ADO.NET

ADO.NET

- ADO .NET (ActiveX Data Objects .NET) набор классов, реализующих программные интерфейсы для облегчения подключения к базам данных из приложения

Три стороны ADO.NET

MERA

- Подключенный уровень (Connected layer) :
  - ✔ Явное подключение к хранилищу данных
- Автономный уровень (disconnected layer):
  - ✔ Работа с копией данных из хранилища. Подключение открывается только для изменения данных
- Entity Framework:
  - ✔ Сокрытие низкоуровневых деталей работы с базой данных

Реляционные базы данных. Основные понятия

- ✔ столбец (поле, атрибут)
- ✔ строка (запись, кортеж)
- ✔ таблица
- ✔ первичный ключ таблицы (primary key)
- ✔ внешний ключ таблицы (foreign key)

Доступ к данным (упрощенная схема)

Приложение

Поставщик данных (Data Provider)

База данных

Поставщик данных (Data Provider)

MERA

Набор типов классов поставщика данных:

- ✔ Connection – обеспечивает подключение к БД;
- ✔ Command – для управления БД; позволяет выполнять команды SQL или хранимые процедуры;
- ✔ DataReader – предоставляет доступный только для однонаправленного чтения набор записей, подключенный к БД;
- ✔ DataAdapter – заполняет отсоединенный объект DataSet или DataTable и обновляет его содержимое.
- ✔ Parameter - именованный параметр в параметризованном запросе

Имеющиеся в .Net поставщики данных

- SQL Server - предоставляет оптимизированный доступ к базам данных SQL Server (версии 7.0 и выше)
- OLE DB - предоставляет доступ к любому источнику данных, который имеет драйвер OLE DB. Это включает базы данных SQL Server версий, предшествующих 7.0
- Oracle – устарел. Используйте DP.NET (Oracle Data Provider для .NET) производства Oracle, который доступен на веб-сайте http://www.oracle.com
- *ODBC* - предоставляет доступ к любому источнику данных, имеющему драйвер ODBC

Работа в подключенном режиме

Подключение к базе данных. Строка соединения

```
// Создание открытого подключения
using (SqlConnection cn = new SqlConnection()
{
   cn.ConnectionString = @"Data Source=n102933\SQLEXPRESS2012;Initial
   Catalog=AutoLot;Integrated Security=SSPI;Pooling=False";
   cn.Open();
   // Работа с базой данных
   cn.Close();
 }
```

Подключение к базе данных. Строка соединения

```
// Создание строки подключения с помощью объекта построителя
SqlConnectionStringBuilder connect = new SqlConnectionStringBuilder();
connect.InitialCatalog = "Autolot";
connect.DataSource = @"(local)\SQLEXPRESS";
connect.ConnectTimeout = 30;
connect.IntegratedSecurity = true;
```

Элемент <connectionStrings>

```
<connectionStrings>
    <add name="AutoLotSqlProvider"
connectionString="Data Source=MICROSOF-1EA29E\SQLEXPRESS;
Initial Catalog=AutoLot;Integrated Security=True;Pooling=False"/>
    <add name="AutoLotOleDbProvider“
connectionString="Provider=SQLOLEDB;Data
Source=MICROSOF1EA29E\SQLEXPRESS;
Initial Catalog=AutoLot;Integrated Security=True;Pooling=False"/>

</connectionStrings>
```

Элемент <connectionStrings>

```
 // Получение строки подключения из *.config
string cnStr =
ConfigurationManager.ConnectionStrings["AutoLotSqlProvider"].
   ConnectionString;
```

Подключение к базе данных. Класс Connection

MERA

Выполняет обмен данными между базой данных и приложением

Свойства:

- ✔ ConnectionString
- ✔ ConnectionTimeout
- ✔ DataBase
- ✔ State

Методы:

- ✔ Open() – открытие соединения
- ✔ Close() – закрытие соединения
- ✔ BeginTransaction()

Модель работы в подключенном режиме

```
using (SqlConnection cn = new SqlConnection())
{
   cn.ConnectionString = connect.ConnectionString;
   try
   {
    //Открыть подключение
    cn.Open();
   }
   catch (SqlException ex)
   {
   }
   finally
   {
    // Гарантировать освобождение подключения
    cn.Close();
   }
}
```

Запросы (Queries)

- Запросы, которые не возвращают записей:

```
UPDATE Customers Set CompanyName = 'NewHappyName'
                 WHERE CustomerID = '007‘
```

```
CREATE TABLE myTable ( Field1 int NOT NULL Field2 varchar() )
```

- Запросы, возвращающие значения из базы данных

```
SELECT CustomerID,
    CompanyName,
    ContactName
FROM Customers
WHERE Phone = '222-3322'
```

Класс Command

Класс Command позволяет выполнить запросы к базе данных (выборку, обновление, дополнение, удаление и т. д.).

Класс Command

MERA

Свойства:

✔CommandType:

- CommandType.Text (по умолчанию)- операторы SQL ;
- CommandType.TableDirect – работа с конкретной таблицей;
- CommandType.StoredProcedure – вызов хранимой в БД процедуры.

✔CommandText содержит:

- текст оператора SQL (для типа CommandType.Text);
- имя таблицы (для CommandType.TableDirect);
- имя хранимой процедуры с параметрами (для CommandType.StoredProcedure);

✔Connection – ссылка на открытое соединение (объект Connection);

✔Parameters – коллекция параметров запроса

Создание экземпляра Command

```
//Открыть подключение
cn.Open();

// Создание объекта команды с помощью конструктора
string strSQL = "Select * From Inventory";
SqlCommand myCommand = new SqlCommand(strSQL, cn);

// Создание еще одного объекта команды с помощью свойств
SqlCommand testCommand = new SqlCommand();
testCommand.Connection = cn;
testCommand.CommandText = strSQL;

//Открыть подключение
cn.Close();
```

Основные методы выполнения Command

- ✔ ExecuteReader() - выполняет оператор SELECT, создает и возвращает ссылку на объект DataReader который содержит результат выполнения запроса.
- ✔ ExecuteNonQuery() - выполняет операторы INSERT, DELETE, UPDATE на языке SQL (возвращает количество обработанных записей)
- ✔ ExecuteScalar() – возвращает первую строку первого столбца в результирующем наборе (используя функции COUNT, AVG, MIN, MAX, SUM);

Метод ExecuteNonQuery(). Пример

```
string strSQL = "UPDATE Customers SET LastName =
  'Johnson' WHERE LastName = 'Walton'";
SqlCommand myCommand = new SqlCommand(strSQL, cn);
int i = myCommand.ExecuteNonQuery();
```

Метод ExecuteReader(). Пример

```
string strSQL = "SELECT * FROM Inventory";
SqlCommand myCommand = new SqlCommand(strSQL, cn);
SqlDataReader dr = myCommand.ExecuteReader();
while (dr.Read())
{
  Console.WriteLine("ID: {0} Car Pet Name: {1}",
  dr[0], dr[3]);
}
```

Задание параметров с помощью типа DbParameter

В SQL запросе в Command.Text можно задавать переменные – параметры

Параметры позволяют менять SQL запрос без переписывания его текста

Параметры используются при вызове хранимой процедуры для передачи входных данных и получения результатов

Задание параметров с помощью типа DbParameter

- Для Odbc поля параметра задаются символами «?»:

```
select EmpId, Title, FirstName, LastName
from Employees where (FirstName = ?, LastName = ? )
```

- Для OleDbCommand и SqlCommand используется именованные поля параметров - @Xxxxx:

```
select EmpId, Title, FirstName, LastName
from Employees
where (FirstName = @First, LastName = @Last )
```

Добавление параметров

MERA

```
string strSQL = string.Format("Insert Into Inventory" +
"(CarID, Make, Color, PetName) Values(@CarId, @Make, @Color, @PetName)");

SqlCommand testCommand = new SqlCommand(strSQL, cn);
SqlParameter param = new SqlParameter();
param.ParameterName = "@CarID";
param.Value = id;
param.SqlDbType = SqlDbType.Int;
testCommand.Parameters.Add(param);
```

Добавление параметров

```
testCommand.Parameters.AddWithValue("@CarId", 1212);
testCommand.Parameters.AddWithValue("@Make", "Skoda");
testCommand.Parameters.AddWithValue("@Color", "Grey");
testCommand.Parameters.AddWithValue("@PetName", "Skoda");
```

Хранимые процедуры (Stored Procedures)

Хранимая процедура (stored procedure) — это именованный блок SQL-кода, хранимый в базе данных

- ✔ В одной процедуре можно сгруппировать несколько запросов;
- ✔ В одной процедуре можно сослаться на другие сохраненные процедуры, что упрощает процедуры обращения к БД;
- ✔ Выполняются быстрее, чем индивидуальные предложения SQL.

Пример вызова хранимой процедуры

```
using (SqlCommand cmd = new SqlCommand("GetPetName", cn))
{
    cmd.CommandType = CommandType.StoredProcedure;
    // Входной параметр.
    SqlParameter param = new SqlParameter();
    param.ParameterName = "@carID";
    param.SqlDbType = SqlDbType.Int;
    param.Value = 1212;
    param.Direction = ParameterDirection.Input;
    cmd.Parameters.Add(param);
    SqlDataReader dr = cmd.ExecuteReader();
    while (dr.Read())
    {
      Console.WriteLine("Car Pet Name: {0}", dr[0]);
    }
}
```

Пример вызова хранимой процедуры

```
USE [AutoLot]
GO
/****** Object:  StoredProcedure [dbo].[GetPetName]    Script Date:
   11/13/2015 11:21:16 AM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[GetPetName]
@carID int
AS
SELECT PetName from Inventory
where CarID = @carID
```

Транзакции

Транзакция — это набор операций в базе данных, которые должны быть либо все выполнены, либо все не выполнены

Транзакции. Пример

MERA

```
 // Выборка имени по идентификатору клиента
string fName = string.Empty;
string lName = string.Empty;
SqlCommand cmdSelect =
new SqlCommand(string.Format("Select * from Customers where CustID
   = {0}", custId), cn);
using (SqlDataReader dr = cmdSelect.ExecuteReader())
{
    if (dr.HasRows)
    {
        dr.Read();
        fName = (string)dr["FirstName"];
        lName = (string)dr["LastName"];
    }
    else return;
}
```

Транзакции. Пример

```
// Создание объектов команд для каждого шага операции.
SqlCommand cmdRemove = new SqlCommand(
string.Format("Delete from Customers where CustID = {0}", custId), cn);
SqlCommand cmdInsert = new SqlCommand(string.Format("Insert Into
   CreditRisks" + "(CustID, FirstName, LastName) Values" +
    "({0}, '{1}', '{2}')", custId, fName, lName), cn);
```

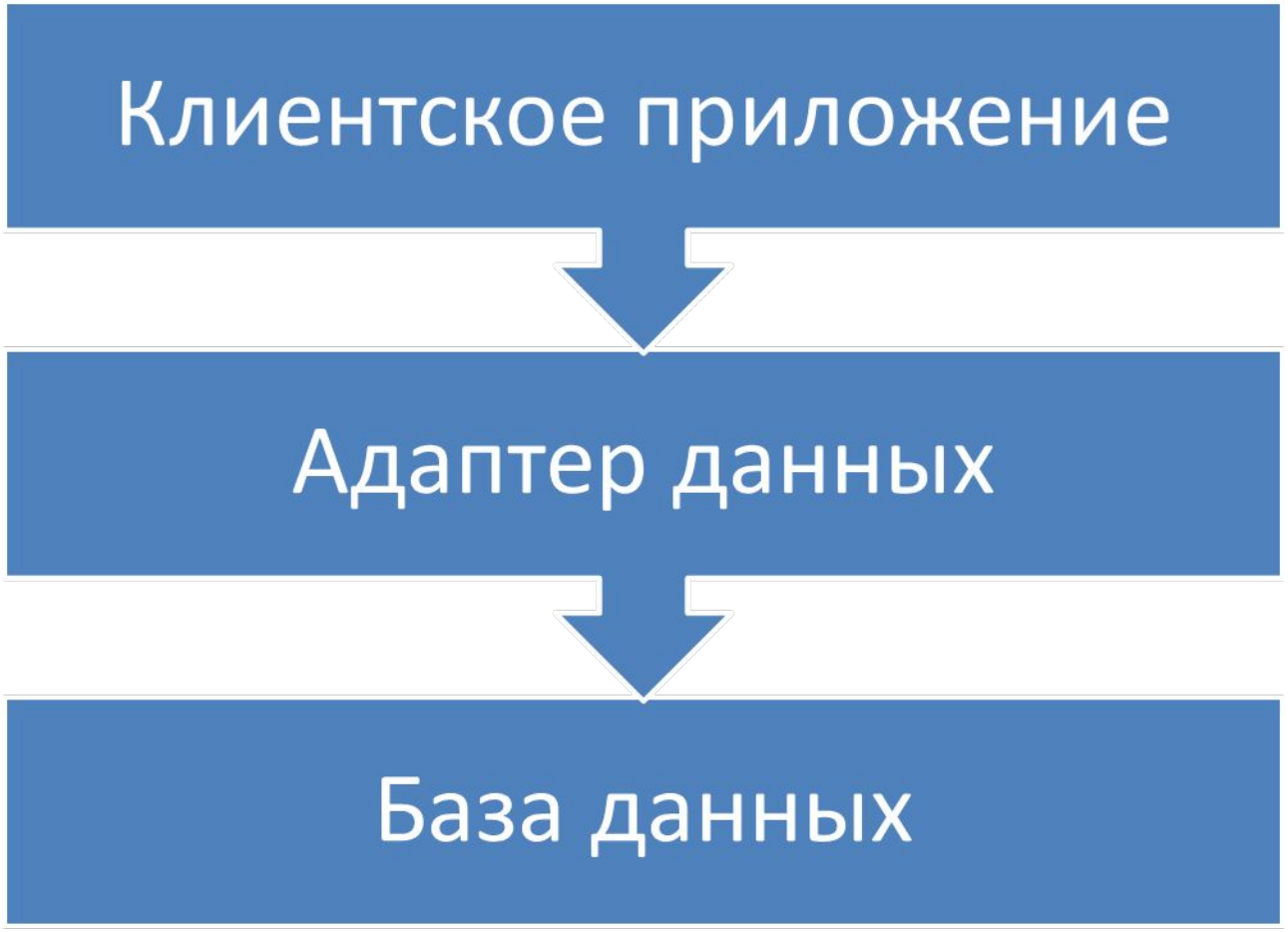
Транзакции. Пример

MERA

```
SqlTransaction tx = null;
try
{
    tx = cn.BeginTransaction();
    // Включение команд в транзакцию
    cmdInsert.Transaction = tx;
    cmdRemove.Transaction = tx;
    // Выполнение команд.
    cmdInsert.ExecuteNonQuery();
    cmdRemove.ExecuteNonQuery();
    tx.Commit();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
    // При возникновении любой ошибки выполняется откат транзакции.
    tx.Rollback();
}
```

Работа в отключенном режиме

Схема работы в отключенном режиме



Объект DataSet

представляет собой контейнер для любого количества объектов DataTable, каждый из которых содержит коллекцию объектов DataRow и DataColumn

<u>Способы создания объекта DataSet:</u>

- Создать программным образом
- Загрузить из файла
- Загрузить из БД с помощью адаптера данных

Объект DataSet

```
DataSet carsInventoryDS = new DataSet("Car Inventory");
carsInventoryDS.ExtendedProperties["TimeStamp"] = DateTime.Now;
carsInventoryDS.ExtendedProperties["DataSetID"] = Guid.NewGuid();
carsInventoryDS.ExtendedProperties["Company"] = "Мой магазин";
```

Объект DataColumn

Представляет один столбец в объекте DataTable

Множество всех объектов DataColumn, содержащихся в данном объекте DataTable, содержит всю информацию схемы таблицы

Объект DataColumn

```
DataColumn carIDColumn = new DataColumn("CarID",
  typeof(int));
//строковое значение для отображения при выводе
  данных
carIDColumn.Caption = "Car ID";
carIDColumn.ReadOnly = true;
carIDColumn.AllowDBNull = false;
carIDColumn.Unique = true;
```

Объект DataColumn

```
carIDColumn.AutoIncrement = true;
carIDColumn.AutoIncrementSeed = 0;
carIDColumn.AutoIncrementStep = 1;

// Добавление объектов DataColumn в DataTable
DataTable inventoryTable = new
  DataTable("Inventory");
inventoryTable.Columns.AddRange(new DataColumn[] {
  carIDColumn,carMakeColumn, carColorColumn,
  carPetName });
```

Объект DataRow

Представляет конкретные данные в таблице

Невозможно напрямую создать объект типа DataRow:

```
// Ошибка! Нет общедоступного конструктора!
DataRow dr = new DataRow();
```

Объект DataRow

```
// Добавление строк в таблицу Inventory
DataRow carRow = inventoryTable.NewRow();
carRow["Make"] = "BMW";
carRow["Color"] = "Black";
carRow["PetName"] = "Hamlet";
inventoryTable.Rows.Add(carRow);
```

Объект DataTable

Представляет одну таблицу

Содержит схему и данные

Чтение данных из DataTable с помощью DataTableReader

MERA

```
// Создание объекта DataTableReader
DataTableReader dtReader =
  inventoryTable.CreateDataReader();

while (dtReader.Read())
{
   for (int i = 0; i < dtReader.FieldCount; i++)
     Console.Write("{0}\t",
     dtReader.GetValue(i).ToString());
}
dtReader.Close();
```

Адаптер данных

MERA

- ✔ Заполняет объект DataSet объектами DataTable, получая значения из базы данных
- ✔ Отправляет измененные DataTable назад в базу данных для обработки
- ✔ Управляет подключением к базе данных

Адаптер данных

```
DataSet CarsDS = new DataSet();
SqlDataAdapter dataAdapter = new SqlDataAdapter();
dataAdapter.SelectCommand = selectCmd;

// executes SelectCommand, DeleteCommand,
  UpdateCommand
dataAdapter.Fill(CarsDS);

// code to modify data in CarsDS here

dataAdapter.Update(CarsDS); // updates database
```

LINQ to DataSet

Чтобы использовать LINQ to DataSet нужно получить объект

DataTable, совместимый с LINQ с помощью метода расширения

AsEnumerable()

(определен в сборке System.Data.DataSetExtensions .dll)

```
var cars = from car in inventoryTable.AsEnumerable()
           where car.Field<string>("Color") == "Black"
           select new
           {
               PetName = car.Field<string>("PetName"),
               Make = car.Field<string>("Make")
            };
```

Задание

Модифицировать программу Книжная картотека так, чтобы
данные о книгах хранились в базе данных(база данных на ваше
усмотрение)

Должна быть возможность изменить строку подключения к базе
данных, не пересобирая приложения (строка подключения
должна храниться в конфигурационном файле).

Выбор модели доступа к данным (присоединённая / отсоединенная) на ваше усмотрение.

Q & A