Работа с БД

## Технология ADO.Net

ADO.NET (Active Data Objects) — это наименование набора классов, которые используются с С# и .NET Framework для доступа к данным в реляционном, таблично-ориентированном формате. Это включает реляционные базы данных, такие как Microsoft SQL Server и Microsoft Access, наряду с другими базами данных и даже не реляционными источниками данных. Технология ADO.NET интегрирована в .NET Framework и спроектирована для использования с любым языком .NET, в особенности — С#.

История развития (для понимания)

1. ODBC (Open Database Connectivity — открытый интерфейс взаимодействия с базами данных). Когда появились первые системы управления данными, каждая система имела собственную библиотеку функций для прямого взаимодействовали с базами данных. Однако программистам необходимо было быть знакомыми с разными библиотеками для каждой базы данных, с которой им приходилось работать. Кроме того, если компания заменяла используемую базу данных, все ее приложения нужно было полностью переписывать. ODBC предоставил полный набор функций, которые разработчики могли применять с любыми системами баз данных. Разработчикам теперь достаточно было знать, как пользоваться одним набором функций (функциями ODBC); и если компания заменяла свою систему управления базами данных, все, что приходилось изменить — это код подключения к базе данных.
2. Но ODBC не позволял обращаться к данным других типов, которые не размещались в строгом порядке строк и столбцов, а порой вообще не имели регулярной структуры. Технология **OLE DB** работает способом, аналогичным ODBC, предоставляя слой абстракции между базой данных и приложениями, которым нужен доступ к данным. Клиентское приложение взаимодействует с источником данных, которым может быть традиционная база данных или любое другое место хранения данных, через поставщика OLE DB для данного источника данных. Данные из любого источника предоставляются приложению в табличном формате — как будто они поступили из базы данных. Вдобавок, поскольку OLE DB обеспечивает доступ к данным, предоставляемым существующими драйверами ODBC, он может быть использован и для доступа к базам, поддерживаемым ODBC.
3. ADO (ActiveX Data Objects) — это просто тонкий слой, располагающийся поверх OLE DB, который позволяет программам, написанным высокоуровневых языках, обращаться к данным OLE DB.
4. Технология ADO.NET появилась вместе с .NET Framework 1.0, и эволюционировала вместе с различными версиями .NET с добавлением поставщиков данных, постепенно при этом совершенствуясь.

ADO.NET спроектирован для работы в многоуровневых приложениях. Сегодня это — наиболее распространенная архитектура для бизнес-приложений и приложений электронной коммерции. В многоуровневой архитектуре разные части прикладной логики распределены по разным уровням или слоям, и взаимодействуют только с соседними слоями.

Одним из наиболее распространенных подходов является трехуровневая модель, которая состоит из:

* **Слоя данных.** Содержит базу данных и код доступа к данным.
* **Бизнес-слоя.** Содержит бизнес-логику, который определяет уникальную ункциональность приложения, и абстрагирует ее от других слоев. Этот слой иногда называют средним слоем.
* **Слоя представления.** Обеспечивает пользовательский интерфейс и контроль потока управления приложения, наряду с такими вещами, как проверка пользовательского ввода.

## Обзор классов и объектов ADO.NET

Классы ADO.NET подразделяются на классы объектов-поставщиков данных .NET и классы объектов потребителей.

* **Объекты-поставщики** специфичны для каждого типа источников данных —действительное чтение и запись информации в источники данных выполняется специфичными для поставщика объектами.
* **Объекты-потребители** — это те, что применяются для доступа и манипуляции данными после того, как они прочитаны в память.

Объекты-поставщики требуют наличия активного соединения. Они используются для чтения данных, а затем, в зависимости от потребностей, можно работать с данными в памяти через объекты-потребители. Можно также обновлять данные в источнике данных, используя объекты поставщика для записи изменений обратно в источник данных.

Таким образом, объекты-потребители оперируют в автономном (отключенном) режиме можно работать с данными в памяти даже тогда, когда соединение с базой данных закрыто.

### Объекты-поставщики

1. Объект **Connection** — обычно первый объект, который используется, еще до применения большинства прочих объектов ADO.NET; он предоставляет базовое соединение с вашим источником данных.
2. Объект **Command** используется для передачи команды источнику данных.
3. Объект **CommandBuilder** используется для построения команд SQL для модификации данных из объектов на основе однотабличного запроса.
4. Объект **DataReader** читает однонаправленные, доступные только для чтения потоки данных.
5. Объект **DataAdapter** выполняет различные операции, специфичные для источника данных, включая обновление измененных данных, наполнение объектов DataSet и другие операции.

### Объекты-потребители

Объекты-потребители не связаны ни с каким определенным поставщиком данных .NET и находятся в пространстве имен System.Data.

1. Объект **DataSet** представляет набор связанных таблиц, воспринимаемых как единое целое в рамках приложения. С помощью этого объекта можно быстро получать все необходимые данные из каждой таблицы, рассматривать и изменять их, будучи отключенным от сервера, а затем обновлять сервер изменениями в одной эффективной операции. DataSet имеет средства, которые позволяют обращаться к объектам более низкого уровня, представляющим индивидуальные таблицы и отношения – **DataTable и DataRelation.**
2. Объект **DataTable** представляет одну из таблиц в DataSet и имеет средства, позволяющие вам обращаться к его строкам и столбцам.
3. Объект **DataColumn.** Представляет один столбец в таблице.
4. Объект **DataRow**. Представляет одну строку таблицы
5. Объект **DataRelation** представляет отношение между двумя таблицами, установленное через общий столбец. Например, таблица Orders может иметь столбец СиstomerID, идентифицирующий заказчика, разместившего заказ. Объект DataRelation может быть создан как представляющий отношение между Customers и Orders через разделенный столбец Customer ID (внешний ключ).

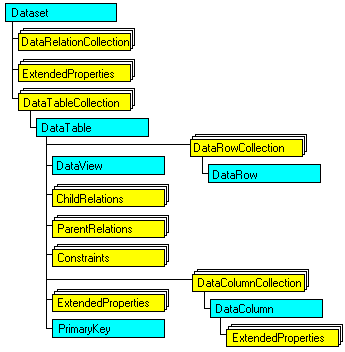


Рисунок Архитектура ADO.Net

## Создание БД

Создание БД выполняется в среде СУБД MS SQL Server Management Studo.

При открытии программы MS SQL Server Management Studio появляется форма, представленная на рисунке 2, на которой необходимо подтвердить соединение, нажав на кнопку «Connect».



Рисунок Окно для подтверждения соединения с СУБД

Далее средствами СУБД создаются таблицы БД и связи между таблицами (рис.3).

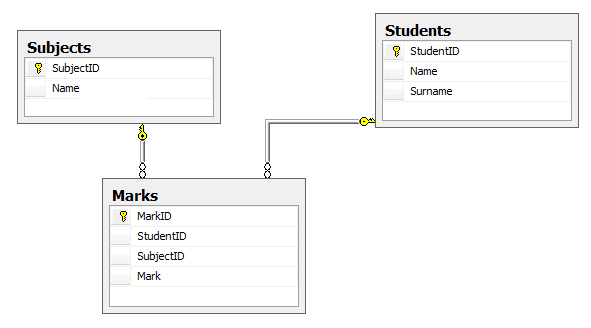


Рисунок 3 Схема базы данных в среде MS SQL Server Management Studio

## Создание проекта приложения

Проект приложения создается в среде MS Visual Studio как проект Windows Forms Application.

Для связи с СУБД необходимо разместить на форме элемент управления sqlConnection. Он предоставляет собой открытое подключение к базе данных SQL Server (не используется Integrated Security = true, и объект SqlCredential, содержащий идентификатор пользователя и пароль).

В свойствах элемента управления sqlConnection необходимо выбрать свойство Connection String и в нем выбрать из списка нужную БД. После этого выбранная БД станет доступной в окне Server Explorer (рис. 4).

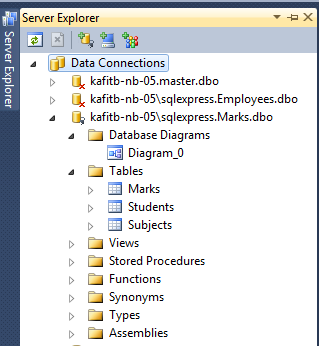


Рисунок 4 Окно Server Explorer с выбранной БД

## Создание формы

На форме приложения необходимо разместить элементы управления для вывода данных из таблиц. Вывод данных будет выполняться с помощью элемента DataGrid. Выбор нужной таблицы будет выполняться с помощью меню (рис . 5).

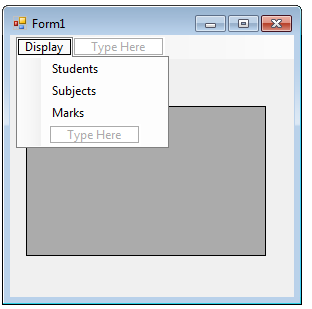


Рисунок Форма для отображения данных БД

## Обмен данными между формой и приложением

Для обмена информацией между приложением и БД используются следующие объекты:

* **DataSe**t — объект потребителя (приложение), центральный объект ADO.NET; все операции используют его. DataSet содержит набор объектов **DataTable**, представляющих таблицы базы данных, с которыми вы работаете. Объект **DataTable** может быть всего один.
* Каждый объект **DataTable** имеет дочерние объекты **DataRow** и **DataColumn**, представляющие строки и столбцы таблицы базы данных.
* **SqlDataAdapter** –  объект поставщика, используется для взаимодействия с DataSet для извлечения и сохранения данных. Метод  Fill изменяет данные в DataSet для соответствия их источнику данных, метод Update, изменяет данные в источнике данных данных для их соответствия DataSet, с помощью выполняемых над источником SQL инструкций.
* **SqlCommand** – объект поставщика, используется для передачи команды источнику данных (БД).
* **sqlConnection** – объект, предоставляющий подключение к источнику данных (БД).

Создадим управляющий класс ControlDataBase, в который поместим статические методы для выполнения обмена данными с БД.

### Просмотр данных (Read)

Для чтения данных из таблицы напишем метод:

public static DataTable DisplayTable(string tableName, SqlConnection sqlConnection)

{

SqlCommand command = sqlConnection.CreateCommand();

command.CommandText = "SELECT \* FROM " + tableName;

DataTable dt = new DataTable();

SqlDataAdapter adapter = new SqlDataAdapter(command);

try

{

adapter.Fill(dt);

}

catch

{

MessageBox.Show("Connection failed!", "Error!");

}

finally

{

sqlConnection1.Close();

}

return dt;

}

Рассмотрим операторы метода.

* SqlCommand command = sqlConnection.CreateCommand(); – создает пустую команду, связанную с подключением sqlConnection.
* command.CommandText = "SELECT \* FROM " + tableName; – задает SQL инструкцию, выполняемую для источника данных.
* DataTable dt = new DataTable(); – создает новый объект DataTable.
* SqlDataAdapter adapter = new SqlDataAdapter(command); – создает новый объект SqlDataAdapter и инициализирует его командой command. SqlDataAdapter представляет набор выполняемых над данными команд и подключения базы данных, которые используется для заполнения DataSet и обновления базы данных SQL Server.
* adapter.Fill(dt); - добавляет или обновляет строки в объекте DataSet в соответствии с командой command.
* sqlConnection.Close(); – закрывает подключение.

Для выполнения этого метода будут использоваться обработчики событий меню.

//обработчик для вывода таблицы Students

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

{

dataGridView1.DataSource =ControlDataBase.DisplayTable ("Students", sqlConnection1);

}

### Добавление записей в БД (Create)

Для добавления записей добавим диалоговые окна для ввода информации в каждую таблицу. На рис 6. представлена форма для ввода данных в таблицу Students.

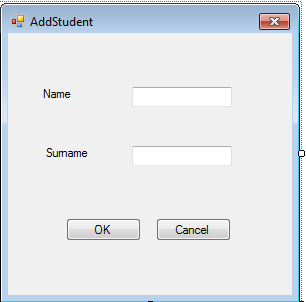


Рисунок . Форма для ввода данных в таблицу Students.

При вводе данных в таблицу необходимо предусмотреть события в управляющих элементах (Textbox) для проверки правильности ввода текста в поля Name и Surname.

Кроме того, в форму AddStudent необходимо добавить свойства для обмена информацией, полученной из полей Name и Surname с другими формами.

public string \_Name

{

get { return textBox1.Text; }

}

public string \_Surname

{

get { return textBox2.Text; }

}

Для того, чтобы добавить запись, введенную с помощью формы AddStudent в БД необходимо создать соответствующий SQL-запрос и с реализовать его помощью функции InsertStudent класса ControlDataBase

public static void InsertStudent(string \_name, string \_surname, SqlConnection sqlConnection)

{

SqlCommand command = sqlConnection.CreateCommand();

command.CommandText = "INSERT INTO Students (name, surname) VALUES (@name, @surname)";

command.Parameters.Add("name", SqlDbType.NVarChar).Value = \_name;

command.Parameters.Add("surname", SqlDbType.NVarChar).Value = \_surname;

sqlConnection.Open();

try

{

command.ExecuteNonQuery();

}

finally

{

sqlConnection.Close();

}

}

В функции используются операторы:

* SqlCommand command = sqlConnection.CreateCommand(); – создает пустую команду, связанную с подключением sqlConnection.
* command.CommandText = "INSERT INTO Students (name, surname) VALUES (@name, @surname)"; – задает SQL инструкцию, выполняемую для источника данных. Инструкция содержит SQL-запрос, который добавляет строку в таблицу БД.

В запросе:

список (name, surname) – список полей, которые добавляются в таблицу БД,

(@name, @surname) – список значений, которые должны быть подставлены в указанные поля, @значение – именованный параметр.

command.Parameters.Add("name", SqlDbType.NVarChar).Value = name;

command.Parameters.Add("surname", SqlDbType.NVarChar).Value = surname; – параметры используются для передачи значений в выражения SQL, обеспечивая проверку типов. В данном примере выполняется проверка преобразования переменных \_name и \_surname типа string в данные типа NVarChar.

* sqlConnection.Open(); — открывает подключение к базе данных со значениями свойств, определяемыми объектом ConnectionString.
* command.ExecuteNonQuery(); – метод ExecuteNonQuery позволяет вносить изменения в базу данных, не используя DataSet, с помощью операторов UPDATE, INSERT или DELETE.

Данная функция будет вызываться в обработчике соответствующего пункта меню:

private void studentsToolStripMenuItem1\_Click(object sender, EventArgs e)

{

AddStudent addStudentForm = new AddStudent();

addStudentForm.ShowDialog();

if (addStudentForm.DialogResult==DialogResult.OK)

{

ControlDataBase.InsertStudent(addStudentForm.\_Name, addStudentForm.\_Surname, sqlConnection1);

}

}

### Удаление записей из таблицы (Delete)

Работу пользователя при удалении записей из таблицы можно организовать следующим образом:

1. Открыть таблицу, из которой будут удаляться записи с помощью соответствующей команды меню, например, Display -> Students.
2. Выделить нужную запись, нажав левой клавишей мыши на строку с записью.
3. Выбрать пункт меню «Delete».

Функция для удаления строки из таблицы БД будет выглядеть следующим образом:

public static void DeleteRecord(int deleteID, string currentTable, string currentID, SqlConnection sqlConnection)

{

SqlCommand command = sqlConnection.CreateCommand();

string strCommand = string.Format("DELETE FROM {0} WHERE {1} = '{2}'", currentTable, currentID, deleteID);

command.CommandText = strCommand;

sqlConnection.Open();

try

{

command.ExecuteNonQuery();

}

finally

{

sqlConnection.Close();

}

}

Параметрами функции являются:

int deleteID – номер удаляемой записи в таблице,

string currentTable – название текущей таблицы,

string currentID – название ключевого столбца таблицы в БД,

SqlConnection sqlConnection – подключение к БД.

Для формирования SQL инструкции используется метод string.Format, который создает команду DELETE и подставляет в нее необходимые параметры.

Для того, чтобы определить название таблицы и ее ключевой столбец, в главную форму приложения необходимо добавить соответствующие переменные и присвоить им требуемые значения в обработчике команды меню для просмотра соответствующей таблицы.

string currentTable = "";

string currentID = "";

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

{

dataGridView1.DataSource = ControlDataBase.DisplayTable ("Students", sqlConnection1);

currentTable = "Students";

currentID = "StudentID";

}

В обработчике команды меню Delete нужно определить номер текущей строки и вызвать функцию DeleteRecord().

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

{

int deleteID = (int)dataGridView1[0, dataGridView1.CurrentRow.Index].Value;

ControlDataBase.DeleteRecord(deleteID, currentTable,currentID, sqlConnection1);

}

### Редактирование записей в таблице (Update)

Работу пользователя при редактировании записей в таблице можно организовать также как и удаление:

1. Открыть таблицу, в которой будут редактироваться записи с помощью соответствующей команды меню, например, Display -> Students.
2. Выделить нужную запись, нажав левой клавишей мыши на строку с записью.
3. Выбрать пункт меню «Update».

Функция для редактирования строки в таблицы БД будет выглядеть следующим образом:

public static void UpdateStudent(int updateID, string \_name, string \_surname, SqlConnection sqlConnection)

{

SqlCommand command = sqlConnection.CreateCommand();

string strcom = string.Format("UPDATE Students SET name=(@name), surname=(@surname) WHERE StudentID='{0}'", updateID);

command.CommandText = strcom;

command.Parameters.Add("name", SqlDbType.NVarChar).Value = \_name;

command.Parameters.Add("surname", SqlDbType.NVarChar).Value = \_surname;

sqlConnection.Open();

try

{

command.ExecuteNonQuery();

}

finally

{

sqlConnection.Close();

}

}

Параметрами функции являются:

int updateID – номер редактируемой записи в таблице,

string \_name – новое значение поля name,

string \_surname – новое значение поля surname,

SqlConnection sqlConnection – подключение к БД.

В обработчике команды меню Update нужно определить номер текущей строки и вызвать функцию UpdateStudent().

Для ввода новых значений для обновляемой записи используется форма AddStudent.

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

{

switch (currentTable)

{

case "Students":

{

int updateID = (int)dataGridView1[0, dataGridView1.CurrentRow.Index].Value;

AddStudent dlg = new AddStudent();

dlg.ShowDialog();

if (dlg.DialogResult == DialogResult.OK)

{

ControlDataBase.UpdateStudent(updateID, dlg.\_Name, dlg.\_Surname, sqlConnection1);

}

else

{

return;

}

break;

}

}

}

### Формирование запроса к БД

Для получения информации из БД используется тот же механизм, что и при чтении данных из БД.

В класс ControlDataBase добавим метод

public static DataTable TestQuery(SqlConnection sqlConnection)

{

SqlCommand command = sqlConnection.CreateCommand();

command.CommandText = "SELECT Students.Surname FROM Students";//команда может быть другая

DataTable dt = new DataTable();

SqlDataAdapter adapter = new SqlDataAdapter(command);

sqlConnection.Open();

try

{

adapter.Fill(dt);

}

catch

{

MessageBox.Show("Connection failed!", "Error");

}

finally

{

sqlConnection.Close();

}

return dt;

}

В главную форму добавим команду Query и ее обработчик.

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

{

dataGridView1.DataSource = ControlDataBase.TestQuery(sqlConnection1);

}