Урок № 2: Работа в присоединённом режиме

Содержание урока:

1. Класс DbConnection
   a. Цели и задачи класса DbConnection и его потомков
   b. Анализ методов и свойств
   c. Пример соединения с источником данных
2. Класс DbCommand
   a. Цели и задачи класса DbCommand и его потомков
   b. Анализ методов и свойств
   c. Пример посылки запроса
3. Примеры вставки, обновления, удаления данных
4. Класс DbDataReader
   a. Цели и задачи класса DbDataReader и его потомков
   b. Анализ методов и свойств
   c. Пример получения данных
5. Использование параметров
   a. Класс DbParameter
      - Цели и задачи класса DbParameter и его потомков
      - Анализ методов и свойств
      - Пример передачи параметров
   b. Использование и вызов хранимых процедур
6. Использование транзакций
   a. Класс DbTransaction
      - Цели и задачи класса DbTransaction и его потомков
      - Анализ методов и свойств
      - Пример работы с транзакциями
   b. Использование механизма транзакций аппарата СУБД
7. Сохранение и получение значений BLOB в базе данных

В этом уроке Вы узнаете, каким образом можно осуществить взаимодействие Вашего приложения с серверами баз данных для получения, обновления и удаления данных из таблиц, осуществлять вызов хранимых процедур и получение результатов их работы, обрабатывать ошибки при взаимодействии с СУБД.

Существует два принципиально разных методики для взаимодействия приложений с серверами баз данных.

Во-первых, это классический способ взаимодействия в присоединённом режиме, когда приложение и сервер имеют постоянное надёжное подключение. Именно этот способ взаимодействия мы будем рассматривать в рамках этого урока.

А во-вторых, это новый, появившийся в ADO.Net способ взаимодействия, при котором связь с сервером может осуществляться эпизодически, а данные кэшируются на клиенте, называемый отсоединённым режимом, рассмотрение которого будет проведено в последующих уроках.

Соединение с БД (объект Connection)

Для начала работы с базами данных в первую очередь необходимо, чтобы приложение установило соединение с целевой БД.

В Microsoft .Net Framework существует набор классов, специально предназначенных для выполнения соединения с соответствующими серверами баз данных. Все эти классы расположены в пространстве имён **System.Data**:

- **System.Data.SqlClient.SqlConnection** (MS SQL Server)
- **System.Data.OleDb.OleDbConnection** (соединение через OleDb)
- **System.Data.Odbc.OdbcConnection** (соединение с использованием технологии ODBC)
- **System.Data.OracleClient.OracleConnection** (соединение с сервером Oracle)

Все эти классы унаследованы от общего для них абстрактного базового класса **System.Data.Common.DbConnection**:

C#

```
public abstract class DbConnection : Component, IDbConnection, IDisposable
```

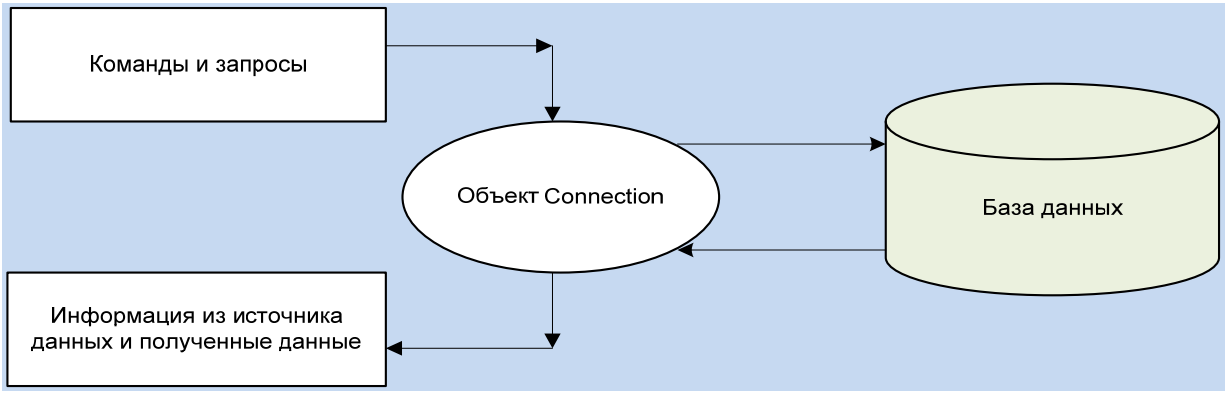
Кроме того, сторонними разработчиками могут быть реализованы соответствующие классы для соединения с другими СУБД, например класс Connection в ADODB для сервера Firebird или Interbase (соответствующую

dll можно скачать с сайта ibase.ru, или можно использовать IbProvider: http://www.ibprovider.com/rus/download.html).

Все эти классы предназначены для создания соединения с серверами соответствующих баз данных, поддержания соединения и его завершения в приложениях, написанных для .Net Framework.

Давайте подробнее рассмотрим, что из себя представляет объект класса **DbConnection**.

Connection – это канал, по которому приложение отправляет команды и запросы к базе данных, а также получает результаты выполнения запросов от базы. Объект типа Connection самостоятельно не получает и не обновляет данные, не получает запросов и не содержит результатов их выполнения.



Существуют два основных способа создания объектов типа Connection: создание подключения средствами мастеров VisualStudio и создание объектов типа Connection программным путём.

При создании объекта Connection в первую очередь необходимо определить, к какому источнику данных Вы будете осуществлять подключение. Как я уже описывал выше, для присоединения к источнику данных можно использовать один из классов, унаследованных от **System.Data.Common.DbConnection**: **SqlConnection**, **OleDbConnection**, **OdbcConnection** или **OracleConnection**.

Давайте поближе познакомимся с этими классами, ниже я приведу свойства и методы класса **DbConnection** и унаследованных от него классов:

Основные свойства класса **DbConnection**:

Свойство	Описание
String ConnectionString	Возвращает или задаёт строку, используемую для открытия подключения.
int ConnectionTimeout	Только для чтения. Возвращает время ожидания установки подключения до завершения попыток подключения и генерирования ошибки.
String Database	Только для чтения. Возвращает имя текущей базы данных после открытия подключения или имя базы данных, указанное в строке подключения до его открытия.
String DataSource	Только для чтения. Возвращает имя сервера, к которому производиться подключение.
ConnectionState State	Только для чтения. Возвращает строку, описывающую подключение.

С остальными свойствами класса **DbConnection** Вы можете познакомиться, обратившись к MSDN.

Основные методы класса **DbConnection**:

Метод	Описание
BeginDbTransaction	Начинает явную транзакцию в базе данных.
BeginTransaction	
ChangeDatabase	Изменяет текущую базу данных для открытого подключения.
Close	Закрывает текущее подключение.
CreateCommand	Создаёт и возвращает объект System.Data.Common.DbCommand, предназначенный для выполнения запросов через текущее подключение..
CreateDbCommand	
EnlistTransaction	Входит в указанную транзакцию как распределённая транзакция
GetSchema	Возвращает информацию схемы источника данных класса System.Common.DbConnection.
Open	Открывает подключение к базе данных с настройками, указанными в System.Data.Common.DbConnection.ConnectionString.

С остальными методами класса **DbConnection** Вы можете познакомиться, обратившись к MSDN

Поскольку класс **System.Data.Common.DbConnection** является абстрактным классом, то создавать объекты этого класса невозможно. Вы должны использовать для создания подключения к базе данных объекты конкретных классов, унаследованных от **DbConnection**:

```
System.Data.Common.DbConnection conn;

try
{
  conn = new System.Data.SqlClient.SqlConnection(
  "Data Source=(local); Initial Catalog=_Library; Integrated Security=SSPI;");
  conn.Open();
  /*Работаем с базой данных*/
}

catch (System.Data.Common.DbException ex)
{
  MessageBox.Show(ex.Message);
}
```

Обращаю Ваше внимание на обязательную в данном случае обработку исключительных ситуаций. Дело в том, что исключительная ситуация в данном коде может возбуждаться по причинам, не зависящим от функциональности Вашего приложения, например в случае проблем с соединением или с запуском службы SqlServer. Поэтому перехват **System.Data.Common.DbException** является обязательным!!!

И ещё один пример создания объектов Connection: **OdbcConnection** и **OracleConnection** (в последнем случае нужно не забыть подключить в References **System.Data.OracleClient**):

```
System.Data.Common.DbConnection conn=null;
try{
  switch (ConnType){
     case 0:  conn = new System.Data.Odbc.OdbcConnection(
                        "Dsn=_AllPhonesDnepropetrovsk;");
              break;
     case 2:  conn = new System.Data.OracleClient.OracleConnection(
                        "Data Source=Oracle9i; Integrated Security=yes;");
              break;
  }
  conn.Open();
          /*Здесь пишем код для работы с базой данных*/
  if(conn!=null)
     conn.Close();
}
catch (System.Data.Common.DbException ex){
              MessageBox.Show(ex.Message);
}
```

Как видно, работа с различными классами, унаследованными от **System.Data.Common.DbConnection** принципиально не отличается между собой. Вызов соответствующего конструктора и задание строки соединения, вызов метода Open для создания соединения и метода Close для его закрытия.

Осталось совсем немного – правильно написать **ConnectionString** для того или иного провайдера. Во времена юности технологии ADO это приходилось делать вручную, читая соответствующие спецификации в MSDN.

При написании строки соединения обратите внимание на следующие её составляющие:

Data Source – имя сервера;
Initial Catalog – имя базы данных;
Integrated Security – способ проверки пользователя (Windows или встроенная в сервер);
Dsn – для ODBC имя источника данных(DSN);
Provider – для OLEDB имя провайдера.
User ID – имя пользователя для входа
Password – пароль

Конкретное содержание строки подключения зависит от типа источника данных.

Для облегчения написания строки соединения в Microsoft ADO.Net Framework 2.0 появился набор классов унаследованных от **System.Data.Common.DbConnectionStringBuilder**, предназначенных для формирования строки соединения путём заполнения соответствующих полей класса:

Провайдер	Класс, унаследованный от ConnectionStringBuilder
System.Data.SqlClient	SqlConnectionStringBuilder
System.Data.OleDb	OleDbConnectionStringBuilder

System.Data.Odbc	OdbcConnectionStringBuilder
System.Data.OracleClient	OracleConnectionStringBuilder

Работать с ConnectionStringBuilder очень просто, покажу Вам это на примере **SqlConnectionStringBuilder:**

```
SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();
builder.InitialCatalog = "_Library"; //Задаём InitialCatalog
builder.IntegratedSecurity = true;   //Задаём режим проверки подлинности
builder.DataSource = "(local)";      //Задаём сервер

SqlConnection connection = new SqlConnection(); //Создаём объект Connection
connection.ConnectionString = builder.ConnectionString;

try{
    connection.Open();  //Выполняем соединение
}
catch (System.Data. SqlClient.SqlException ex)
{
    MessageBox.Show(ex.Message);
}
```

Выполнение команд (объект Command)

Выполнить соединение с источником данных для полноценной работы с ним мало. Хотелось бы выполнить запросы SQL, добавить в базу данные.

Для облегчения разработки кода выполнения запросов к базе данных в ADO.Net также предусмотрен набор классов, унаследованных от единого абстрактного класса **System.Data.Common.DbCommand:**

C#

```
public abstract class DbCommand : Component, IDbCommand, IDisposable
```

Каждый из этих классов предназначен для выполнения SQL запроса на соответствующем источнике данных (SqlServer, Oracle, ODBC) и расположен в соответствующем пространстве имён .Net Framework:

- **System.Data.SqlClient.SqlCommand;**
- **System.Data.OleDb.OleDbCommand;**
- **System.Data.Odbc.OdbcCommand;**
- **System.Data.OracleClient.OracleCommand.**

Давайте подробнее рассмотрим основные методы, при помощи которых осуществляется взаимодействие класса **DbCommand** (и его наследников) с базами данных:

Cancel	Пытается отменить выполнение объекта DbCommand.
CreateDbParameter	Создает новый экземпляр объекта DbParameter.
CreateParameter	Создает новый экземпляр объекта DbParameter.
ExecuteDbDataReader	Выполняет текст команды применительно к подключению.
ExecuteNonQuery	Выполняет оператор SQL применительно к объекту подключения.
ExecuteReader	Перегружен. Выполняет свойство CommandText применительно к свойству Connection и возвращает объект DbDataReader.
ExecuteScalar	Выполняет запрос и возвращает первый столбец первой строки результирующего набора, возвращаемого запросом. Все другие столбцы и строки игнорируются.
Prepare	Создает подготовленную (или скомпилированную) версию команды для источника данных.

Более подробно Вы можете познакомиться с методами класса **System.Data.Common.DbCommand** в MSDN.

Ниже перечислены основные свойства класса **System.Data.Common.DbCommand:**

CommandText	Возвращает или задает SQL запрос, выполняющийся применительно к источнику данных
CommandTimeout	Возвращает или задает время ожидания перед завершением попытки выполнить команду и генерацией исключения.
CommandType	Указывает или определяет способ интерпретации свойства CommandText.
Connection	Возвращает или задает объект DbConnection, используемый этим объектом DbCommand.
DbConnection	Возвращает или задает объект DbConnection, используемый этим объектом DbCommand.
DbParameterCollection	Получает коллекцию объектов DbParameter.
DbTransaction	Возвращает или задает свойство DbTransaction внутри которого выполняется этот объект DbCommand.
Parameters	Получает коллекцию объектов DbParameter.
Transaction	Возвращает или задает свойство DbTransaction внутри которого выполняется этот объект DbCommand.
UpdatedRowSource	Возвращает или задает способ применения результатов команды к объекту DataRow, если он используется методом Update объекта DbDataAdapter.

Более подробно Вы можете познакомиться со свойствами класса **System.Data.Common.DbCommand** в MSDN

Ниже я приведу пример того, как выполнить запрос в базе данных SQL Server на создание таблицы. Поскольку запрос CREATE TABLE не возвращает никаких результатов, то для выполнения запроса я применил метод класса **SqlCommand ExecuteNonQuery**

```
private void buttonSQL_Click(object sender, EventArgs e)
{
   SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();
   builder.InitialCatalog = "_Library"; //Задаём InitialCatalog
   builder.IntegratedSecurity = true;   //Задаём режим проверки подлинности
   builder.DataSource = "(local)";      //Задаём сервер

   SqlConnection connection = new SqlConnection();//Создаём объекn
Connection
            connection.ConnectionString = builder.ConnectionString;

   try
   {
       connection.Open();       //Выполняем соединение с сервером
       SqlCommand command = new SqlCommand(  //Создаём объект Command
           @"CREATE TABLE tbFaculties(Id INT IDENTITY,
           NAME VARCHAR(10))", connection);

       command.ExecuteNonQuery(); //Выполняем запрос

       connection.Close();  //Закрываем соединение
    }

    catch (System.Data.Common.DbException ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }
}
```

Аналогично работе с **System.Data.SqlClient.SqlCommand** можно работать и с другими классами, такими как**, System.Data.OleDb.OleDbCommand, System.Data.Odbc.OdbcCommand, System.Data.OracleClient.OracleCommand**;

Небольшой пример для **System.Data.OleDb.OleDbCommand, System.Data.Odbc.OdbcCommand** я приведу ниже. Во всех примерах просто создаётся соединение и выполняется запрос на создание простой таблицы в базе данных. Что касается **Odbc.OdbcCommand**, то выполнение запросов в нём ничем не отличается от остальных объектов Command.

```
private void buttonOleDb_Click(object sender, EventArgs e)
{
   /* Создаём соединение */
   OleDbConnection connection = new OleDbConnection();
   connection.ConnectionString = @"Provider=Microsoft.Jet.OLEDB.4.0;
   data source=d:\library.mdb";

   try{
        connection.Open(); //выполняем соединение

        OleDbCommand command = new OleDbCommand(  //Создаём объект Command
                  @"CREATE TABLE tbFaculties(Id COUNTER,
                   NAME VARCHAR(10))", connection);

        command.ExecuteNonQuery(); //Выполняем запрос
        connection.Close();
   }
   catch (System.Data.OleDb.OleDbException ex)
   {
        MessageBox.Show(ex.Message);
        return;
   }
}

private void buttonOdbc_Click(object sender, EventArgs e)
{
   /* Создаём Command */
   OdbcCommand comm = new OdbcCommand();
   /* и под него соединение (не забудте создать DSN с именем _Test)*/
   comm.Connection = new OdbcConnection("Dsn=_Test;");

   try{
      //в свойство  CommandText вносим наш SQL запрос
      comm.CommandText =
        @"CREATE TABLE tbFaculties(Id INT PRIMARY KEY, NAME VARCHAR(10))";

      comm.Connection.Open(); //выполняем соединение
      comm.ExecuteNonQuery(); //выполняем запрос
      comm.Connection.Close();//отсоединяемся
   }
   catch (System.Data.Common.DbException ex)
   {
       MessageBox.Show(ex.Message);
       return;
   }
}
```

Все вышеприведенные примеры находятся в папке Source Вашего урока.

Добавление, модификацию, и удаление данных в источнике проще всего организовать путём выполнения запросов INSERT, UPDATE, DELETE соответственно.

Давайте напишем небольшой пример простого телефонного справочника с функцией добавления и просмотра информации об абоненте. Информация будет храниться в файле SimplePhoneBook.accdb (access 2007), файл базы вы найдёте в папке Source, я буду использовать OleDbConnection, строка соединения для Access 2007: "Provider= Microsoft.ACE.OLEDB.12.0; Data Source=SimplePhoneBook.accdb; Persist Security Info=False". В случае, если у Вас Access 2003, то создайте базу данных SimplePhoneBook.mdb и измените строку подключения на следующую: "Provider=Microsoft.Jet.OLEDB.4.0; data source= SimplePhoneBook.mdb"

Пример программы вы также найдёте в папке Source. Обратите внимание, что соединение с источником данных происходит при загрузке формы, а отсоединение происходит при закрытии формы. Сам объект **OleDbConnection** сделан членом класса.

Добавление нового абонента в базу может быть реализовано примерно такой функцией:

```
internal void AddDataInDatabase(String Name,
      String Surname, String EMail, String Phone)
{
     try
     {
         OleDbCommand command = connection.CreateCommand();
         command.CommandText = String.Format(
              @"INSERT INTO tbPhones (Name, Surname, EMail, Phone)
              VALUES ('{0}', '{1}', '{2}', '{3}')",
              Name, Surname, EMail, Phone);
              command.ExecuteNonQuery();
     }
     catch (OleDbException ex)
     {
         MessageBox.Show(ex.Message);
     }
}
```

Нам понадобиться реализовать удаление записи из базы:

```
private void DeleteDataInDatabase(int Id){
   try{
        OleDbCommand command = connection.CreateCommand();
        command.CommandText = String.Format(
               @"DELETE FROM tbPhones WHERE Id= {0}", Id);
        command.ExecuteNonQuery();
   }
   catch (OleDbException ex)
   {
         MessageBox.Show(ex.Message);
   }
}
```

Заодно можно реализовать и изменение отдельных полей в базе данных.

```
private void UpdateName(int Id, String Name, String Surname){
    try{
        OleDbCommand command = connection.CreateCommand();
        command.CommandText = String.Format(
            @"UPDATE tbPhones SET Name={1}, Surname={2} WHERE Id={0}",
            Id, Name, Surname);
        command.ExecuteNonQuery();
    }
    catch (OleDbException ex){
        MessageBox.Show(ex.Message);
    }
}
private void UpdateMail(int Id, String EMail){
    try{
        OleDbCommand command = connection.CreateCommand();
        command.CommandText = String.Format(
             @"UPDATE tbPhones SET EMail={1} WHERE Id={0}", Id, EMail);
        command.ExecuteNonQuery();
    }
    catch (OleDbException ex){
        MessageBox.Show(ex.Message);
    }
}
private void UpdatePhone(int Id, String Phone){
    try{
        OleDbCommand command = connection.CreateCommand();
        command.CommandText = String.Format(
            @"UPDATE tbPhones SET Phone={0} WHERE Id={1}", Id, Phone);
        command.ExecuteNonQuery();
    }
    catch (OleDbException ex){
        MessageBox.Show(ex.Message);
    }
}
```

Получение информации из базы данных. Класс DbDataReader

Метод **ExecuteNonQuery** класса **DbCommand** позволяет выполнить запрос в базе данных в том случае, если в результате выполнения запроса объект **DbCommand** не возвращает результата выполнения запроса.

В случае, если в результате выполнения запроса предполагается, что мы получаем набор данных, то необходимо выполнить **ExecuteScalar**, который выполняет запрос и возвращает первый столбец первой строки результирующего набора, возвращаемого запросом. Все другие столбцы и строки игнорируются. **ExecuteScalar** обычно применяется для выполнения запросов, возвращающих результат выполнения агрегатной функции.

```
private int GetRowCount()
{
    try
    {
        OleDbCommand comm = connection.CreateCommand();
        comm.CommandText = @"SELECT COUNT(*) FROM tbPhones";

        return (int)comm.ExecuteScalar();
    }
    catch(OleDbException ex)
    {
        MessageBox.Show(ex.Message);
    }
    return 0;
}
```

В случае, если предполагается получение набора данных, содержащий более одного столбца и одной записи, то для выполнения запроса у объекта класса **DbCommand** вызывается метод **ExecuteReader**, который возвращает объект класса, унаследованного от **System.Data.Common.DbDataReader**. В зависимости от конкретного типа объекта **DbCommand**, это может быть объект одного из следующих классов:

- **System.Data.SqlClient.SqlDataReader**
- **System.Data.OleDb.OleDbDataReader**
- **System.Data.Odbc.OdbcDataReader**
- **System.Data.OracleClient.OracleDataReader**

Давайте подробнее познакомимся с методами и свойствами класса **System.Data.Common.DbDataReader.**

Основные свойства класса **DbDataReader**

Свойство	Описание
Depth	Получает значение, указывающее глубину вложенности для текущей строки
FieldCount	Получает значение, указывающее глубину вложенности для текущей строки
HasRows	Получает значение, указывающее, содержит ли этот объект DbDataReader одну или несколько строк
IsClosed	Получает значение, указывается, является ли объект DbDataReader закрытым
Item	Перегружен. Получает значение заданного столбца в виде экземпляра класса Object.
RecordsAffected	Получает количество строк, которые были изменены, вставлены или удалены при выполнении оператора SQL.
VisibleFieldCount	Получает количество нескрытых полей в DbDataReader.

Основные методы класса **DbDataReader**

Метод	Описание
Close	Закрывает объект DbDataReader
CreateObjRef	Создает объект, который содержит всю необходимую информацию для создания прокси-сервера, используемого для взаимодействия с удаленным объектом. (Унаследовано от MarshalByRefObject.)
GetBoolean	Получает значение заданного столбца в виде логического значения
GetByte	Получает значение заданного столбца в виде байта.
GetBytes	Считывает поток байтов из указанного столбца, начиная с местоположения, указанного параметром dataOffset, в буфер, начиная с местоположения, указанного параметром bufferOffset

GetChar	Получает значение заданного столбца в виде одного символа
GetChars	Считывает поток символов из указанного столбца, начиная с местоположения, указанного параметром dataIndex, в буфер, начиная с местоположения, указанного параметром bufferIndex
GetData	Возвращает объект DbDataReader для запрошенного порядкового номера столбца.
GetDataTypeName	Получает имя типа данных указанного столбца.
GetDateTime	Получает значение заданного столбца в виде объекта DateTime.
GetDbDataReader	Возвращает объект DbDataReader для запрошенного порядкового номера столбца, который может быть переопределен с помощью зависящей от поставщика реализации
GetDecimal	Получает значение заданного столбца в виде объекта Decimal.
GetDouble	Получает значение заданного столбца в виде числа двойной точности с плавающей запятой
GetEnumerator	Возвращает значение IEnumerator, которое можно использовать для итерации элементов строк в модуле чтения данных.
GetFieldType	Получает тип данных указанного столбца.
GetFloat	Получает значение указанного столбца в виде числа одиночной точности с плавающей запятой
GetGuid	Возвращает значение заданного столбца в виде глобального уникального идентификатора (GUID)
GetInt16	Получает значение заданного столбца в виде 16-битового целого числа со знаком
GetInt32	Получает значение заданного столбца в виде 32-битового целого числа со знаком
GetInt64	Получает значение заданного столбца в виде 64-битового целого числа со знаком
GetName	Получает имя столбца при наличии заданного порядкового номера (с нуля) столбца

GetOrdinal	Получает порядковый номер столбца при наличии заданного имени столбца
GetProviderSpecificFieldType	Возвращает зависящий от поставщика тип поля заданного столбца
GetProviderSpecificValue	Получает значение заданного столбца в виде экземпляра класса Object
GetProviderSpecificValues	Получает зависящие от поставщика столбцы атрибутов в коллекции для текущей строки
GetSchemaTable	Возвращает объект DataTable, описывающий метаданные столбца объекта DbDataReader
GetString	Получает значение заданного столбца в виде экземпляра класса String
GetValue	Получает значение заданного столбца в виде экземпляра класса Object.
GetValues	Получает все атрибуты столбцов в коллекции для текущей строки.
IsDBNull	Получает значение, указывающее, содержит ли столбец несуществующие или пропущенные значения
NextResult	Перемещает считыватель к следующему результату при считывании результатов выполнения пакетных операторов
Read	Перемещает считыватель на следующую запись в наборе результатов

Для получения данных из объекта **DbDataReader** вызывается метод **Read**, и в случае, если этот метод возвращает true, производиться считывание данных из объекта **DbDataReader** построчно по одной записи, переход к следующей записи происходит при повторном вызове метода **Read**.

Выборка данных по полям производиться при помощи методов Get… (вместо многоточия – соответствующий тип, например GetString(int) выбирает значение из поля со строчным значением, а GetInt32(int) – из поля типа INT), принимающих порядковый номер поля в полученном наборе данных.

Узнать порядковый номер поля по его имени позволяет метод GetOrdinal(string)

Посмотрите на пример кода, в котором производиться выборка данных из телефонной книги и помещение их в ListView:

```
private void ReadData()
{
   try
   {
      OleDbCommand comm = connection.CreateCommand();
      comm.CommandText = @"SELECT Id,
      Name, Surname, EMail, Phone
      FROM tbPhones";

      OleDbDataReader reader = comm.ExecuteReader(); //Получение ридера

      while (reader.Read())
      {
         ListViewItem item = listView1.Items.Add(new ListViewItem());
         item.Text = reader.GetString(reader.GetOrdinal("Name"));
         item.SubItems.Add(reader.GetString(reader.GetOrdinal("Surname")));
         item.SubItems.Add(reader.GetString(reader.GetOrdinal("EMail")));
         item.SubItems.Add(reader.GetString(reader.GetOrdinal("Phone")));
      }
      reader.Close();

   }
   catch(OleDbException ex)
   {
      MessageBox.Show(ex.Message);
   }
}
```

Полный текст примера Вы как всегда можете найти в папке Source. В папке Source.Databases вы найдёте файлы баз данных в формате assess 2007 и access 2003. Не забудьте их положить в папку с исполнимым файлом.

Вызовы хранимых процедур, использование класса DbParametr

У объектов **DbCommand** есть свойство **Parameters**, представляющее собой коллекцию параметров для выполнения SQL запроса. Параметры запроса представляют из себя объекты производные от типа **System.Data.Common.DbParameter**. Конкретные реализации производных от **DbParameter** классов находятся в соответствующих пространствах имён

- **System.Data.SqlClient.SqlParameter;**
- **System.Data.OleDb.OleDbParameter;**
- **System.Data.Odbc.OdbcParameter;**
- **System.Data.OracleClient.OracleParameter;**

Объект **DbParameter** предназначен для передачи в DbCommand параметров запросов и используется для создания параметризированных запросов. Абсолютно незаменимым **DbParameter** является при вызове хранимых процедур.

Параметр можно описать как тип переменной, используемый для передачи и получения значений между приложением и базой данных. Так же как и переменные, параметры имеют тип, определяемый свойством **DbType** (типы определены в перечислении **System.Data.SqlDbType**).

При выполнении запросов параметры могут использоваться как для передачи данных в базу (Input параметры), так и для получения данных из базы.

Чтобы добавить параметр к запросу его необходимо создать и правильно проинициализировать. Необходимо указать как минимум имя параметра (у MS SQL Server имена параметров начинаются с символа «@», например «@login»), его тип и значение (для Input параметров). Кроме того необходимо указать направление (**ParameterDirection**) **Input**, **Output**, **InputOutput** или **ReturnValue** – соответственно для входных, выходных или двунаправленных параметров, или для значения, возвращаемого из хранимой процедуры.

После создания параметра, он добавляется в коллекцию Parameters класса DbCommand.

Давайте поближе познакомимся с методами и свойствами класса **DbPerameter**:

Свойство	Описание
DbType	Возвращает или задает значение DbType параметра.
Direction	Возвращает или задает значение, определяющее, предназначен ли параметр только для ввода, только для вывода, является ли он двунаправленным или возвращается хранимой процедурой.
IsNullable	Возвращает или задает значение, показывающее, может ли параметр принимать значения NULL.
ParameterName	Возвращает или задает имя DbParameter.
Size	Возвращает или задает наибольший размер данных в столбце (в байтах).
SourceColumn	Возвращает или задает имя исходного столбца, который сопоставляется с объектом DataSet и используется для загрузки и возврата свойства Value.
SourceColumnNullMapping	Возвращает или задает значение, показывающее, может ли исходный столбец содержать значение NULL. Это позволяет объекту DbCommandBuilder правильно генерировать операторы Update для столбцов, которые могут содержать значения null.
SourceVersion	Возвращает или задает значение DataRowVersion, используемое при загрузке свойства Value.
Value	Возвращает или задает значение параметра.

Из собственных методов класса **DbPerameter** можно указать только один: **ResetDbType(),** который сбрасывает свойство DbType к его исходным параметрам.

Ниже я приведу примеры работы с классом SqlParameter и SqlCommand в коде предназначенном для вызова хранимых процедур GetMessage и GetMessageCount из базы данных. В первом случае хранимая процедура выполняет обычный SELECT, а во втором случае записывает результат выполнения в OUTPUT параметр.

Как всегда код примера в папке Source, а база в папке Databases.

```
private void GetMessageButton_Click(object sender, EventArgs e)
{
   try
   {
      listView1.Items.Clear();

      SqlConnection connection = new SqlConnection(
      "Data Source=(local); Initial Catalog=Messenger; Integrated Security=SSPI;");

      SqlCommand command=connection.CreateCommand();
      command.CommandType = CommandType.StoredProcedure;
      command.CommandText="GetMessages";

      SqlParameter LoginParam = new SqlParameter();
      LoginParam.ParameterName="@login";
      LoginParam.SqlDbType=SqlDbType.VarChar;
      LoginParam.Direction=ParameterDirection.Input;
      LoginParam.Value=textBoxLogin.Text;

      SqlParameter PwdParam = new SqlParameter();
      PwdParam.ParameterName="@pwd";
      PwdParam.SqlDbType=SqlDbType.VarChar;
      PwdParam.Direction=ParameterDirection.Input;
      PwdParam.Value=textBoxPwd.Text;

      command.Parameters.Add(LoginParam);
      command.Parameters.Add(PwdParam);

      connection.Open();

      SqlDataReader reader = command.ExecuteReader();

      while (reader.Read())
      {
         ListViewItem item = listView1.Items.Add(new ListViewItem());
         item.Text = reader.GetString(reader.GetOrdinal("Login"));
         item.SubItems.Add(reader.GetString(reader.GetOrdinal("Message")));
      }
   }
   catch (SqlException ex)
   {
      MessageBox.Show(ex.Message);
   }
}
```

```
private void GetMessageCountButton_Click(object sender, EventArgs e)
{
    SqlConnection connection = new SqlConnection(
        "Data Source=(local); Initial Catalog=Messenger; Integrated
Security=SSPI;");

    SqlCommand command = connection.CreateCommand();
    command.CommandType = CommandType.StoredProcedure;
    command.CommandText = "GetMessagesCount";

    SqlParameter LoginParam = new SqlParameter();
    LoginParam.ParameterName = "@login";
    LoginParam.SqlDbType = SqlDbType.VarChar;
    LoginParam.Direction = ParameterDirection.Input;
    LoginParam.Value = textBoxLogin.Text;

    SqlParameter PwdParam = new SqlParameter();
    PwdParam.ParameterName = "@pwd";
    PwdParam.SqlDbType = SqlDbType.VarChar;
    PwdParam.Direction = ParameterDirection.Input;
    PwdParam.Value = textBoxPwd.Text;

    SqlParameter CountParam = new SqlParameter();
    CountParam.ParameterName = "@counter";
    CountParam.SqlDbType = SqlDbType.Int;
    CountParam.Direction = ParameterDirection.Output;

    command.Parameters.Add(LoginParam);
    command.Parameters.Add(PwdParam);
    command.Parameters.Add(CountParam);

    connection.Open();

    command.ExecuteNonQuery();
    textBox1.Text = String.Format("{0} rows", CountParam.Value.ToString());
    connection.Close();
}
```

Использование транзакций

Часто возникает ситуация, когда необходимо произвести над базой данных несколько действий, при этом, если при выполнении хотя бы одного запроса произошла ошибка, то необходимым бывает отменить не только то действие, при котором произошла ошибка, но и всю последовательность операций.

Представьте себе следующее, покупатель пришёл в магазин, купил товар и расплатился за него кредитной картой. При этом в базе данных магазина должна добавиться запись о продаже единицы товара, а в банке – соответственно запись о перечислении денег на счёт магазина. То есть мы имеет две разные операции, при этом, если вторая операция закончилась неудачей, то должна быть отменена также и первая – о продаже.

Чтобы автоматизировать подобное выполнение команд на практике используется механизм транзакций.

Транзакции – это команды, которые выполняются одним пакетом. При этом, если одна из команд завершиться неудачно, то будет прервано выполнение всей транзакции, а изменения внесённые другими командами будут откатаны. Транзакции используются для обеспечения целостности данных в базе.

Для того, чтобы начать транзакцию вызывается метод у объекта класса производного от **DbConnection**.

Метод **BeginTransaction** возвращает нам объект класса, унаследованного от System**.Data.Common. DbTransaction**. Соответственно, в зависимости от конкретного типа **DbConnection** мы получим объект одного из следующих классов:

- **System.Data.SqlClient.SqlTransaction;**
- **System.Data.OleDb.OleDbTransaction;**
- **System.Data.Odbc.OdbcTransaction;**
- **System.Data.OracleClient.OracleTransaction**;

С этого момента начинается выполнение транзакции, все изменения, вносимые в базу данных, будут выполняться в рамках транзакции.

Если всё пойдёт успешно, мы сможем применить все внесённые изменения, вызвав метод **Commit()** у текущей транзакции, или отменить все действия вызовом метода **Rollback()**.

```
void TestTransaction()
{
   SqlConnection connection = new SqlConnection(
       "Data Source=(local); Initial Catalog=Test; Integrated Security=SSPI;");

   connection.Open();

   /*Начинаем транзакцию - получаем объект SqlTransaction*/
   SqlTransaction trans = connection.BeginTransaction();

   SqlCommand command = connection.CreateCommand();

   //Делаем несколько действий над базой (любых)
   command.CommandText = "UPDATE tbData Set Name=''";
   command.ExecuteNonQuery();

   command.CommandText = "UPDATE tbUsers Set Login=''";
   command.ExecuteNonQuery();

   command.CommandText = "UPDATE tbUsers Set Pwd=''";
   command.ExecuteNonQuery();

   if (MessageBox.Show("Применить изменения ?", "Вопрос",
           MessageBoxButtons.YesNo, MessageBoxIcon.Warning,
           MessageBoxDefaultButton.Button2) == DialogResult.Yes)
      {
         trans.Commit();//применяем все изменения в базе
      }
      else
      {
         trans.Rollback();//откатываем изменения
      }
      connection.Close();
}
```

Сохранение и получение значений BLOB в базе данных

Данный раздел посвящен работе с объектами BLOB (большими бинарными данными). Объекты BLOB в базах данных – это фотографии, музыка, документы, которые можно сохранять непосредственно в базе данных.

В отличие от выполнения запросов и хранимых процедур, которые возвращают «простые» типы данных, работать с двоичными объектами намного сложнее.

Для помещения данных в базу обычно используются битовые массивы, сам же объект при считывании чаще всего помещается в MemoryStream или в FileStream.

Ниже я приведу пример записи объекта Image в базу данных. Функция считывает jpeg файл с файловой системы, создаёт уменьшенное изображение и помещает его в базу вместе с полным путём к файлу и пример чтения таких данных из файла.

Выбор файла и помещение данных в базу происходит при нажатии кнопки на форме.

Показ содержимого нашего PreView – при выборе соответствующего пункта в ListView;

Полный текст примера и база данных находятся в папке Source

Обратите внимание на помещение значения Id в свойство Tag у ListViewItem – такой приём может не раз пригодиться

```
private void button1_Click(object sender, EventArgs e)
{
   OpenFileDialog dlg = new OpenFileDialog();
   dlg.Filter = "Jpeg Files (*.jpg)|*.jpg|All Files (*.*)|*.*";
   if (dlg.ShowDialog(this) == DialogResult.OK)
   {
      try
      {
          Bitmap myBmp = new Bitmap(dlg.FileName);

          Image.GetThumbnailImageAbort myCallBack =
                 new Image.GetThumbnailImageAbort(ThumbnailCallBack);

          Image imgPreview =
                 myBmp.GetThumbnailImage(200, 200, myCallBack, IntPtr.Zero);

          MemoryStream ms = new MemoryStream();

          imgPreview.Save(ms, System.Drawing.Imaging.ImageFormat.Bmp);

          ms.Flush();
          ms.Seek(0, SeekOrigin.Begin);

          BinaryReader br=new BinaryReader(ms);

          byte[] image = br.ReadBytes((int)ms.Length);

          SqlCommand comm = new SqlCommand();
          comm.Connection = conn; //conn - SqlConnection - член класса

          comm.CommandText = @"INSERT INTO tbImages (Path, ImgPreView)
                  VALUES (@Path, @Image)";

          comm.Parameters.Add("@Path", SqlDbType.NChar, 260).Value = dlg.FileName;
          comm.Parameters.Add("@Image", SqlDbType.Image, image.Length).Value = image;

          comm.ExecuteNonQuery();

          ms.Close();

      }
      catch (SqlException ex)
      {
          MessageBox.Show(ex.Message);
      }

   }
}

public bool ThumbnailCallBack()
{
    return false;
}
```

```
private void listView1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (listView1.SelectedIndices.Count < 1)
         return;

  try
  {
    int id = (int)listView1.SelectedItems[0].Tag;

     SqlCommand comm = new SqlCommand();
     comm.Connection = conn;//conn.CreateCommand();

     comm.CommandText = @"SELECT ImgPreView FROM tbImages WHERE Id="+id.ToString();

     SqlDataReader reader = comm.ExecuteReader();

     MemoryStream ms = new MemoryStream();
     int buffSize = 100;

     byte[] buffer = new byte[buffSize];

     while (reader.Read())
     {
        int retval;
        int startIndex = 0;

        while (true)
        {
           retval = (int)reader.GetBytes(0, startIndex, buffer, 0, buffSize);
           if (retval <1) break;

           ms.Write(buffer, 0, retval);
           ms.Flush();
           startIndex += retval;
         }
      }

      if (ms.Length < 1) return;

      pictureBox1.Image = Image.FromStream(ms);
      ms.Close();

      reader.Close();
   }
   catch (SqlException ex)
   {
      MessageBox.Show(ex.Message);
   }
}
```

Домашнее задание

Разработать приложение для автоматизации учёта сотрудников в отделе кадров предприятия.

1. Приложение должно содержать информацию о сотруднике: фамилия, имя отчество, дата рождения, должность, номер приказа о приёме на работу и номер приказа об увольнении.
2. В базе данных также должна храниться фотография сотрудника.
3. Доступ к данным должен быть сделан через хранимые процедуры.
4. Необходимо ограничить доступ к данным путем раздачи прав пользователям системы