Работа с базами данных. ADO.NET

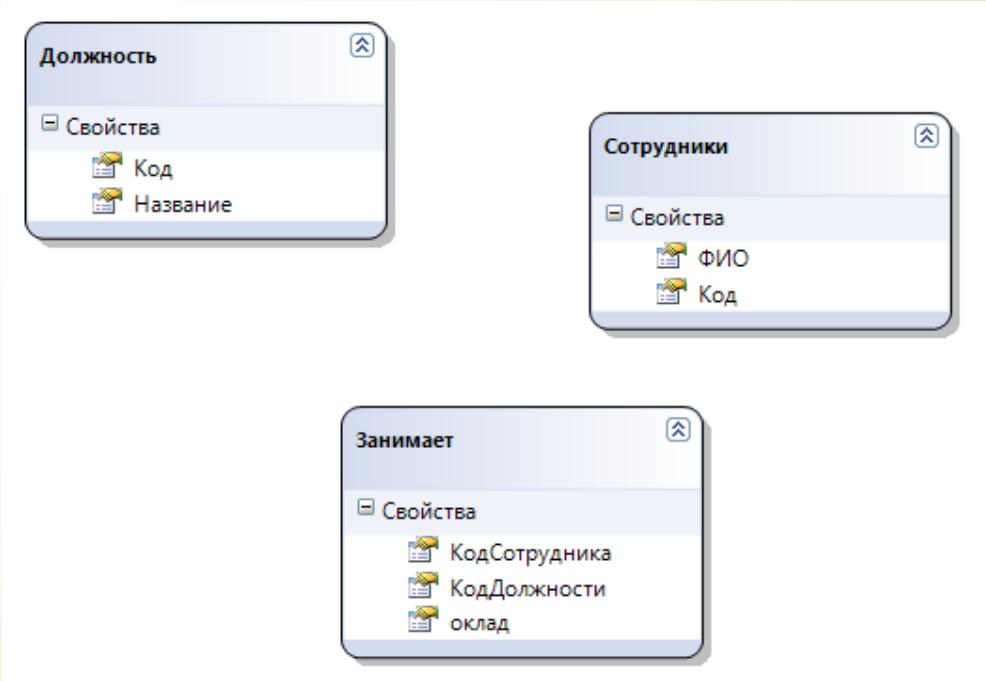
Доцент Евдошенко О.И.

Создание базы данных SQL Server

1. Добавить в проект новый элемент «База данных, основанная на службах»
2. Используя SQL – запросы создать таблицы и заполнить их данными
3. Добавить в проект новый элемент «Классы LINQ to SQL»
4. Перетащить таблицы на реляционный конструктор объектов

Реляционный конструктор объектов создает классы и применяет специфические для LINQ to SQL атрибуты, чтобы иметь функциональные возможности LINQ to SQL (возможности передачи данных и редактирования).

LINQ to SQL представляет технологию доступа и управления реляционными данными. Данная технология позволяет составлять запросу к БД в удобной форме с помощью операторов LINQ, которые затем трансформируются в sql-выражения. Ключевыми объектами здесь являются сущности, которые хранятся в базе данных, контекст данных и запрос LINQ.



Создание LINQ запросов для выборки данных

```
DataClasses1DataContext db = new DataClasses1DataContext();

        var spisok = from table1 in db.Сотрудники
    join table2 in db.Занимает on table1.Код equals table2.КодСотрудника
    join table3 in db.Должность on table2.КодДолжности equals table3.Код
                    where table3.Название == "повар"
                    select new { table1.ФИО };

    foreach (var dol in spisok)
        textBox1.Text += dol.ФИО;
```

Создание LINQ запросов для выборки данных

```
DataClasses1DataContext db = new DataClasses1DataContext();
```

```
 var spisok = from table1 in db.Сотрудники
join table2 in db.Занимает on table1.Код equals table2.КодСотрудника
join table3 in db.Должность on table2.КодДолжности equals table3.Код
                where table1.ФИО == "Иванов"
                select new { table3.Название };
```

```
 foreach (var dol in spisok)
     textBox1.Text += dol.Название;
```

Создание LINQ запросов для выборки данных

```
DataClasses1DataContext db = new DataClasses1DataContext();


          var spisok = from table1 in db.Сотрудники
        join table2 in db.Занимает on table1.Код equals table2.КодСотрудника
        join table3 in db.Должность on table2.КодДолжности equals table3.Код
                    where table2.оклад ==
                   (
                       (from table4 in db.Занимает
                        select table4.оклад).Min()
                    )

                    select new { table3.Название };


          foreach (var dol in spisok)
              textBox1.Text += dol.Название;
```

Создание LINQ запросов для выборки данных

```
DataClasses1DataContext db = new DataClasses1DataContext();


var spisok = from table1 in db.Сотрудники

 join table2 in db.Занимает on table1.Код equals table2.КодСотрудника
 join table3 in db.Должность on table2.КодДолжности equals table3.Код

              group table1 by table3.Название into g
              select g;


   foreach (var dol in spisok)
              {
                  textBox1.Text += dol.Key.ToString();
                  foreach (var fio in dol)
                      textBox1.Text += fio.ФИО;
              }
```

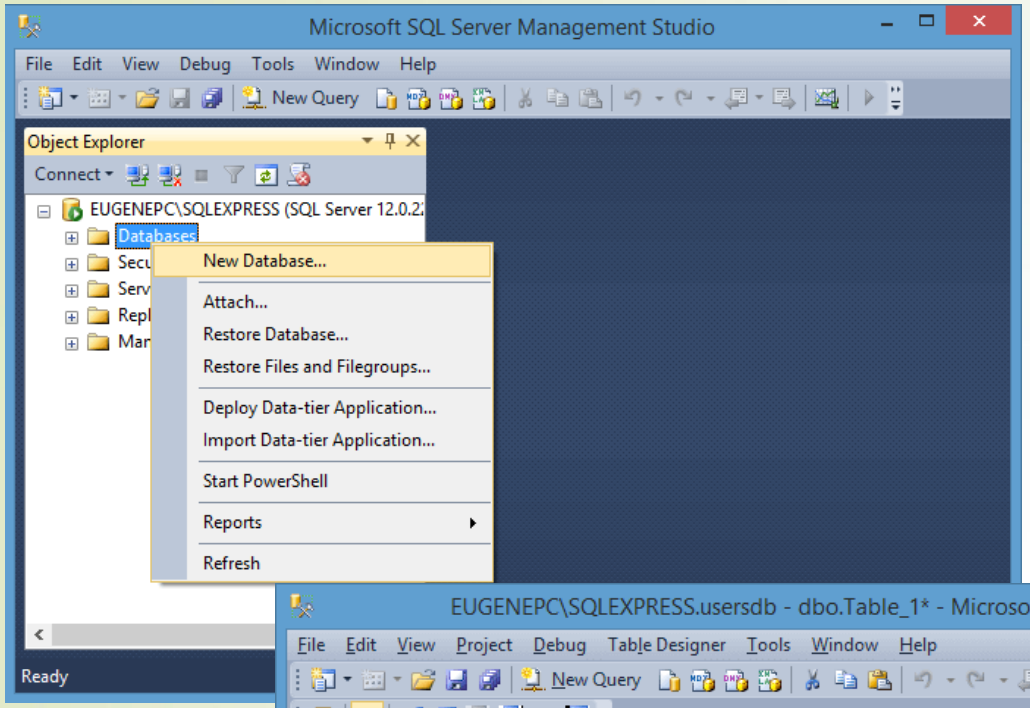
Создание LINQ запросов для выборки данных

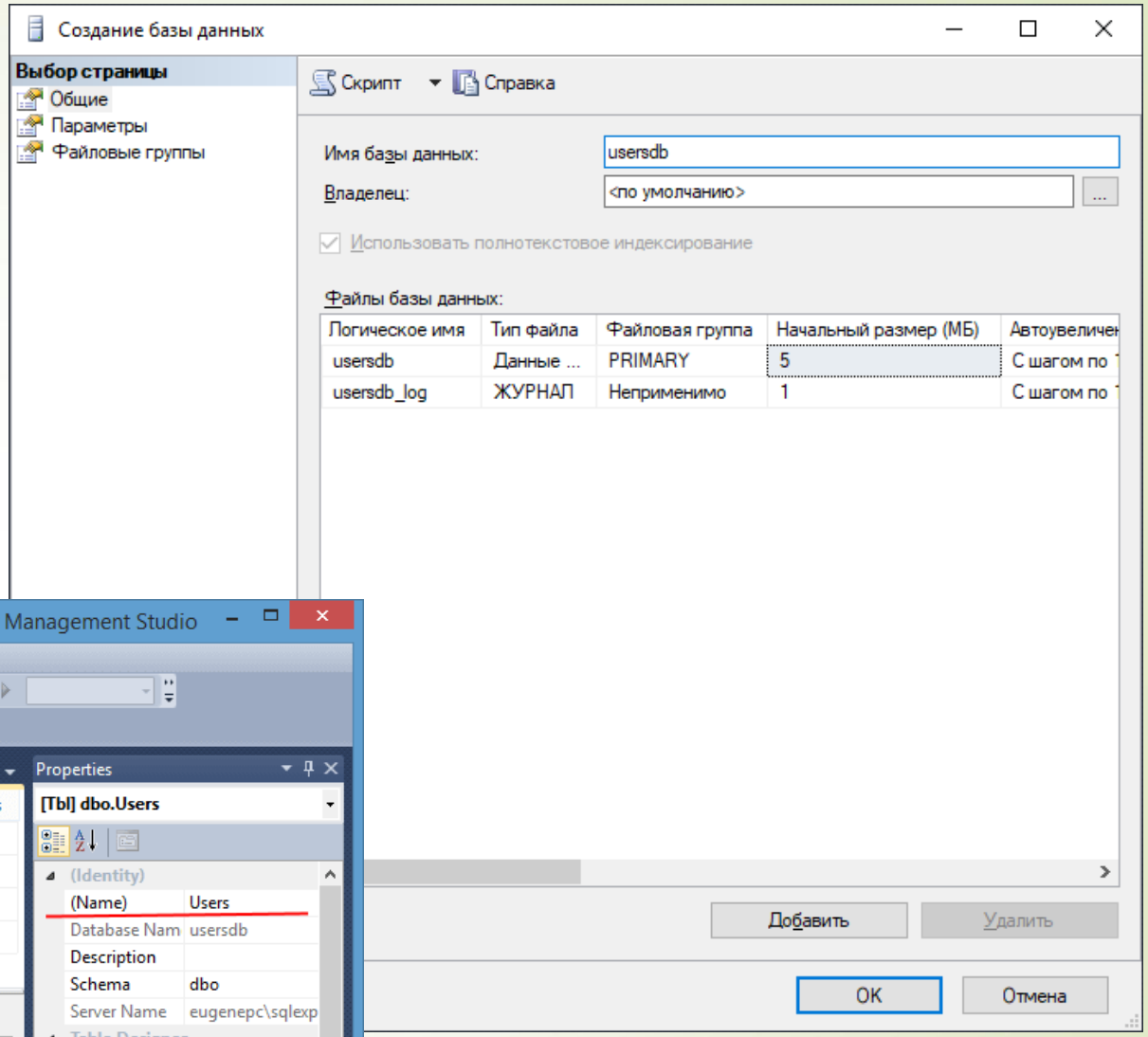
```
var spisok = from table1 in db.Сотрудники
      join table2 in db.Занимает on table1.Код equals table2.КодСотрудника
      join table3 in db.Должность on table2.КодДолжности equals table3.Код
             where table2.оклад >
                    (
                        (from table4 in db.Занимает
                         select table4.оклад).Average()
                     )

             select new { table1,table3 };


            foreach (var dol in spisok)
                textBox1.Text += dol.table1.ФИО+"-"+dol.table3.Название;
```

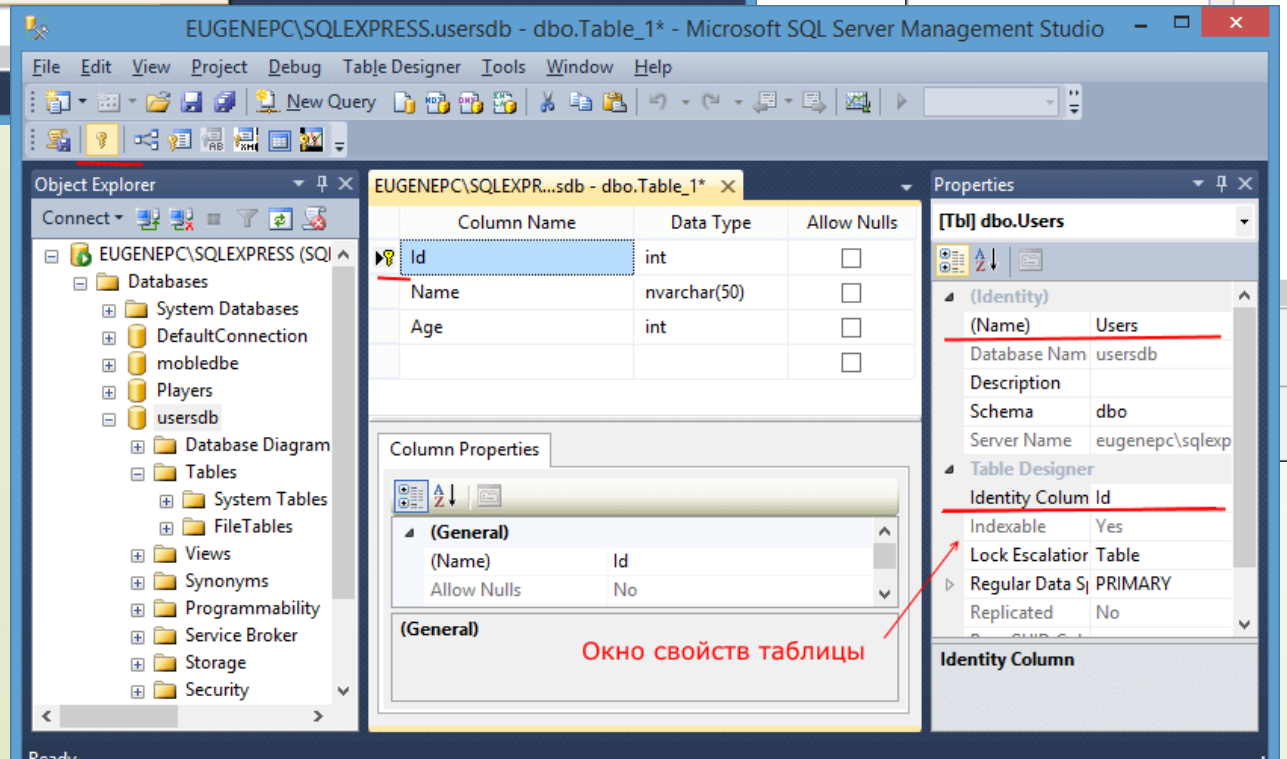
Создание LINQ запросов для выборки данных

```
var spisok = from table1 in db.Сотрудники
        join table2 in db.Занимает on table1.Код equals table2.КодСотрудника
        join table3 in db.Должность on table2.КодДолжности equals table3.Код
             where table2.оклад == (
                         (from table4 in db.Занимает
                          where table4.КодДолжности==table3.Код
                          select table4.оклад).Max()
                         )

             select new { table1.ФИО, table3.Название };


             dataGridView.DataSource = spisok;
```

Создание базы данных







Строка подключения

Первым делом нам надо определить строку подключения, предоставляющая информацию о базе данных и сервере, к которым предстоит установить подключение:

```
class Program
{
    static void Main(string[] args)
    {
        string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=usersdb;Integrated Security=True";
    }
}
```

```
string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=usersdb;User Id = sa; Password = 1234567fd";";
```

Строка подключения представляет набор параметров в виде пар **ключ=значение**.

В данном случае для подключения к ранее созданной базе данных определяем строку подключения из трех параметров:

Data Source: указывает на название сервера. По умолчанию это ".\SQLEXPRESS". Поскольку в строке используется слеш, то в начале строки ставится символ @. Если имя сервера базы данных отличается, то соответственно его и надо использовать.

Initial Catalog: указывает на название базы данных на сервере.

Integrated Security: устанавливает проверку подлинности.

Строка подключения

Жесткое кодирование строки подключения (то есть ее определение в коде приложения), как правило, редко используется. Гораздо более гибкий путь представляет определение ее в специальных конфигурационных файлах приложения. В проектах десктопных приложений это файл **App.config**, а в веб-приложениях это в основном файл **Web.config**.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
    </startup>
  <connectionStrings>
   <add name="DefaultConnection" connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=usersdb;Integrated Security=True"
       providerName="System.Data.SqlClient"/>
  </connectionStrings>
</configuration>
```

Каждая строка подключения имеет название, определяемое с помощью атрибута **name**. В данном случае строка подключения называется **"DefaultConnection"**. Название может быть произвольное.

Атрибут **connectionString** собственно хранит строку подключения, то есть весь тот текст, который мы выше определяли в методе **Main**. И третий атрибут **providerName** задает пространство имен провайдера данных. Так как мы будем подключаться к базе данных **MS SQL Server**, то соответственно мы будем использовать провайдер для SQL Server, функциональность которого заключена в пространстве имен **System.Data.SqlClient**.

Создание подключения

```
string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=new_base;Integrated Security=True";
```

```
// Создание подключения
SqlConnection connection = new SqlConnection(connectionString);
try
{
    // Открываем подключение
    connection.Open();
   MessageBox.Show("Подключение открыто");
}
catch (SqlException ex)
{
    MessageBox.Show(ex.Message);
}
finally
{
    // закрываем подключение
    connection.Close();
    MessageBox.Show("Подключение закрыто...");
}
```

В качестве альтернативного метода можно использовать конструкцию using, которая автоматически закрывает подключение:

```
using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();
            MessageBox.Show("Подключение открыто");
        }
```

```
string connectionString =
ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString;

using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    MessageBox.Show("Подключение открыто");
}
```

Выполнение команд и SqlCommand

Для выполнения команды потребуется sql-выражение и объект подключения:

```
string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=new_base;Integrated Security=True";
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();
            SqlCommand command = new SqlCommand();
            command.CommandText = "SELECT * FROM Users";
            command.Connection = connection;
            command.ExecuteReader();
        }
```

Чтобы выполнить команду, необходимо применить один из методов **SqlCommand**:

• **ExecuteNonQuery**: просто выполняет sql-выражение и возвращает количество измененных записей. Подходит для sql-выражений INSERT, UPDATE, DELETE.

• **ExecuteReader**: выполняет sql-выражение и возвращает строки из таблицы. Подходит для sql-выражения SELECT.

• **ExecuteScalar**: выполняет sql-выражение и возвращает одно скалярное значение, например, число. Подходит для sql-выражения SELECT в паре с одной из встроенных функций SQL, как например, Min, Max, Sum, Count.

Выполнение команд и SqlCommand

Добавление объектов

```
string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=new_base;Integrated Security=True";
        string sqlExpression = "INSERT INTO Users (Name, Age) VALUES ('Tom', 18)";

        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();
            SqlCommand command = new SqlCommand(sqlExpression, connection);
            int number = command.ExecuteNonQuery();
            MessageBox.Show("Добавлено объектов: "+number);
        }
```

Обновление объектов

```
string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=new_base;Integrated Security=True";

        string sqlExpression = "UPDATE Users SET Age=20 WHERE Name='Tom'";
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();
            SqlCommand command = new SqlCommand(sqlExpression, connection);
            int number = command.ExecuteNonQuery();
            MessageBox.Show("Обновлено объектов: "+number);
        }
```

Чтение результатов запроса и SqlDataReader

```
string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=new_base;Integrated Security=True";

    string sqlExpression = "SELECT * FROM Users";
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        connection.Open();
        SqlCommand command = new SqlCommand(sqlExpression, connection);
        SqlDataReader reader = command.ExecuteReader();

        if (reader.HasRows) // если есть данные
        {
            // выводим названия столбцов
            Console.WriteLine("{0}\t{1}\t{2}", reader.GetName(0), reader.GetName(1), reader.GetName(2));


            while (reader.Read()) // построчно считываем данные
            {
                object id = reader.GetValue(0);
                object name = reader.GetValue(1);
                object age = reader.GetValue(2);

                Console.WriteLine("{0} \t{1} \t{2}", id, name, age);
                MessageBox.Show(name.ToString());
            }
        }

    }
```

После завершения работы с **SqlDataReader** надо его закрыть методом **Close()**. И пока один **SqlDataReader** не закрыт, другой объект **SqlDataReader** для одного и того же подключения использовать нельзя.

Чтение результатов запроса и SqlDataReader

```
while (reader.Read())
{
    object id = reader["id"];
    object name = reader["name"];
    object age = reader["age"];
    Console.WriteLine("{0} \t{1} \t{2}", id, name, age);
}
```

Вывод данных в DataGridView

```
DataTable table = new DataTable();
string sqlExpression = "SELECT * FROM Users";
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    SqlCommand command = new SqlCommand(sqlExpression, connection);
    SqlDataReader reader = command.ExecuteReader();

    if (reader.HasRows) // если есть данные
    {

        table.Load(reader);

    }

    dataGridView1.DataSource = table;
}
```

Типизация результатов SqlDataReader

```
int id = reader.GetInt32(0);
string name = reader.GetString(1);
int age = reader.GetInt32(2);
```

Тип sql	Тип .NET	Метод
bigint	Int64	GetInt64
char	String и Char[]	GetString и GetChars
datetime	DateTime	GetDateTime
decimal	Decimal	GetDecimal
float	Double	GetDouble
image и long varbinary	Byte[]	GetBytes и GetStream
int	Int32	GetInt32
money	Decimal	GetDecimal
nchar	String и Char[]	GetString и GetChars
ntext	String и Char[]	GetString и GetChars
numeric	Decimal	GetDecimal

Тип sql	Тип .NET	Метод
nvarchar	String и Char[]	GetString и GetChars
real	Single (float)	GetFloat
smalldatetime	DateTime	GetDateTime
smallint	Intl6	GetIntl6
long varchar	String и Char[]	GetString и GetChars
timestamp	Byte[]	GetBytes
varchar	String и Char[]	GetString и GetChars

Получение скалярных значений

```
string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=new_base;Integrated Security=True";
        string sqlExpression = "SELECT COUNT(*) FROM Users";
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();
            SqlCommand command = new SqlCommand(sqlExpression, connection);
            object count = command.ExecuteScalar();

            command.CommandText = "SELECT MIN(Age) FROM Users";
            object minAge = command.ExecuteScalar();

        }
```

Параметризация запросов

```
string connectionString = @"Data Source=.\SQLEXPRESS;Initial Catalog=new_base;Integrated Security=True";
        int age = 23;
        string name = "Петр";
        string sqlExpression = "INSERT INTO Users (Name, Age) VALUES (@name, @age)";
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();
            SqlCommand command = new SqlCommand(sqlExpression, connection);
            // создаем параметр для имени
            SqlParameter nameParam = new SqlParameter("@name", name);
            // добавляем параметр к команде
            command.Parameters.Add(nameParam);
            // создаем параметр для возраста
            SqlParameter ageParam = new SqlParameter("@age", age);
            // добавляем параметр к команде
            command.Parameters.Add(ageParam);
            int number = command.ExecuteNonQuery();
        }
```

Выходные параметры запросов

Параметры, которые используются в командах, могут быть нескольких типов. Тип параметра задается с помощью свойства **Direction** объекта **SqlParameter**. Данное свойство принимает одно из значений перечисления **ParameterDirection**:

Input: параметр является входным, то есть предназначен для передачи значений в sql-выражение запроса. Это значение по умолчанию для всех параметров

InputOutput: параметр может быть как входным, так и выходным.

Output: параметр является выходным, то есть используется для возвращения запросом каких-либо значений

ReturnValue: параметр представляет результат выполнения выражения или хранимой процедуры

```
int age = 23;
string name = "Игорь";
string sqlExpression = "INSERT INTO Users (Name, Age) VALUES (@name, @age);SET @id=SCOPE_IDENTITY()";
```

SCOPE_IDENTITY()	Возвращает последнее значение идентификатора, вставленное в столбец идентификаторов в той же области. Областью является модуль, что подразумевает хранимую процедуру, триггер, функцию или пакет.

Выходные параметры запросов

```
using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();
            SqlCommand command = new SqlCommand(sqlExpression, connection);
            // создаем параметр для имени
            SqlParameter nameParam = new SqlParameter("@name", name);
            // добавляем параметр к команде
            command.Parameters.Add(nameParam);
            // создаем параметр для возраста
            SqlParameter ageParam = new SqlParameter("@age", age);
            // добавляем параметр к команде
            command.Parameters.Add(ageParam);
            // параметр для id
            SqlParameter idParam = new SqlParameter
            {
                ParameterName = "@id",
                SqlDbType = SqlDbType.Int,
                Direction = ParameterDirection.Output // параметр выходной
            };
            command.Parameters.Add(idParam);

            command.ExecuteNonQuery();

            // получим значения выходного параметра
            MessageBox.Show("Id нового объекта: "+idParam.Value);
        }
```

Работа с хранимыми процедурами

```
CREATE PROCEDURE [dbo].[sp_InsertUser]
    @name nvarchar(50),
    @age int
AS
    INSERT INTO Users (Name, Age)
    VALUES (@name, @age)

    SELECT SCOPE_IDENTITY()
GO
```

Выражение **SCOPE_IDENTITY()** возвращает id добавленной записи, поэтому на выходе из процедуры мы получим id новой записи. И завершается процедура ключевым словом GO.

```
CREATE PROCEDURE [dbo].[sp_GetUsers]
AS
    SELECT * FROM Users
GO
```

Работа с хранимыми процедурами

```
string sqlExpression = "sp_InsertUser";
        string name = "Люба";
        int age = 12;
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();
            SqlCommand command = new SqlCommand(sqlExpression, connection);
            // указываем, что команда представляет хранимую процедуру
            command.CommandType = System.Data.CommandType.StoredProcedure;
            // параметр для ввода имени
            SqlParameter nameParam = new SqlParameter
            {
                ParameterName = "@name",
                Value = name
            };
            // добавляем параметр
            command.Parameters.Add(nameParam);
            // параметр для ввода возраста
            SqlParameter ageParam = new SqlParameter
            {
                ParameterName = "@age",
                Value = age
            };
            command.Parameters.Add(ageParam);

            var result = command.ExecuteScalar();
            // если не надо возвращать id
            //var result = command.ExecuteNonQuery();
        }
```

```
string sqlExpression = "sp_GetUsers";

        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();
            SqlCommand command = new SqlCommand(sqlExpression, connection);
            command.CommandType = System.Data.CommandType.StoredProcedure;
            var reader = command.ExecuteReader();
            if (reader.HasRows)
            {
                while (reader.Read())
                {
                }
            }
            reader.Close();
        }
```

Выходные параметры хранимых процедур

```
CREATE PROCEDURE [dbo].[sp_GetAgeRange]
    @name nvarchar(50),
    @minAge int out,
    @maxAge int out
AS
    SELECT @minAge = MIN(Age), @maxAge = MAX(Age) FROM Users WHERE Name LIKE '%' + @name + '%'
GO
```

Параметры **@minAge** и **@maxAge** являются выходными благодаря указанию ключевого слова **out** в их определении. Через них собственно и будем получать минимальный и максимальный возраст.

```
string sqlExpression = "sp_GetAgeRange";

SqlCommand command = new SqlCommand(sqlExpression, connection);
command.CommandType = CommandType.StoredProcedure;

SqlParameter nameParam = new SqlParameter
{
    ParameterName = "@name",
    Value = name
};
command.Parameters.Add(nameParam);
```

Выходные параметры хранимых процедур

```
SqlParameter minAgeParam = new SqlParameter
        {
           ParameterName = "@minAge",
           SqlDbType = SqlDbType.Int // тип параметра
        };
        // указываем, что параметр будет выходным
        minAgeParam.Direction = ParameterDirection.Output;
        command.Parameters.Add(minAgeParam);

        // определяем второй выходной параметр
SqlParameter maxAgeParam = new SqlParameter
        {
           ParameterName = "@maxAge",
           SqlDbType = SqlDbType.Int
        };
        maxAgeParam.Direction = ParameterDirection.Output;
        command.Parameters.Add(maxAgeParam);



            command.ExecuteNonQuery();

                    MessageBox.Show("Минимальный возраст: "+ command.Parameters["@minAge"].Value);
                    MessageBox.Show("Максимальный возраст: " + command.Parameters["@maxAge"].Value);
```

Транзакции

Транзакции позволяют выполнять ряд операций в виде одного целостного пакета. И если хотя бы одна из этих операций завершится неудачно, то произойдет откат выполнения остальных операций.

```
connection.Open();
        SqlTransaction transaction = connection.BeginTransaction();

        SqlCommand command = connection.CreateCommand();
        command.Transaction = transaction;

        try
            {
               // выполняем две отдельные команды
               command.CommandText = "INSERT INTO Users (Name, Age) VALUES('Tim', 34)";
               command.ExecuteNonQuery();
               command.CommandText = "INSERT INTO Users (Name, Age) VALUES('Kat', 31)";
               command.ExecuteNonQuery();

               // подтверждаем транзакцию
               transaction.Commit();
               Console.WriteLine("Данные добавлены в базу данных");
            }
            catch (Exception ex)
            {
               Console.WriteLine(ex.Message);
               transaction.Rollback(); // откат транзакции
            }
```

Работа с SqlDataAdapter и DataSet

Для получения данных через объект **SqlDataAdapter** необходимо организовать подключение к БД и выполнить команду SELECT. Есть несколько способов создания **SqlDataAdapter**.

SqlDataAdapter adapter = new SqlDataAdapter();
Можно использовать конструктор без параметров, а команду SELECT и подключение установить позже

SqlDataAdapter adapter = new SqlDataAdapter(command);
Можно передать в конструктор объект SqlCommand

SqlDataAdapter adapter = new SqlDataAdapter(sql, connection);
Можно в конструкторе установить sql-выражение SELECT и объект SqlConnection

SqlDataAdapter adapter = new SqlDataAdapter(sql, connectionString);
Можно в конструкторе установить sql-выражение SELECT и строку подключения

```
string sql = "SELECT * FROM Users";
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();
            // Создаем объект DataAdapter
            SqlDataAdapter adapter = new SqlDataAdapter(sql, connection);

        }
```

Работа с SqlDataAdapter и DataSet

DataSet представляет хранилище данных, с которыми можно работать независимо от наличия подключения, а **SqlDataAdapter** заполняет **DataSet** данными из БД.

```
using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();
            // Создаем объект DataAdapter
            SqlDataAdapter adapter = new SqlDataAdapter(sql, connection);
            // Создаем объект Dataset
            DataSet ds = new DataSet();
            // Заполняем или обновляем строки в Dataset
            adapter.Fill(ds);
            // Отображаем данные
            dataGridView1.DataSource = ds.Tables[0];
        }
```

dataGridView1.SelectionMode = DataGridViewSelectionMode.FullRowSelect;
Выделяет всю строку при её выделении

dataGridView1.AllowUserToAddRows = false;
Запрет добавления новой строки

SqlCommandBuilder и сохранение изменений DataSet в базе данных

Для модификации данных в БД в соответствии с изменениями в DataSet SqlDataAdapter использует команды InsertCommand, UpdateCommand и DeleteCommand. Можно определить для этих команд sql-выражения, либо воспользоваться классом SqlCommandBuilder, который позволяет автоматически сгенерировать нужные выражения.

```
string sql = "SELECT * FROM Users";
connection.Open();
SqlDataAdapter adapter = new SqlDataAdapter(sql, connection);
DataSet ds = new DataSet();
adapter.Fill(ds);

DataTable dt = ds.Tables[0];
// добавим новую строку
DataRow newRow = dt.NewRow();
newRow["Name"] = "Алиса";
newRow["Age"] = 24;
dt.Rows.Add(newRow);
```

```
foreach (DataColumn column in dt.Columns)
    Console.Write("\t{0}", column.ColumnName);

// перебор всех строк таблицы
foreach (DataRow row in dt.Rows)
{
    // получаем все ячейки строки
    var cells = row.ItemArray;
    foreach (object cell in cells)
        Console.Write("\t{0}", cell);
 }
```

SqlCommandBuilder и сохранение изменений DataSet в базе данных

```
// создаем объект SqlCommandBuilder
SqlCommandBuilder commandBuilder = new SqlCommandBuilder(adapter);
adapter.Update(ds);
// альтернативный способ - обновление только одной таблицы
//adapter.Update(dt);
// заново получаем данные из бд
// очищаем полностью DataSet
ds.Clear();
// перезагружаем данные
adapter.Fill(ds);
```

Получение sql-выражения используемых команд:

При необходимости мы можем получить sql-выражения используемых команд:

```
Console.WriteLine(commandBuilder.GetUpdateCommand().CommandText);
Console.WriteLine(commandBuilder.GetInsertCommand().CommandText);
Console.WriteLine(commandBuilder.GetDeleteCommand().CommandText);
```

команда обновления будет выглядеть так:

```
UPDATE [Users] SET [Name]=@p1, [Age]=@p2 WHERE (([Id]=@p3) AND ([Name]=@p4) AND ([Age]=@p5))
```

Команда вставки:

```
INSERT INTO [Users] ([Name],[Age]) VALUES (@p1, @p2)
```

Команда удаления:

```
DELETE FROM [Users] WHERE (([Id]=@p1) AND ([Name]=@p2) AND ([Age]=@p3))
```