**Лабораторная работа №5.**

**Связывание приложения на С# с базой данных под управлением PostgreSQL**

**Цели лабораторной работы:**

* Познакомиться c библиотекой в C# для связывания приложения с БД;
* Изучить некоторые шаблоны проектирования, связанные с работой с БД;
* Освоить на практике основы взаимодействия с БД под управлением PostgreSQL в приложении на C#.

**Задание на лабораторную работу:**

При выполнении работы нужно использовать БД, созданную в предыдущих лабораторных работах. Необходимо создать приложение на языке программирования С# с графическим интерфейсом. Для любой одной таблицы в БД, содержащей хотя бы один внешний ключ на другую таблицу, приложение должно:

* Выводить строки выбранной таблицы;
* Предоставлять любой фильтр по значениям строк (например, «Дата с … по …»);
* Предоставлять возможность добавить новую строку в таблицу;
* Предоставлять возможность удалить строку из таблицы.

Требования к реализации:

* В выводе таблицы должны показываться осмысленные значения (например, если таблица «Чек» ссылается на таблицу «Товар», нужно вывести не id товара, а его название);
* При добавлении новой записи пользователь также должен не ввести значение внешнего ключа, а выбрать строку из списка;
* Или сохранение, или удаление строки должно быть реализовано с помощью функции на PL/pgSQL;
* Фильтроваться значения должны с помощью SQL-запроса, а не в готовой коллекции на клиенте;
* При разработке нужно использовать шаблоны проектирования, связанные с работой с БД.

Отчет должен содержать диаграмму разработанных классов, организующих взаимодействие с БД, их исходный код, а также экранные формы работы приложения.

**1. Создание проекта**

Разработку приложения будем проводить в Visual Studio. Для создания проекта выбираем: Файл -> Создать -> Проект -> Visual C# -> Приложение Windows Forms (.NET Framework), в графе «имя» задаем имя и жмем «Ок».

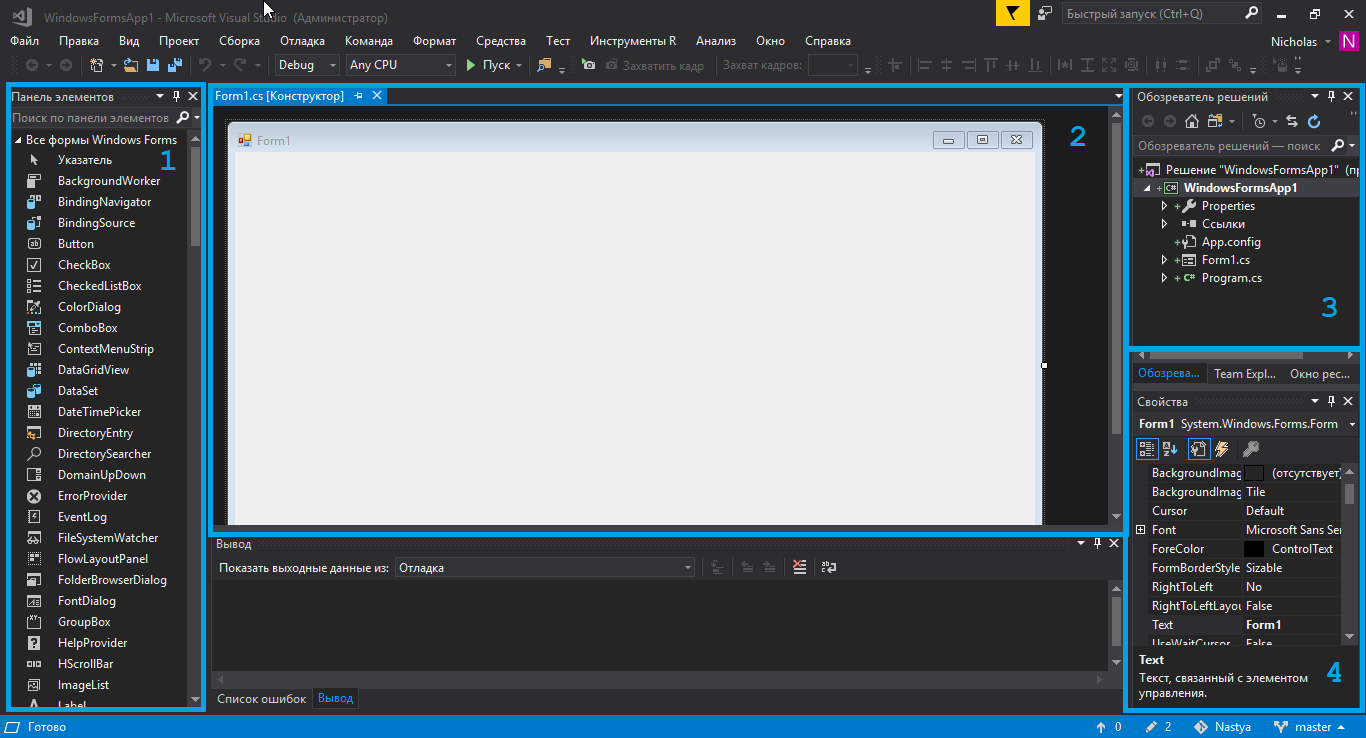


Рисунок 1

После создания проекта у нас открывается окно (Рисунок 1). Рассмотрим, что там есть:

1. Элементы, известные еще с Lazarus/Delphi, например: Button, Label, CheckBox. Некоторые компоненты имеют другие имена (в сравнении Lazarus/Delphi), например: Edit (Lazarus) – TextBox (Visual Studio).
2. Непосредственно сама форма, на которой мы будем размещать наши элементы.
3. Обозреватель решений – тут лежат все файлы проекта.
4. Свойства выделенных объектов. Можно задать цвет, шрифт, размер, видимость и т.д.

**2. Создание формы**

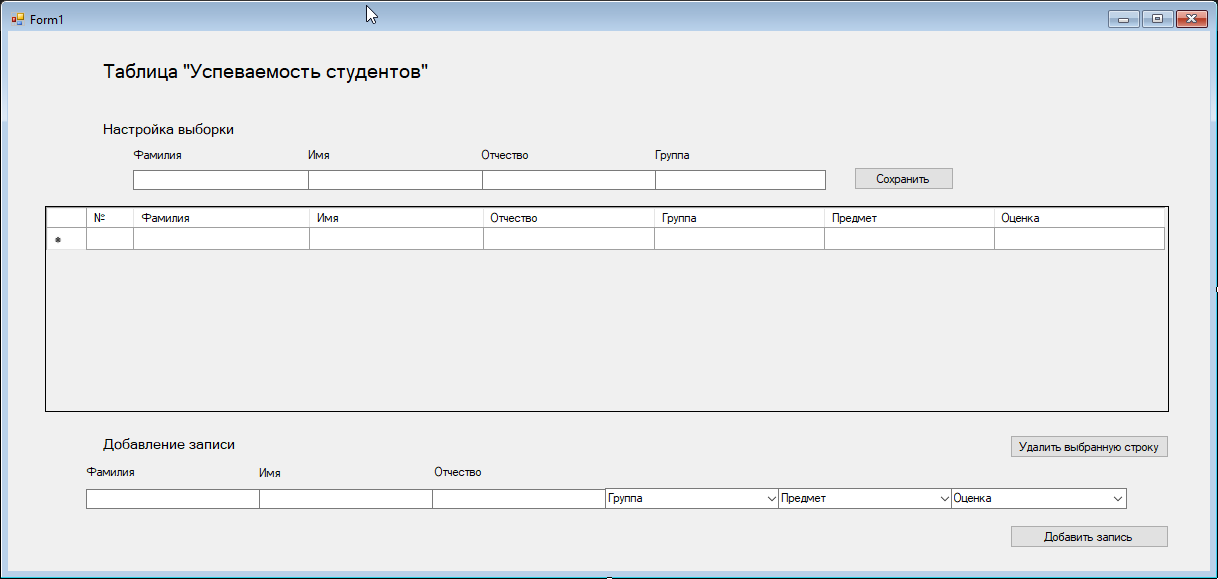


Рисунок 2

Разобравшись с созданием проекта, приступим к форме. Из компонентов нам потребуется: Label, TextBox, Button, ComboBox, а так же DataGridView, который сейчас рассмотрим более подробно.

DataGridView – настраиваемая таблица для отображения данных. Она имеет множество свойств, который сильно помогают в разработке приложений для работы с базами данных.

Установив DataGridView на форму, появится следующее окно (Рисунок 3), которое позволит настроить таблицу для дальнейшего ее заполнения.

Далее, жмем «Добавить столбец», появится окно настройки стобца (Рисунок 4). В поле «Текст заголовка» указываем название стобца, в поле «Тип» можно выбрать вид ячеек. Изначально стоит обычный TextBox, однако можно выбрать и ComboBox. В поле «Имя» указываем имя столбца, которое в дальнейшем будем использовать при написании программы.

Остальные настройки в данной лабораторной работе не потребуются, но, в случае надобности информация о том, как с ними работать, она легко найдется в интернете.

С DataGriedView стало все понятно, теперь можно расположить все элементы по форме и перейти к следующему пункту.

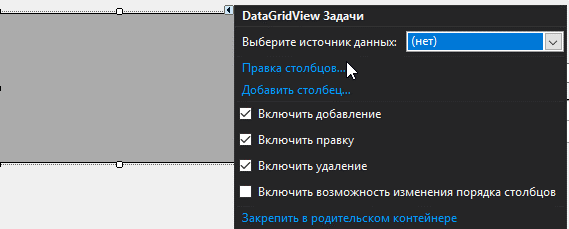


Рисунок 3

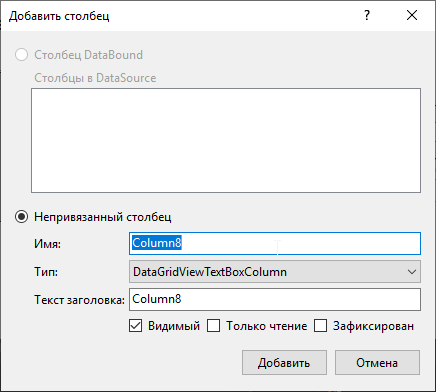


Рисунок 4

**3. Подключение библиотеки NPGSql**

NPGSql – библиотека, предназначенная для упрощения связи C#-приложения и базы данных Postgresql.

Поскольку в Visual Studio нет NPGSql, то скачаем ее, например, [отсюда](http://pgfoundry.org/frs/?group_id=1000140). И качаем Npgsql-2.2.3-net45.zip, разархивируем и переносим в удобное для нас место.

Теперь нам надо «подружить» Visual Studio и NPGSql. Для этого, в обозревателе решений (Рисунок 5), жмем ПКМ на «Ссылки», затем «Добавить ссылку…», «Обзор» (Рисунок 6). В скачанной библиотеке выделяем два файла (Рисунок 7) и жмем «Добавить».

Пишем в коде: using Npgsql; using Mono.Security;

Библиотека подключена, переходим к написанию кода.

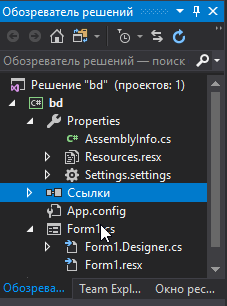


Рисунок 5

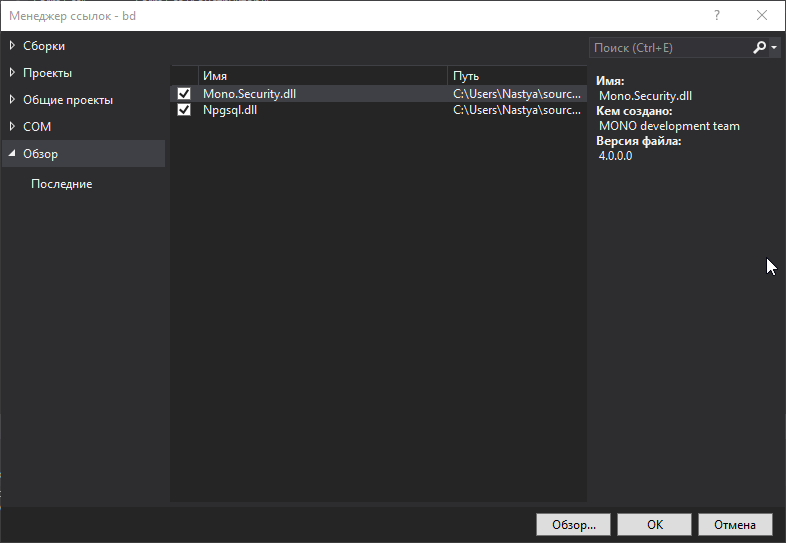


Рисунок 6

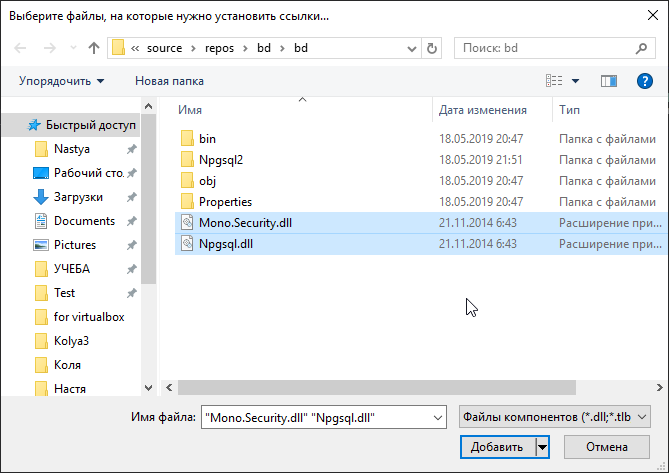


Рисунок 7

**4. Соединение с БД**

Соединение открывается с помощью конструктора *NpgsqlConnection* класса Npgsql Connection, одна из перегрузок которого принимает на вход имя сервера, порт, логин, пароль и имя БД.

Ниже показан пример получения соединения:

using Npgsql;

using Mono.Security;

public class NpgsqlUserManual

{

public static void Conect()

{

*// Сведения базы данных должен соответствовать определенному формату  
 // localhost - адрес БД, 5432 - порт БД, lab\_db - имя БД*

string sql = “Server = **127.0.0.1**; Port = **5432**; User Id = **lab\_user**; Password = **lab\_user**; Database = **lab\_db**”;

*// Установка соединения с БД*

NpgsqlConnection conn = new NpgsqlConnection(sql);

conn.Open();

*// Тут можно делать запросы, используя соединение*

*// Всегда соединение должно быть в итоге закрыто!*

conn.Close();

}

При работе с Npgsql всегда нужно закрывать соединение в конце работы приложения. Это очень важно, поскольку без ручного закрытия соединения могут «повисать в воздухе», пока приложение не завершит работу или соответствующие сессии не будут завершены на стороне СУБД. Поскольку количество одновременных подключений к БД, как правило, ограничено, это может привести к появлению ошибок, которые показывают себя только при нагрузочном тестировании или во время эксплуатации приложения.

БД, к которой совершается подключение, имеет структуру, описанную в предыдущих лабораторных работах.

**5. Выполнение запросов на чтение данных**

С помощью конструктора *NpgsqlCommand* класса Command Npgsql можно создать экземпляр класса. Инициализирует новый экземпляр класса Command Npgsql с текстом запроса. Это экземпляр представляет инструкцию или функцию SQL (хранимую процедуру) для выполнения в базе данных PostgreSQL.

Существует 4 перегрузки данного конструктора:

* NpgsqlCommand () инициализирует новый экземпляр класса Command Npgsql .
* *NpgsqlCommand (String)* инициализирует новый экземпляр класса Command Npgsql с текстом запроса.
* *NpgsqlCommand (String, NpgsqlConnection)* инициализирует новый экземпляр NpgsqlCommand класса с текстом запроса и Npgsql Connection .
* NpgsqlCommand (String, NpgsqlConnection, NpgsqlTransaction) Инициализирует новый экземпляр  NpgsqlCommand класса с текстом запроса, в NpgsqlConnection , и NpgsqlTransaction.

Далее необходимо вызвать метод ExecuteReader() класса NpgsqlCommand. С помощью этого метода происходит получение данный, которые следует записать в объект класса NpgsqlDataReader.

После выполнения запроса желательно вызвать метод *close()* у выражения, чтобы освободить ресурсы.

Ниже показан пример использования *NpgsqlCommand (String, NpgsqlConnection)* иExecuteReader():

*/\*  
 В результате выполнения метода получим:  
  
 ID = 1; фамилия = Петров; имя = Иван; отчество = Викторович*

*ID = 2; фамилия = Иванов; имя = Петр; отчество = Алексеевич  
 ID = 3; фамилия = Сидоров; имя = Андрей; отчество = Иванович  
 \*/*

public static void Read()

{

string sql = “Server = **127.0.0.1**; Port = **5432**; User Id = **lab\_user**; Password = **lab\_user**; Database = **lab\_db**”;

*// Установка соединения с БД*

NpgsqlConnection conn = new NpgsqlConnection(sql);

NpgsqlCommand com = new NpgsqlCommand(“SELECT t.\* FROM public. students t”, conn);

conn.Open();

NpgsqlDataReader reader;

reader = com.ExecuteReader();

string[,] nums = new string[4,5];

int a = 0;

while (reader.Read())

{

try

{

nums[a, 0] = reader.GetInt64(0).ToString();

nums[a, 1] = reader.GetString(2);

nums[a, 2] = reader.GetString(3);

nums[a, 3] = reader.GetString(4);

}

catch { }

a++;

}

conn.Close();

}

}

Внутри *NpgsqlDataReader* существует курсор, который перемещается по полученной выборке. Изначально курсор стоит на позиции до первого элемента. С помощью метод *read()* курсор можно перемещать вперед. Метод возвращает true, если следующий элемент существует и курсор переместился.

Чтение значений из *объекта NpgsqlDataReader* осуществляется разными функциями для разных типов данных:

* GetInt64 (Int32) получает значение указанного столбца в виде 64-разрядного целого числа со знаком.
* GetInt32 (Int32) получает значение указанного столбца в виде 32-разрядного целого числа со знаком.
* GetFloat (Int32) получает значение указанного столбца как число с плавающей запятой одинарной точности.
* GetString (Int32) gолучает значение указанного столбца как экземпляр System. Строка .

Для того, чтобы узнать тип столбца существует функция GetFieldType (Int32).

Часто в запросы необходимо передавать параметры. Для этого существует класс *Npgsql Parameter*. При создании запроса указываются ссылки на параметры. Пример использования *Npgsql Parameter* показан ниже:

*/\*  
 Этот пример вернет:  
  
 ID = 1; фамилия = Петров; имя = Иван; отчество = Викторович   
 \*/*public static void Read()

{

string sql = “Server = **127.0.0.1**; Port = **5432**; User Id = **lab\_user**; Password = **lab\_user**; Database = **lab\_db**”;

*// Установка соединения с БД*

NpgsqlConnection conn = new NpgsqlConnection(sql);

NpgsqlCommand com = new NpgsqlCommand(“SELECT \* FROM students WHERE ID = @p1”, conn);

var npgSqlParameterId

npgSqlParameterId = new NpgsqlParameter("@p1",NpgsqlTypes.NpgsqlDbType. Bigint);

npgSqlParameterId.**Value** = 1;

com.Parameters.Add(npgSqlParameterId);

conn.Open();

NpgsqlDataReader reader;

reader = com.ExecuteReader();

string[,] nums = new string[4,5];

int a = 0;

while (reader.Read())

{

try

{

nums[a, 0] = reader.GetInt64(0).ToString();

nums[a, 1] = reader.GetInt32(1).ToString();

nums[a, 2] = reader.GetString(2);

nums[a, 3] = reader.GetInt32(3).ToString();

}

catch { }

a++;

}

conn.Close();

}

}

Через *NpgsqlParameter* можно передавать и значения более сложных типов, чем примитивные, но рассмотрение таких действий не входит в курс лабораторных работ.

**6. Выполнение запросов на изменение данных**

Запросы на изменение данных можно точно также выполнять через экземпляры *NpgsqlCommand*, *Npgsql Parameter*.

Пример выполнения запроса insert показан ниже:

*/\*  
 Добавиться в БД*

*фамилия = Котов; имя = Сергей; отчество = Викторович  
  
 \*/*public static void Add(string sub)

{

string sql = “Server = **127.0.0.1**; Port = **5432**; User Id = **lab\_user**; Password = **lab\_user**; Database = **lab\_db**”;

*// Установка соединения с БД*

NpgsqlConnection con = new NpgsqlConnection(sql);

con.Open();

NpgsqlCommand com = new NpgsqlCommand(**"**INSERT INTO students(first\_name, middle\_name, second\_name) VALUES( **(**@p1**,** @p2**,** @p3**)"**

, con)

var a = new NpgsqlParameter("@p1",NpgsqlTypes. NpgsqlDbType.Varchar);

var b = new NpgsqlParameter("@p2",NpgsqlTypes. NpgsqlDbType.Varchar);

var c = new NpgsqlParameter("@p3",NpgsqlTypes. NpgsqlDbType. Varchar);

a.**Value** = **"Котов"**;

b.**Value** = **"Сергей"**;

c.**Value** = **"Викторович"**;

com.Parameters.Add(a);

com.Parameters.Add(b);

com.Parameters.Add(c);

com.ExecuteNonQuery();

con.Close();

}