Оглавление

[Подключение к базе данных MS SQL Server 2](#_Toc93890231)

[Выполнение запросов 6](#_Toc93890232)

[Параметризация запроса 9](#_Toc93890233)

[Типизация значений 12](#_Toc93890234)

[Агрегатные функции в запросе 14](#_Toc93890235)

[SqlDataAdapter и DataSet 16](#_Toc93890236)

[Сохранение изменений, внесенных в таблицу через DataGridView 20](#_Toc93890237)

[Приложение 1 24](#_Toc93890238)

[Приложение 2 26](#_Toc93890239)

[Приложение 3 30](#_Toc93890240)

[Приложение 4 32](#_Toc93890241)

[Приложение 5 34](#_Toc93890242)

# Подключение к базе данных MS SQL Server

Для подключения к базе данных MS SQL Server используется класс System.Data.SqlClient.SqlConnection. Перед созданием объекта подключения к БД необходимо задать строку подключения. Строка подключения – это строка, которая содержит информацию, необходимую ODBC драйверу/OLE DB провайдеру для подключения к источнику данных.

Основные параметры строки подключения:

* Server – адрес сервера MS SQL. Для указания локальной базы данных используется ключевое слово (localdb);
* Application Name: название приложения.
* AttachDBFileName: хранит полный путь к прикрепляемой базе данных.
* Connect Timeout: период в секундах, в который происходит попытка подключения к базе данных.
* Database: хранит название базы данных.
* Trusted\_Connection: задает режим аутентификации. Если значение равно true, то для подключения к базе данных будет использоваться текущая учетная запись Windows.

Пример строки подключения:

String connectionString = "Server=(localdb)[\\mssqllocaldb](file:///\\mssqllocaldb); Database=master; Trusted\_Connection=True"

Через пакетный менеджер NuGet необходимо установить пакет System.Data.SqlClient

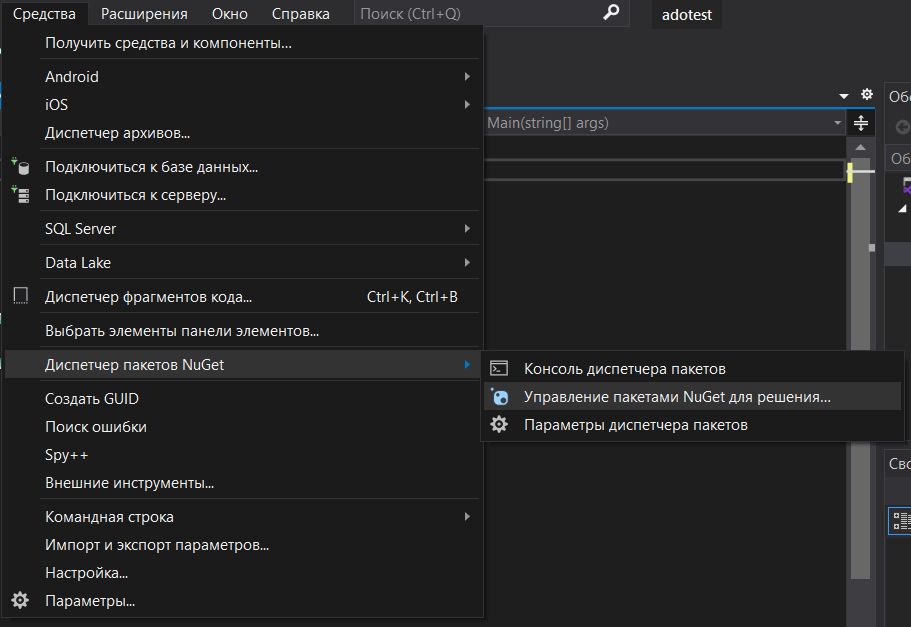


Рисунок 1 – Пакетный менеджер NuGet

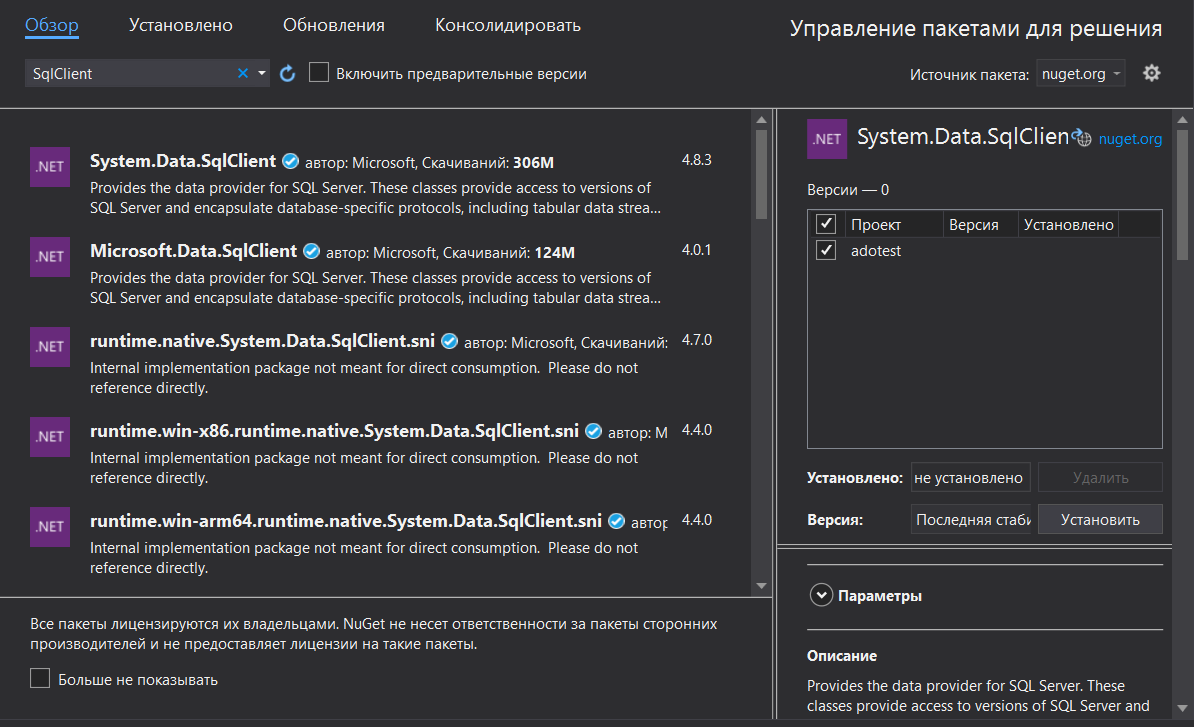


Рисунок 2 – Установка пакета System.Data.SqlClient

После чего необходимо подключить данный пакет к проекту.



Рисунок 3 – Подключение пакета System.Data.SqlClient к проекту

После чего мы можем подключиться к базе данных используя экземпляр класса SqlConnection.

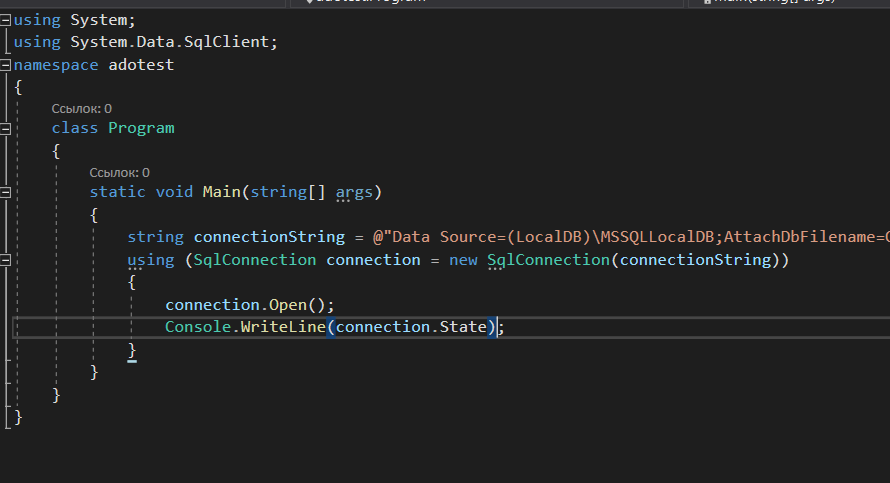


Рисунок 4 – Подключение к базе данных

В конструктор класса передаем строку подключения, после чего открываем соединение, используя метод Open(). Выведем в консоль состояние подключение.

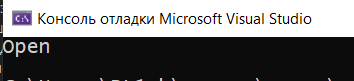


Рисунок 5 – Вывод состояния подключения

Подключение создано.

# Выполнение запросов

Для того чтобы выполнить запрос к базе данных и получить результат его выполнения используются классы SqlCommand и SqlDataReader.

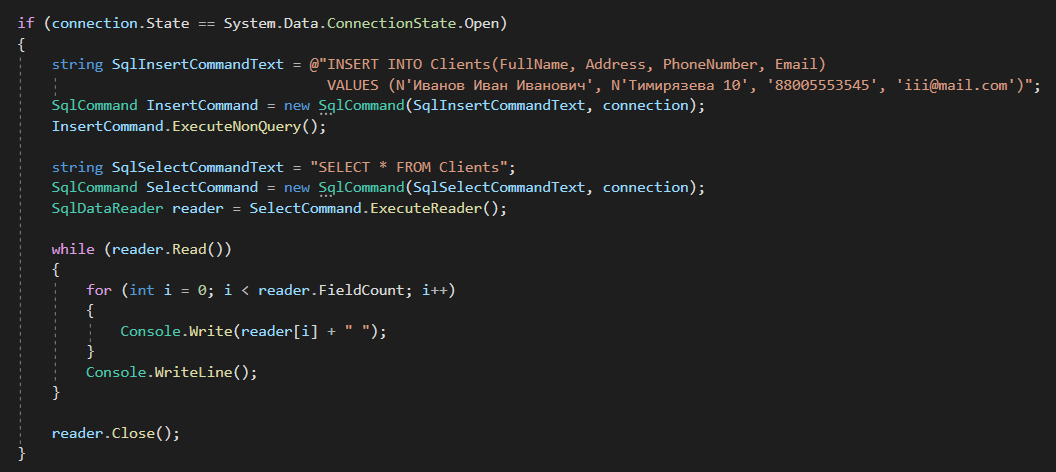


Рисунок 6 – Выполнение запросов

Проверяем состояние подключения. Если подключение открыто, то выполняем запросы. Создается объект SqlCommand, в конструктор передается текст запроса и объект подключения, после чего необходимо вызвать один из этих методов:

* ExecuteNonQuery() – выполняет запрос и возвращает количество задетых строк. Используется с запросами типа INSERT, DELETE, UPDATE, CREATE.
* ExecuteReader() – возвращает строки из таблицы. Используется для запроса типа SELECT.
* ExecuteScalar() – возвращает скалярное значение. Используется если запрос содержит функцию SELECT и одну из встроенных агрегатных функций SQL, таких как: Min, Max, Sum, Count.

В данном примере:

* SqlInsertCommandText – текст запроса с выражением INSERT INTO.
* InsertCommand – объект класса SqlCommand, который в конструкторе принимает как аргумент текст запроса (SqlInsertCommandText) и объект класса SqlConnection (connection) который отвечает за подключения к базе данных.
* InsertCommand.ExecuteNonQuery() – выполняем запрос.
* SqlSelectCommandText – текст запроса с выражением SELECT.
* SelectCommand – объект класса SqlCommand, который в конструкторе принимает как аргумент текст запроса (SqlSelectCommandText) и объект класса SqlConnection (connection) который отвечает за подключения к базе данных.
* SelectCommand.ExecuteReader() – выполняем запрос. В результат выполнения метода возвращается объект класса SqlDataReader, который содержит в себе строки – результ выполнения запроса.
* reader – непосредственно сам объект класса SqlDataReader, который мы используем для того чтобы получить все строки таблицы.

После выполнения запросов, используя объект reader мы можем считать строки. Для этого воспользуемся циклом while (reader.Read()). Метод Read считывает следующую строку и возвращает true, если в таблице еще содержаться строки. Используя индексатор мы можем обратиться к столбцу, указав его порядковый номер. Свойство reader.FieldCount возвращает количество столбцов в текущей строке. Выводим на экран содержимое таблицы.

После использования SqlDataReader, необходимо его закрыть вызвав метод Close(). У одного объекта SqlConnection может быть только один открытый SqlDataReader.



Рисунок 7 – Результат выполнения запросов

В результате выполнения программы мы видим строку, которую только что добавили.

# Параметризация запроса

Для того чтобы передать параметры в запрос, используется класс SqlParameter.

Введем данные с клавиатуры.

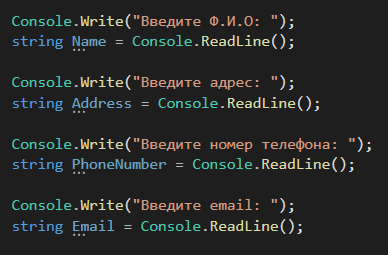


Рисунок 8 – Ввод данных с клавиатуры

После чего создадим параметры. SqlParameter в конструкторе принимает название параметра и его значение.

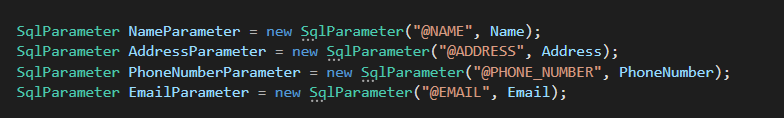


Рисунок 9 – Объявление параметров

После этого создадим запрос, добавим к нему параметры и выполним его.

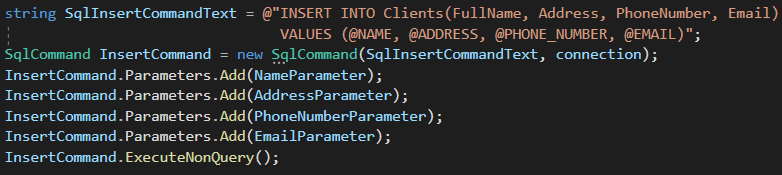


Рисунок 10 – Добавление параметров в запросе

Для того, чтобы добавить параметр к запросу, необходимо обратиться к коллекции параметров Parameters и вызвать метод Add, который как аргумент принимает объект SqlParameter.

После добавим вывод содержимого таблицы на экран.

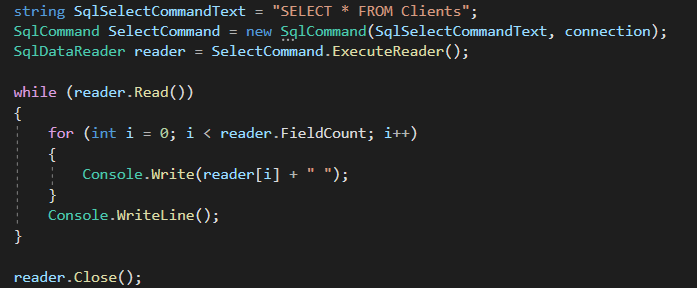


Рисунок 11 – Вывод строк таблицы на экран

В результате выполнения программы, после ввода данных с клавиатуры, мы видим, что в таблицу была добавлена еще одна запись с введенными нами данными.

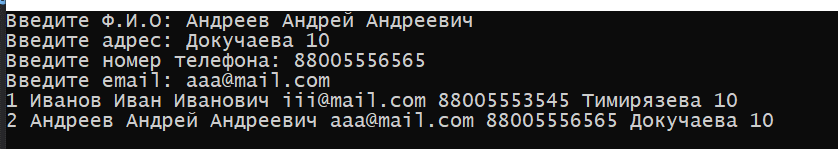


Рисунок 12 – Результат работы программы

# Типизация значений

По умолчанию, SqlDataReader возвращает тип Object. Для того чтобы вернуть значение определенного типа, предусмотрены различные методы. Например:

GetInt32() – принимает как аргумент индекс столбца в таблице, возвращает значение типа int.

GetString() - принимает как аргумент индекс столбца в таблице, возвращает значение типа string.

GetByte() - принимает как аргумент индекс столбца в таблице, возвращает значение типа byte.

GetDouble() – принимает как аргумент индекс столбца в таблице, возвращает значение типа double.

GetDateTime() – принимает как аргумент индекс столбца в таблице, возвращает значение типа DateTime.

GetDecimal() – принимает как аргумент индекс столбца в таблице, возвращает значение типа decimal

Каждому типу SQL соответствует тип в C#. Например:

|  |  |
| --- | --- |
| **SQL** | **C#** |
| Int | int |
| Money | decimal |
| datetime | DateTime |
| varchar | string |
| bit | bool |
| float | double |

Таблица 1 – Сопоставление типов SQL и типов C#

Данный фрагмент программы получает Id клиента (в таблице имеет тип int) в текущей строке и умножает его на два, выводит результат операции на экран.

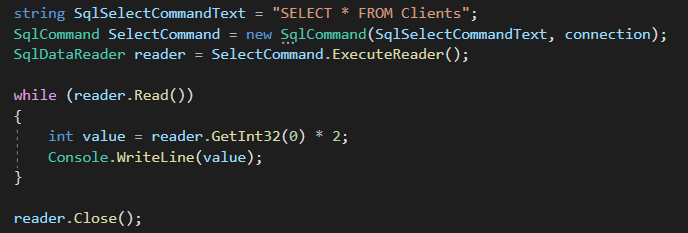


Рисунок 13 – Получение ID клиента и умножение его на 2

В результате выполнения программы мы видим, что были получены необходимые значения.

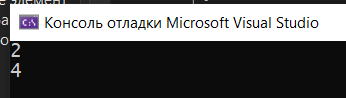


Рисунок 14 – Результат умножения Id клиентов на 2

# Агрегатные функции в запросе

Для получения результата выполнения агрегатной функции в запросе используется метод ExecuteScalar().

Метод класса SqlCommand ExecuteScalar() возвращает объекта типа object.

Выполним запрос, содержащий агрегатную функцию и получим результат её выполнения. Выведем его на экран.

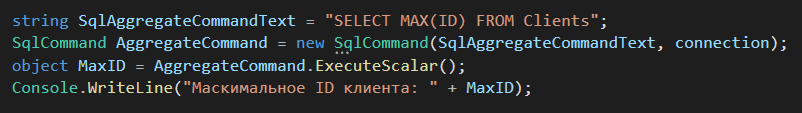


Рисунок 15 – Агрегатная функция в запросе

Для того чтобы манипулировать полученным значением как числом необходимо конвертировать его в тип int. Для этого в классе Convert содержится статический метод ToInt32. После конвертирования, мы можем произвести необходимые манипуляции со значением, например, возвести в пятую степень и вывести на экран.

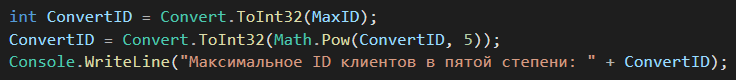


Рисунок 16 – Преобразование типов и математические манипуляции

В результате выполнения программы мы видим, что было выведено максимальное ID клиента и оно было возведено в пятую степень.

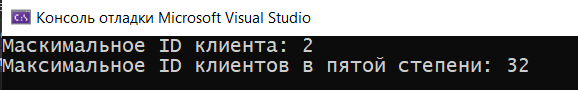


Рисунок 17 – Результат работы программы с запросом, содержащим агрегатную функцию

# SqlDataAdapter и DataSet

DataSet является типом, который содержит в себе таблицы DataTable. DataTable содержит в себе строки и столбцы (DataRow и DataColumn).

Для того чтобы загрузить данные из таблицы базы данных в DataSet используется класс SqlDataAdapter. Он принимает как аргументы конструктора текст SELECT запроса и объект подключения. Для непосредственно заполнения DataSet используется метод Fill, который как аргумент принимает объект DataSet.

Для демонстрации работы с DataSet и SqlDataAdapter создадим приложение WindowsForms. На форме расположим DataGridView.

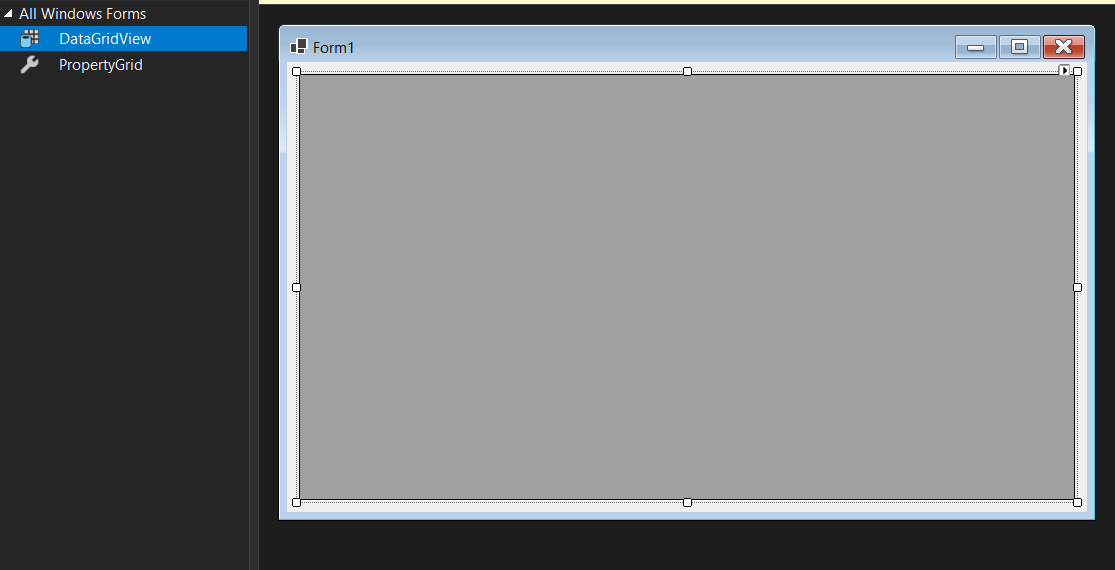


Рисунок 18 – Приложение Windows Forms. Форма с объектом DataGridView

Подключим через пакетный менеджер NuGet пакет System.Data.SqlClient. Подключим к проекту пакеты System.Data и System.Data.SqlClient.

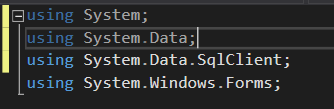


Рисунок 19 – Подключение пакетов к приложению

В классе формы создадим поля классов SqlConnection, SqlDataAdapter, DataSet.

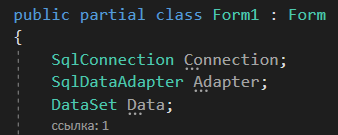


Рисунок 20 – Объявление полей класса

В конструкторе формы создаем подключение к базе данных. Создаем строковую переменную, содержащую SELECT запрос к базе данных.

В конструктор SqlDataAdapter передаем текст запроса и объект подключение.

Далее вызовем конструктор DataSet и метод SqlDataAdapter Fill, куда передадим DataSet.

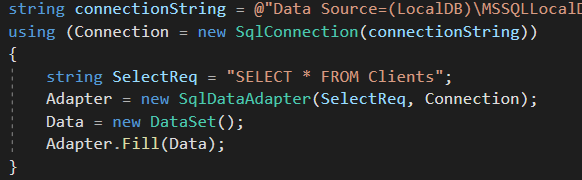


Рисунок 21 – Заполнение DataSet данными из таблицы

Для того чтобы вывести на форму данные из DataSet, свойству DataSource экземпляра DataGridView, что располагается на форме, присваиваем таблицу с индексом 0 из DataSet.

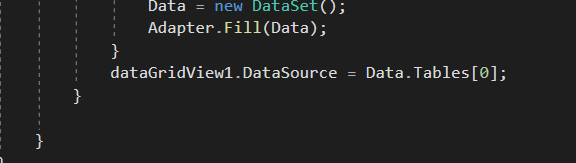


Рисунок 22 – Вывод данных на форму

В результате при запуске программы мы увидим данные из таблицы.

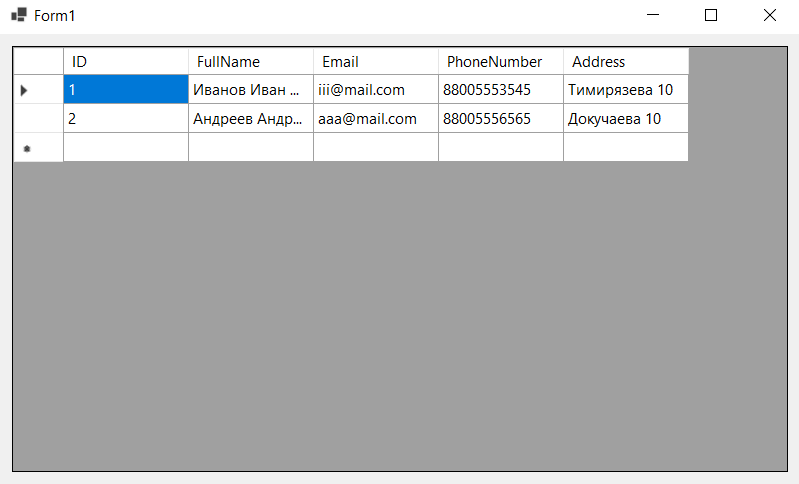


Рисунок 23 – Результат работы программы.

# Сохранение изменений, внесенных в таблицу через DataGridView

Добавим на форму кнопку “Сохранить”.

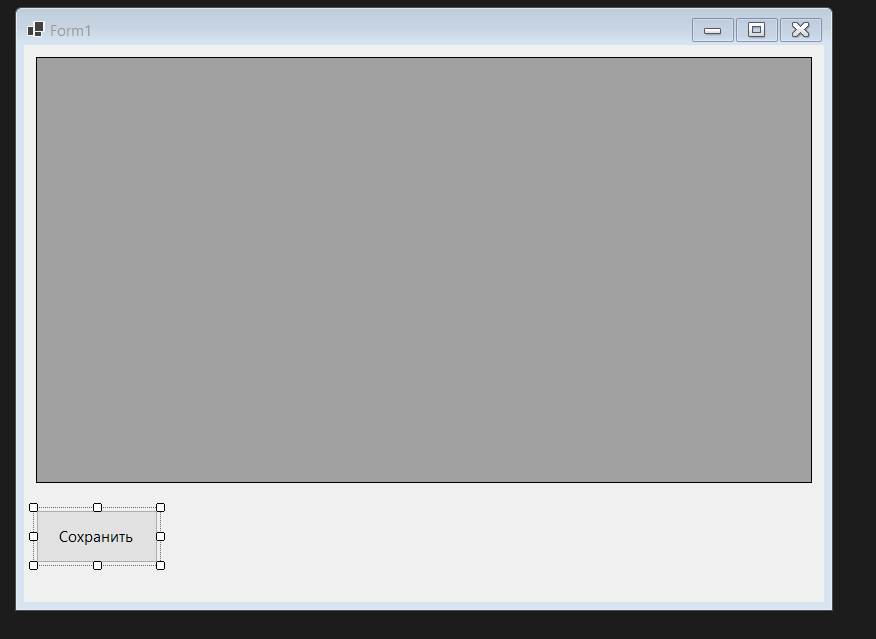


Рисунок 24 – Добавление кнопки сохранить

При двойном нажатии на нее в конструкторе формы создается метод, который вызывается при нажатии на нее во время работы программы.

Внесем изменения в конструктор. Не используем конструкцию using для экземпляра SqlConnction, так как нам необходимо подключение к базе данных во время работы с программой. Создадим объект класса SqlCommandBuilder. Данный класс позволяет автоматически сгенерировать запросы добавления, изменения и удаления данных для соответствующего экземпляра SqlDataAdapter, который передается в конструктор.

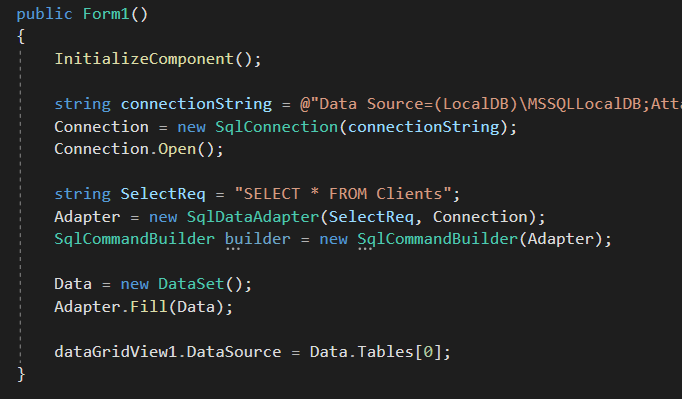


Рисунок 25 – Создание экземпляра SqlCommandBuilder

В свойствах формы создаем метод для события FormClosed.

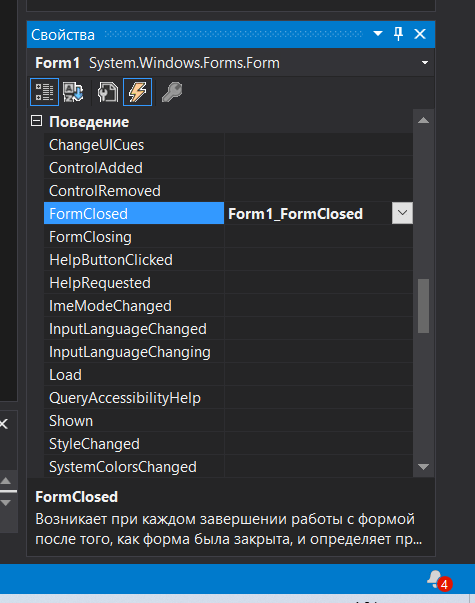


Рисунок 26 – Событие FormClosed

В коде метода закрываем соединение с базой данных.

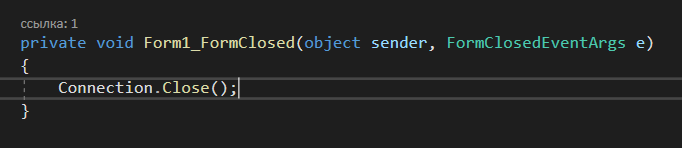


Рисунок 27 – Закрываем подключение к базе данных

В методе кнопки вызываем метод Update объекта класса SqlDataAdapter, передаем DataSet как аргумент.

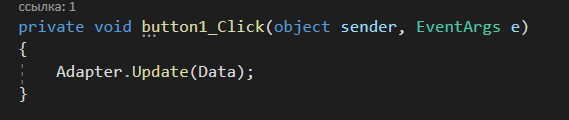


Рисунок 28 – Обновление базы данных

Запускаем программу. Вносим изменения в таблицу. Добавляем нового клиента. Нажимаем сохранить.

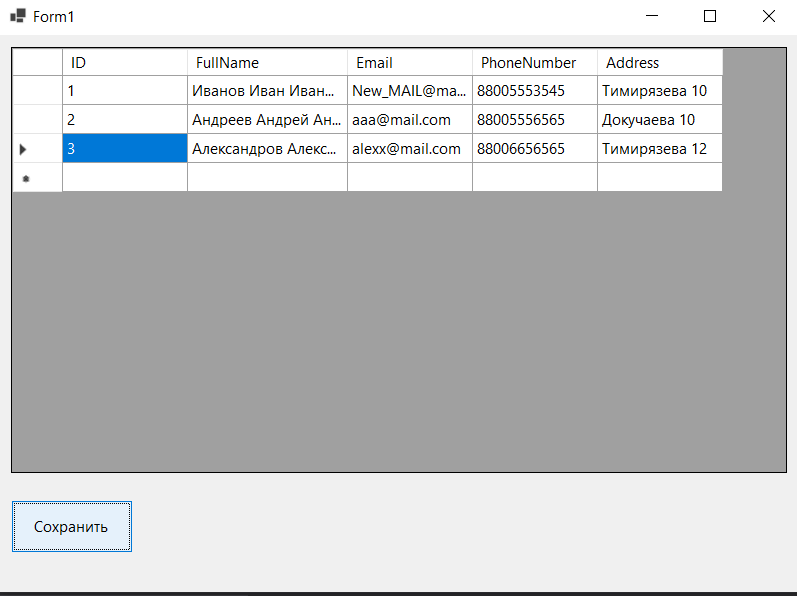


Рисунок 29 – Вносим изменения в таблицу

После перезапуска программы видим что изменения сохранились.

# Приложение 1

Пример 1

using System;

using System.Data.SqlClient;

namespace adotest

{

class Program

{

static void Main(string[] args)

{

string connectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\DL1ch\OneDrive\Documents\Lab2.mdf;Integrated Security=True;Connect Timeout=30";

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

if (connection.State == System.Data.ConnectionState.Open)

{

string SqlInsertCommandText = @"INSERT INTO Clients(FullName, Address, PhoneNumber, Email)

VALUES (N'Иванов Иван Иванович', N'Тимирязева 10', '88005553545', 'iii@mail.com')";

SqlCommand InsertCommand = new SqlCommand(SqlInsertCommandText, connection);

InsertCommand.ExecuteNonQuery();

string SqlSelectCommandText = "SELECT \* FROM Clients";

SqlCommand SelectCommand = new SqlCommand(SqlSelectCommandText, connection);

SqlDataReader reader = SelectCommand.ExecuteReader();

while (reader.Read())

{

for (int i = 0; i < reader.FieldCount; i++)

{

Console.Write(reader[i] + " ");

}

Console.WriteLine();

}

reader.Close();

}

}

}

}}

# Приложение 2

Пример 2

using System;

using System.Data.SqlClient;

namespace adotest

{

class Program

{

static void Main(string[] args)

{

string connectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\DL1ch\OneDrive\Documents\Lab2.mdf;Integrated Security=True;Connect Timeout=30";

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

if (connection.State == System.Data.ConnectionState.Open)

{

Console.Write("Введите Ф.И.О: ");

string Name = Console.ReadLine();

Console.Write("Введите адрес: ");

string Address = Console.ReadLine();

Console.Write("Введите номер телефона: ");

string PhoneNumber = Console.ReadLine();

Console.Write("Введите email: ");

string Email = Console.ReadLine();

SqlParameter NameParameter = new SqlParameter("@NAME", Name);

SqlParameter AddressParameter = new SqlParameter("@ADDRESS", Address);

SqlParameter PhoneNumberParameter = new SqlParameter("@PHONE\_NUMBER", PhoneNumber);

SqlParameter EmailParameter = new SqlParameter("@EMAIL", Email);

string SqlInsertCommandText = @"INSERT INTO Clients(FullName, Address, PhoneNumber, Email)

VALUES (@NAME, @ADDRESS, @PHONE\_NUMBER, @EMAIL)";

SqlCommand InsertCommand = new SqlCommand(SqlInsertCommandText, connection);

InsertCommand.Parameters.Add(NameParameter);

InsertCommand.Parameters.Add(AddressParameter);

InsertCommand.Parameters.Add(PhoneNumberParameter);

InsertCommand.Parameters.Add(EmailParameter);

InsertCommand.ExecuteNonQuery();

string SqlSelectCommandText = "SELECT \* FROM Clients";

SqlCommand SelectCommand = new SqlCommand(SqlSelectCommandText, connection);

SqlDataReader reader = SelectCommand.ExecuteReader();

while (reader.Read())

{

for (int i = 0; i < reader.FieldCount; i++)

{

Console.Write(reader[i] + " ");

}

Console.WriteLine();

}

reader.Close();

}

}

}

}

}

# Приложение 3

Пример 3

using System;

using System.Data.SqlClient;

namespace adotest

{

class Program

{

static void Main(string[] args)

{

string connectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\DL1ch\OneDrive\Documents\Lab2.mdf;Integrated Security=True;Connect Timeout=30";

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

if (connection.State == System.Data.ConnectionState.Open)

{

string SqlSelectCommandText = "SELECT \* FROM Clients";

SqlCommand SelectCommand = new SqlCommand(SqlSelectCommandText, connection);

SqlDataReader reader = SelectCommand.ExecuteReader();

while (reader.Read())

{

int value = reader.GetInt32(0) \* 2;

Console.WriteLine(value);

}

reader.Close();

}

}

}

}

}

# Приложение 4

Пример 4

using System;

using System.Data.SqlClient;

namespace adotest

{

class Program

{

static void Main(string[] args)

{

string connectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\DL1ch\OneDrive\Documents\Lab2.mdf;Integrated Security=True;Connect Timeout=30";

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

if (connection.State == System.Data.ConnectionState.Open)

{

string SqlAggregateCommandText = "SELECT MAX(ID) FROM Clients";

SqlCommand AggregateCommand = new SqlCommand(SqlAggregateCommandText, connection);

object MaxID = AggregateCommand.ExecuteScalar();

Console.WriteLine("Маскимальное ID клиента: " + MaxID);

int ConvertID = Convert.ToInt32(MaxID);

ConvertID = Convert.ToInt32(Math.Pow(ConvertID, 5));

Console.WriteLine("Максимальное ID клиентов в пятой степени: " + ConvertID);

}

}

}

}

}

# Приложение 5

Пример 5-6

using System;

using System.Data;

using System.Data.SqlClient;

using System.Windows.Forms;

namespace DataSetTest

{

public partial class Form1 : Form

{

SqlConnection Connection;

SqlDataAdapter Adapter;

DataSet Data;

public Form1()

{

InitializeComponent();

string connectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\DL1ch\OneDrive\Documents\Lab2.mdf;Integrated Security=True;Connect Timeout=30";

Connection = new SqlConnection(connectionString);

Connection.Open();

string SelectReq = "SELECT \* FROM Clients";

Adapter = new SqlDataAdapter(SelectReq, Connection);

SqlCommandBuilder builder = new SqlCommandBuilder(Adapter);

Data = new DataSet();

Adapter.Fill(Data);

dataGridView1.DataSource = Data.Tables[0];

}

private void button1\_Click(object sender, EventArgs e)

{

Adapter.Update(Data);

}

private void Form1\_FormClosed(object sender, FormClosedEventArgs e)

{

Connection.Close();

}

}

}