](#_Toc519554178)

[Импорт данных средствами SSMS 22](#_Toc519554179)

[Подключение БД в Visual Studio 35](#_Toc519554180)

[Работа с запросами SQL 39](#_Toc519554181)

[Добавление данных. Оператор insert 39](#_Toc519554182)

[Выборка данных. Оператор select 39](#_Toc519554183)

[Использование хранимых процедур для добавления данных (адаптировать для общего случая) 40](#_Toc519554184)

[Автоматизированное тестирование 43](#_Toc519554185)

[Работа с системами контроля версий. Git. 43](#_Toc519554186)

[Тест C# 43](#_Toc519554187)

[Типовые задачи 43](#_Toc519554188)

[Создание одностраничной навигации 43](#_Toc519554189)

[Визуальное перетаскивание объектов (Drag and drop) 45](#_Toc519554190)

[Проверка подлинности пользователя (аутентификация) 45](#_Toc519554191)

# WPF, язык разметки XAML

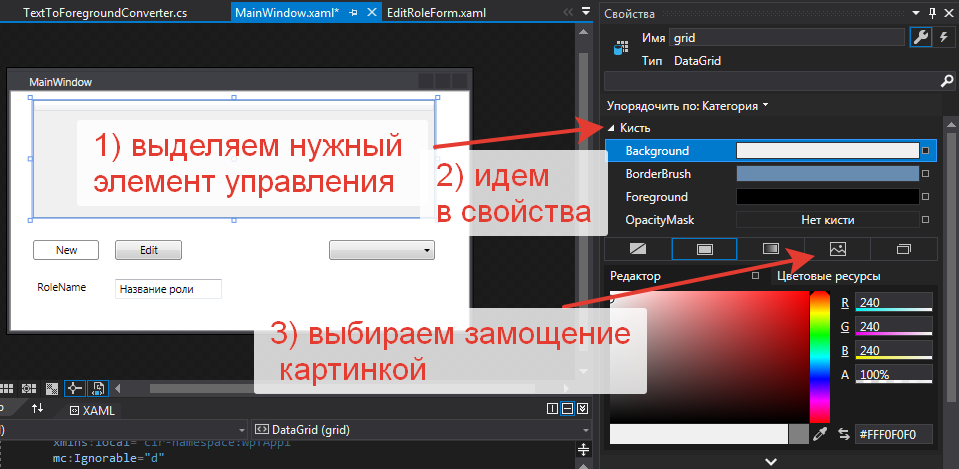
Базовую информацию о WPF можно получить по ссылке: <https://metanit.com/sharp/wpf/>

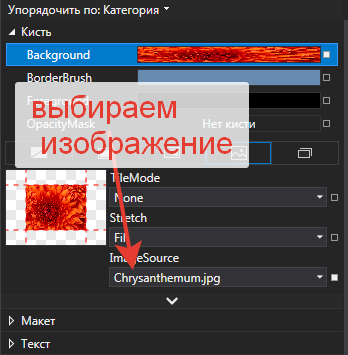
## Установка стартового окна

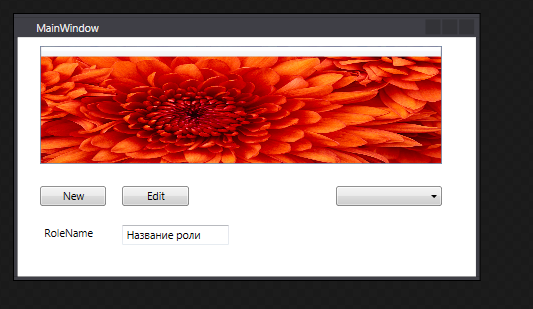
Окно, которое показывает программа по умолчанию, задается в App.xaml, свойство StartupUri

## Фоновая картинка объекта

Если для какого-то объекта нужно установить картинку как фоновую, необходимо выделить этот объект, далее в свойствах установить свойство Background, выбрав необходимую картинку (картинка должна быть добавлена в проект):







Считывание данных с формы, преобразование типов

Преобразование из одного типа данных в другой осуществляется классом Convert, например, если есть строка, которая содержит число (например textbox) и его нужно преобразовать в int, это будет выглядеть следующим образом:

int digit = Convert.ToInt32( textBox.Text );

обратное действие:

textBox.Text = Convert.ToString( 4390 );

## Задание текста по умолчанию (Placeholder)

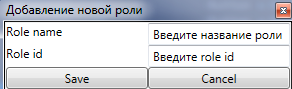
**Пример к данному разделу расположен в архиве BindingExamples\Example2. Перед запуском не забудьте поменять строку подключения.**

За отображение некоторого значения, в случае если не задано основное значение поля, отвечают параметры биндинга FallbackValue и TargetNullValue:

<TextBox Text="{Binding Path=RoleName, FallbackValue=Введите название роли}" />

<TextBox Text="{Binding Path=RoleId, FallbackValue=Введите role id}"/>

Запустим, проверим работу при добавлении новой записи:



Работает, однако, для ввода текста нужно вручную стирать текст-заместитель. Чтобы исправить это, обработаем событие получение фокуса GotFocus полем ввода:

<TextBox Text="{Binding Path=RoleName, FallbackValue=Введите название роли}"

GotFocus="TextBox\_GotFocus" />

В обработчике пропишем удаление текста при получении фокуса. Удалять будем только в том случае, если до сих пор не задано значение соответствующего поля в записи:

private void TextBox\_GotFocus(object sender, RoutedEventArgs e)

{

var data = \_dataRow["RoleName"] as string;

if ( data == null )

{

txtRoleName.Text = "";

}

}

Чуть более сложный вариант приведен в проекте BindingExamples\Example2для поля RoleId.

# Работа с базами данных средствами ADO .NET

## Общие сведения о работе с БД средствами ADO .NET

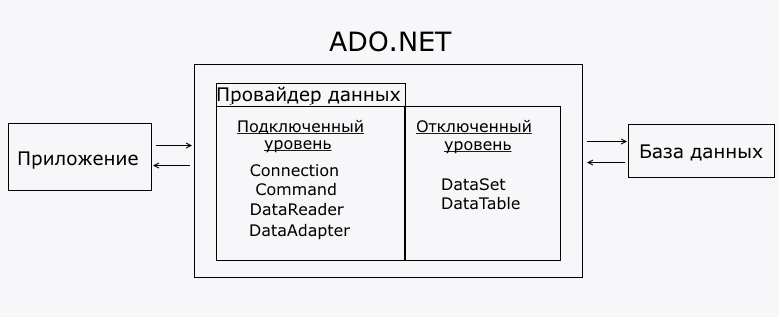
ADO.NET предоставляет собой технологию работы с данными, которая основана на платформе .NET Framework. Эта технология представляет нам набор классов, через которые мы можем отправлять запросы к базам данных, устанавливать подключения, получать ответ от базы данных и производить ряд других операций.

Причем важно отметить, что систем управления баз данных может быть множество. В своей сущности они могут различаться. MS SQL Server, например, для создания запросов использует язык T-SQL, а MySQL и Oracle применяют язык PL-SQL. Разные системы баз данных могут иметь разные типы данных. Также могут различаться какие-то другие моменты. Однако функционал ADO.NET построен таким образом, чтобы предоставить разработчикам унифицированный интерфейс для работы с самыми различными СУБД.

Основу интерфейса взаимодействия с базами данных в ADO.NET представляет ограниченный круг объектов: Connection, Command, DataReader, DataSet и DataAdapter.

* С помощью объекта Connection происходит установка подключения к источнику данных.
* Объект Command позволяет выполнять операции с данными из БД.
* Объект DataReader считывает полученные в результате запроса данные.
* Объект DataSet предназначен для хранения данных из БД и позволяет работать с ними независимо от БД.
* Объект DataAdapterявляется посредником между DataSet и источником данных.

Главным образом, через эти объекты и будет идти работа с базой данных.

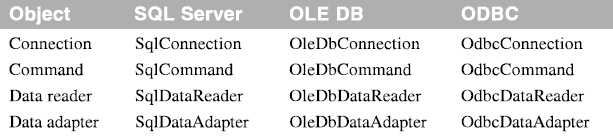


Однако чтобы использовать один и тот же набор объектов для разных источников данных, необходим соответствующий провайдер данных. Собственно через провайдер данных в ADO.NET и осуществляется взаимодействие с базой данных. Причем для каждого источника данных в ADO.NET может быть свой провайдер, который собственно и определяет конкретную реализацию вышеуказанных классов.

Основные пространства имен, которые используются в ADO.NET:

* System.Data: определяет классы, интерфейсы, делегаты, которые реализуют архитектуру ADO.NET
* System.Data.Common: содержит классы, общие для всех провайдеров ADO.NET
* System.Data.Design: определяет классы, которые используются для создания своих собственных наборов данных
* System.Data.Odbc: определяет функциональность провайдера данных для ODBC
* System.Data.OleDb: определяет функциональность провайдера данных для OLE DB
* System.Data.Sql: хранит классы, которые поддерживают специфичную для SQL Server функциональность
* System.Data.OracleClient: определяет функциональность провайдера для баз данных Oracle
* System.Data.SqlClient: определяет функциональность провайдера для баз данных MS SQL Server
* System.Data.SqlServerCe: определяет функциональность провайдера для SQL Server Compact 4.0
* System.Data.SqlTypes: содержит классы для типов данных MS SQL Servera
* Microsoft.SqlServer.Server: хранит компоненты для взаимодействия SQL Server и среды CLR

Функционально классы ADO.NET можно разбить на два уровня: подключенный и отключенный. Каждый провайдер данных .NET реализует свои версии объектов Connection, Command, DataReader, DataAdapter и ряда других, который составляют подключенный уровень. То есть с помощью них устанавливается подключение к БД и выполняется с ней взаимодействие. Как правило, реализации этих объектов для каждого конкретного провайдера в своем названии имеют префикс, который указывает на провайдер:



## Общие правила связки элементов управления и источников данных

Если нужно установить содержимое какого-то объекта, смотрите свойства Source, ItemsSource, Content (как правило, если выводится один элемент и этот элемент может быть заменен чем-то другим, например кнопкой, т.е. графическим элементом)

Перед прочтением следующих разделов рекомендуется ознакомиться с общим понятием привязки данных в WPF: https://metanit.com/sharp/wpf/11.php

## Вывод данных в DataGrid (ADO. NET)

1. Подключаем базу данных (в Visual Studio - Server Explorer (обозреватель серверов), DataConnections (подключения данных) – Add Connection, меняем DataSource на Sql Server, выбираем сервер БД, базу данных)
2. Создаем dataSet, настраиваем запрос
3. Прописываем dataset и table adapter для соответствующей таблицы в классе формы
4. Добавляем в xaml разметке DataGrid, даем имя, например, dataGrid
5. В конструкторе формы или в обработчике события Loaded прописываем связь между датагридом и таблицей БД: dataGrid.ItemsSource = dataSet.Имя\_таблицы;
6. Заполняем таблицу: da.Fill( dataSet.Имя\_таблицы); da – имя адаптера данных для соответствующей таблицы.
7. В датагрид устанавливаем AutoGenerateColumns="False"
8. Прописываем между тегами открытия и закрытия датагрида содержимое столбцов:

<DataGrid.Columns>

<DataGridTextColumn Header="Заголовок" Binding="{Binding Path=Имя\_столбца}"/>

</DataGrid.Columns>

1. Чтобы перезагрузить содержимое таблицы, сначала чистим ее: dataSet.Имя\_таблицы.Clear(), и повторно загружаем данные (шаг 6)

## Настройка и работа с ComboBox/ListBox (ADO. NET)

1) Создаем dataSet, настраиваем запрос, в запросе ***должно*** извлекаться ключевое поле!!!

2) Прописываем dataset и table adapter в классе формы, создаем их, загружаем их содержимое с помощью da.Fill( dataSet.имя\_таблицы);

3) в xaml разметке даем ComboBox’у имя, например comboBox1

4) там же устанавливаем свойство DisplayMemberPath="имя\_отображаемого\_поля" и SelectedValuePath = “имя\_ключевого\_столбца”;

5) в коде (в конструкторе формы или обработчике события Loaded формы) прописываем источник данных: comboBox1.ItemsSource = dataSet.имя\_таблицы;

6) Если нас интересует только значение ключевого поля (id) выбранного элемента, можем написать:

int id = (int) comboBox1.SelectedValue;

или, чтобы наоборот, задать выбранный элемент по его id:

comboBox2.SelectedValue = “R”;

Если же нам нужно значение всех полей выбранной записи, нужно помнить, что он представляет собой элемент типа DataRowView, поэтому проводим его преобразование:

DataRowView row = comboBox1.SelectedItem as DataRowView;

Это же можно сделать другой записью:

DataRowView row = ( DataRowView) comboBox1.SelectedItem;

Имея строку данных, извлекаем из нее интересующее нас ключевое поле:

int id = Convert.ToInt32( row[“поле с id”] );

или

string strId = Convert.ToString( row[“поле с id”] );

## Добавление данных (без биндинга)

1) Создаем dataSet, настраиваем запрос

2) Прописываем dataset и table adapter в классе формы, создаем их, но содержимое загружать не обязательно!!!

3) Вместо этого, при нажатии кнопки сохранить, сначала считываем все записи с полей в локальные переменные, например:

string name = nameBox.Text;

int age = Convert.ToInt32( ageBox.Text );

и т.д.

4)Делаем нужные проверки

5)Создаем новую строку в базе:

DataRowView row = dataSet.имя\_таблицы.DefaultView.AddNew();

row[“имя поля1”] = name;

row[“имя поля2”] = age;

и т.п., заполняем все поля по их именам

6)заканчиваем редактирование строки:

row.EndEdit();

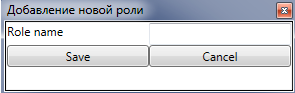
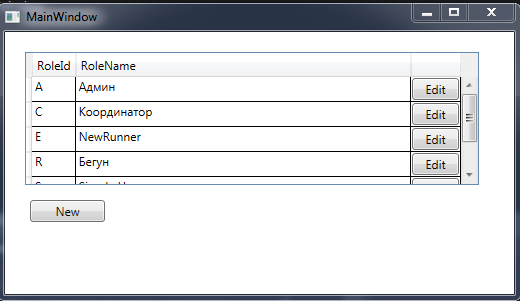
7) обновляем содержимое БД:

da.Update( dataSet.имя\_таблицы);

## Редактирование данных (биндинг) (ADO. NET)

**Пример к данному разделу расположен в архиве BindingExamples\Example1. Перед запуском не забудьте поменять строку подключения.**

Допустим, мы хотим связать строку базы данных (DataRowView), с формой редактирования/добавления записи.

За связку полей данных с объектом используется свойство DataContext. Относительно этого элемента будет производиться поиск связанных свойств.

Рассмотрим процедуру редактирования. При редактировании нужно определить текущий элемент и передать его на вторую форму.

Выбранный элемент DataGrid’а определяется свойством SelectedItem, представляющем из себя запись данных (класс DataRowView). Форма редактирования должна принимать и хранить объект такого типа. Поэтому, определим поле \_dataRow для хранения данных нашей записи в классе формы редактирования:

public partial class EditRoleForm : Window

{

DataRowView \_dataRow; //наша будущая строка данных

public EditRoleForm()

{

InitializeComponent();

}

}

EditRoleForm – название класса формы редактирования.

Конструктор формы по умолчанию не предусматривает передачи параметров, поэтому добавим новую реализацию конструктора с параметром типа DataRowView или object.

public EditRoleForm(object dataRow)

{

InitializeComponent();

\_dataRow = dataRow as DataRowView; //сохраняем переданную строку в переменную класса

}

Пропишем связь между элементами управления и нашей записью из БД, для этого будем использовать биндинг. Для этого сделаем свойство Text элемента TextBox зависимым, для этого укажем свойство, с которым оно будет связано:

<TextBox Text="{Binding Path=RoleName}" …/>

Осталось указать биндингу, откуда брать указанное свойство RoleName, для указания этой зависимости предназначено свойство DataContext, которое должно ссылаться на объект-источник данных. В нашем случае это \_dataRow, поэтому немного изменяем конструктор:

public EditRoleForm(object dataRow)

{

InitializeComponent();

\_dataRow = dataRow as DataRowView; //сохраняем переданную строку в переменную класса

DataContext = \_dataRow; //связываем элементы управления с записью БД

}

При нажатии на кнопку «сохранить» завершаем редактирование данных, сохраняем изменения в БД, используя TableAdapter:

\_dataRow.EndEdit(); //завершаем редактирование строки (не обязательно?)

RoleTableAdapter rta = new RoleTableAdapter(); //создаем объект для внесения обновлений

rta.Update(\_dataRow.Row); //вносим изменения в БД

Close(); //закрываем окно

При нажатии на кнопку «отмена» закрываем окно.

При закрытии окна проверяем состояние строки таблицы. Если строка до сих пор в режиме редактирования, значит, выход осуществлен по кнопке «отмена», следовательно, следует произвести отмену изменений:

\_dataRow.CancelEdit(); (не обязательно?)

Вернемся в основную форму, по нажатию на кнопку Edit мы должны передать текущую строку для редактирования на форму, описанную выше. Поэтому, в обработчике кнопки Edit пропишем:

EditRoleForm editForm = new EditRoleForm( grid.SelectedItem );

editForm.ShowDialog();

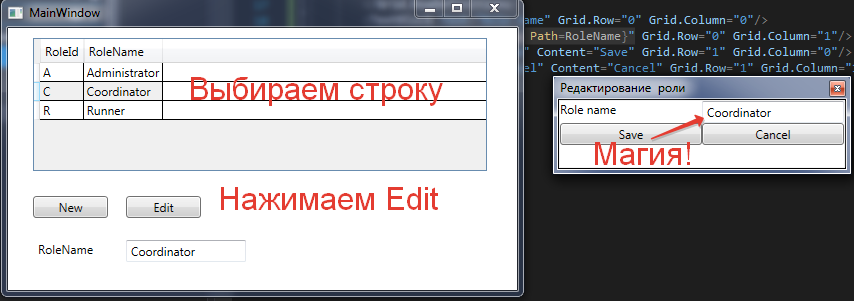
Если вы не использовали конструкции \_dataRow.BeginEdit(); \_dataRow.EndEdit(); \_dataRow.CancelEdit(); вам потребуется обновить содержимое таблицы, тогда обработчик кнопки Edit примет вид:

EditRoleForm editForm = new EditRoleForm( grid.SelectedItem );

editForm.ShowDialog();

rta.Fill(dataSet.Role); //rta – адаптер данных

Запускаем, проверяем:



Добавление данных выполняется аналогично, будет отличаться только источник записи данных, браться она будет не из таблицы, а создаваться новая. Сделать это можно в конструкторе формы по умолчанию:

/// <remarks>

/// Если вызывается конструктор по умолчанию, создается новая запись

/// </remarks>

public EditRoleForm()

{

InitializeComponent();

Title = "Добавление новой роли";

DataSet1.RoleDataTable dt = new DataSet1.RoleDataTable();

\_dataRow = dt.DefaultView.AddNew();

DataContext = \_dataRow;

}

Теперь, для добавления новых записей, будет форма EditRole будет создаваться конструктором по умолчанию, в случае редактирования в конструктор будет передаваться редактируемая запись.

Также следует учесть, что после добавления новой записи следует перезагрузить содержимое таблицы, чтобы она отобразилась (самый простой на данный момент способ).

# C#

## Глобальные переменные (Email, Role)

Если нужно сохранить информацию, доступную из нескольких страниц, можно использовать публичные статические переменные (на самом деле это не очень хорошо). Например, для хранения email текущего пользователя, его роли, эти переменные можно определить в классе App (приложение) в файле App.xaml.cs:

public partial class App:Application

{

public static string Email;

public static string Role;

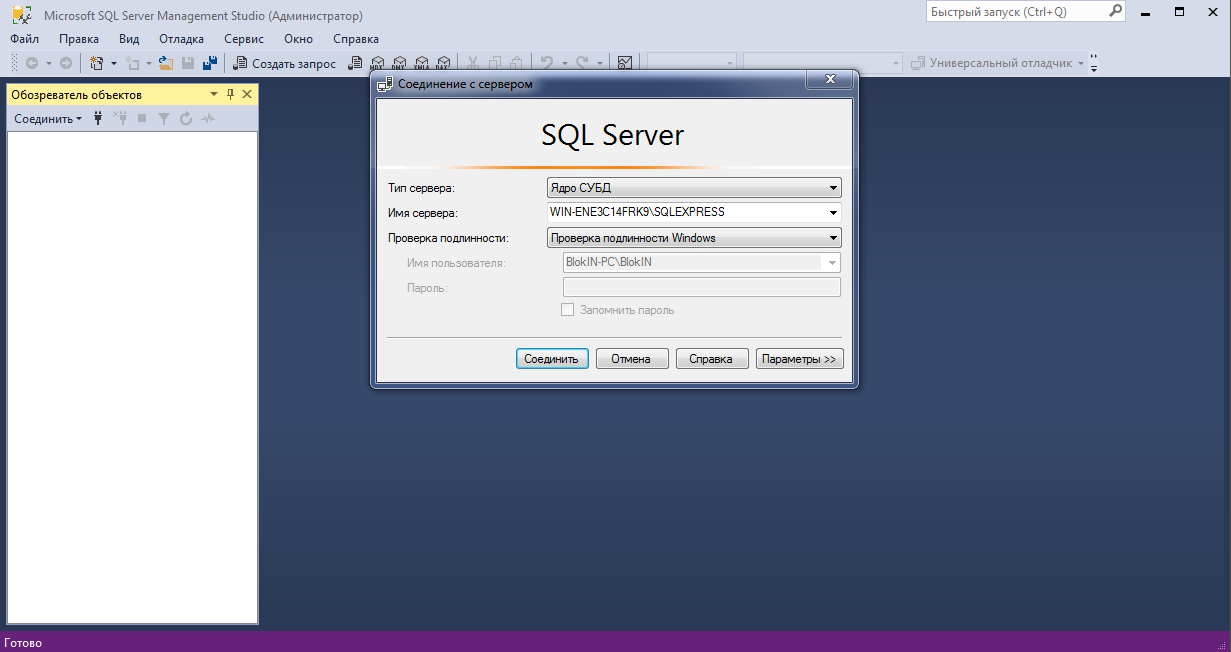
}

Везде, где нужен к ним доступ обращение будет производиться по именам App.Email , App.Role.

# Работа с БД (создание, импорт данных, SQL)

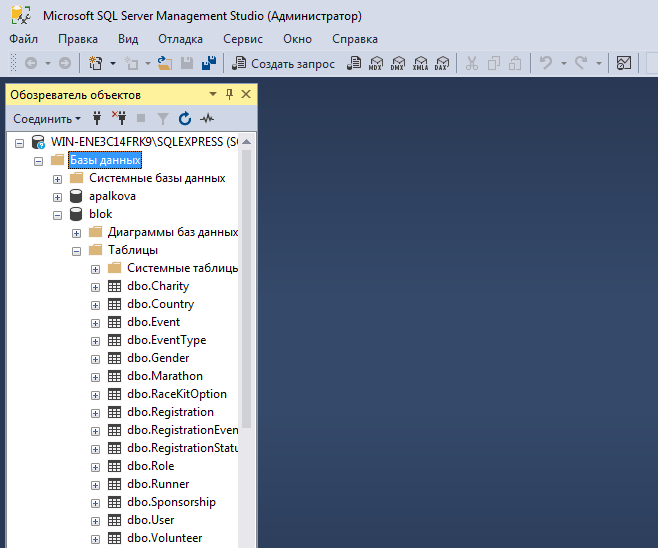
## SQL Server Management Studio (SSMS)

Запускаете SSMS, откроется окно, показанное на рисунке ниже:

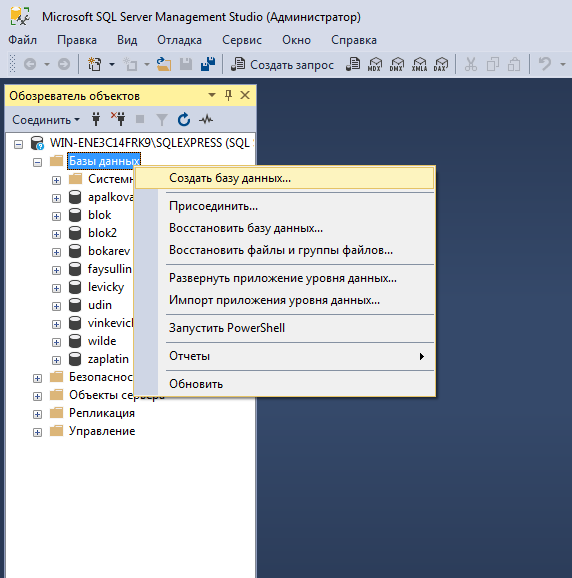


Если ваша база данных располагается на том же компьютере, на котором вы работаете, выбираете проверку подлинности Windows. В противном случае проверку подлинности SQL сервер, используя выданные вам логин и пароль.

После подключения, в обозревателе объектов будет представлен список доступных баз данных, таблиц и других объектов, как показано на рисунке ниже:

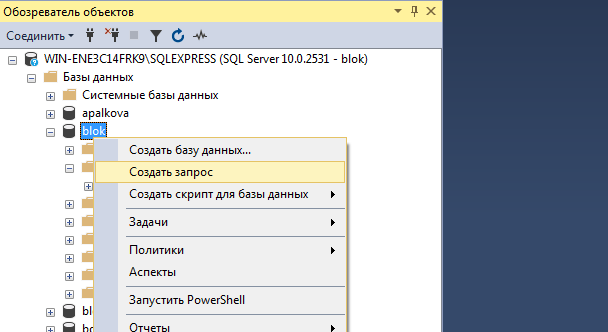


## Создание базы данных из SQL скрипта

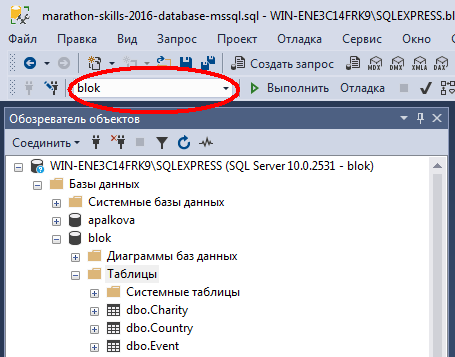
В случае, если подключение выполняется впервые, для дальнейшей работы необходимо создать базу данных. Для этого кликните правой кнопкой мыши по элементу списка «Базы данных» и выберите пункт «Создать базу данных», как показано на рисунке ниже:

В появившемся окне, введите название базы данных и нажмите «Ок».

Осталось заполнить таблицу, для этого выбираете созданную базу данных, и создаете к ней запрос:



Откройте SQL скрипт на генерацию таблиц БД, расположенный в папке с заданием (скрипта будет 2, выбирать тот, который для MSSQL), нажмите выполнить. При этом, убедитесь, что активна ваша БД в выпадающем списке:



Либо в самом начале запроса пропишите команду

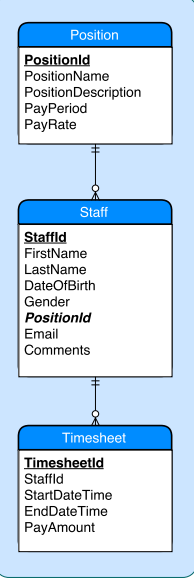
USE имя\_бд

где имя\_бд – название ранее созданной базы данных.

## Создание таблицы в SSMS согласно ER диаграмме и словарю данных

Даны ER диаграмма и словарь данных, нужно создать таблицы, связать их соответствующим образом

ER диаграмма:

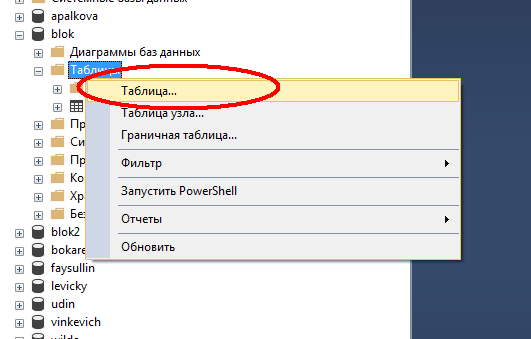


Словарь данных:

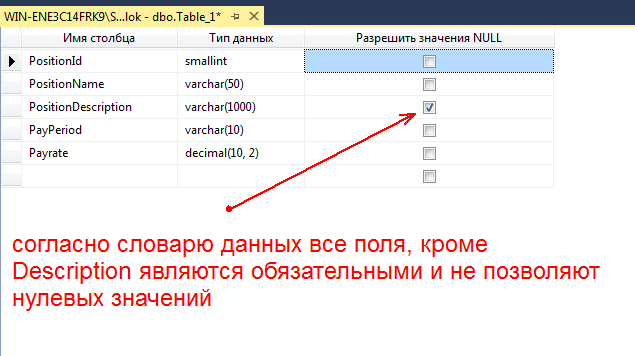
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Position (Позиция)** | |  | |  | | |
| **FIELD NAME** | **DATA TYPE / FIELD SIZE** | | **REQUIRED?** | | **NOTES** |
| **PositionId** | SMALLINT | | Y | | Auto increment |
| **PositionName** | VARCHAR (50) | | Y | |  |
| **PositionDescription** | VARCHAR (1000) | |  | |  |
| **PayPeriod** | VARCHAR (10) | | Y | |  |
| **Payrate** | DECIMAL (10,2) | | Y | |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Staff (Персонал)** | |  |  | |
| **KEY** | **FIELD NAME** | **DATA TYPE / FIELD SIZE** | **REQUIRED?** | **NOTES** |
| PK | **StaffId** | INT | Y | Auto increment |
|  | **FirstName** | VARCHAR (80) | Y |  |
|  | **LastName** | VARCHAR (80) | Y |  |
|  | **Email** | VARCHAR (100) | Y |  |
|  | **DateOfBirth** | DATETIME | Y |  |
|  | **Gender** | VARCHAR (10) | Y |  |
| FK | **PostionId** | SMALLINT | Y | Foreign key to Position |
|  | **Comments** | VARCHAR (2000) |  |  |
| **Timesheet (Расписание)** | |  |  | |
| **KEY** | **FIELD NAME** | **DATA TYPE / FIELD SIZE** | **REQUIRED?** | **NOTES** |
| PK | **TimesheetId** | INT | Y | Auto increment |
| FK | **StaffId** | INT | Y | Foreign key to Staff |
|  | **StartDateTime** | DATETIME |  |  |
|  | **EndDateTime** | DATETIME |  |  |
|  | **PayAmount** | DECIMAL (10,2) |  |  |

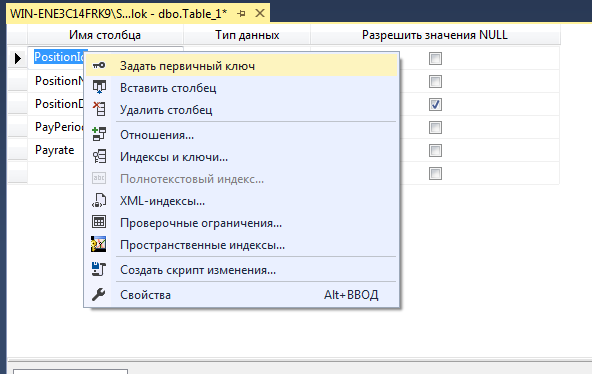
Как видно, в словаре данных указаны нужные типы данных. Создадим, для примера, таблицу Position:



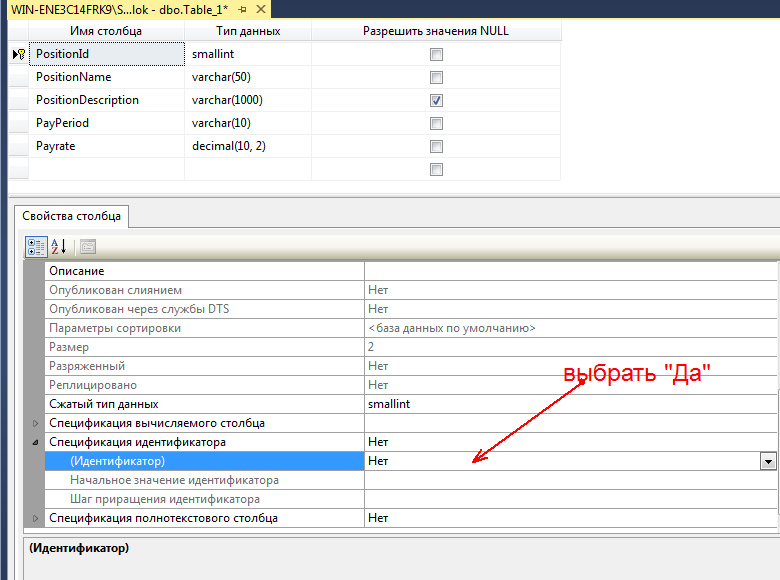
Добавляем столбцы согласно ER диаграмме и словарю данных:



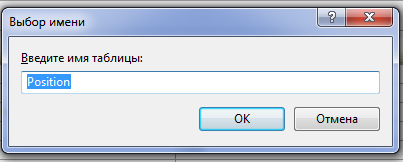
Указываем поле PositionID как ключевое:



Указываем, что поле PositionId – автоинкремент, т.е. счетчик:



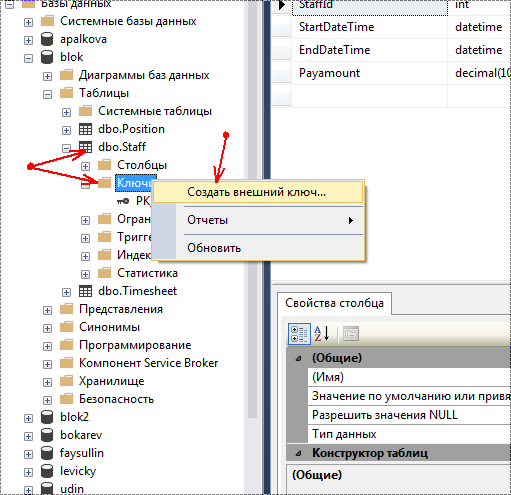
Нажимаем сохранить, вводим нужное имя таблицы:



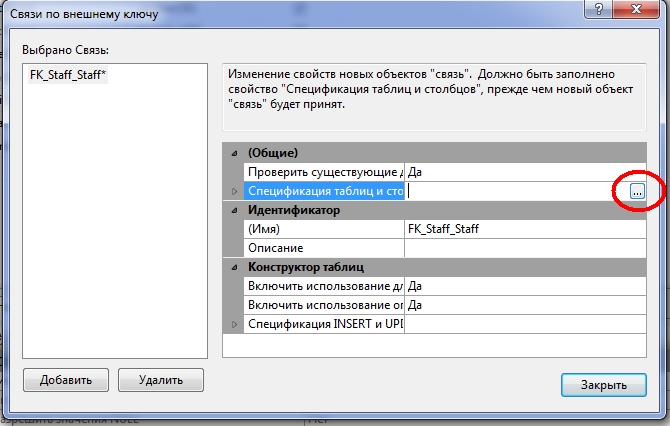
Аналогично делаем оставшиеся 2 таблицы.

Когда таблицы созданы, осталось прописать внешние ключи. В словаре данных они отмечены как FK, таких ключа у нас 2, в таблице Staff – PositionId, в таблице Timesheet – StaffId.

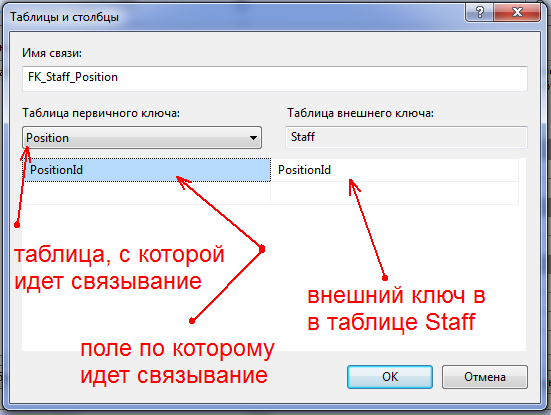
Для этого в таблице, содержащей такой ключ, делаем следующее:



Далее:



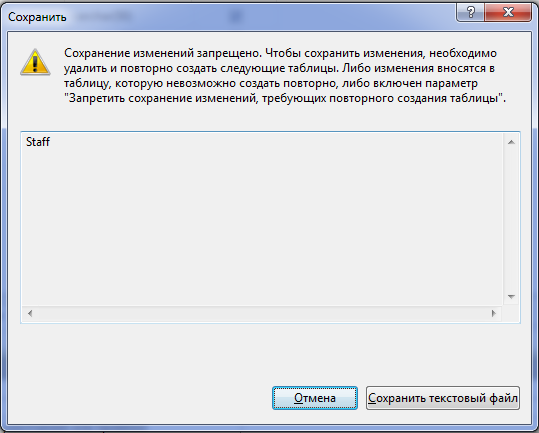
Настраиваем:



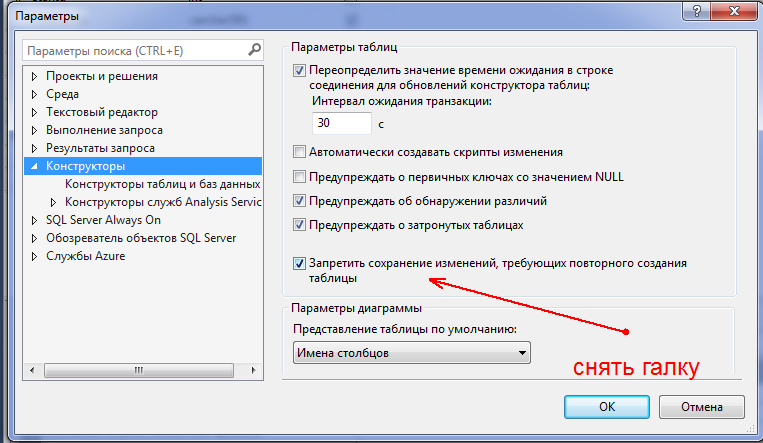
Жмем «ОК»

Нажимаем кнопку сохранить, подтверждаем свои намерения.

**Если вы где то допустили ошибку и хотите изменить таблицу, жмете по ней правой кнопкой - проект, снова откроется редактор, вносите изменения, однако, скорее всего вы получите ошибку:**



Чтобы отключить ее. идем в настройки SSMS: Сервис – параметры – конструкторы, и снимаем соответствующий чекбокс:

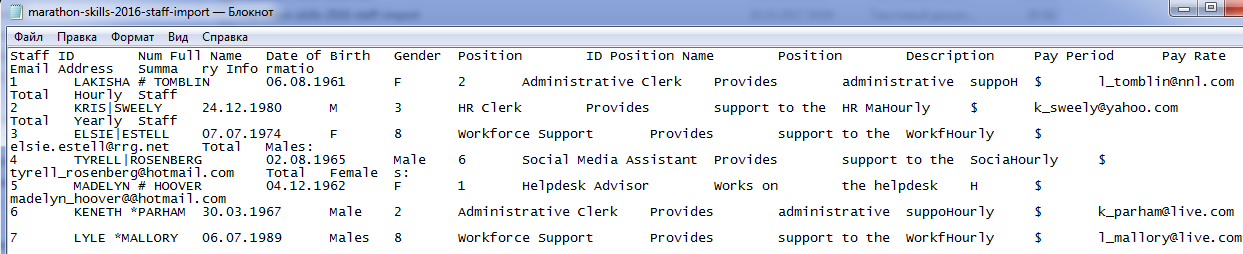


## Импорт данных средствами SSMS

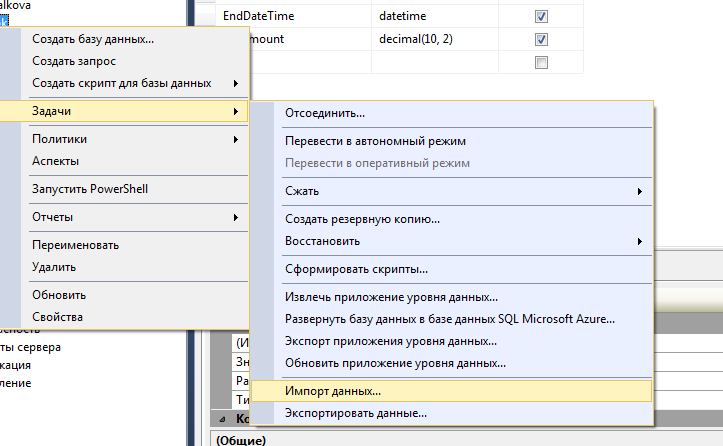
*\*будут использованы таблицы, созданные в разделе «Создание таблицы в SSMS согласно ER диаграмме и словарю данных»*

К таблицам будут даны данные для заполнения, однако даны они будут не в формате SQL скрипта, а скорее всего в виде «сырых», необработанных данных, текстовых файлов. В лучшем случае csv/excel.

Рассмотрим случай, когда данные находятся в txt файле. Файл приложен к проекту: «marathon-skills-2016-staff-import». Открыв его в блокноте, увидим нечто некрасивое:

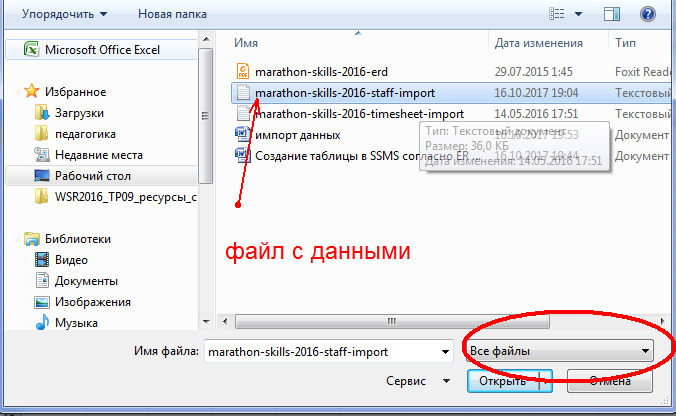


В некоторых случаях, импортировать можно и такие файлы:

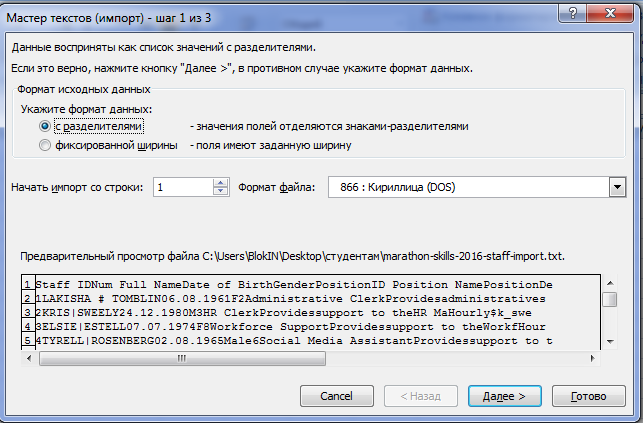


В этом случае, в качестве источника данных нужно выбрать «Источник Неструктурированный файл», настроить поля и произвести импорт.

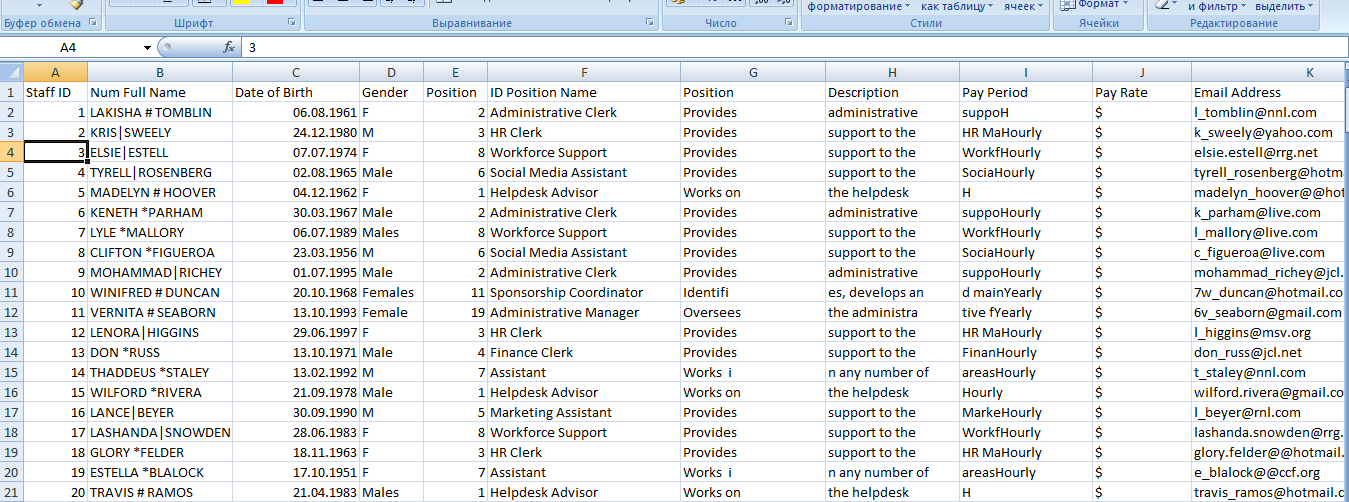
Однако, лучше, данные сначала обработать. Для этого, откроем файл в екселе:



Настраиваем импорт:

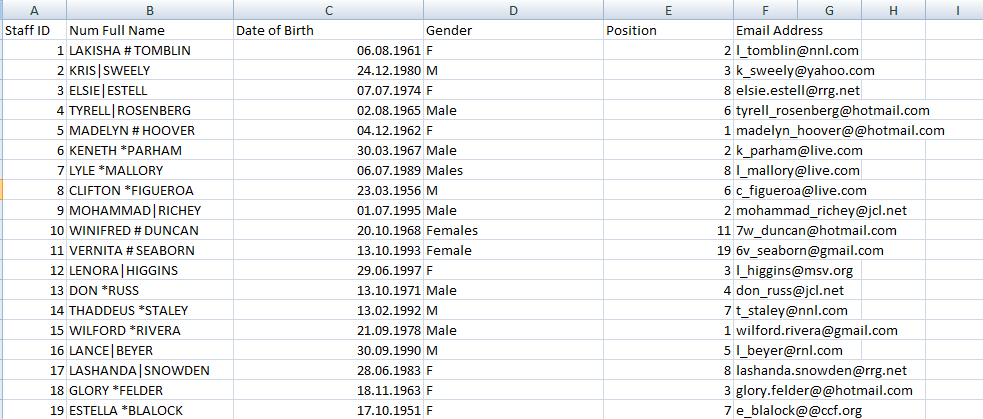


В данном примере импорт проходит успешно без настроек, однако, возможно, придется поиграть с параметрами. На выходе не совсем корректно определились имена столбцов:

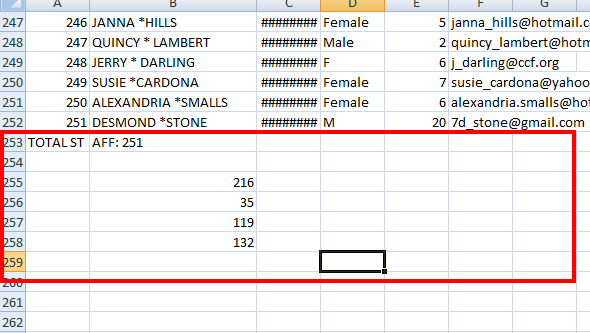


Как видно, данные в этой таблице не нормализованы, т.е. содержат данные из таблиц Staff (сотрудники) и Position (должности). Поэтому, разделим исходную таблицу на 2, выбирая соответствующие колонки:

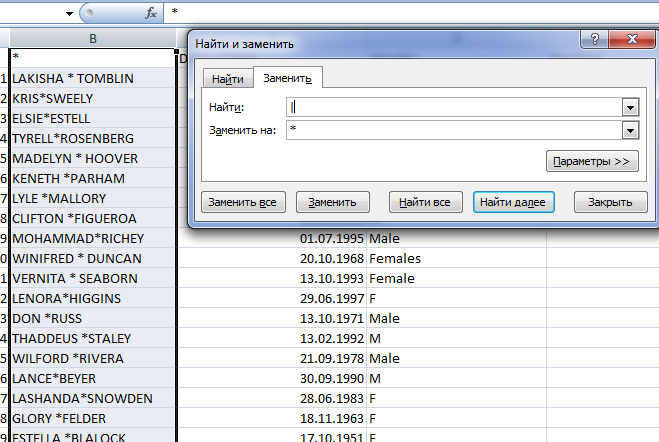
К таблице Staff относятся столбцы A (ключевое поле),B (полное имя), C (дата рождения), D (пол), E (код должности), Email. Как видно, поле B позже должно быть разделено еще на 2 – имя и фамилия. Копируем их в отдельный файл:



Перед дальнейшими действиями проверим данные на корректность, видно что в конце содержится «мусор», удаляем его:



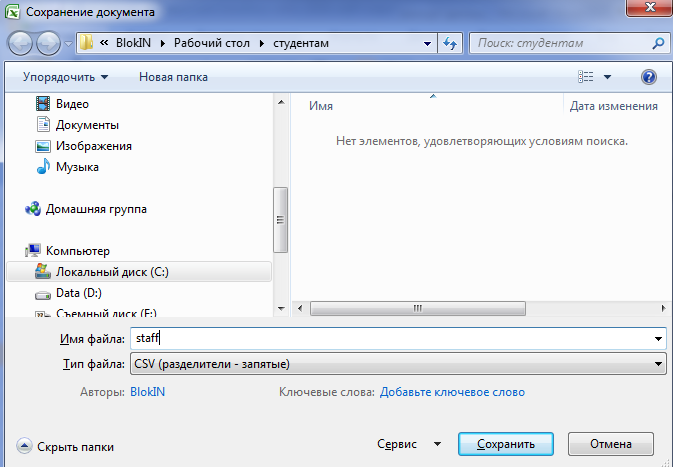
Вернемся к разделению колонки, видно, что в колонке с именем имя разделено разными разделителями (масло масляное, ага ☺ ). Заменим их на один:



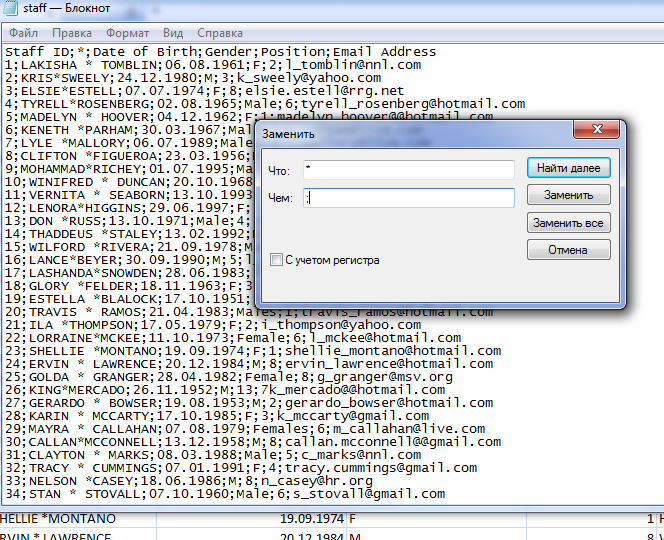
Теперь разделим колонку на 2, для этого есть 2 способа:

1. Сложный в действиях, но проще запоминающийся

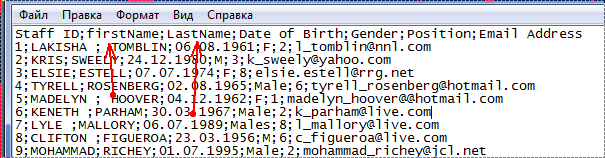
Сохраняем файл в формате с расширением .csv:



Открываем файл в блокноте, делаем замену нашего символа-разделителя на ; :



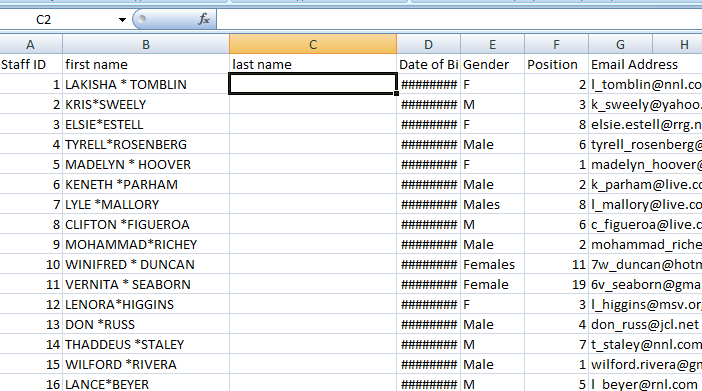
И исправляем имена колонок в первой строке:



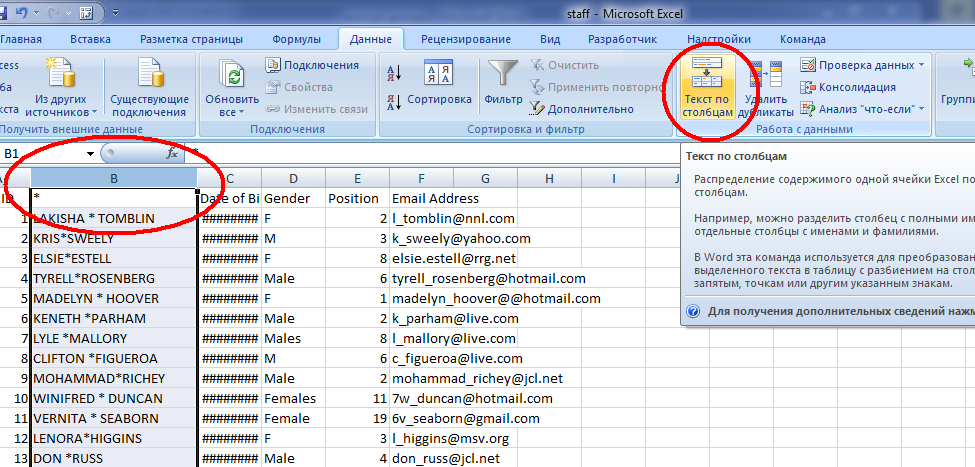
Сохраняем. Открываем в екселе, пересохраняем в формате excel

1. Более быстрый, но сложнее запоминающийся:

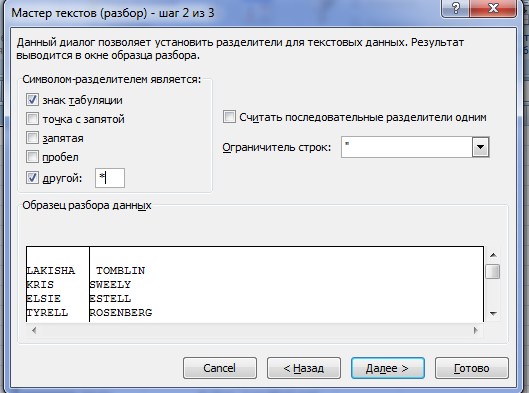
Добавляете новый столбец, в первой строке правите имя будущего столбца:



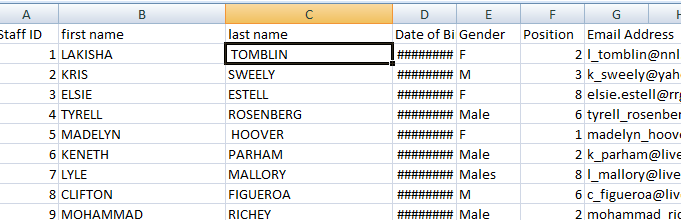
В екселе на вкладке данные – Текст по столбцам:



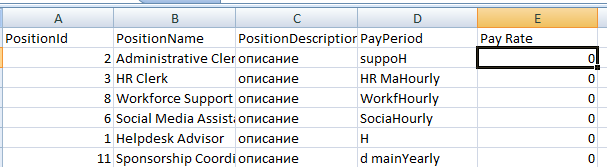
В качестве разделителя указываете выбранный символ:



Profit!!:

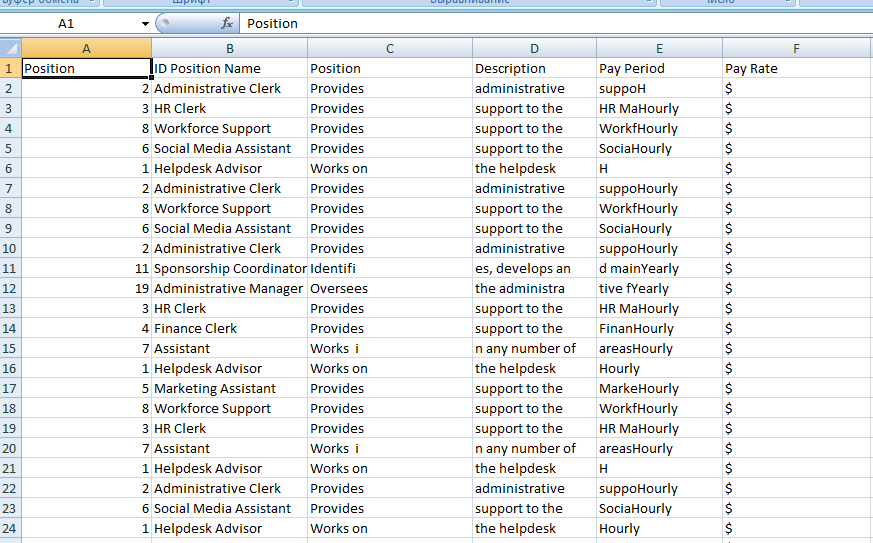


**Приводим имена столбцов в соответствие с именами в БД:**

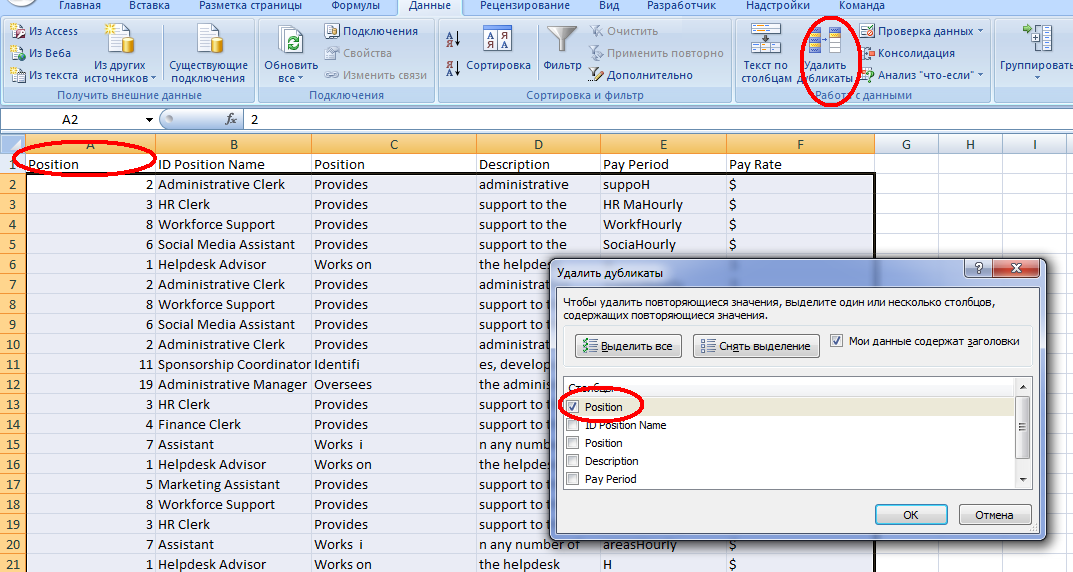
****

**Сохраняем результат в формате excel.**

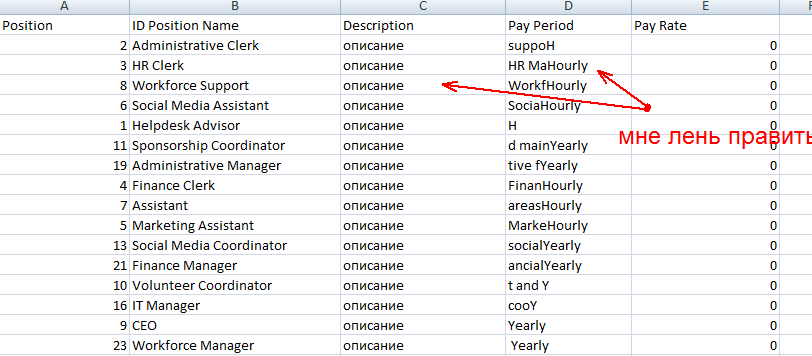
**Вернемся к исходной таблице, выберем из нее данные должностей, т.е. таблицы Position (обращая внимание на возможный мусор, в виде сводных данных):**

****

Удалим дубликаты (вкладка данные – удалить дубликаты) и укажем поле, в котором их искать:

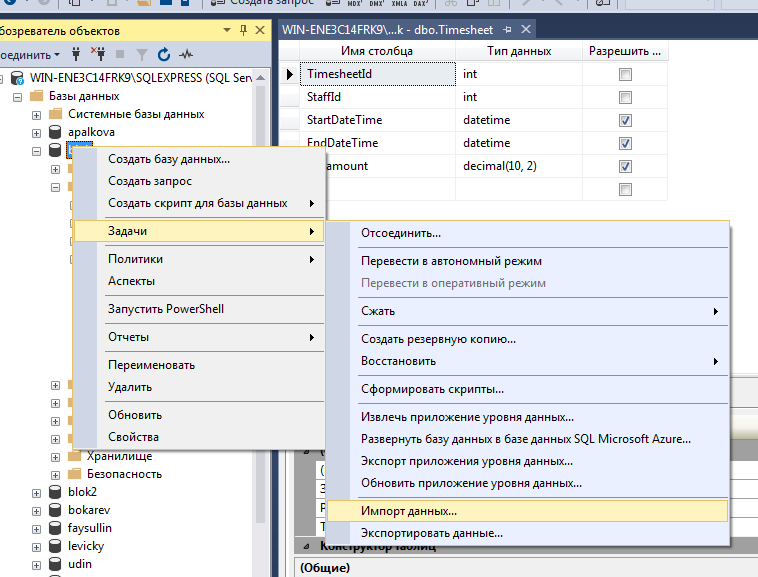


Устраняем косяки (поле Description занимает колонки C,D, и немного E, PayPeriod – сливается с частью Description в колонке E, в PayRate вместо цифр $):

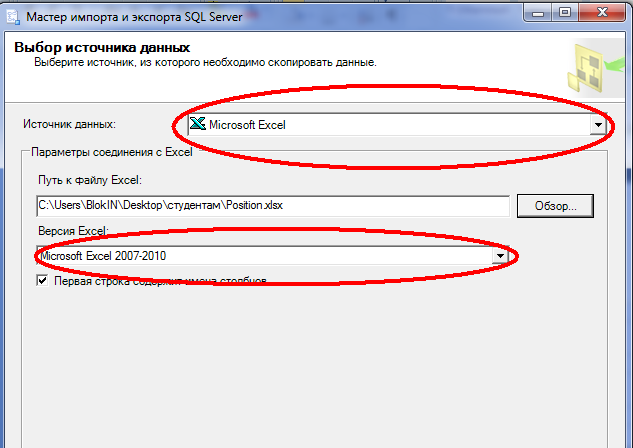


**Сохраняем в формате excel**

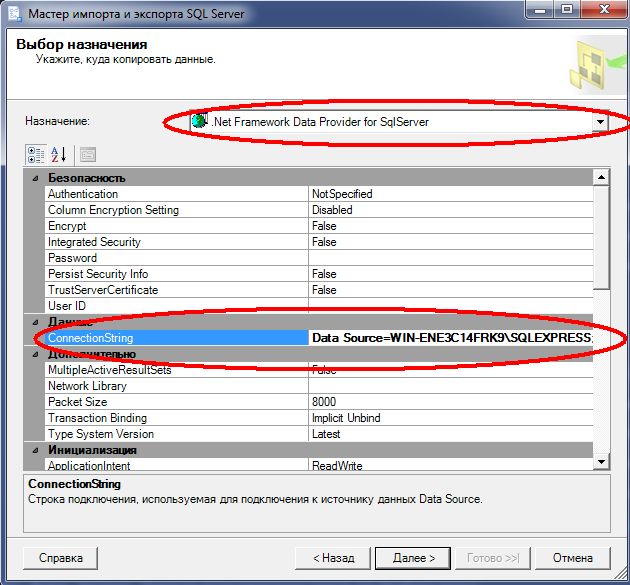
**Теперь в SSMS начинаем импорт. Импортировать начинаем со справочников (в нашем примере сначала нужно загрузить профессии, а потом сотрудников:**

****

В качестве источника данных выбираем excel файл:

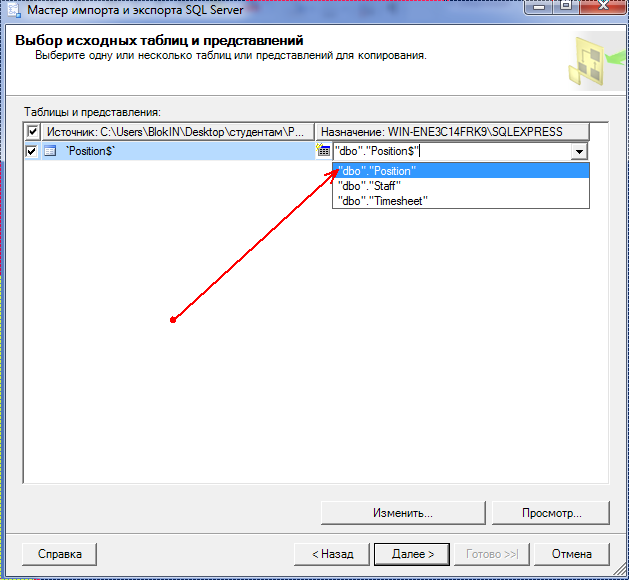


В качестве приемника данных выбираете SQL сервер, прописываете строку подключения (параметры строки можно взять в Visual Studio – обозреватель серверов, свойства):

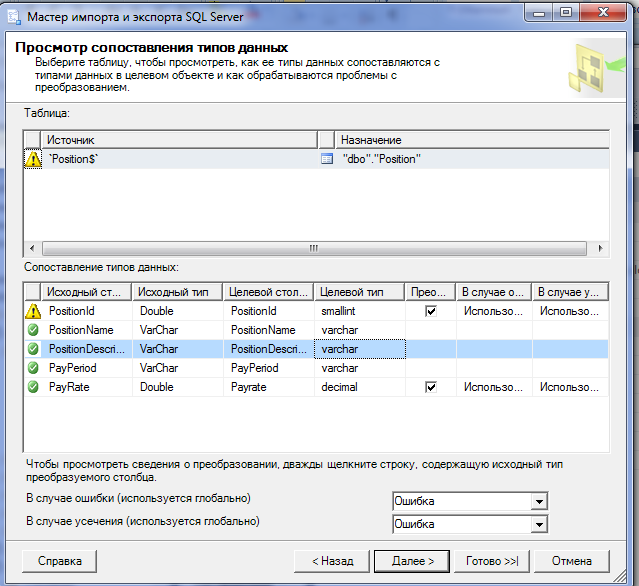


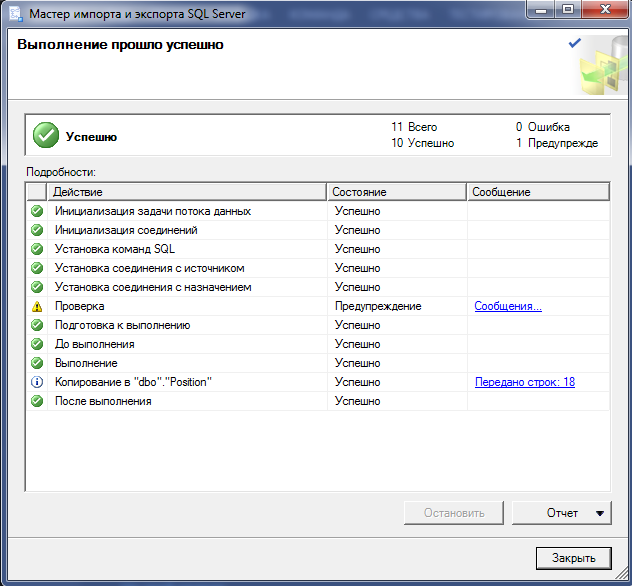
Нажимаете Далее -> Далее

Выбираете вашу таблицу для импорта (именно выбираете, по умолчанию создается новая):

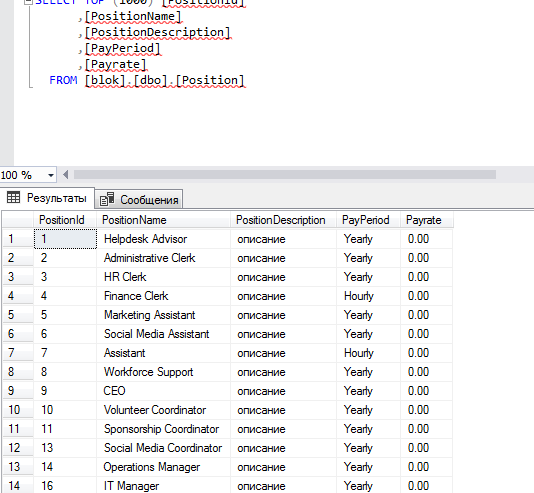


Далее:





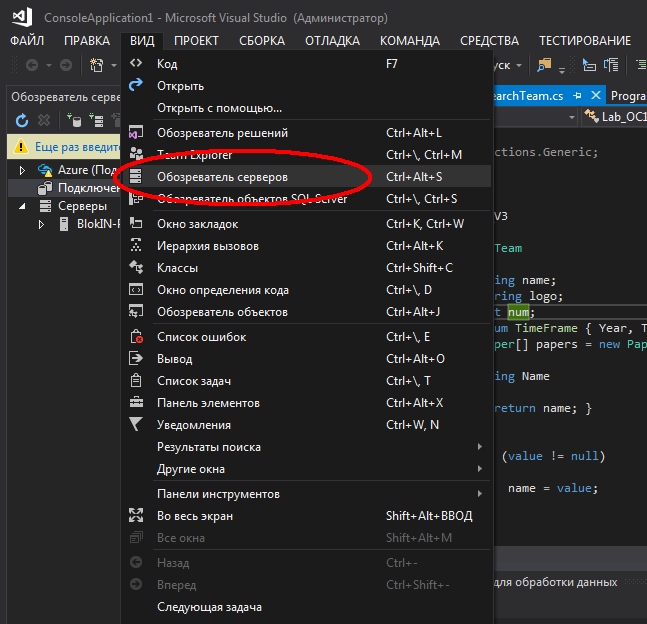
Делаем выборку, данные в таблице:



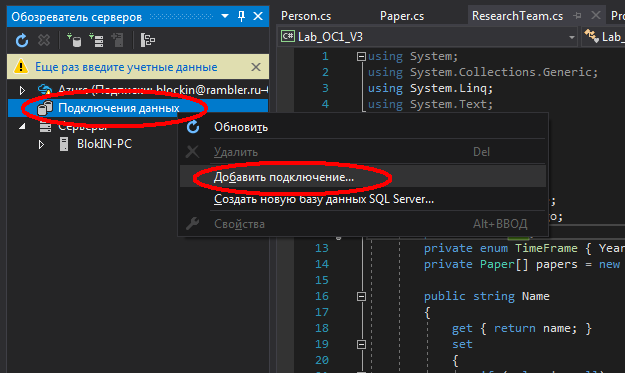
Аналогично добавляем данные для сотрудников (Staff), сформированные ранее (не забыв назвать колонки как в БД).

## Подключение БД в Visual Studio

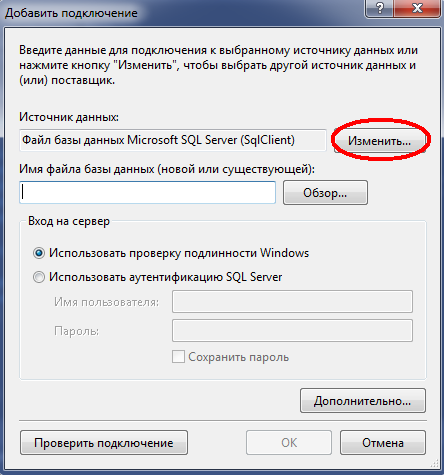
Запустите Visual Studio, откройте «Обозреватель серверов». Если его нет, в главном меню выберите «Вид» -> «Обозреватель серверов»:

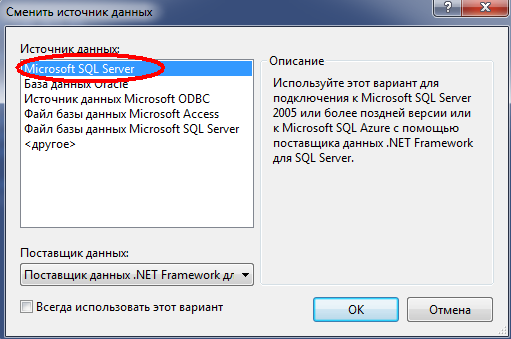


В обозревателе серверов, добавьте новое подключение данных:



На след шаге нажмите «Изменить»:

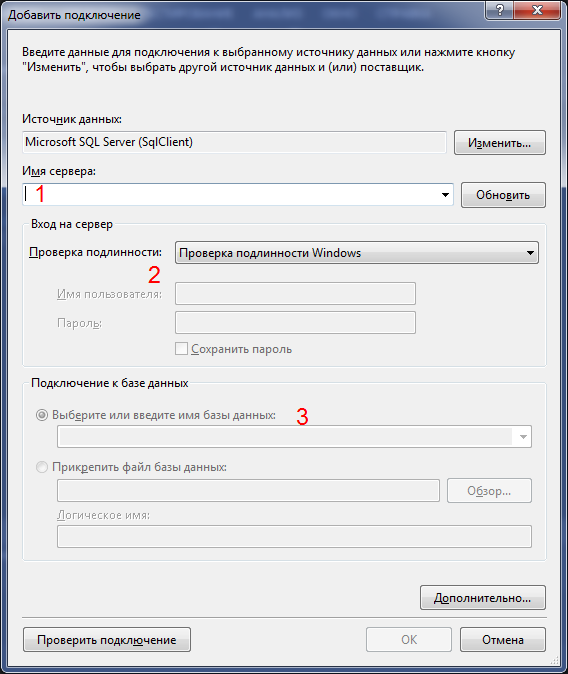


В качестве источника данных выберите Microsoft SQL Server: 

В следующем диалоговом окне произведите настройку SQL Server’а. Для этого нужно указать расположение сервера в поле 1. Если в выпадающем списке нет вашего сервера, попробуйте запустить SSMS, найти его там, потом скопировать имя сервера в поле 1.

В поле 2 выбираете тип аутентификации аналогично настройке запуска SSMS.

**НЕ ЗАБУДЬТЕ** указать имя базы данных, которую вы планируете использовать, в выпадающем списке 3.



## Работа с запросами SQL

### Добавление данных. Оператор insert

Оператор insert имеет следующий синтаксис (на самом деле здесь представлена упрощенный вид запроса):

INSERT INTO <имя таблицы>[(<имя столбца>,...)]

VALUES (<значение столбца>,…)

Как видно из представленного синтаксиса, список столбцов не является обязательным (об этом говорят квадратные скобки в описании синтаксиса). В том случае, если он отсутствует, список вставляемых значений должен быть полный, то есть обеспечивать значения для всех столбцов таблицы. При этом порядок значений должен соответствовать порядку, заданному оператором **CREATE TABLE** для таблицы, в которую вставляются строки. Кроме того, эти значения должны относиться к тому же типу данных, что и столбцы, в которые они вносятся. В качестве примера рассмотрим вставку строки в таблицу Product, созданную следующим оператором **CREATE TABLE**:

Приведем пример запроса insert:

insert into Marathon( MarathonName, CityName, CountryCode, YearHeld )

values ('Просто марафон', 'Новосибирск','RUS',2017)

Этот же запрос в сокращенной форме, без указания порядка столбцов:

insert into Marathon

values ('Просто марафон', 'Новосибирск','RUS',2017)

### Выборка данных. Оператор select

Оператор **SELECT** осуществляет выборку из базы данных и имеет наиболее сложную структуру среди всех операторов языка SQL.

**Простая выборка**

SELECT \* FROM PC;

который осуществляет выборку всех записей из объекта БД табличного типа с именем РС. При этом столбцы и строки результирующего набора не упорядочены. Чтобы упорядочить поля результирующего набора, их следует перечислить через запятую в нужном порядке после слова SELECT:

1. SELECT price, speed, hd, ram, cd, model, code
2. FROM PC;

**Объединение нескольких таблиц. Оператор inner join**

**Фильтрация результатов запроса**

Горизонтальную выборку реализует предложение **WHERE** предикат, которое записывается после предложения **FROM**. При этом в результирующий набор попадут только те строки из источника записей, для каждой из которых значение предиката равно **TRUE**. То есть предикат проверяется для каждой записи. Например, запрос «получить информацию о частоте процессора и объеме оперативной памяти для компьютеров с ценой ниже $500» можно сформулировать следующим образом:

В последнем запросе был применен предикат сравнения с использованием операции сравнения «<» (меньше чем). Кроме этой операции сравнения могут использоваться: «=» (равно), «>» (больше), «>=» (больше или равно), «<=» (меньше или равно) и «<>» (не равно). Выражения в предикатах сравнения могут содержать константы и любые поля из таблиц, указанных в предложении **FROM**. Символьные строки и константы типа дата/время записываются в апострофах.

Для выборки строк, частично удовлетворяющих запросу, можно использовать оператор like

**Сортировка результата запроса**

SELECT DISTINCT speed, ram

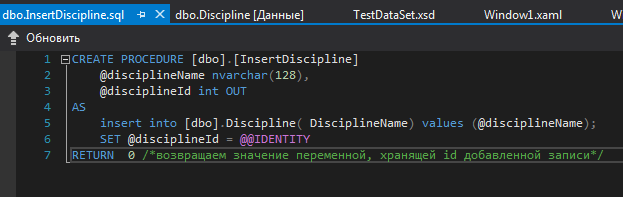
FROM PC

ORDER BY ram DESC;

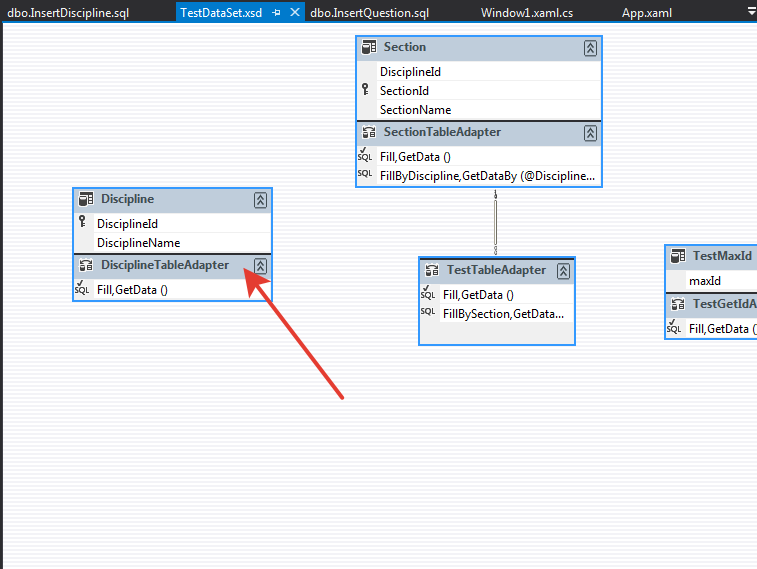
### Использование хранимых процедур для добавления данных (адаптировать для общего случая)

1. Создаем хранимую процедуру с возвращаемым значением

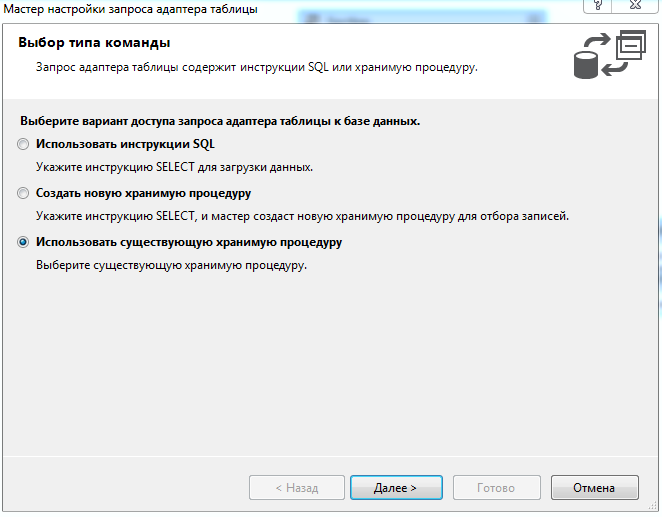
Кроме параметров, значения которых будут добавлены в таблицу, передаем еще один - @disciplineId с модификатором OUT. OUT указывает, что этот параметр будет передан из хранимой процедуры во внешнюю среду, т.е. является выходным. Через этот параметр мы будет передавать id добавленной записи. Инициализируется @disciplineId значением @@IDENTITY (id последней добавленной записи) с помощью оператора SET (можно было использовать и select).



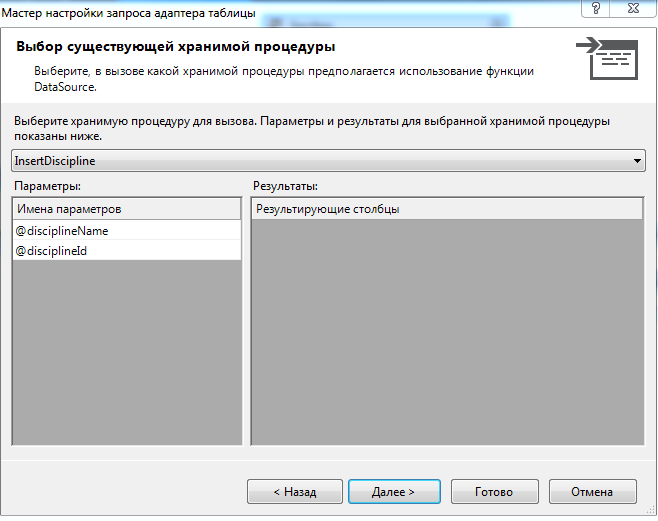
1. Открываем нужный датасет и в нем находим нужную таблицу:



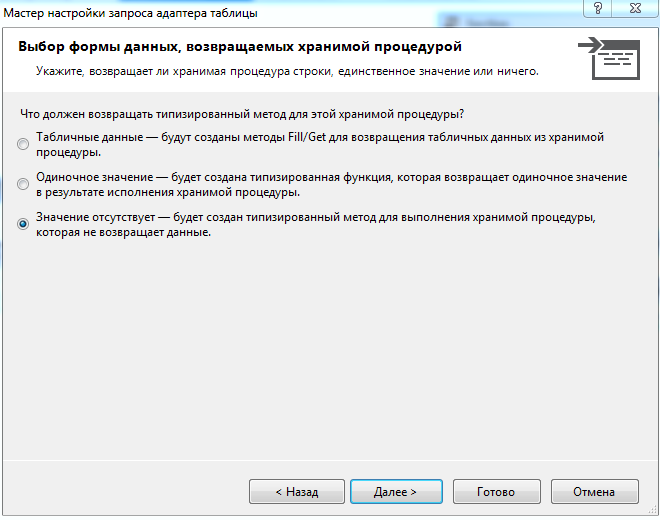
1. Правый клик по table adapte’у, создать запрос, там выбираем «Использовать существующую хранимую процедуру»:



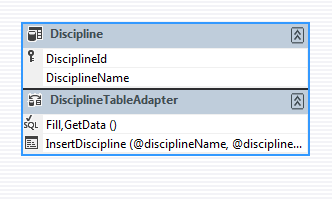
1. Выбираем созданную процедуру из списка:



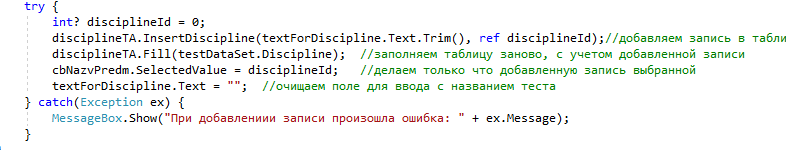
1. Так как процедура у нас ничего не возвращает (id передаем через параметр), выбираем третий пункт:



1. На оставшихся пунктах со всем соглашаемся, в результате видим, что в тейбл адаптер была добавлена новая запись, описывающая нашу процедуру:



1. Используем созданную процедуру:



# Типовые задачи

## Проверка подлинности пользователя (аутентификация)

См. проект “ресурсы\примеры типовых проектов\Авторизация”.