ЛАБОРАТОРНАЯ РАБОТА №7 “РАЗРАБОТКА ПИЛОЖЕНИЯ, ИСПОЛЬЗУЮЩЕГО ТЕХНЛОГИЮ ADO.NET”

1. Теоретическая часть

Преимущества и нововведения в ADO.NET
Использование разъединенной модели доступа к данным.

В клиент-серверных приложениях традиционно используется технология доступа к источнику данных, при которой соединение с базой поддерживается постоянно. Однако после широкого распространения приложений, ориентированных на Интернет, выявились некоторые недостатки такого подхода. Попробуем выявить некоторые из них.

Соединения с базой данных требуют выделения системных ресурсов, что может быть критично при большой нагрузке сервера. Хотя постоянное соединение позволяет несколько ускорить работу приложения, общий убыток от растраты системных ресурсов сводит преимущество на нет.
Специфика веб приложений не позволяет серверу в каждый момент времени знать, что необходимо пользователю. То есть до следующего запроса сервер не имеет представления, нужно ли еще поддерживать соединение.
Опыт разработчиков показал, что приложения с постоянным соединением с источником данных черезвычайно трудно поддаются масштабированию.
Хотя существуют и другие недостатки, приведенные наиболее существенны. Все эти проблемы порождаются постоянным соединением с базой данных и решаются в ADO.NET, где используется другая модель доступа. Теперь соединение устанавливается лишь на то короткое время, когда необходимо проводить операции над базой данных.

Следует признать, что новая технология иногда все же проигрывает традиционной. Для этих случаев рекомендовано (не только мною, но и Microsoft) использовать ADO. Примерами таких приложений служат програмы проводящие частые и объемные изменения содержания записей - заказ билетов, например (подробнее о причинах непригодности разъединенной модели в этом случае см. следующий раздел).

Хранение данных в объектах DataSet.

При работе с базой данных нам чаще всего приходится работать не с одной, а несколькими записями. Более того, данные эти могут собираться из различных таблиц. В разъединенной модели доступа к базе данных не имеет смысла соединяться и источником данных при каждом обращении. Исходя из

этого, представляется логичным хранить несколько строк и обращаться к ним при необходимости. Для этих целей и используется DataSet.

DataSet представляет собой, по сути, упрощенную реляционную базу данных и может выполнять наиболее типичные для таких баз данных операции. Теперь, в отличие от Recordset мы можем хранить в одном DataSet сразу несколько таблиц, связи между ними, выполнять операции выборки, удаления и обновления данных. Безусловно, разъединенная модель не позволяет постоянно отслеживать изменения в базе данных, производимые другими пользователями. Это может привести к ошибкам в таких приложениях, где информация должна обновляться каждый момент - заказ билетов или продажа ценных бумаг. Однако в любую секунду может быть получена свежая информация из базы данных через вызов метода FillDataSet. Таким образом, DataSet остается черезвычайно удобным для самого широкого класса приложений: когда необходимо получить данные из базы и как-либо обработать их.

Глубокая интеграция с XML.

Все более широко распространяющийся XML играет важнейшую роль в ADO.NET и приносит еще несколько преимуществ по сравнению с традиционным подходом.

Заметим для начала, что практически любой XML файл может быть использован как источник данных и на его основе может быть создан DataSet. точно также при передаче данных между компонентами или сохранении их в файл используется XML.

Программист, работающий с ADO.NET не обязательно должен иметь опыт работы с XML или познания в этом языке. Все операции остаются прозрачными для разработчика.

Так как XML имеет текстовое представление, это позволяет передавать его по протоколам типа HTTP через брандмауэры. Дело в том, что системы защиты обычно настроены на фильтрацию двоичной информации, текстовая же легко пропускается, что облегчает создание распределенных приложений.

XML представляет собой промышленный стандарт, поддерживаемый практически любой современной платформой, что позволяет передавать данные любому компоненту, умеющему работать с XML, и выполняющемуся под любой операционной системой.

При передаче больших объемов информации через COM возникает проблема приведения типов данных, так как COM поддерживает лишь ограниченный их набор. Действительно, COM маршаллинг может требовать длительной обработки, что негативно сказывается на

производительности приложения. XML же поддерживает неограниченное число типов и не требует их конверсии, что позволит ускорить процесс передачи данных.

Практическое применение ADO.NET

ADO.NET поддерживает два типа источников данных - SQL Managed Provider и ADO Managed Provider. SQL Managed Provider применяется для работы с Microsoft SQL Server 7.0 и выше, ADO Managed Provider - для всех остальных баз данных.

SQL Managed Provider - работает по специальному протоколу, называемому TabularData Stream (TDS) и не использует ни ADO, ни ODBC, ни какую-либо еще технологию. Ориентированный специально на MS SQL Server, протокол позволяет увеличить скорость передачи данных и тем самым повысить общую производительность приложения.

ADO Managed Provider - предназначен для работы с произвольной базой данных. Однако за счет универсальности есть проигрыш по сравнению с SQL Server Provider, так что при работе с SQL Server рекомендовано использовать специализированные классы. В данном обзоре мы коснемся ADO Managed Provider лишь мельком, указав только существующие незначительные различия, так как наиболее употребимой базой данных представляется SQL Server 7.0 или 2000, а разница заключается, в основном, в именовании.

Вне зависимости от того, какой поставщик вы будете использовать, вам придется работать с тремя общими объектами для взаимодействия с БД:

- Command Object
- DatasetCommand
- Connection Object

Connection Object

Следующие примеры кода иллюстрируют инициализацию соединения с БД:

```
using System.Data;
using System.Data.SQL;
public class SQLConnect {
private String connString = "";
private SQLConnection dataConn = null;
public string openConnection(String dbConnectString) {
connString = dbConnectString;
try {
dataConn = new SQLConnection(connString);
dataConn.Open();
return "SQL Server Data Connection Opened";

}
catch (Exception e) {
```

```
return(e.ToString());
}
finally {
if (dataConn != null) {
dataConn.Close();
}
}
}
```

} // End Class

Command Object

Хотя установка соединения с БД важна, но нам необходимо средство для получения данных. Здесь нам поможет Command object, используемый в ADO+ так же, как и в ADO

Ниже показаны примеры использования Command Object:

```
using System.Data;
using System.Data.SQL;
public class SQLConnect {
private SQLConnection dataConn = null;
private SQLDataReader reader = null;
public string openConnection(HttpResponse Response,String
dbConnectString,String cmdString) {
try {
dataConn = new SQLConnection(dbConnectString);
SQLCommand sqlCmd = new SQLCommand(cmdString,dataConn);
dataConn.Open();
sqlCmd.Execute(out reader);
Response.Write("<table><tr><td><b>ID</b></td>");
Response.Write("<td><b>Name</b></td></tr>");
while (reader.Read()) {
Response.Write("<tr>");
Response.Write("<td>" +reader["CategoryID"].ToString());
Response.Write("</td>");
Response.Write("<td>");
Response.Write(reader["CategoryName"].ToString());
Response.Write("</td>");
Response.Write("</tr>");
}
Response.Write("</table>");
return "<p>SQL Server Data Connection Opened";
}
catch (Exception e) {
return(e.ToString());
}

finally {
if (reader != null) {
reader.Close();
```

```
}
if (dataConn != null) {
dataConn.Close();
}
}
}
} // End Class
```

Command Object имеет множество конструкторов, принимающих различные параметры. В наших примерах мы передавали строку с SQL скриптом и Connection Object .

Пара слов о DataReader Object - в отличие от Dataset он использует forwardonly доступ и хранит только одну строку в памяти в каждый момент времени. Так как он читает построчно, то вынужден поддерживать соединение (очень похоже на обычный Recordset).

При инициализации объекта устанавливаются различные свойства по умолчанию. Например, поле CommandType устанавливается в CommandType.Text. Однако легко можно вызывать и сохраненные процедуры.

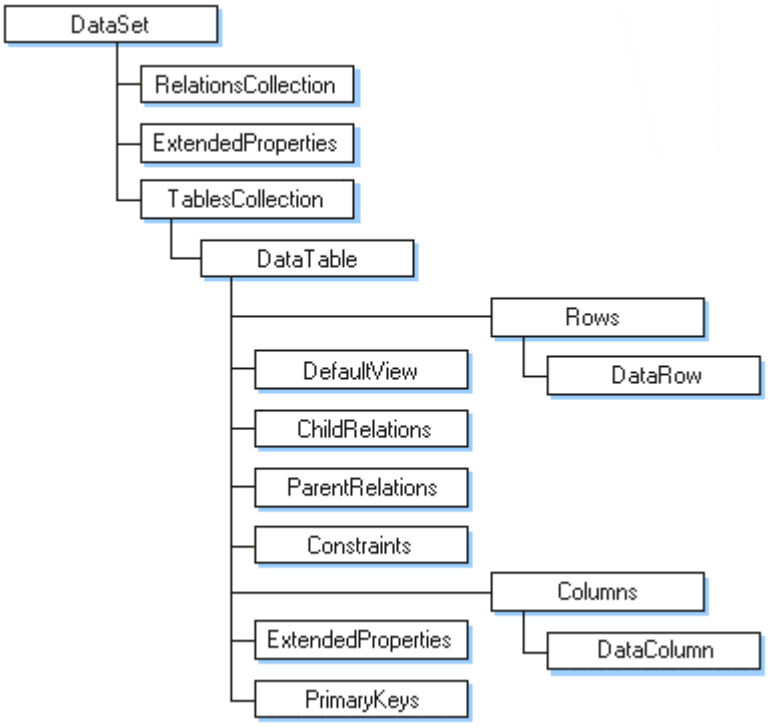
```
using System.Data;
using System.Data.SQL;
public class SQLConnect {
private SQLConnection dataConn = null;
private SQLDataReader reader = null;
public string openConnection(HttpResponse Response,String
dbConnectString,String cmdString) {
try {
dataConn = new SQLConnection(dbConnectString);
SQLCommand sqlCmd = new SQLCommand(cmdString,dataConn);
sqlCmd.CommandType = CommandType.StoredProcedure;
SQLParameter param = sqlCmd.Parameters.Add(new
SQLParameter("@CustomerID",SQLDataType.Char, 5));
param.Direction = ParameterDirection.Input;
sqlCmd.Parameters["@CustomerID"].Value = "ALFKI";
dataConn.Open();
sqlCmd.Execute(out reader);
Response.Write("<table><tr><td><b>Product Name</b></td>");
Response.Write("<td><b>Total</b></td></tr>");
while (reader.Read()) {
Response.Write("<tr>");

Response.Write("<td>"+reader["ProductName"].ToString());
Response.Write("</td>");
Response.Write("<td>"+reader["Total"].ToString());
Response.Write("</td></tr>");
}
Response.Write("</table>");
return "<p>SQL Server Data Connection Opened";
```

```
}
catch (Exception e) {
return(e.ToString());
}
finally {
if (reader != null) {
reader.Close();
}
if (dataConn != null) {
dataConn.Close();
}
}
}
} // End Class
```

DataSet Object

Основные особенности DataSet Object:
1)Dataset не взаимодействует напрямую с истчником данных. 2)Поддерживает отношения между различными таблицами.
3) Вы можете перемещаться по таблицам с помощью этих отношений
4) Для передачи данных используется XML.
5) Помимо расшрения передаваемых тиипов, это позволяет передаваемому объекту не испытывать воздействия брандмауэров.

Ниже приведена иллюстрация объектной модели Dataset:



Как видно, Dataset имеет более сложную структуру, нежели Recordset Object. Схема наглядно показывает возможности хранения нескольких таблиц и программного добавления связей между ними.

Но как Dataset заполняется первый раз? Очень просто! Для этого используется метод FillDataset() объекта DatasetCommand. Метод получает два параметра - имя Dataseta который должен быть заполнен и имя таблицы, котора будет добавлена в Dataset.

Код ниже иллюстрирует заполнение Dataset на примере SQL provider:

```
public class SQLConnect {
public string openConnection(HttpResponse Response,String
dbConnectString,String cmdString) {
try {
Dataset dsOrders = new Dataset();
SQLConnection dataConn = new
SQLConnection(dbConnectString);
SQLDatasetCommand dsCmd = new
SQLDatasetCommand(cmdString,dataConn);
dsCmd.FillDataset(dsOrders,"Orders");

// Loop through Dataset "Orders" table
Response.Write("<table><tr><td><b>Order ID</b></td></tr>");
foreach (DataRow Order in dsOrders.Tables["Orders"].Rows) {
Response.Write("<tr>");
Response.Write("<td>" + Order["OrderId"].ToString() + "</td>");
Response.Write("</tr>");
}
Response.Write("</table>");
return "<p>SQL Server Data Connection Opened";
}
catch (Exception e) {
return(e.ToString());
}
}
} //End Class
```

Код во многом похож на использованный при работе с DataReader, хотя соединение и не поддерживается во время перебора записей. Видно что после заполнения Dataset мы просматриваем все строчки таблицы.

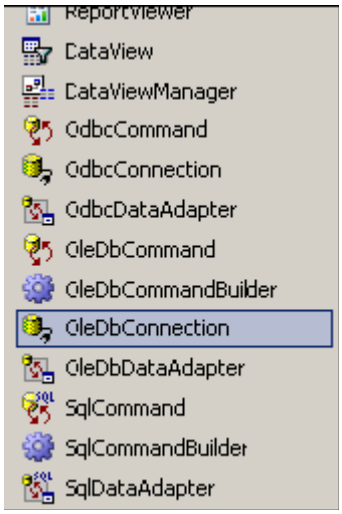
2. Практическая часть

Цель лабораторной работы – приобрести навыки разработки приложений ADO.NET и на примере данной лабораторной работы ознакомиться с основными принципами разработки таких приложений для платформы .NET.

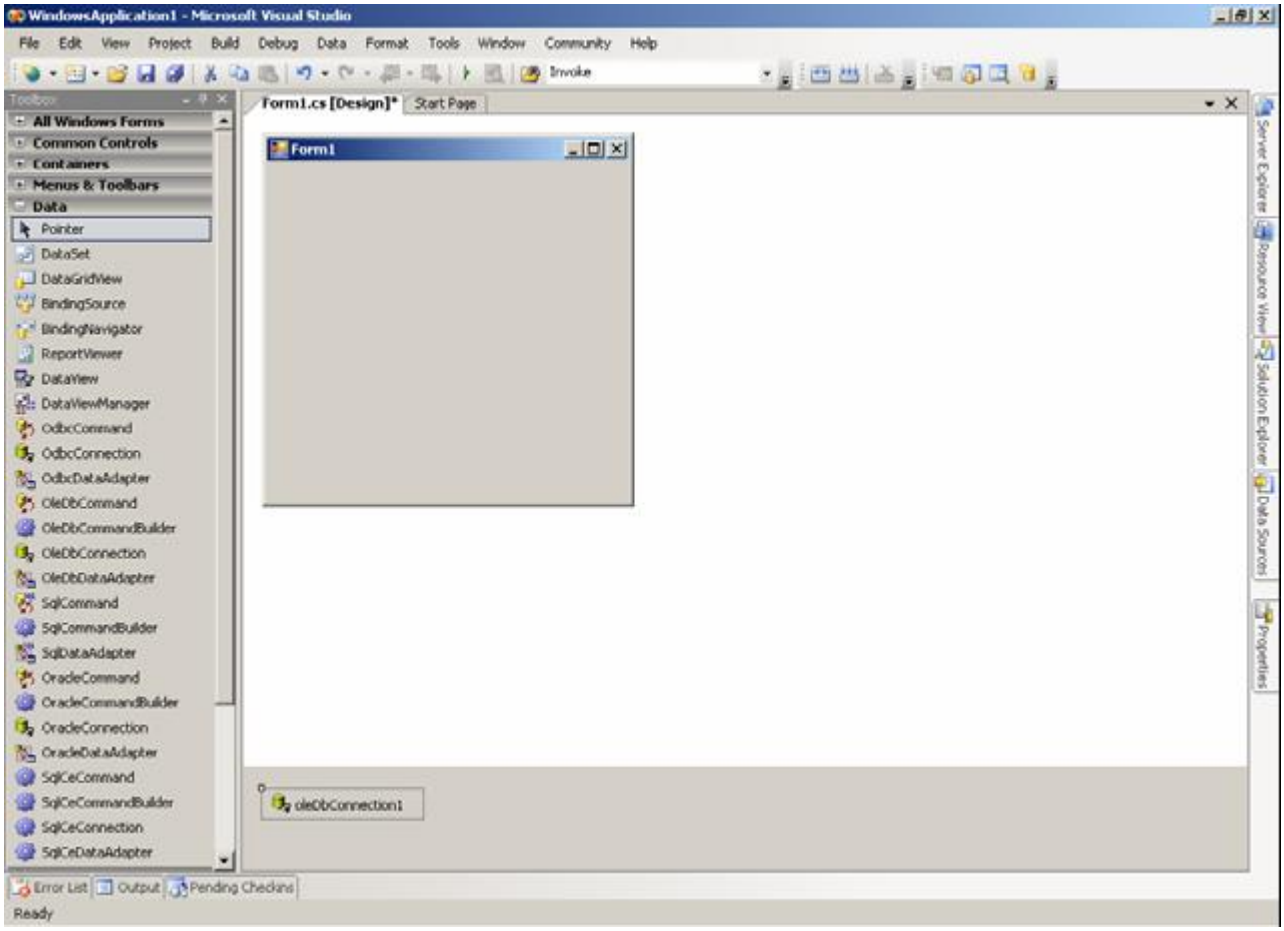
Приступим к написанию программы, реализующей взаимодействие с базой данных по средствам механизмов ADO.NET.

Создадим новое приложение Windows. Для этого в главном меню среды программирования Visual Studio 2005 выберем элемент меню **File->New Project…** В появившимся окне в поле выберем язык разработки Visual C#, а в качестве типа приложения – Windows Application. В поле Name зададим имя нашего приложения, а в поле Location – его месторасположение. После проведения вышеописанных операций подтвердим свои действия нажатием на кнопку OK и, тем самым завершим создание нового проекта.

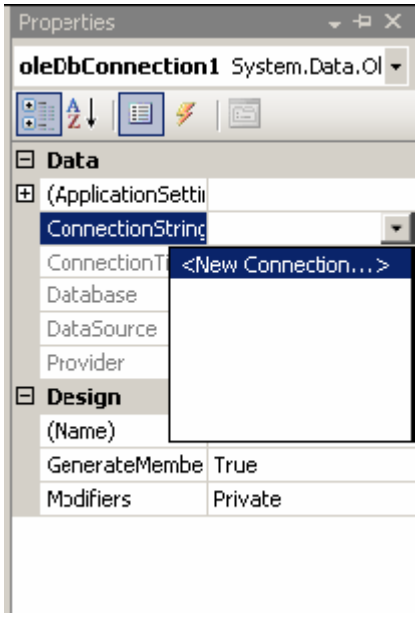
Установим соединение с нашей базой данных. В окне Toolbox выберем закладку Data и в ней элемент управления OleDbConnection и мышью перетащим данный элемент в форму приложения:



После этого в нижней части формы приложения появится объект с именем oleDbConnection1:



Этот объект соединения с источником данных OLE DB. Сделаем этот объект текущим в форме, для чего щелкнем по нему мышью. В окне Properties найдем свойство ConnectionString(строка соединения) и в поле данных нажмем кнопку выпадающего списка. В появившимся списке выберем строку New Connection:



В появившимся окне Add Connection укажем месторасположение нашей базы данных на жестком диске, а также логиин и пароль к нашей базе данных (при необходимости). Проверку соединения с базой данных можно осуществить с помощью кнопки Test Connection.

Строка соединения с базой данных будет выглядеть так:

```
Provider=Microsoft.Jet.OLEDB.4.0;DataSource=C:\comp2univer.mdb
```

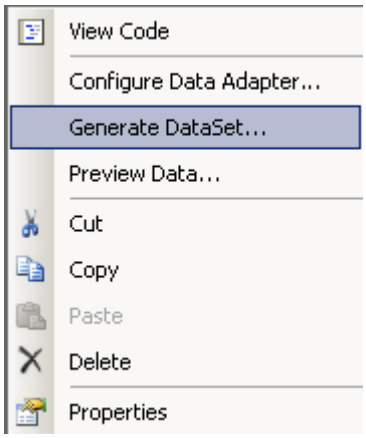
Далее создадим адаптер данных – промежуточное звено между набором данных (еще не созданным) и таблицей Kaf нашей базы данных. В окне Toolbox на закладке Data выберем объект OleDbDataAdapter и перетащим его в форму. На экране появится окно мастера настройки и создания адаптеров данных (Data Adapter Configuration Wizard). Нажмем кнопку Next. В следующем окне выберем радиокнопку Use SQL Statements и нажмем кнопку Next. В следующем окне (Generate SQL Statements) в поле данных введем следующую команду языка SQL:

```
SELECT kaf_id, kaf_name,comps
FROM Kafedra
```

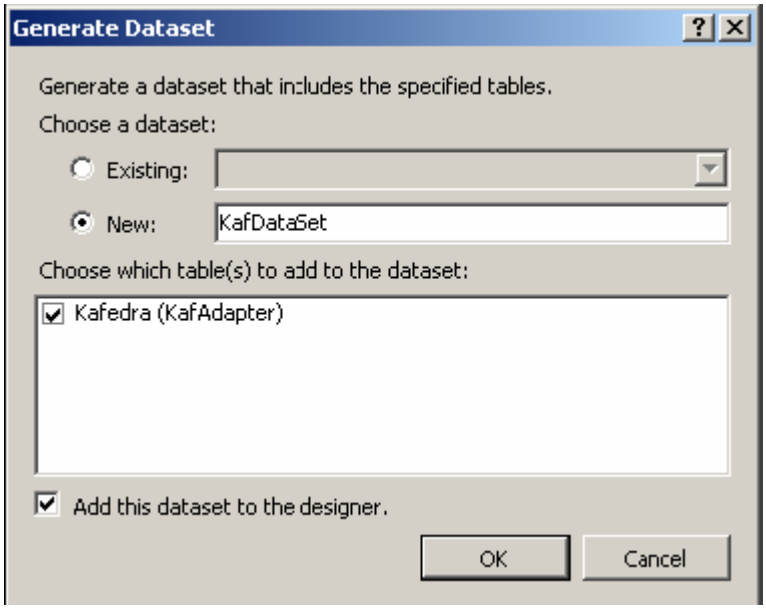
В следующем окне нажмем кнопку Finish. Наш адаптер данных сконфигурирован.

Выберем созданный нами адаптер данных oleDbDataAdaper1 и перейдем в окно его свойств. Изменим имя адаптера на KafAdapter.

Сгенерируем набор данных (DataSet). Выберем адаптер данных и правым щелчком мыши перейдем в контекстное меню, в котором выберем пункт Generate DataSet…



В появившимся окне установим значения полей, как показано на следующем рисунке:



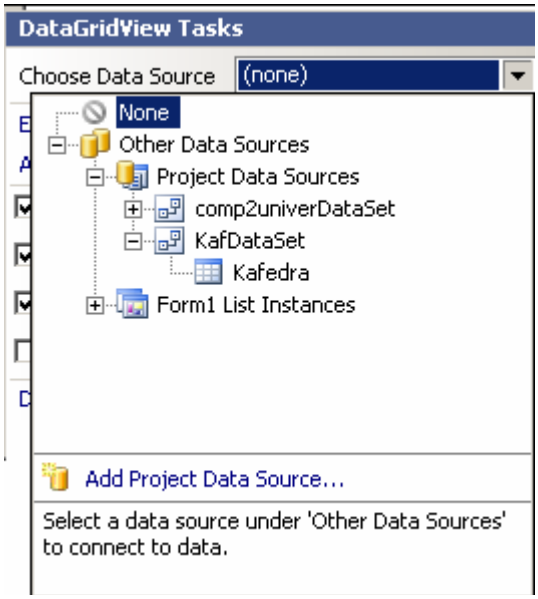
После этого в приложение будет добавлен класс набора данных с именем KafDataSet.

Зальем в нашу таблицу Kaf набора данных KafDataSet1 записи из одноименной таблицы в нашей базе данных. Для этого дважды щелкнем мышью по полю формы и в обработчике события загрузки формы зададим следующий код:

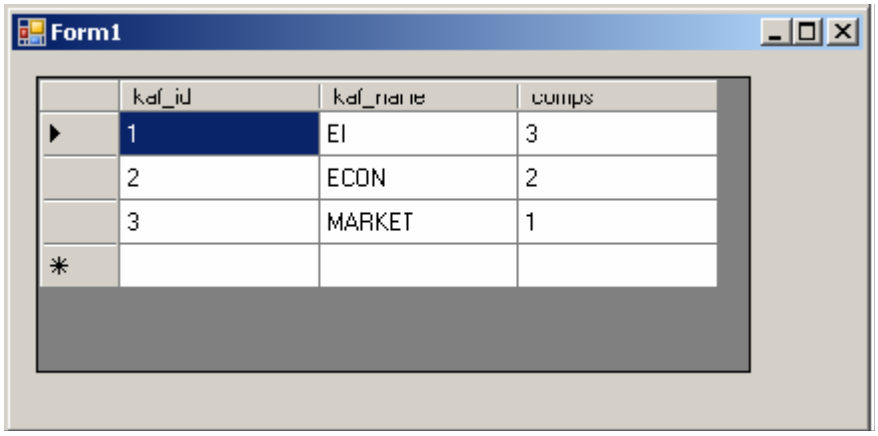
```
private void Form1_Load(object sender, EventArgs e)
{
KafAdapter.Fill(kafDataSet1);
     }
```

Далее перейдем в окно Toolbox на закладку Data, выберем элемент управления DataGridView и перетащим его на нашу форму. Сразу после

перенесения элемента на форму перед нами появится окно выбора набора данных с которым будет ассоциирован наш DataGridView:

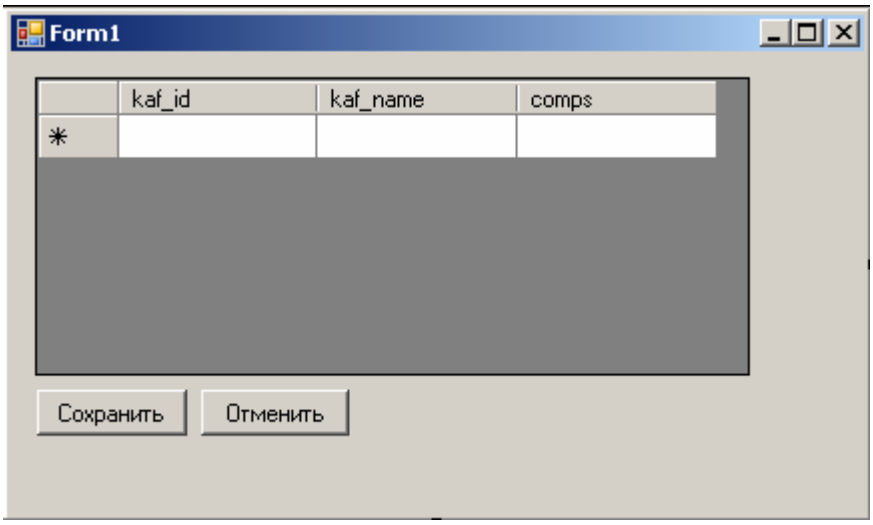


В списке адаптер данных выберем созданный нами набор данных KafDataSet, а в нем таблицу, с которой он ассоциирован – Kafedra. После завершения вышеописанной операции наша форма должна принять следующий вид:



Теперь на нашей форме отображаются данные из таблицы Kafedra нашей базы данных.

Добавим на нашу форму две кнопки для сохранения в базе данных изменений, вносимых через DataGridView1 и для отмены этих изменений. В окне Toolbox выберем закладку Common Controls, а в ней – элементы управления Button и перетащим две кнопки на нашу форму. В окне свойств кнопок присвоим кнопкам новые названия: btn_save и btn_cancel соответственно. После добавления кнопок форма должна принять следующий вид:



Напишем обработчик события для кнопки сохранения:

```
private void btn_save_Click(object sender, EventArgs e)
{
KafAdapter.Update(kafDataSet1, "Kafedra");
}
```

Прокомментируем данный обработчик: здесь с помощью метода `Update()` адаптера данных мы обновляем нашу таблицу базы данных данными из созданного нами объекта `DataSet`, передавая названия таблицы и набора данных в данную функцию в качестве параметров.

Напишем обработчик событий для отмены изменений, вносимых в таблицу базы данных:

```
private void btn_cancel_Click(object sender, EventArgs e)
{
kafDataSet1.Kafedra.RejectChanges();
}
```

В данном фрагменте кода мы с помощью метода `RejectChanges()` производим откат изменений, вносимых в таблицу Kafedra набора данных kafDataSet1.

В итоге у нас получилось приложение, отображающее данные из выбранной нами таблицы, предусматривающее добавление новых даннх в таблицу базы данных, редактирование,удаление и сортировку данных, а также сохранение изменеий в базе данных и откат нежелетельных изменений.

Индивидуальные задания

1. Разработать приложение учета продажи авиабилетов в аэропорту.
2. Разработать приложение учета продаж компьютеров.

3. Разработать приложение учета посадок деревьев в районах города Минска.
4. Разработать приложение учета продажи лотерейных билетов различных лотерей.
5. Разработать приложение учета отгрузки стройматериалов.
6. Разработать приложение учета закупок оргтехники подразделениями университета.
7. Разработать приложение учета закупки канцелярских товаров кафедрами.
8. Разработать приложение учета продажи компакт-дисков.
9. Разработать приложение учета сдачи студентами лабораторных работ.
10. Разработать приложение учета посещаемости лекций студентами.

Вопросы для самопроверки

1. Особенности архитектуры ADO.NET .
2. Отсоединенные объекты.
3. Для чего предназначен объект DataTable?
4. Для чего предназначены методы Fill и Update и что они получают в качестве параметров?
5. Для чего предназначен объект Parameter?
6. Для чего предназначен объект DataSet?
7. Для чего предназначен объект Connection?
8. Каких поставщиков данных можно использовать в ADO.NET?
9. В чем заключается различие между поставщиками данных OLE DB и ODBC?
10. С помощью какого элемента управления в форме отображаются данные, хранящиеся в таблице базы данных?