**Лабораторная работа №16**

**Соединение базы данных PostgreSQL с приложением на языке C#**

**Подготовка**

Прежде чем начать работу по созданию приложения необходимо завершить некоторые подготовительные этапы.

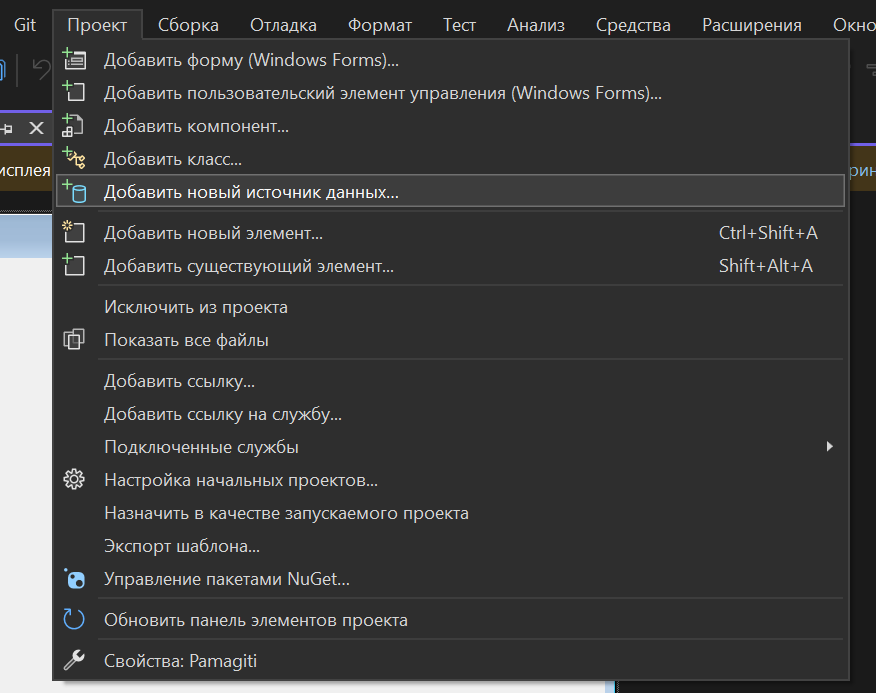
Для начала необходимо установить драйвер ODBC для используемого сервера. В нашем случае для этого необходимо запустить Stackbuilder, который находится с папке сервера по пути …\App\PgSQL\bin

После установки требуемого драйвера, можно продолжать работу.

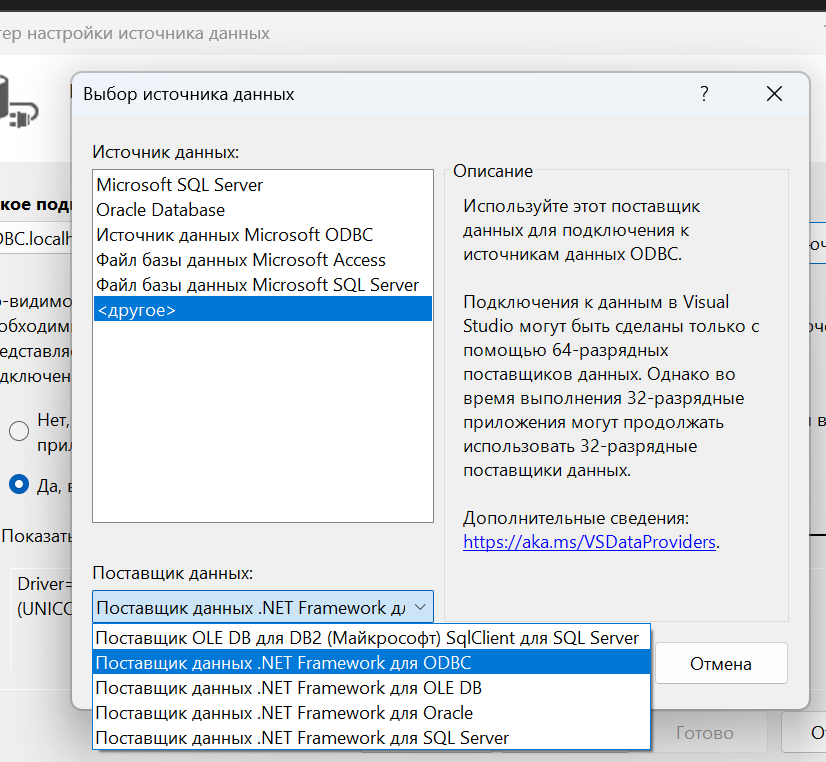
1. Создаем проект Приложение Windows Forms (.Net framework)
2. В качестве целевой платформы выбираем .Net Framework 4.8

Перед нами появится первая форма нашего приложения, на которой мы в будущем реализуем функционал авторизации.

Следующим шагом будет соединение программного продукта с базой данных. Для этого в верхней части экрана в меню проект необходимо выбрать пункт – добавить источник данных



В последующих меню выбираем: база данных-набор данных и нажимаем на кнопку создать подключение. В появившемся меню необходимо в верхней части выбрать «другое», а в нижнем выпадающем списке – поставщик данных .Net Framework для ODBC



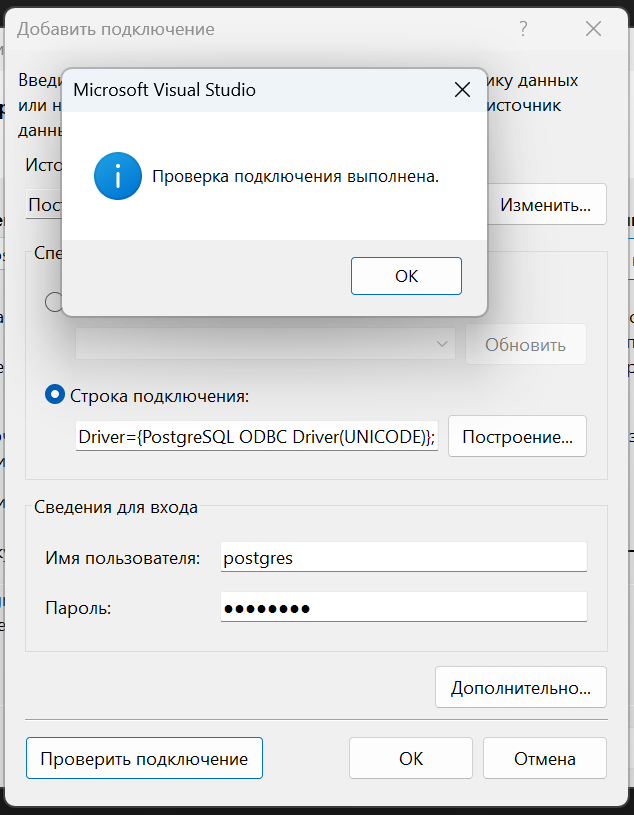
В следующем окне необходимо переключить пункт на «строка подключения» и заполнить её в следующем формате:

Driver={PostgreSQL ODBC Driver(UNICODE)};Server=<server>;Port=<port>;Database=<database>;UID=<user id>;PWD=<password>

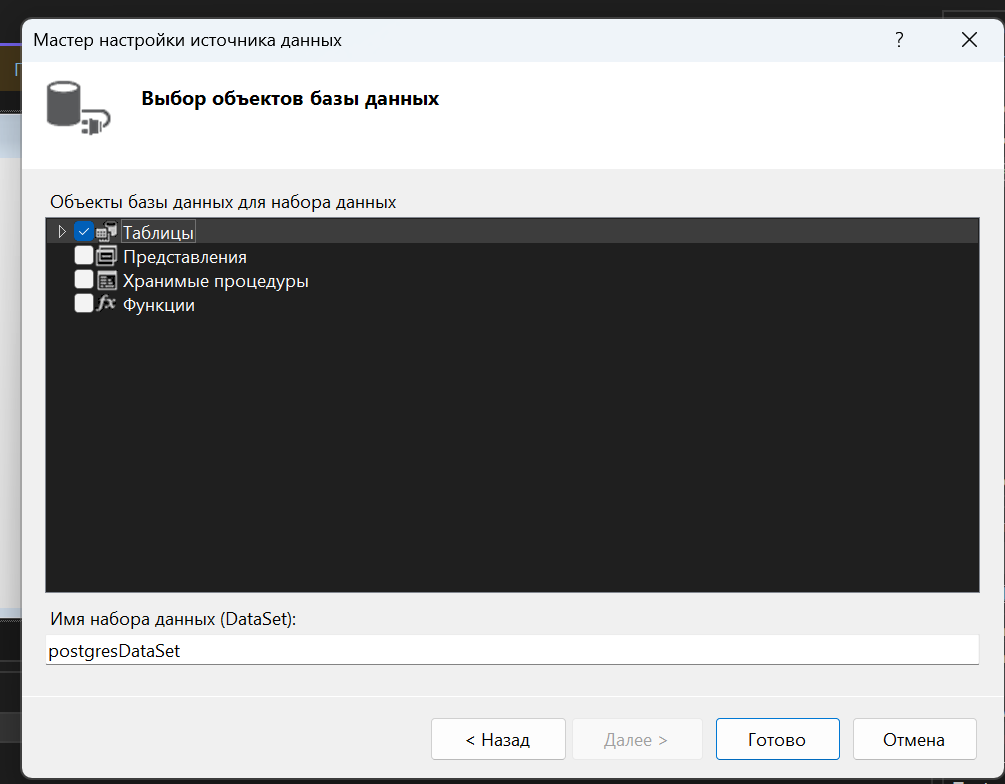
**Пример:**

Driver={PostgreSQL ODBC Driver(UNICODE)};Server=127.0.0.1;Port=5432;Database=postgres;UID=postgres;PWD=postgres

Проверить правильность введённых данных можно нажав кнопку «проверка подключения»

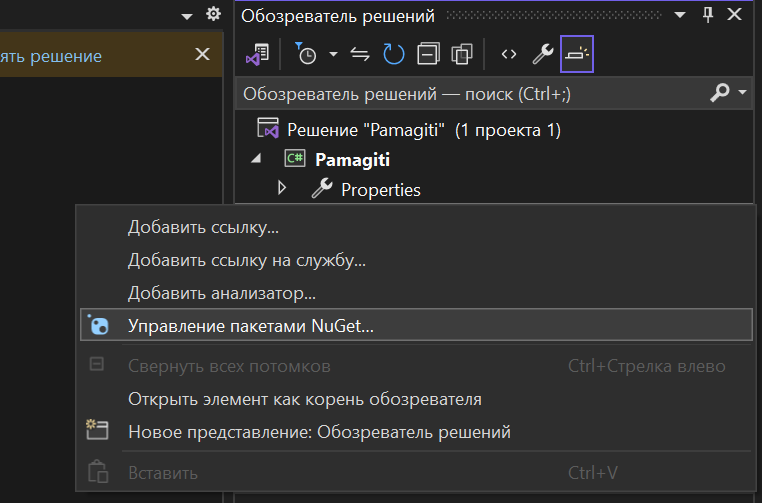


Прожав кнопки «ОК» - «Далее» - «Далее» появится выбор объектов, которые необходимо добавить в источник данных программы.

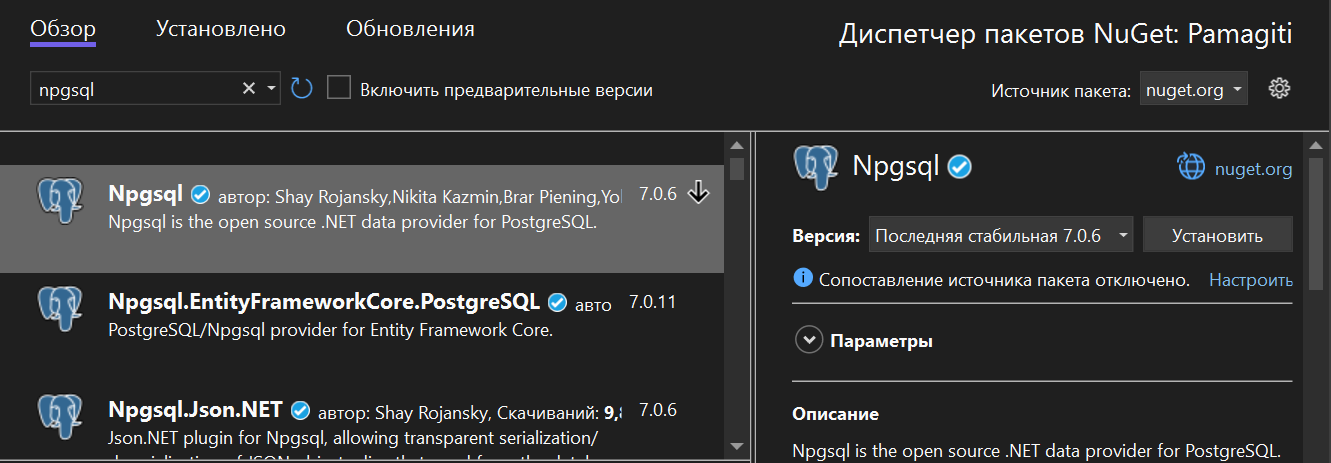


Для удобной работы с базой данных в приложении необходимо установить дополнительную библиотеку из менеджера пакетов.

Для этого в правой части окна проекта необходимо кликнуть правой кнопкой мыши по пункту «Ссылки» и выбрать «управление пакетами»



На появившемся экране в строке поиска пишем «npgsql» и устанавливаем.



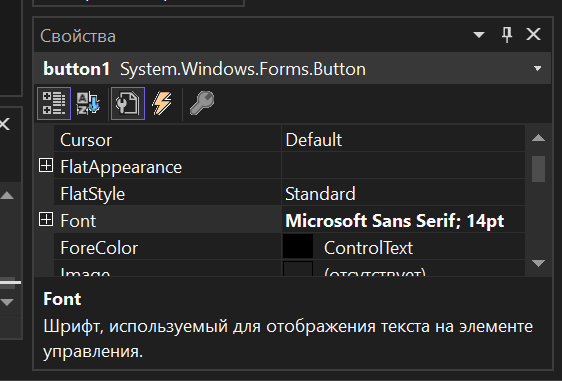
Ура! Мы готовы страдать!

**Авторизация**

Первым шагом будет создание формы авторизации. Для этого необходимо поместить на форму компоненты для ввода данных и кнопку с панели элементов.

Перенесём из панели элементов компонент button, 2 экземпляра textbox для ввода данных и 2 экземпляра label для подписей.

Шрифт, размер и текст компонентов можно изменить в окне свойств



Кликнув два раза по кнопке, будет создано событие, срабатывающее при нажатии.

Для корректной работы необходимо добавить библиотеку Npgsql к форме.

Приступаем к созданию

В данный момент мы предусматриваем, что все данные текущим и будущих пользователей базы данных будут хешированы, но пока нам необходимо реализовать данную функцию. Поэтому первый вариант авторизации будет упрощён.

Код метода авторизации, срабатывающего по нажатию кнопки представлен ниже. Если введённые данные найдены в таблице пользователей, то будет открыта форма 2, если же нет, то пользователю будет выведено сообщение об ошибке.

NpgsqlConnection conn = new NpgsqlConnection("server=localhost; port=5432; database=postgres;username=postgres; password=postgres; ");

conn.Open();

NpgsqlCommand command = new NpgsqlCommand("select id from usrs where logn=(:l) and pass=(:p);", conn);

command.Parameters.Add(new NpgsqlParameter("l", DbType.String));

command.Parameters[0].Value = Convert.ToString(textBox1.Text);

command.Parameters.Add(new NpgsqlParameter("p", DbType.String));

command.Parameters[1].Value = Convert.ToString(textBox2.Text);

int id = Convert.ToInt32(command.ExecuteScalar());

conn.Close();

if (id != 0)

{

Form2 frm2 = new Form2();

frm2.Show();

this.Hide();

}

catch

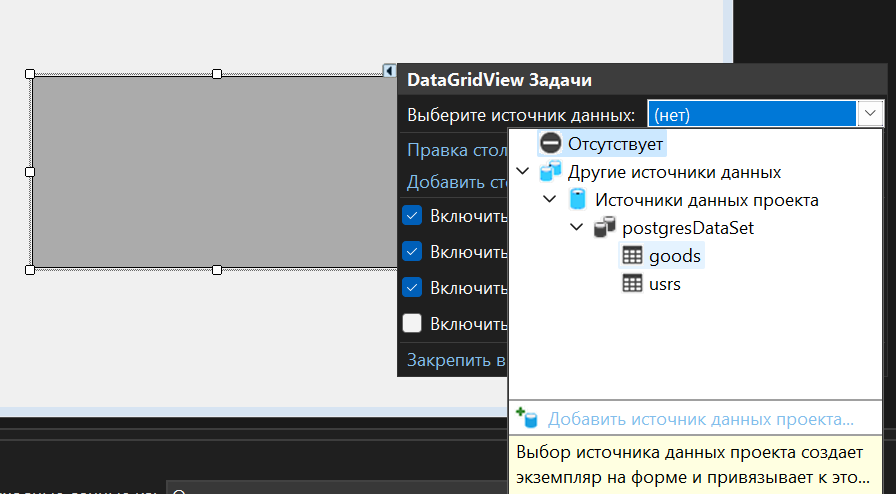
{

}

**Отображение данных**

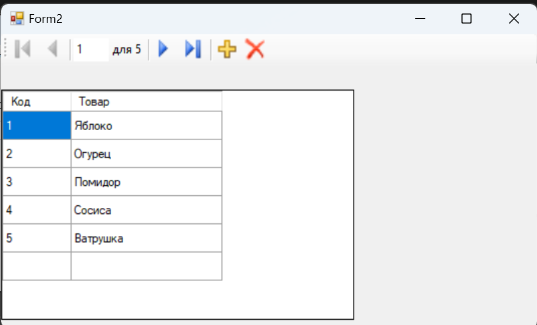
Для отображения данных используем компонент datagridview

Кликнув по компоненту и выбрав выпадающее меню, можно привязать его к выбранной таблице



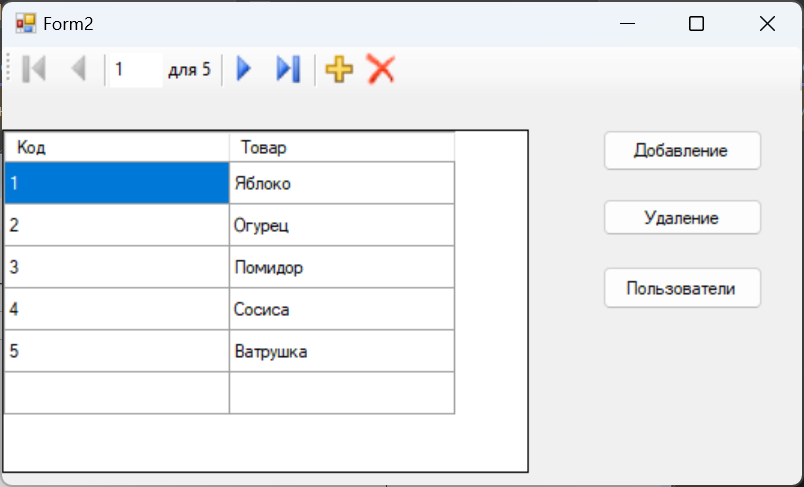
Также для упрощения работы и навигации по записям выбранной таблицы можно использовать компонент BindingNavigator. Его также нужно привязать к BindingSource через меню свойств.

В результате форма с отображением данных будет выглядеть следующим образом



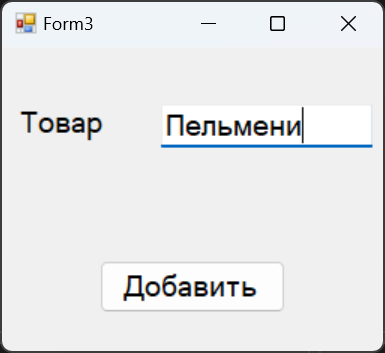
**Добавление данных**

Добавим к нашей программе кнопки для добавления, удаления данных, а также для открытия формы добавления пользователей с использованием шифрования.

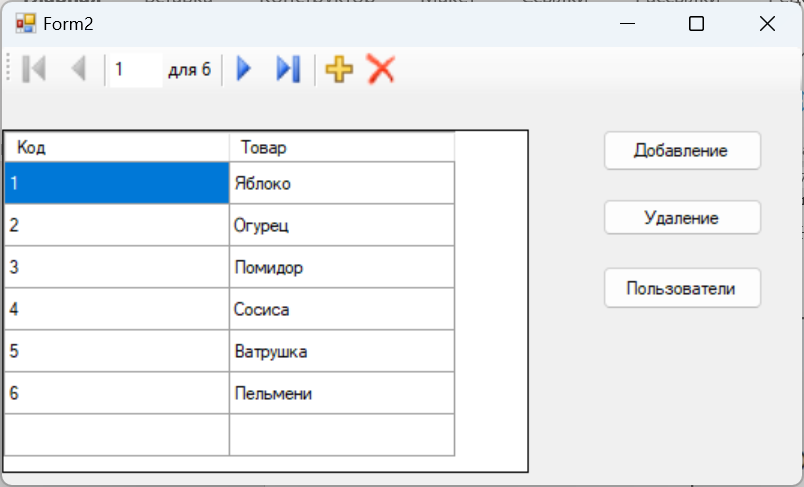


Для каждой функции создадим отдельную форму.

Перейдём на форму добавления данных. Для каждого поля таблицы, в которую будут добавляться данные необходимо добавить поле для ввода (исключениям является ключевое поле, если его значение генерируются автоматически).



В результате в форме отображения данных можно будет увидеть новую запись в текущей таблице.



Код кнопки добавления данных:

NpgsqlConnection conn = new NpgsqlConnection("server=localhost; port=5432; database=postgres;username=postgres; password=postgres; ");

conn.Open();

NpgsqlCommand command = new NpgsqlCommand("insert into goods (fhjgh) values(:tovar);", conn);

command.Parameters.Add(new NpgsqlParameter("tovar", DbType.String));

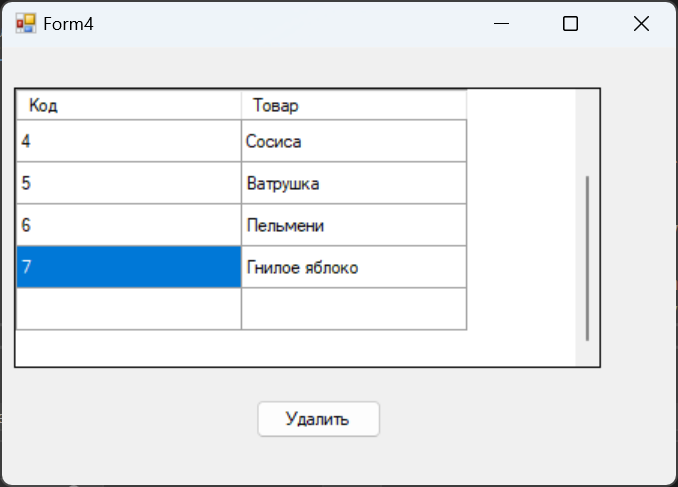
command.Parameters[0].Value = Convert.ToString((textBox1.Text));

command.ExecuteNonQuery();

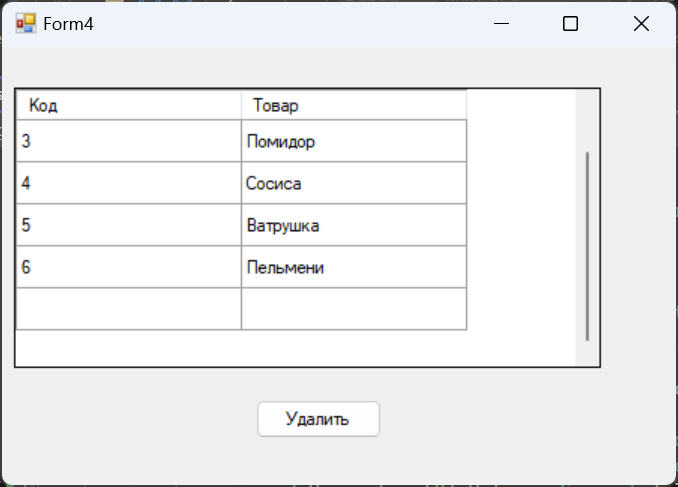
conn.Close();

**Удаление данных**

Существует множество способов удаления данных из базы данных через программу. Независимо от способа реализации, необходимо получить значение какого-либо параметра, позволяющего однозначно идентифицировать запись или записи, подлежащие удалению. В данной работе рассмотрим один из возможных способов с идентификацией требуемой записи по ключевому полю.



После клика по необходимой записи и нажатия кнопки удалить, будет произведения считывание кода данной записи. По этому коду будет произведено удаление.



Код кнопки удаления записи из таблицы:

try

{

NpgsqlConnection conn = new NpgsqlConnection("server=localhost; port=5432; database=postgres;username=postgres; password=postgres; ");

int row = dataGridView1.CurrentRow.Index;

int column = 0;

int id1 = Convert.ToInt32(dataGridView1[column, row].Value);

conn.Open();

NpgsqlCommand command = new NpgsqlCommand("delete from goods where gid=:id1", conn);

command.Parameters.Add(new NpgsqlParameter("id1", DbType.Int32));

command.Parameters[0].Value = Convert.ToInt32(id1);

command.ExecuteNonQuery();

conn.Close();

}

catch { }

**Изменение данных**

Изменение данных, так же, как и удаление можно реализовать множеством способов: например, выгружать выбранную запись в отдельную форму и там производить изменения, или изменять их прямо в форме отображения. Для примера реализуем второй способ. В форме с отображением данных к компоненту DataGridview добавим событие CellEndEdit, которое будет срабатывать по окончанию редактирования значения ячейки таблицы.

Код примера метода на изменение данных:

try

{

NpgsqlConnection conn = new NpgsqlConnection("server=localhost; port=5432; database=postgres;username=postgres; password=postgres; ");

int row = dataGridView1.CurrentRow.Index;

int column = 0;

int id1 = Convert.ToInt32(dataGridView1[column, row].Value);

conn.Open();

NpgsqlCommand command = new NpgsqlCommand("update goods set fhjgh=:tovar where gid=:id1", conn);

command.Parameters.Add(new NpgsqlParameter("tovar", DbType.String));

command.Parameters[0].Value = dataGridView1[1, row].Value.ToString();

command.Parameters.Add(new NpgsqlParameter("id1", DbType.Int32));

command.Parameters[1].Value = id1;

command.ExecuteNonQuery();

conn.Close();

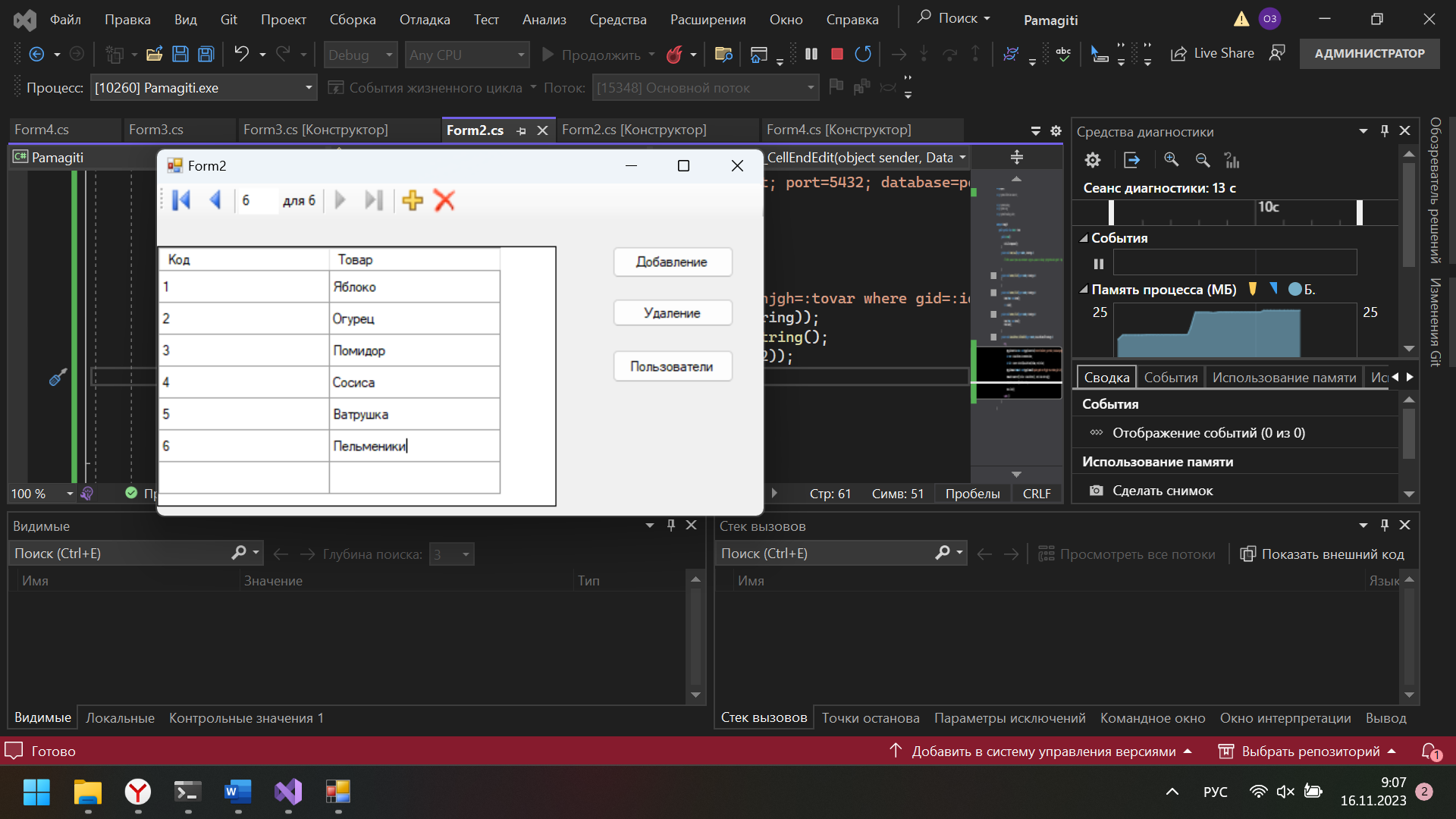
}

catch { }

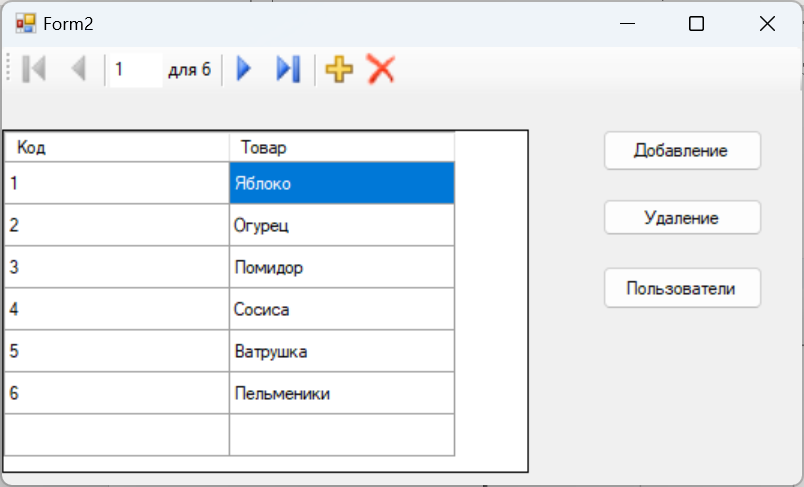
По окончанию изменения значения в ячейке таблицы вызывается событие CellEndEdit, которое посылает запрос на обновление данных выбранной строки в БД.

Результаты выполнения показаны ниже

Редактирование:



После редактирования:



**Защита данных хэшированием**

PostgreSQL обеспечивает шифрование на разных уровнях и даёт гибкость в выборе средств защиты данных в случае кражи сервера, от недобросовестных администраторов или в небезопасных сетях. Шифрование может также требоваться для защиты конфиденциальных данных, например, медицинских сведений или финансовых транзакций.

Самый первый и основной механизм защиты данных это хэширевание паролей.

PostgreSQL имеет встроенную функцию hash, которая генерирует хеш из предоставленной строки.

Альтернативным способом шифрования паролей, а также любых других сведений является использование утилиты pgcrypto. Pgcrypto — это расширение PostgreSQL, которое предоставляет функции для шифрования и дешифрования данных.

Для его активации нужно выполнить команду-запрос:

CREATE EXTENSION pgcrypto;

Эта команда загрузит доступное расширение в вашу текущую базу данных — в том случае, если его в ней еще нет. По факту будет запущен сценарий расширения, добавляющий новые SQL-объекты — функции, типы данных, операторы и методы индексации.

Для того чтобы сделать генерируемый хэш ещё надёжнее, во время операции хэширования к данным прибавляют так называемую соль.

Дело в том, что хэш-функция всегда генерирует одно и то же значения для конкретных входных данных. Из этой особенности следует несколько проблем:

Два одинаковых пароля у двух разных пользователей имеют один и тот же хэш. А лучше бы им быть разными.

Чтобы не выполнять лишние вычисления по хэшированию словаря во время брутфорса, хакеры используют так называемые радужные таблицы — предварительно хэшированные словари с часто используемыми паролями.

Для обхода данных проблем и используется соль. Соль представляет собой псевдослучайную строку, которая и обеспечивает уникальность хэша на выходе.

В PostgreSQL для этого есть специальная функция — gen\_salt(), в качестве аргумента которой передается тип криптографического алгоритма:

md5 (MD5)

des (DES)

xdes (Extended DES)

bf (Blowfish)

Для самого же шифрования используется функция crypt().

Каждый раз, когда пользователь создает новый или меняет существующий пароль, PostgreSQL должна сохранить его хэш.

Сама генерация выполняется с помощью встроенной функции crypt(). У нее есть 2 аргумента:

строка пароля

строка соли

Соответственно, вместе с функцией генерацией соли получение хэша выглядит следующим образом:

SELECT crypt('password', gen\_salt('md5'));

Кстати, если по каким-то причинам вы не хотите делать хэш уникальным, достаточно просто подставлять константное значение во второй аргумент:

SELECT crypt('password’, 'nothing');

Проверка пароля

проверка пароля выполняется той же самой функцией хэширования. Отличаются только аргументы.

Например, чтобы проверить пароль «password» на соответствие его же хэшу, выполняется команда:

SELECT crypt('password’, хэш);

Обратите внимание, здесь не указывается соль — только пароль и хэш. На выходе этой команды получается тот же самый хэш, что был создан ранее — и который был указан вторым аргументом.

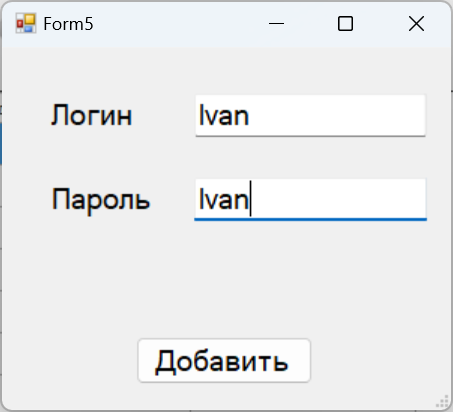
Однако, если пароль неверный, хэш будет отличаться:

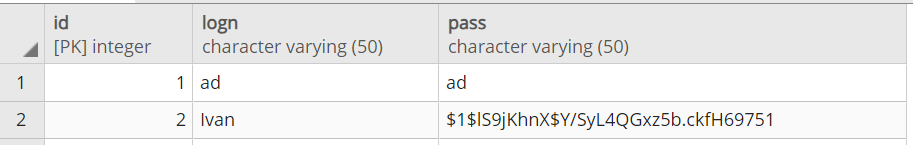
SELECT crypt('another\_password’, хэш);

Добавляем пользователя с шифрованием

Для примера создадим пользователя с шифрованием пароля по алгоритму md5.

При вводе логина и пароля Ivan - Ivan , в самой базе данных в результате шифрования будут получены следующие результаты:





Пример кода вставки пользователя с шифрованием пароля:

NpgsqlConnection conn = new NpgsqlConnection("server=localhost; port=5432; database=postgres;username=postgres; password=postgres; ");

conn.Open();

NpgsqlCommand command = new NpgsqlCommand("insert into usrs (logn,pass) values(:logn,crypt(:pass,gen\_salt('md5')));", conn);

command.Parameters.Add(new NpgsqlParameter("logn", DbType.String));

command.Parameters[0].Value = Convert.ToString((textBox1.Text));

command.Parameters.Add(new NpgsqlParameter("pass", DbType.String));

command.Parameters[1].Value = Convert.ToString((textBox2.Text));

command.ExecuteNonQuery();

conn.Close();

Авторизация

Так как мы заменили алгоритм создания пользователя в базу данных, добавив шифрование, для корректной работы приложения необходимо модифицировать метод авторизации.

При авторизации пользователя его учетные данные извлекаются из базы данных, после чего хэши паролей сопоставляются:

SELECT (password = crypt(введенный\_пароль, password))

Пример кода авторизации с учётом хэширования пароля:

try

{

NpgsqlConnection conn = new NpgsqlConnection("server=localhost; port=5432; database=postgres;username=postgres; password=postgres; ");

conn.Open();

NpgsqlCommand command = new NpgsqlCommand("select id from usrs where logn=(:l) and pass=(crypt(:p,pass));", conn);

command.Parameters.Add(new NpgsqlParameter("l", DbType.String));

command.Parameters[0].Value = Convert.ToString(textBox1.Text);

command.Parameters.Add(new NpgsqlParameter("p", DbType.String));

command.Parameters[1].Value = Convert.ToString(textBox2.Text);

int id = Convert.ToInt32(command.ExecuteScalar());

conn.Close();

if (id != 0)

{

Form2 frm2 = new Form2();

frm2.Show();

this.Hide();

}

else { throw new Exception(); }

}

catch

{

}