Лабораторная работа

Разработка приложений для работы с СУБД

Оглавление

Работа с базами данных

Цель лабораторной работы: изучить средства разработки приложений для работы с базами данных.

Средства: Visual Studio 2013 (Express), Microsoft SQL Server.

В наше время на рынке существует большое количество сервисов, предоставляющих функции серверов базы данных. По популярности в первую тройку входят **Oracle, MySQL, Microsoft SQL Server**. Каждый из этих движков баз, данных поддерживает стандартные и унифицированные технологии (как язык запросов **SQL**), так и обладает расширением функционала, (например: **Transact-SQL**, **PL/SQL**). К тому же, кампания, разработчик сервера баз данных встраивает свои продукты в собственные средства разработки приложений, обеспечивая функционал для быстрого, высокоуровневого доступа к данным для разработки приложений. Лабораторная работа посвящена работе с MS SQL Server используя модифицированный SQL – Transact-SQL и средствами для взаимодействия с системами управления базами данных (СУБД), входящими в платформу **.Net Framework 4.5**.

Платформа .NET определяет ряд пространств имен, которые позволяют взаимодействовать с реляционными базами данных. Все вместе эти пространства имени известны как **ADO.NET**. Платформа .NET поддерживает множество различных поставщиков данных, каждый из которых оптимизирован на взаимодействие с конкретной системой управления базами данных, такой как Microsoft SQL Server, Oracle, MySQL и т.д.

В качестве системы построения интерфейса была выбрана среда **WPF** (Windows Presentation Foundation), появившейся в .Net 3.0 и предоставляющий относительно **Windows Forms** более гибкий и функциональный механизм построения интерфейса. В основе WPF лежит векторная система отрисовки, не зависящая от разрешения и созданная с расчетом на возможности современного графического оборудования. Для создания интерфейса в WPF используется язык разметки **XAML** (Extensible Application Markup Language).

В технологиях WPF и **Silverlight** XAML действительно применяется главным образом для описания пользовательского интерфейса (но не только). Основное назначение XAML заключается в том, чтобы предоставить программистам и специалистам в других областях возможность совместной работы. То есть XAML становится единым языком общения, как правило, опосредованным инструментами разработки и предметно-ориентированными средствами проектирования. Но поскольку язык XAML (и вообще XML) может восприниматься человеком, то с ним можно работать, даже не имея ничего, кроме редактора Блокнот.

Необходимые команды SQL

SQL – язык запросов, позволяющий производить множество операций с данными. Стандартным набором для любой базы данных являются операции:

- Создание таблицы
- Добавление данных в таблицу
- Получение полной таблицы для просмотра
- Выборка данных по столбцам.
- Выборка данных по строкам (фильтрация таблицы)
- Изменение данных в таблице.
- Удаление данных из таблицы.

Не маловажна операция – выборка из нескольких таблиц, но в рамках данной работы рассматриваться не будет.

Рассмотрим вышеперечисленные операции на примере таблицы

Номер зач – кн.	ФИО	Дата поступления	Дата рождения
1001	Пушкин Александр Сергеевич	01.09.2007	14.03.1990
1002	Лермонтов Михаил Юрьевич	01.09.2007	1.10.1989
1003	Гоголь Василий Николаевич	01.09.2012	12.11.1995
1004	Достоевский Федор Михайлович	01.09.2005	4.08.1986

Ниже приведен лишь малая часть функционала SQL. Сам по себе зык довольно обширный, с большим количеством операторов, операций и функций.

Создание таблицы

Для создания таблицы используется команда **CREATE**. Она обладает следующим синтаксисом:

```
CREATE TABLE tablename (
    Column1,  Type, Params
    …
    ColumnN,  Type, Params  )
```

где в качестве Column указывается имя столбца, в качестве Type – тип данных, хранимых в столбце, в Params – дополнительные ограничения на значения данных в столбце, и прочие параметры. Операция CREATE определяет только шапку таблицы, но никак не данные, в ней хранимые. В SQL есть ограниченный набор типов данных, как в любом языке программирования, допиленный в некоторых расширениях языка.

Основные типы данных:

INT или INTEGER	Хранит любое число в диапазоне от -2147683648 до 2147683648
BOOL или BOOLEAN	Булево значение. 0 – false, 1 – true.
FLOAT	Значение с плавающей точкой
CHAR(M)	Хранит строку длинной M
DATE	Позволяет хранить данные

Для создания вышеописанной таблицы нужно выполнить следующую команду:

```
CREATE TABLE Students (
  [Номер зач-кн]     INTEGER NOT NULL,
  [ФИО]              CHAR (50),
  [Дата поступления] DATE,
  [Дата рождения]    DATE,
)
```

Поле [Номер зач-кн] помечено как NOT NULL чтобы предотвратить возможность отсутствия значения в какой-либо строке данного столбца, другими словами – оно является обязательным для ввода, и строка без номера зачетной книжки в таблицу добавлена не будет.

Добавление данных в таблицу

Добавление данных осуществляется с помощью оператора INSERT:

```
INSERT INTO tablename (Column 1, … , Column N) VALUES
                                        (Value 1, … , Value N)
```

Данная команда добавить новую строку сопоставив список столбцов и значений. Если добавление идет во все стобцы, то их можно не указывать в команде:

```
INSERT INTO tablename VALUES (Value 1, … , Value N)
```

Не стоит забывать про тип данных в столбцах!

Добавим в нашу таблицу несколько строк:

```
INSERT INTO Students VALUES (
      1001,
      'Пушкин Александр Сергеевич',
      '1.09.2007',
      '04.03.1990'
)

INSERT INTO Students VALUES (
      1002,
      'Лермонтов Юрий Михайлович',
      '1.09.2007',
      '1.10.1989'
)
```

Аналогично добавляются и другие строки.

Получение полной таблицы

Для получение данных из таблицы используется команда SELECT.

```
SELECT <Column 1, …, Column N>
FROM <Table 1, … , Table M>
WHERE <условие>
```

Данный оператор производит выборку тех строк из отмеченных таблиц в заданных столбцах в которых выполняется введённое условие.

Если нам требуется получить всю таблицу целиком список столбцов можно заменить символом «*».

Итак, для получения полной таблицы запрос будет выглядеть следующим образом:

```
SELECT * FROM <Table>
```

Для получения таблицы из примера получается запрос:

```
SELECT * FROM Students
```

который вернет нам всю табличку целиком, со всеми строками и столбцами.

Выборка столбцов

С помощью оператора SELECT можно получать некоторые столбцы, например:

```
SELECT ФИО,[Дата поступления] FROM Students
```

вернет нам следующую таблицу:

ФИО	Дата поступления
Пушкин Александр Сергеевич	01.09.2007
Лермонтов Михаил Юрьевич	01.09.2007
Гоголь Василий Николаевич	01.09.2012
Достоевский Федор Михайлович	01.09.2005

Выборка строк. Фильтрация

Для получения строк, удовлетворяющих определенным условиям нужно воспользоваться частью WHERE оператора SELECT.

Например, требуется вывести студентов, родившихся в 1990 году и позднее:

```
SELECT * FROM Students
WHERE [Дата рождения]>= '1.01.1990'
```

Номер зач – кн.	ФИО	Дата поступления	Дата рождения
1001	Пушкин Александр Сергеевич	01.09.2007	14.03.1990
1002	Лермонтов Михаил Юрьевич	01.09.2007	1.10.1989
1003	Гоголь Василий Николаевич	01.09.2012	12.11.1995

Условия можно комбинировать логическим оператором AND:

```
SELECT * FROM Students
WHERE [Дата рождения]>= '1.01.1990' AND [Номер зач - кн] < 1003
```

Номер зач – кн.	ФИО	Дата поступления	Дата рождения
1001	Пушкин Александр Сергеевич	01.09.2007	14.03.1990
1002	Лермонтов Михаил Юрьевич	01.09.2007	1.10.1989

Выборку под строкам и столбцам можно комбинировать, создавая новые таблицы во время запросов.

Основные операции сравнения:

Операция	Выполняемая функция
()	Меняет нормальные правила старшинства операций.
=	Проверяет на равенство
!=,^=,<>	Проверяет на неравенство
> <	"Больше чем" и "меньше чем"
>= >=	"Больше или равно" и "меньше или равно"

Основные логические операции:

Операция	Выполняемая функция
( )	Меняет нормальные правила старшинства операций.
NOT	Инвертирует результат логического выражения
AND	Устанавливает в TRUE логическое выражение, если оба условия TRUE.
OR	Устанавливает в TRUE логическое выражение, если одно из условий TRUE.

Изменение данных в таблице

Команда UPDATE позволит изменить данные одного или нескольких полей:

```
UPDATE [top (N)] <Table>
SET Column 1 = Value 1,
     …
    Column M = Value M
[WHERE <условие>]
```

Где: top(M) – M-кратное повторение операции (при однократном - не указывается).

Пример запроса:

```
UPTADE Students
SET ФИО = ‘Борис Николаевич Ельцин’
WHERE [Номер зач-кн = 1001]

SELECT * FROM Students
```

Такой запрос выведет нам таблицу:

Номер зач – кн.	ФИО	Дата поступления	Дата рождения
1001	Борис Николаевич Ельцин	01.09.2007	14.03.1990
1002	Лермонтов Михаил Юрьевич	01.09.2007	1.10.1989
1003	Гоголь Василий Николаевич	01.09.2012	12.11.1995
1004	Достоевский Федор Михайлович	01.09.2005	4.08.1986

Удаление строк

Операция удаления похоже на операцию UPDATE, за одним исключением - в ней нет полей для присвоения новых значений.

```
DELETE FROM <Table>
WHERE <условие>
```

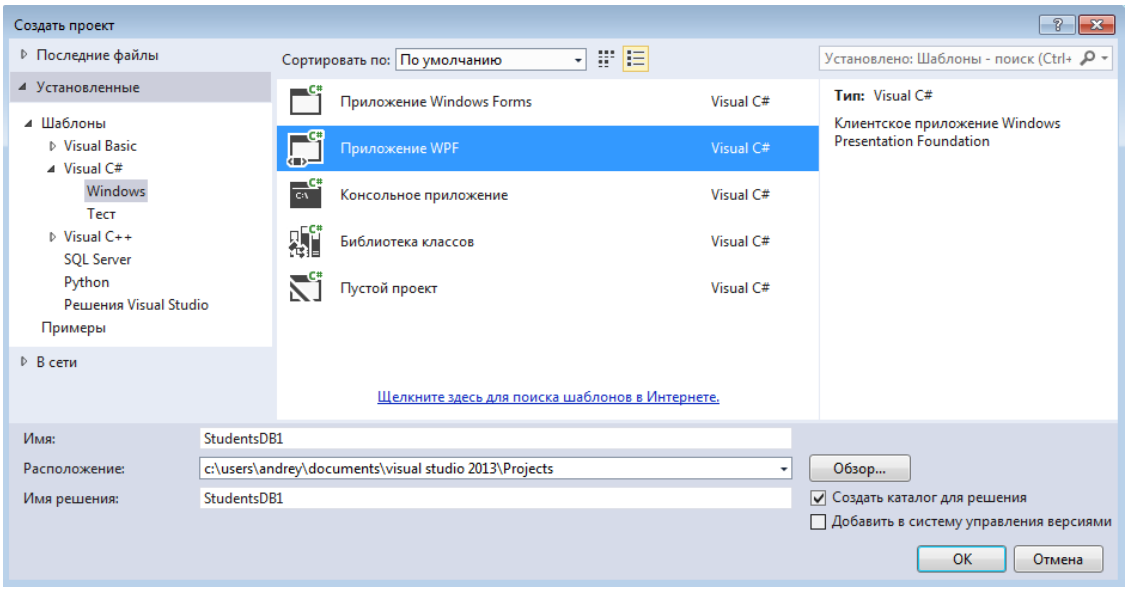
Условие позволяет идентифицировать одну или несколько строк для удаления.

Создание базы данных

В разделе описан процесс создания тестового приложения исходя из задания на лабораторную работу. Описание носит рекомендательно - справочный характер и преследует цель показать пример разработки приложения для взаимодействия с СУБД . В Visual Studio (особенно не Express) имеются разные способы создания баз данных и таблиц (в том числе визуальный).

Создание проекта

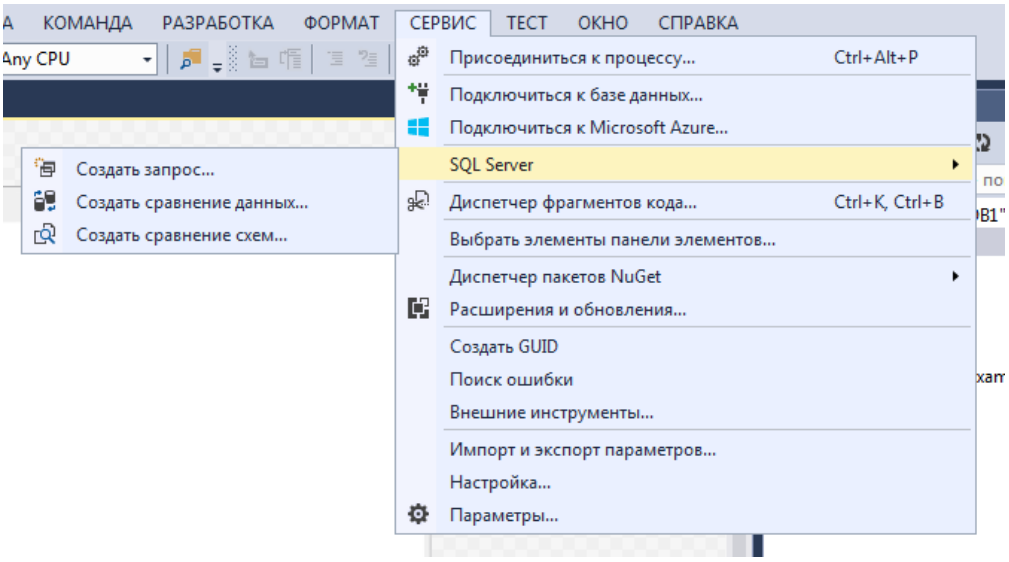
1. Создать новое решение.
2. В качестве типа проекта выбрать «Приложение WPF».
3. Выбрать имя для решения.



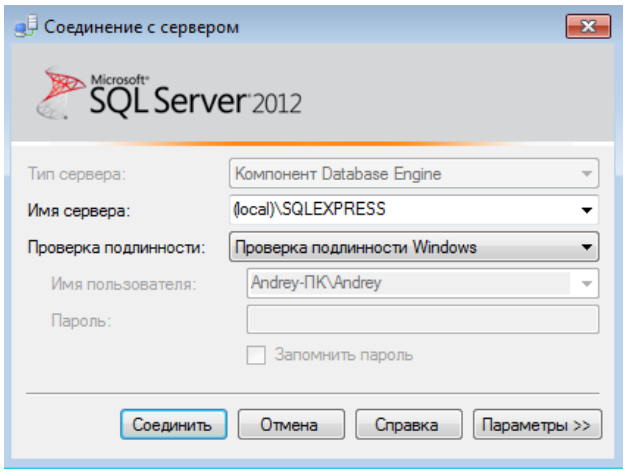
Как только решение будет создано на экране VS появится заготовка окна для создания интерфейса, и окно, содержащее код XAML для точной настройки элементов интерфейса.

Установка соединения с СУБД и создание базы данных

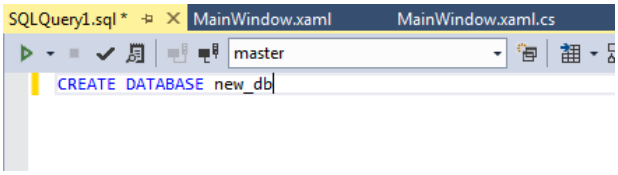
1. В СЕРВИС -> SQL Server выбрать пункт «Создать запрос».



2. В появившемся окне ввести имя сервера (по умолчанию, для сервера установленного на компьютере, на котором ведется разработка приложения - «**(local)\SQLEXPRESS**»). Нажать кнопку «**Соединить**».

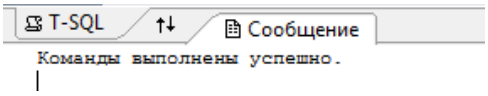


3. Появится окно ввода SQL запросов. В нем можно производить взаимодействие с сервером на зыке запросов, при необходимости сохраняя запросы в *.sql файлы, которые позже, можно присоединить в проекту.
4. Создать новую базу данных, используя команду «**CREATE DATABASE** <database_name>».

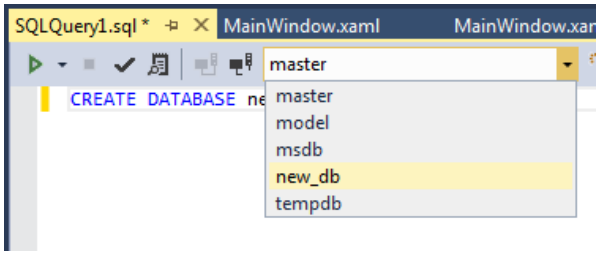


Нажмите кнопку «**Выполнить**».

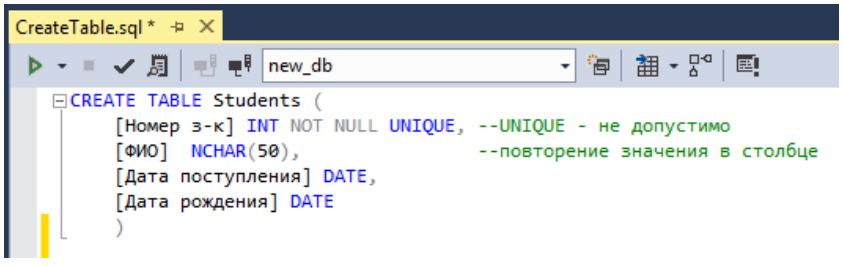
При успешном выполнении команды, на сервере создастся база данных с требуемым именем, и в окне ниже будет подтверждение о успешном выполнении.



5. Выбрать созданную базу данных в выпадающем списке, над полем ввода запросов.

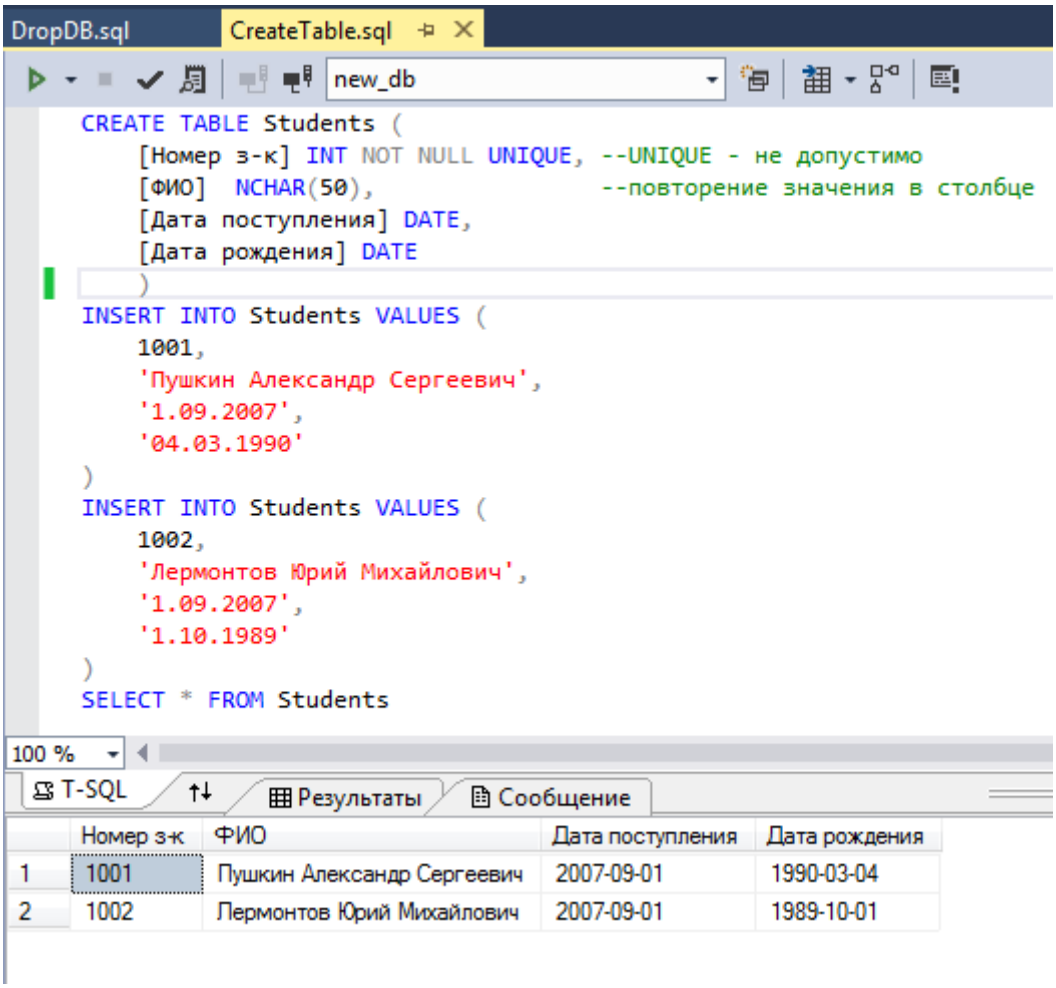


6. С помощью команды «**CREATE TABLE**» создать в базе данных нужную таблицу.



Выполнить запрос. ( )

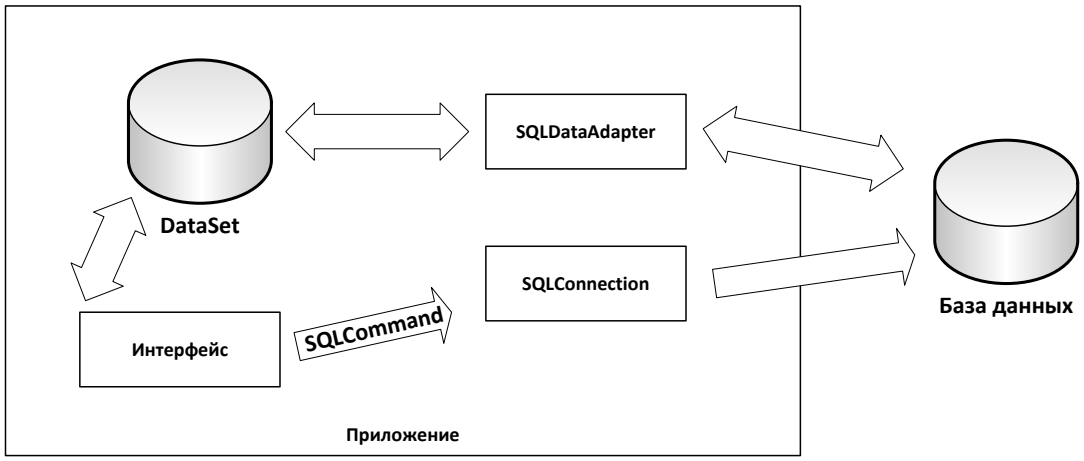
7. Можно проверить, создалась ли таблица с помощью команды **SELECT.**


DropDB.sql
CreateTable.sql
new_db
CREATE TABLE Students (
[Номер з-к] INT NOT NULL UNIQUE, --UNIQUE - не допустимо
[ФИО] NCHAR(50), --повторение значения в столбце
[Дата поступления] DATE,
[Дата рождения] DATE
)
INSERT INTO Students VALUES (
1001,
'Пушкин Александр Сергеевич',
'1.09.2007',
'04.03.1990'
)
INSERT INTO Students VALUES (
1002,
'Лермонтов Юрий Михайлович',
'1.09.2007',
'1.10.1989'
)
SELECT * FROM Students
100 %
T-SQL
Результаты
Сообщение
Номер з-к
ФИО
Дата поступления
Дата рождения
1
1001
Пушкин Александр Сергеевич
2007-09-01
1990-03-04
2
1002
Лермонтов Юрий Михайлович
2007-09-01
1989-10-01

Разработка класса для взаимодействия с базой данных

Один из способов создания приложения для работы СУБД является использование автономного уровня Ado.Net. Суть состоит в использовании класса **DataSet** как локального хранилища данных из СУБД. Связь между ними обеспечивает класс **SqlDataAdapter**, который посредством SQL запросов взаимодействует с базой данных, а результат запросов сохраняет в **DataSet**.

Для выполнения запросов фильтрации, и добавления данных в таблицу можно создавать команды с помощью класса SqlCommand и SqlConnection.



В целях удобство следует выносить все операции для работы с СУБД в отдельный класс, расположенный в отдельном файле. В тестовом приложении это класс **Database** расположенный в **datadase.cs**.

Конструктор класса

Разрабатываемый класс должен содержать все функции для работы с базой данных, скрывая их для последующего использования посредством инкапсуляции. Требуются следующие функции:

- Получение полной таблицы.
- Добавление новых записей.
- Удаление имеющихся записей.
- Получение фильтрованной таблицы.

Так же в класс должен содержать средства отображения результатов работы в виде таблицы значений.

Исходя из этого в класс добавлен атрибут типа DataTable, в котором находится ссылка на объект, визуализирующий таблицу на интерфейсе приложения.

Класс должен содержать объект DataSet, в котором хранятся результаты выполнения запросов к СУБД.

Для установления связи с СУБД требуется экземпляр класса SqlDataAdapter и строка подключения, по этому в качестве атрибутов создаваемого класса требуется объявить еще две переменные.

Исходя из этого, получается заготовка класса Database, в котором есть конструктор, требующий ссылку для DataGrid для последующей отображения данных на интерфейсе.

```
class Database
{
    //ссылка на объект, визуализирующий таблицу на интерфейсе.
    DataGrid dataGrid;

    DataSet ds = new DataSet("Students");

    //строка подключения
    string connectString = "Integrated Security = SSPI;" +
                           "Initial Catalog=new_db;" +
                           "Data Source=(local)\\SQLEXPRESS";

    //адаптер данных
    SqlDataAdapter dAdapt;

    public Database(DataGrid tableGrid)
    {
        dataGrid = tableGrid;

        //создаем новый объект SqlDataAdapter
        dAdapt = new SqlDataAdapter("", connectString);

    }
}
```

Строка подключения "Integrated Security = SSPI;Initial Catalog=new_db; “Source=(local)\\SQLEXPRESS" содержит в себе сведения: название базы данных, с которой будет производится работа (new_db) и путь до СУБД (*(local)\\SQLEXPRESS*).

Получение полной таблицы

В адаптере данных SqlDataAdapter есть атрибут SelectCommand типа SqlCammand. Он в себе хранить команду, выполняющуюся при вызове метода Fill, который заносит результат команды в DataSet.

```
public void GetFullTable()
{
    //задаем команду для выбора всей таблицы.
    dAdapt.SelectCommand.CommandText = "SELECT * FROM Students";

    GetTable();
}

public void GetTable()
{
    ds.Clear();
    try
    {
        //наполняем DataSet запрашиваемой таблицей
        dAdapt.Fill(ds, "Students");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Ошибка!", MessageBoxButton.OK,
                                                    MessageBoxImage.Error);
        return;
    }

    //передаем таблицу в DataGrid
    dataGrid.ItemsSource = ds.Tables[0].DefaultView;
}
```

Вышеприведенная функция соединяется с сервером базы данных, совершает запрос SELECT и записывает результат в объект DataSet, создав в нем таблицу Students. Затем Объект DataGrid связывается с получившейся в результате запроса таблицей в DataSet. Все ошибки про соединении с базой отлавливаются с помощью исключений.

Добавление новых записей

Для удобной организации добавления новых записей был создан вспомогательный класс Student, представляющей собой (информационно) одну строку таблицы:

```
class Student
{
    public int number;
    public string name;
    public DateTime BirthDate;
    public DateTime EnteringDate;
}
```

Экземпляр класса Student формируется при нажатии кнопки добавить из данных введенных пользователем и передается в класс Database, для обработки данных и занесении их (данных) в таблицу. При формировании сложных запросов удобно использовать в них параметры, добавляемые в запрос во время его выполнения. Параметры позволяют использовать один и тот же запрос, подставляя в него разные данные. (Можно просто формировать в качестве запроса строку, преобразуя в строковый формат все необходимые данные, но параметры – более безопасный способ.)

Алгоритм составления команды с параметрами:

1. Создать объект SqlConnection, передав в него строку подключения.
2. Создать объект SqlCommand, передв в него созданный ранее SqlConnection, и строку с запросом, в которой имена параметров начинаются с @.
3. С помощью метода Add у коллекции Parameters в объекте SqlCommand, добавить в коллекцию параметры, используя их имя и тип (тип соответствующего поля в таблице СУБД)(Тип представлен экземплром класса SqlDBType). Метод Add создаст экземпляр SqlParametr и добавит его в коллекцию.
4. Добавить параметру значение. SqlCommand.Paramers[“@имя“] = Значение.
5. Установить соединение с СУБД с помощью метода SqlConnection.Open().
6. У объекта SqlCommand вызвать объект ExecuteScalar().

```
public void AddStudent(Student student)
{

    //создаем текст команды с параметрами
    string cmd = "INSERT INTO Students VALUES (@number, @name,@enter,@birth, )";

    using (SqlConnection connection = new SqlConnection(connectString))
    {
        //создаем команду, используя ее текст и объект подключения
        SqlCommand InsertCmd = new SqlCommand(cmd, connection);

        //вставляем в команду на места параметров фактические значения
        InsertCmd.Parameters.Add("@name", SqlDbType.NVarChar);
        InsertCmd.Parameters["@name"].Value = student.name;

        InsertCmd.Parameters.Add("@number", SqlDbType.NVarChar);
        InsertCmd.Parameters["@number"].Value = student.number;

        InsertCmd.Parameters.Add("@birth", SqlDbType.Date);
```

```
        InsertCmd.Parameters["@birth"].Value =
                                        student.BirthDate.ToString("dd.MM.yyyy");

        InsertCmd.Parameters.Add("@enter", SqlDbType.DateTime2);
        InsertCmd.Parameters["@enter"].Value =
                                    student.EnteringDate.ToString("dd.MM.yyyy");

        try
        {
            //попытка отправить запрос на севрер
            connection.Open();
            InsertCmd.ExecuteScalar();
            GetTable();
        }
        catch (Exception ex)
        {
            // сообщение о неудаче
            MessageBox.Show(ex.Message);
        }
    }
}
```

Удаление записей

Для удаления записей надо сформировать запрос с командой DELETE. Для отправки запроса надо сделать следующие:

1. Создать объект SqlConnection, передав в него строку подключения.
2. Создать объект SqlCommand, передав в него созданный ранее SqlConnection, и строку с запросом.
3. Установить соединение с СУБД с помощью метода SqlConnection.Open().
4. У объекта SqlCommand вызвать объект ExecuteScalar().

Для оптимизации расходов ресурсов создание объекта SqlConnection лучше производить в блоке using () { }, так как в этой операции используются не управляемый код, и не использованные ресурсы после его выполнения не будут удалены сборщиком мусора. Блок using позволяет сборщику мусора понять, на каком этапе не управляемый объект будет не нужен, и высвободить использованную им память.

```
public void DeleteSelectedRow()
{
    //берем выделенню строку из dataGrid
    DataRowView row = (DataRowView)dataGrid.SelectedItem;

    //слздаем текст запроса на удаление
    string cmd = "DELETE FROM Students WHERE [Номер з-к] = " + row.Row.ItemArray[0];

    using (SqlConnection connection = new SqlConnection(connectString))
    {
        //создаем команду
        SqlCommand DelCmd = new SqlCommand(cmd, connection);

        try
        {
            connection.Open();
            DelCmd.ExecuteScalar();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "Ошибка удаления!", MessageBoxButton.OK,
                                                            MessageBoxImage.Error);
```

```
        }

        GetTable();
    }
}
```

Получение фильтрованной таблицы

Для получения фильтрованной таблицы был создан вспомогательный класс TableFilter, который описывает параметры, по которым формируется запрос на фильтрованную таблицу. Объект этого класса создается по нажатию кнопки «Фильтровать» и передается в Database в качестве свойства.

```
class TableFilter
{
    public int minAge;
    public int maxAge;
    public DateTime minEnternigDate;
    public DateTime maxEnteringDate;
}
```

Добавлен в Database закрытый атрибут класса TableFilter:

```
TableFilter filter;
```

```
public TableFilter Filter
{
    set
    {
        filter = value;

        //при пустом фильтре, запрашиваем полную таблицу.
        if (value == null)
        {
            GetFullTable();
            return;
        }

        //формируем базовый запрос со студентами,
        //возраст которых привышает минимульный
        //базовое значение мин. возраста = 0, если не указано другое
        string cmd = "Select * from Students WHERE DATEDIFF(year,[Students].[Дата
                                            рождения],GETDATE()) > " + filter.minAge;

        //далее, приписывем условия через AND
        //для других фильтров (если они указаны)
        if (filter.maxAge > 0)
        {
            cmd += " AND DATEDIFF(year,[Students].[Дата рождения],GETDATE()) < " +
                                                                    filter.maxAge;
        }

        if (filter.minEnternigDate!= new DateTime())
        {
            cmd += string.Format(" AND DATEFROMPARTS ({0},{1},{2}) <= [Students].[Дата
                                                                    поступления]",
                        filter.minEnternigDate.Year, filter.minEnternigDate.Month,
                                                    filter.minEnternigDate.Day);
        }

        if (filter.maxEnteringDate != new DateTime())
```

```
        {
            cmd += string.Format(" AND DATEFROMPARTS ({0},{1},{2}) >= [Students].[Дата
                                                                    поступления]",
                                 filter.maxEnteringDate.Year, filter.maxEnteringDate.Month,
                                                              filter.maxEnteringDate.Day);
        }

        Console.WriteLine(cmd);

        //передаем запрос адаптеру
        dAdapt.SelectCommand.CommandText = cmd;

        //запрашиваем нужную таблицу
        GetTable();
    }
    get
    {
        return filter;
    }
}
```

Перенос строковых констант на другую строчку в VS не возможен! Для переноса используется оператор +. См. листинг с объявлением строки подключении.

В этой функции формование запроса происходит без использований SqlParametr, а с помощью преобразования нужных данных в строковый формат с помощью метода string.Format(…).

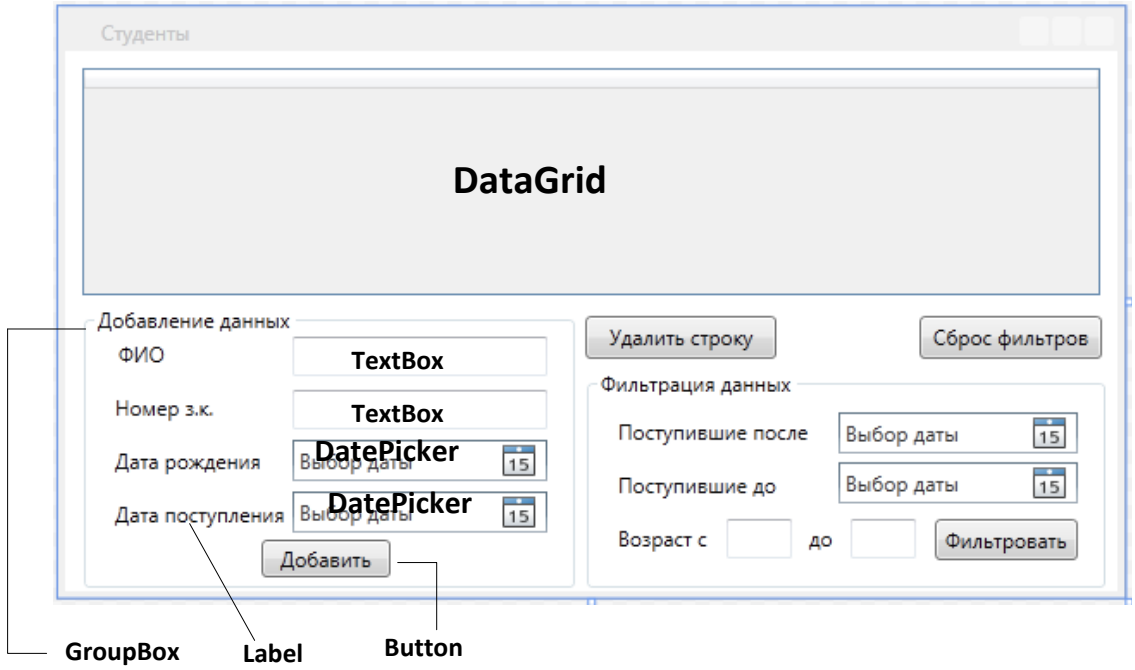
В запросе используются следующие T – Sql функции:

- GETDATE для получения текущей даты;
- DATEFROMPARTS для получения переменной типа DATE из отдельных данных (год, месяц, число) ;
- DATEDIFF для поиска разности между датами (вычисление возраста студента);

Разработка интерфейса.

Используемые компоненты

Для создания интерфейса применялся визуальный редактор, генерирующий код XAML (стандартный конструктор). Ниже обозначены основные типы элементов, использованные в приложении:



Имена элементов интерфейса (XAML):

```
<TextBox x:Name="tbFIO" … />
<TextBox x:Name="tbNumber" … />
<DatePicker x:Name="dpBirthDate" … />
<DatePicker x:Name="dpEnteringDate"…>/>
<Button x:Name="bAddStudent" Content="Добавить" … />
```

```
<DatePicker x:Name="dpEnteringMin" … />
<DatePicker x:Name="dpEnteringMax" … />
<TextBox x:Name="tbAgeMin" … />
<TextBox x:Name="tbAgeMax" … />
<Button x:Name="bFilter" Content="Фильтровать" … />
```

```
<Button x:Name="bDelete" Content="Удалить строку" … />
<Button x:Name="bCancelFilters" Content="Сброс фильтров" … />
```

Так же следует запретить добавление новых строк через DataGrid:

```
<DataGrid CanUserAddRows="False" x:Name="dgTable" … />
```

Связь с классом Database

Весь функционал для работы с СУБД находится в классе Database. Поэтому в классе окна (MainWindow) был объявлен атрибут этого типа, и выполнена его инициализацию в конструкторе.

```
Database db;

public MainWindow()
{
    InitializeComponent();
    db = new Database(dgTable);
    db.GetFullTable();
}
```

Так же в конструкторе мы вызываем функцию GetFullTable() для получения полной таблицы, для его начального отображения.

Связать событие интерфейсного элемента с функцией в коде можно, как и из кода программы, например, используя лямбда выражения или через атрибуты экземпляра соотв. элемента, так и из кода XAML. Связь события нажатия кнопки bAddStudent с функцией bAddStudent_Click приведена на примере ниже.

```
<Button x:Name="bAddStudent" Content="Добавить" … Click="bAddStudent_Click"/>
```

Аналогично, привязаны остальные кнопки к своим функциям.

Для добавления студента сначала формируется объект типа Student. В него с интерфейса записываются данные полей (имя, номер, год рождения, дата поступления). Затем, методом AddStudent(…) этот объект попадает в Database для дальнейшего добавления в базу.

```
private void bAddStudent_Click(object sender, RoutedEventArgs e)
{
    Student stud = new Student();
    try
    {
        stud.name = tbFIO.Text;
        stud.number = Convert.ToInt32(tbNumber.Text);
        stud.BirthDate = (DateTime)dpBirthDate.SelectedDate;
        stud.EnteringDate = (DateTime)dpEnteringDate.SelectedDate;
    }

    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Проверьте введенные данные!", MessageBoxButton.OK,
                                                                   MessageBoxImage.Error);
        return;
    }
    db.AddStudent(stud);
}
```

Для фильтрации сначала формируется объект TableFilter, в который попадает информация о критериях фильтрации. Затем в качестве свойства TableFIlter передается в DataBase

```
private void bFilter_Click(object sender, RoutedEventArgs e)
{
    TableFilter filter = new TableFilter();

    // проверяем введена ли дата
    if (dpEnteringMin.SelectedDate != null)
        filter.minEnternigDate = (DateTime)dpEnteringMin.SelectedDate;

    if (dpEnteringMax.SelectedDate != null)
        filter.maxEnteringDate = (DateTime)dpEnteringMax.SelectedDate;

    try
    {
        filter.maxAge = Convert.ToInt32(tbAgeMax.Text);
    }
```

```
    catch (Exception ex)
    {
        filter.maxAge = 0;
    }

    //если мин. возраст не введен, то он = 0.
    try
    {
        filter.minAge = Convert.ToInt32(tbAgeMin.Text);
    }
    catch (Exception ex)
    {
        filter.minAge = 0;
    }

    //задаем значение свойства
    db.Filter = filter;
}
```

При нажатии «Сброс фильтров» в Database передается нулевой объект null, тем самым сбрасывая фильтры. Затем сбрасываются веденые значения на интерфейсе.

```
private void bCancelFilters_Click(object sender, RoutedEventArgs e)
{
    db.Filter = null;

    dpEnteringMin.SelectedDate = null;
    dpEnteringMax.SelectedDate = null;
    tbAgeMax.Text = "";
    tbAgeMin.Text = "";
}
```

Операция удаления полностью реализована в Database, необходимо только вызвать соответствующий метод.

```
private void bDelete_Click(object sender, RoutedEventArgs e)
{
    db.DeleteSelectedRow();
}
```

Список литературы

[1] «Transcat-SQL Википедия.,» [В Интернете]. Available: https://ru.wikipedia.org/wiki/Transact-SQL.

[2] «DB-Engines Ranking,» [В Интернете]. Available: http://db-engines.com/en/ranking.

[3] «PL/SQL. Википедия,» [В Интернете]. Available: https://ru.wikipedia.org/wiki/PL/SQL.

[4] Э. Троелсен, Язык программирования C# 5.0 и платформа .NET 4.5, 6-е изд., Издательский дом “Вильямс”, 2013.

[5] А. Натан, WPF4. Подробное руководство., Издательство «Символ-Плюс»., 2011.

[6] «Операции в SQL.,» [В Интернете]. Available: http://articles.org.ru/docum/sql_oper.php.

[7] «Типы данных SQL,» [В Интернете]. Available: http://www.site-do.ru/db/sql2.php.

[8] «Сеть разработчиков Майкрософт,» [В Интернете]. Available: http://msdn.microsoft.com/ru-RU/.