Высокоуровневые методы информатики и программирования

Лекция 27

Основы ADO.Net

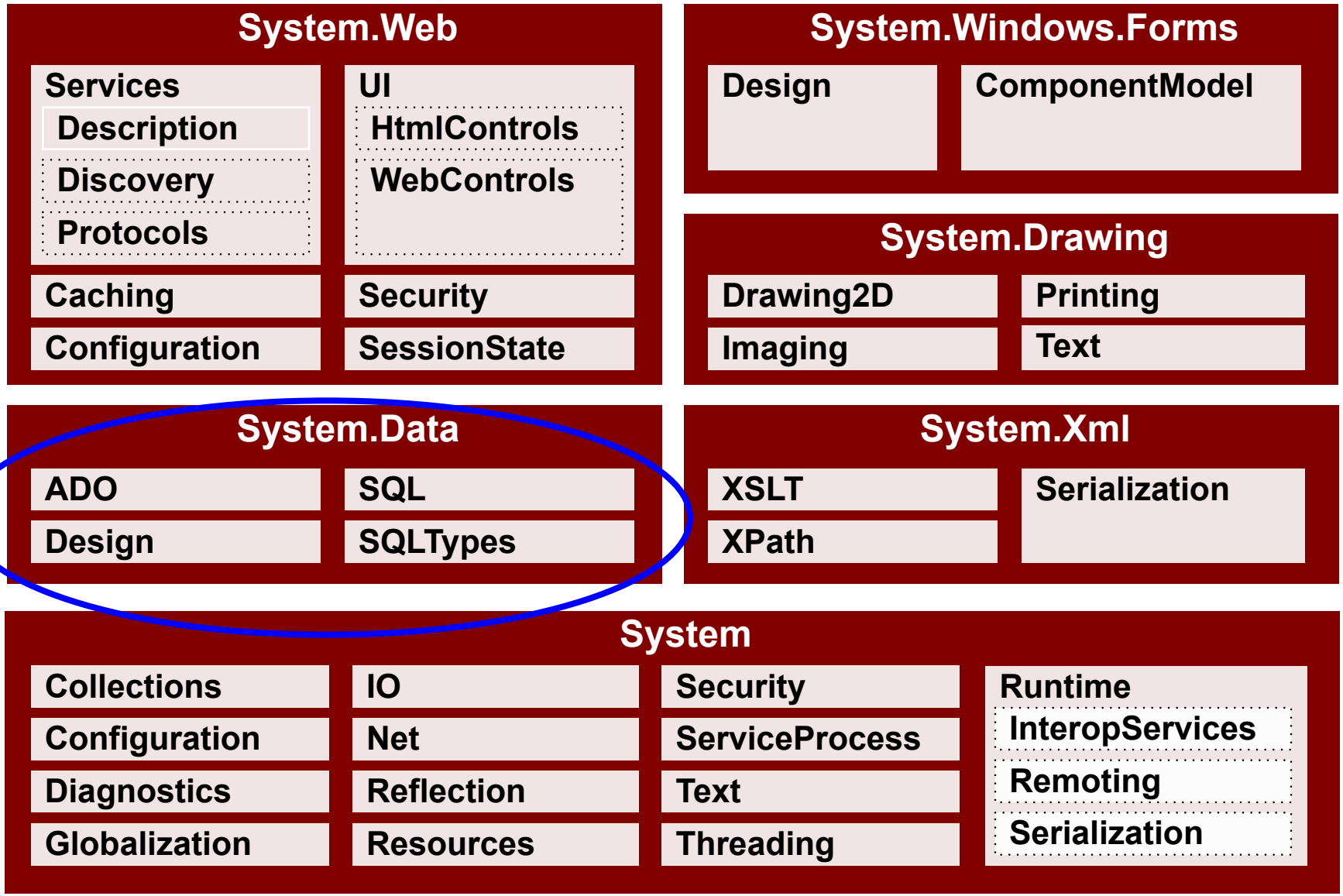
Технологии Microsoft для работы с БД

- ODBC – с использованием драйверов баз данных (описание источников данных);
- OLEDB – с использование COM компонент – провайдеров баз данных;
- ADO - с использование COM компонент – провайдеров баз данных и DataSet класса (отличный от DataSet в ADO.Net);
- ADO.Net – с использованием управляемых провайдеров БД.

Технология ADO.NET

- ADO .NET это набор классов, интерфейсов, структур и перечислений в библиотеке .NET, которые дают возможность доступа к реляционным источникам данных
- Все классы разделены по пространствам имен:
  - System.Data,
  - System.Data.Odbc
  - System.Data.OleDb,
  - System.Data.SqlClient, etc.
- ADO .NET это дальнейшее развитие ADO.
  - Имеет другую объектную модель, но поддерживает те же подходы к выполнению работы!

Пространства имен FCL
(FCL Namespaces)



Провайдер данных

- Связь с базой данных создается и поддерживается при помощи ***провайдеров данных*** (Data Provider).
- Провайдер это набор взаимосвязанных классов, обеспечивающих доступ к данным.
- Любой провайдер состоит из следующего набора типов классов:
  - 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.

Провайдеры данных ADO.NET

- Названия классов провайдера включают префикс перед названием типа класса.
  - Например :
    - OleDb<имяКласса> - для провайдера OleDb
    - Sql<имяКласса> - для провайдера SqlClient

- SQL Server провайдер – специально для работы с сервером Microsoft SQL (пространство System.Data.SqlClient)
- С Microsoft SQL Server можно работать и с помощью классов OLEDB, но менее эффективно

Имеющиеся в .Net провайдеры баз данных

- Odbc Data Provider - провайдер для работы с базами данных по технологии ODBC (System.Data.Odbc,префикс Odbc)
- OleDb Data Provider - провайдер для работы с базами данных по технологии OleDb (System.Data.OleDb префикс OleDb)
- SQL Server Data Provider – провайдер для работы с базами данных SQL Server (System.Data.SQLClient, префикс Sql)
- Oracle Data Provider – провайдер для работы с базами данных Oracle (System.Data.OracleClient, префикс Oracle).

- Odbc, OleDb, SQL Server провайдеры содержатся в компоненте System.Data.

- SQL Server провайдер также содержится и в отдельном компоненте System.Data.SqlClient.
- Oracle провайдер содержится в компоненте System.Data.OracleClient.

Microsoft ADO.NET Data Providers

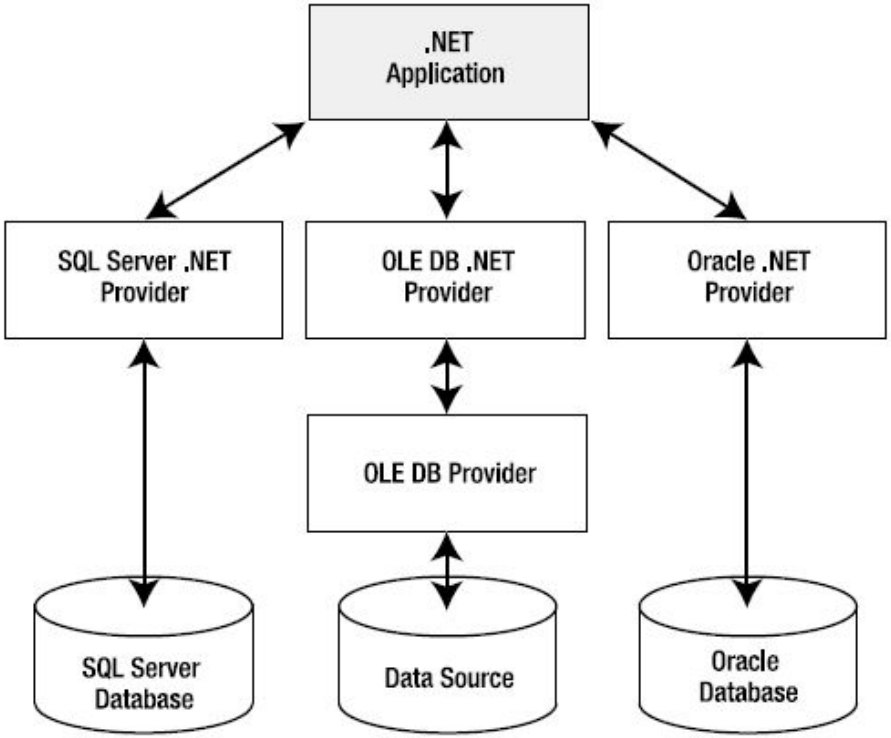
Провайдеры данных	Namespace	Сборка
ODBC	System.Data.Odbc	System.Data.dll
OLE DB	System.Data.OleDb	System.Data.dll
Microsoft SQL Server	System.Data.SqlClient	System.Data.dll
Oracle	System.Data.OracleClient	System.Data. OracleClient.dll

- В пространстве имен System.Data описаны общие классы ADO.Net.
- Например:
  - DataSet
  - DataTable
  - DataRaw
  - DataRelation и т.д.

Задание оператора using для работы с базой данных

- Оператор using должен быть записан до всех других объявлений в файле и не может появиться внутри класса или объявлений модуля

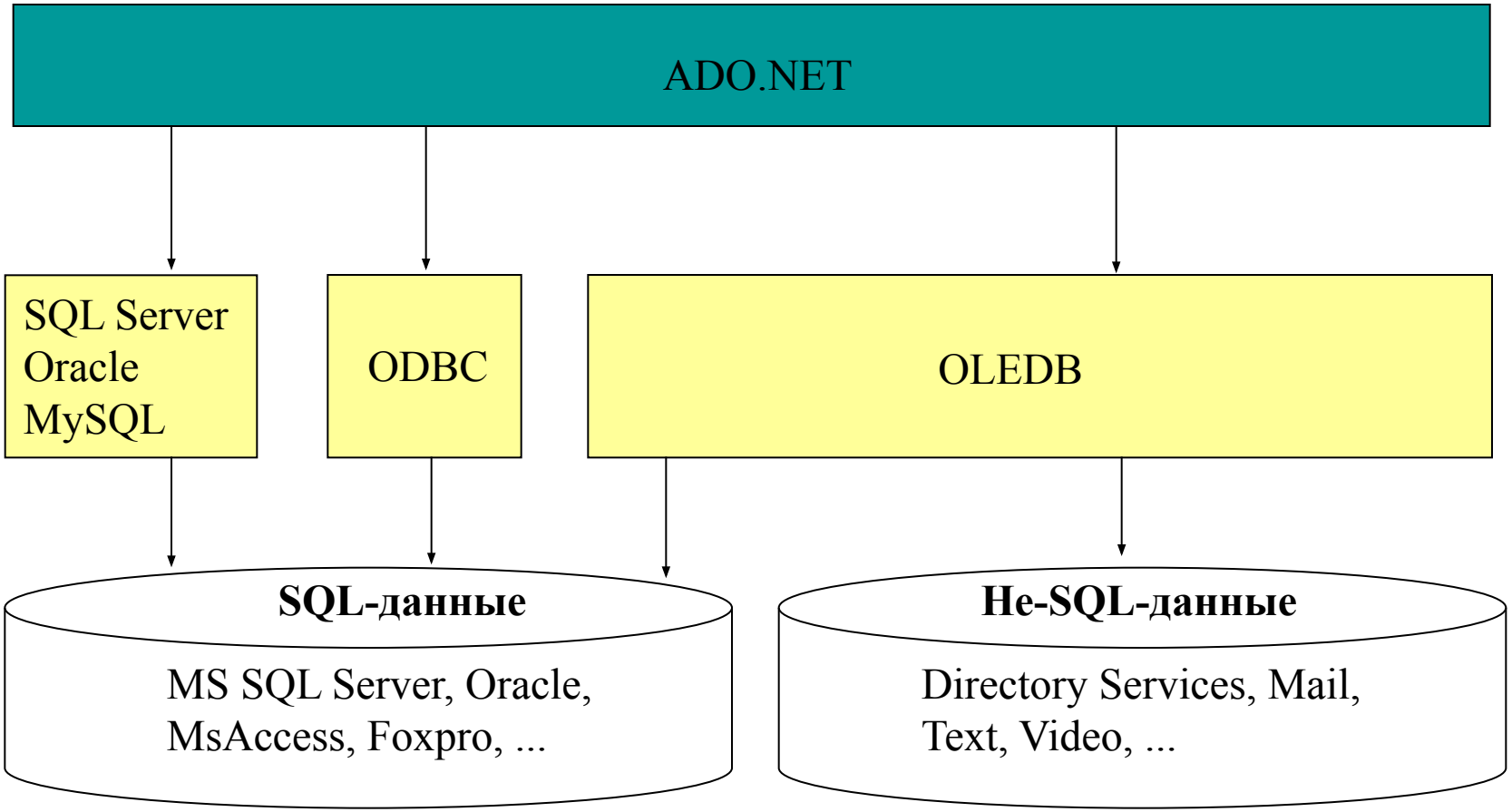
```
using System.Data;
using System.Data.OleDb;
using System.Data.SqlClient;
using System.Data.Odbc;
public class Form1
{
. . .
}
```



- Провайдеры SQL Server и Oracle работают с базой данных самостоятельно.

- Провайдеры ODBC и OleDB работают через старые технологии (или драйверы БД или COM компоненты провайдеры БД).

Использование провайдеров данных для работы с БД



Шаблон работы в соединенном режиме с БД

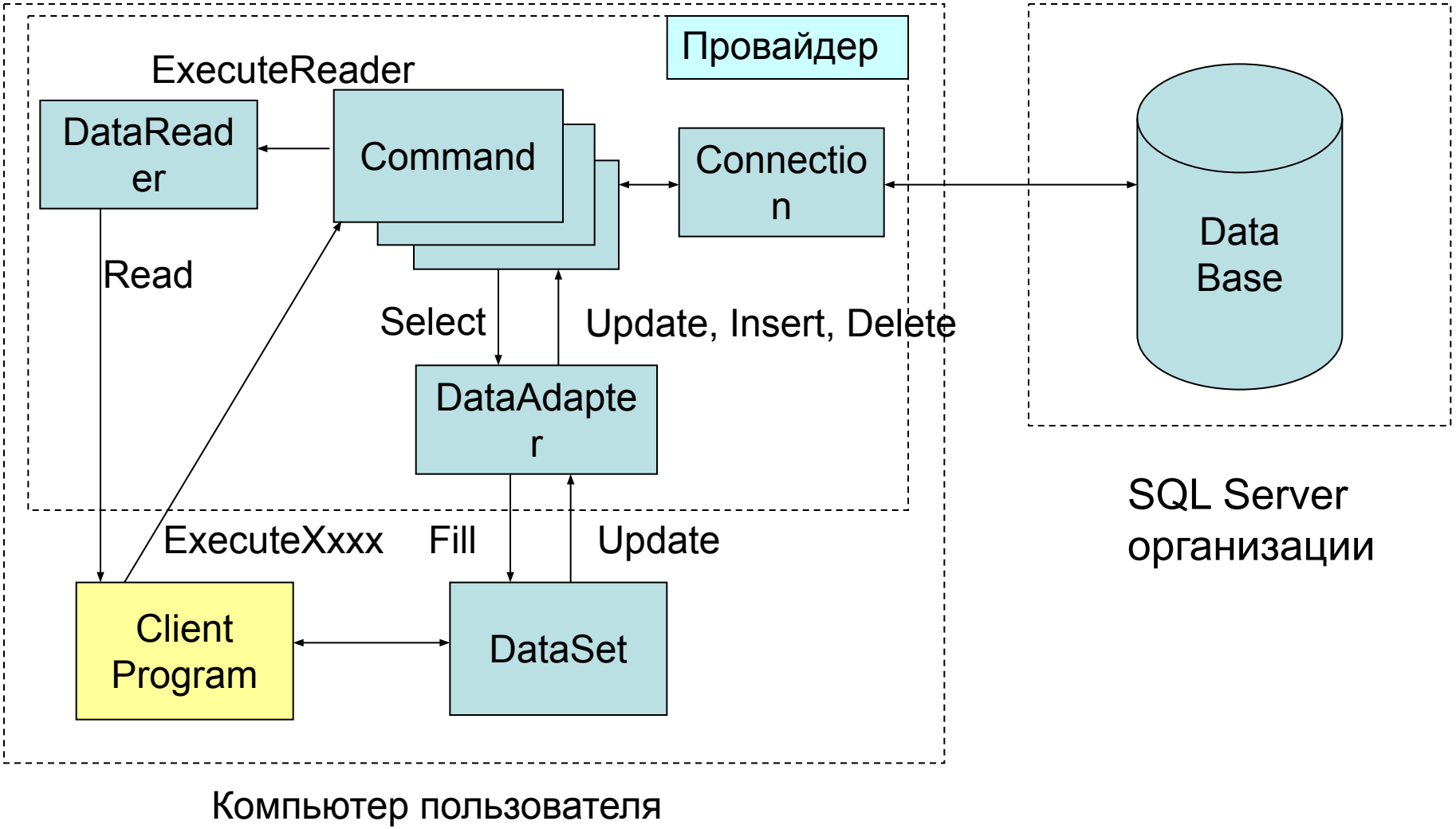
```
1.) Объявление соединения (connection)
try {
        1.) Открытые соединения с БД

        2.) Создание и выполнение команды


        3.) Обработка результатов


        4.) Освобождение ресурсов
} catch ( Exception ) {
        Handle exception
} finally {
        try {
                4.) Закрытие соединения
        } catch (Exception)
                { Handle exception }
}
```

Способы работы с базами данными

- ***С поддержкой соединения*** (Connected): Forward-only, read-only
  - Программа делает запрос, затем читает результаты и обрабатывает их
  - Используется курсор “Firehose” (брандспойт)
  - Используется объект DataReader

- ***С разрывом соединения*** (Disconnected, отсоединенный режим)
  - Программа делает запрос затем читает и сохраняет результаты для обработки, отсоединяется от БД
  - Выполняется работа с данными (добавление, изменение, удаление)
  - Минимизируется время соединения с базой данных
  - Используется объект DataSet
- ***С использованием технологии LINQ***
  - LINQ to DataSet
  - LINQ to SQL

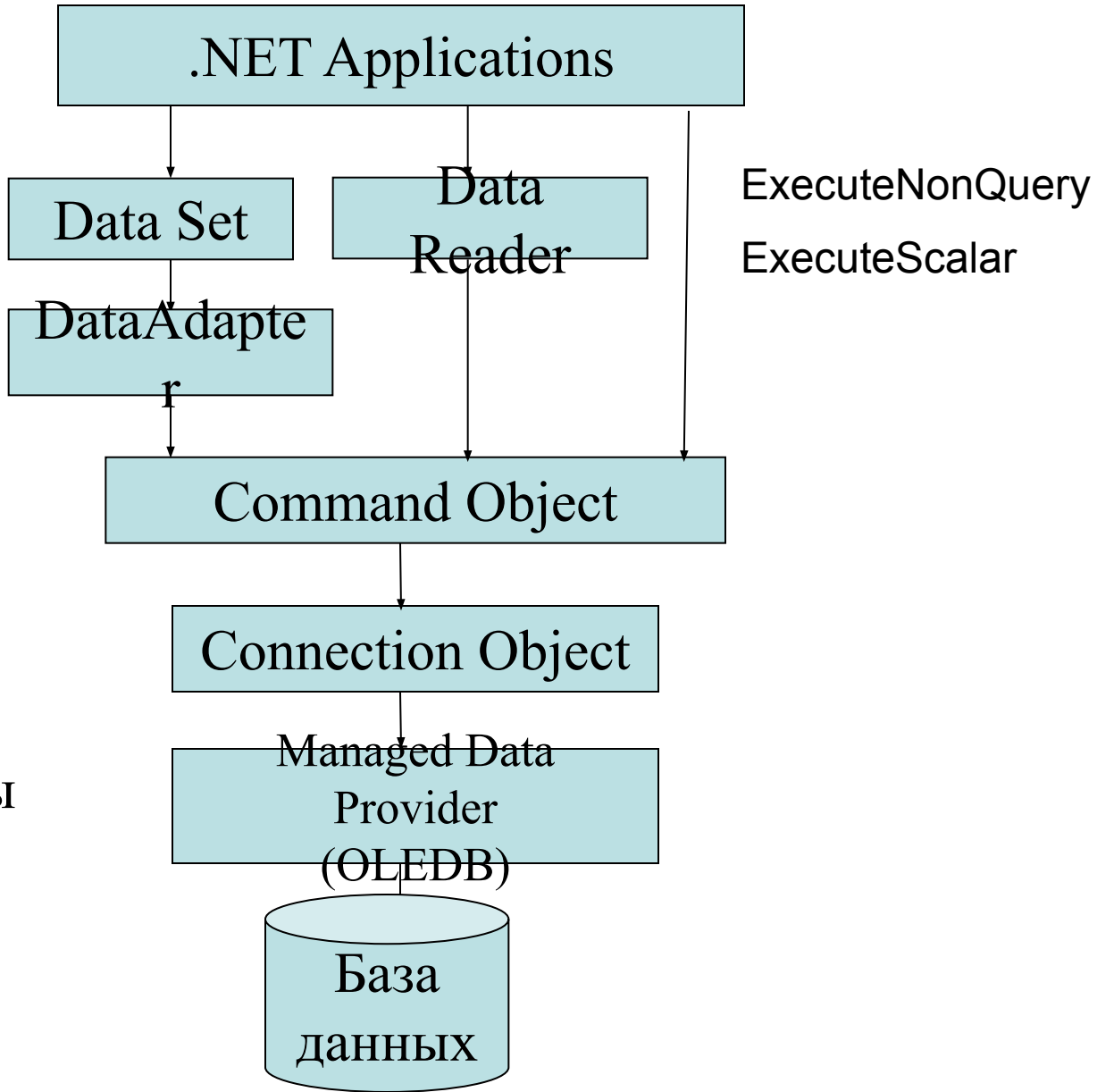
Использование классов ADO.NET



Назначение типов классов ADO.Net

- Connection – выполняет соединение с БД
- Command – подготовка и выполнение SQL команд
  - Parameter - для модификации объекта Command
- DataReader – для быстрого считывания данных из БД
- DataAdapter – содержит набор SQL команд (Select, Insert, Update, Delete) для работы с данными в оперативной памяти и выполняет работу по связи класса Dataset с базой данных
  - CommandBuilder

Объекты ADO.NET

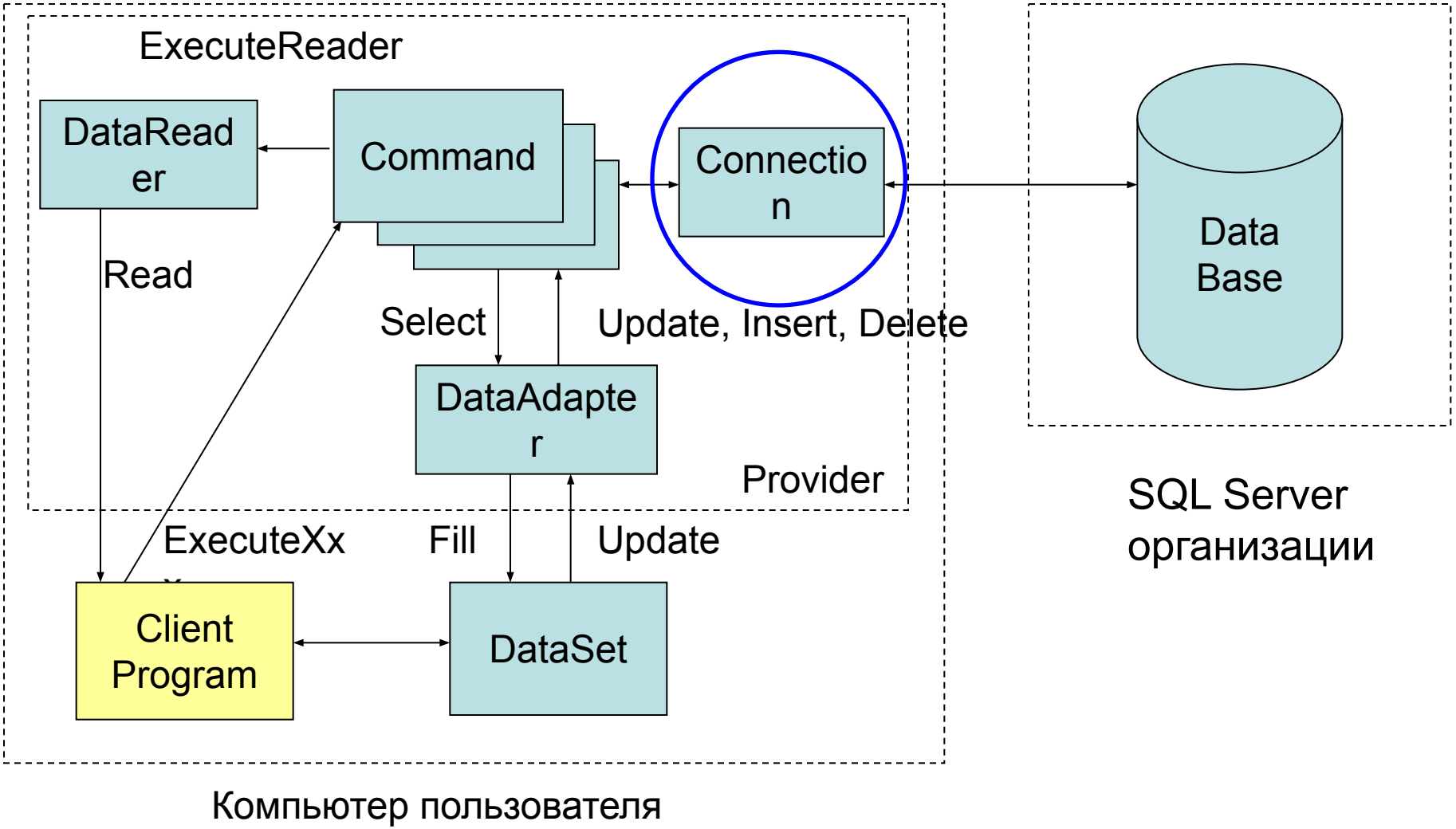


Отсоединенный режим работы с БД

Последовательность работы с данными с поддержкой соединения

- Установить связь с базой данных.
- Выполнить запрос к базе данных.
  - Создать и выполнить команды
- Получить результаты команды.
- Закрыть связь с базой данных

Использование классов ADO.NET



Класс Connection

- выполняет реальный обмен данными между базой данных и приложением
- является часть Data Provider
- свойства
  - ConnectionString
  - ConnectionTimeout
  - Data Base
- методы
  - Open() – открытие соединения
  - Close() – закрытие соединения

Строка соединения

- Объект SqlConnection
  - Server
  - Database (Initial Catalog)
  - uid (User ID)
  - pwd (Password)
- Объект OleDbConnection
  - Provider
  - Data Source (Server)
  - uid (User ID)
  - pwd (Password)

Формат строки соединения

“param1 = val1; param2 = val2; … paramN = valN”

- param – имя параметра строки соединения
- val – значение параметра

Основные параметры строки соединения

- Data Source=(local)\SQLEXPRESS;
  - (local)
  - localhost
  - . (просто точка)
- Initial Catalog = <имя БД>;
- uid=<идентификатор>;
- pwd=<пароль>;
- IntegratedSecurity =True;
  - True
  - ISSP
  - yes
- Provider= … (для ODBC и OLEDB)
- ….

Пример строки соединения

- Для Access

```
Conn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
    Source=" + fld + "\\VbDB.mdb";
```

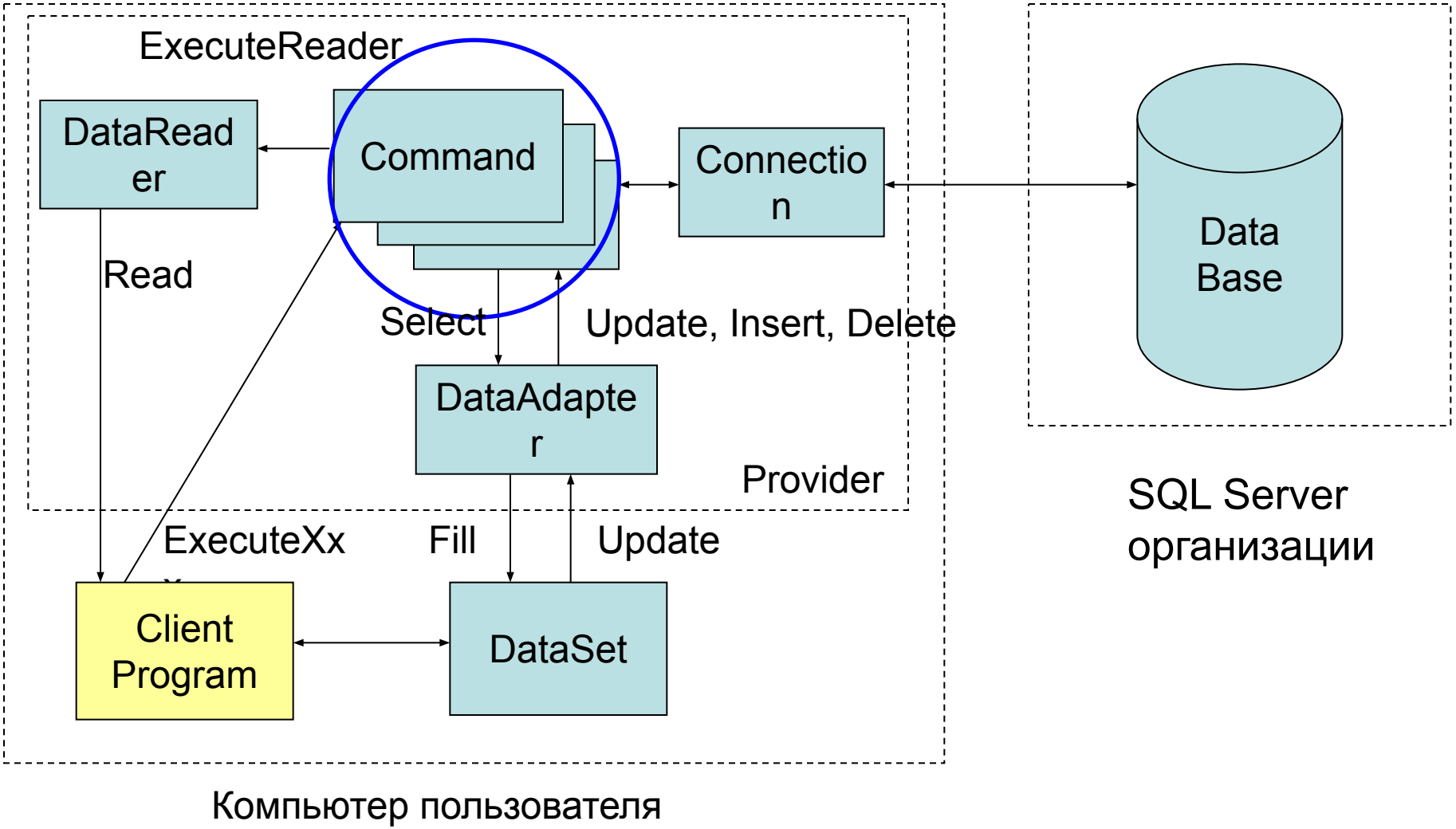
- Для SQL Server

```
con.ConnectionString = @"Data Source=localhost\sqlexpress;Initial
    Catalog=Northwind;Integrated Security=True";
```

Пример использования объекта Connection

```
string strConn;
strConn = "Provider=Microsoft.Jet.OLEDB.4.0;” +
   “Data Source = c:\school.mdb“;
OleDbConnection conn;
conn = new OleDbConnection(strConn);
conn.Open();

. . .

conn.Close();
```

Использование классов ADO.NET



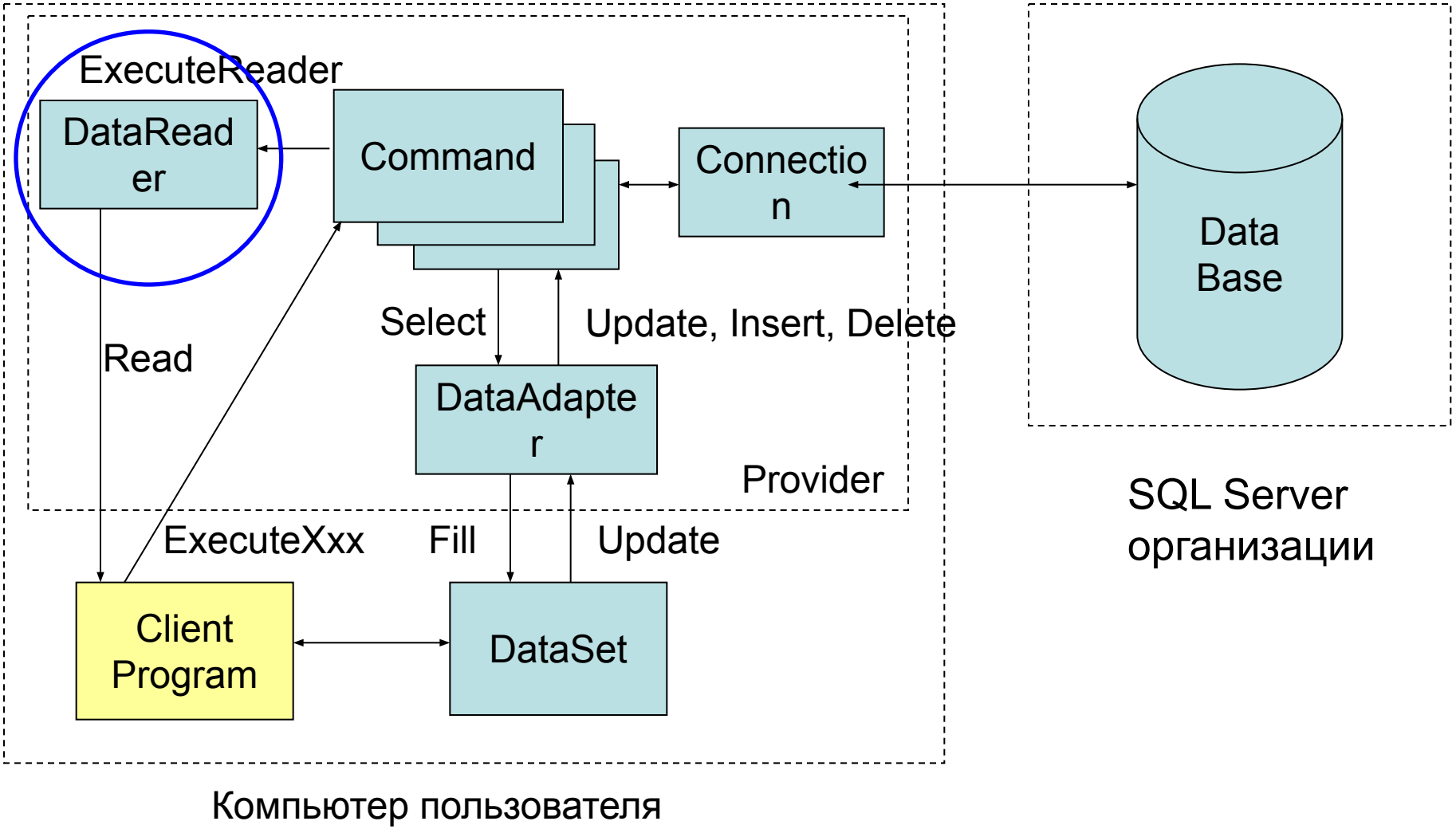
Класс Command

- Класс команд, позволяет выполнить действия с базой данных (выборку, обновление, дополнение, удаление и т.п.).
- Свойства:
  - CommandType:
    - CommandType.Text - 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.

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);

Использование классов ADO.NET



Метод ExecuteReader()

- Создает объект DataReader и возвращает ссылку на него.
- Текст команды должен содержать оператор Select или вызов хранимой процедуры.

Пример вызова метода ExecuteReader()

```
// формируем строку подключения к БД
string strConn ="Provider=Microsoft.Jet.OLEDB.4.0; Data Source = c:\school.mdb“;
OleDbConnection conn = new (strConn);
string strSQL = "select * from books”; //SQL команда
objConn.Open();  // выполняем соединение с БД
// создаем объект класса Command
OleDbCommand cmnd as new OleDbCommand(strSQL,conn);
OleDbDataReader dtReader;
// создаем объект для чтения
dtReader=objComm.ExecuteReader();
```

Класс DataReader

- Объекты данного класса позволяют выполнять только чтение данных из БД, полученных с помощью объекта Command, только в одном направлении (от начала к концу).
- Одновременно объект RataReader дает доступ только к одной записи выборки.
- Можно определить значение поля в записи, используя индексатор
  - dr[n] или dr[“имя поля”]

Объект DataReader

- Для перехода к следующей записи выборки используется метод

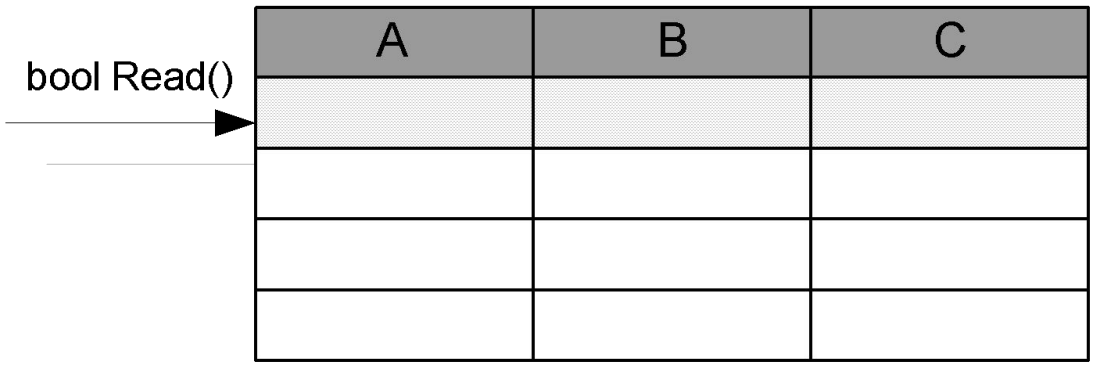
  bool Read() : читает текущую запись и перемещает указатель на следующую запись.
- Если метод Read возвращает true, то следующая запись прочитана, если записи нет, то возвращается false.
- Для окончания работы с объектом должен быть выполнен вызов метода:
  - Close: Окончание работы с данными в DataReader.

Объект DataReader

- Метод ExecuteReader() возвращает ссылку на объект DataReader

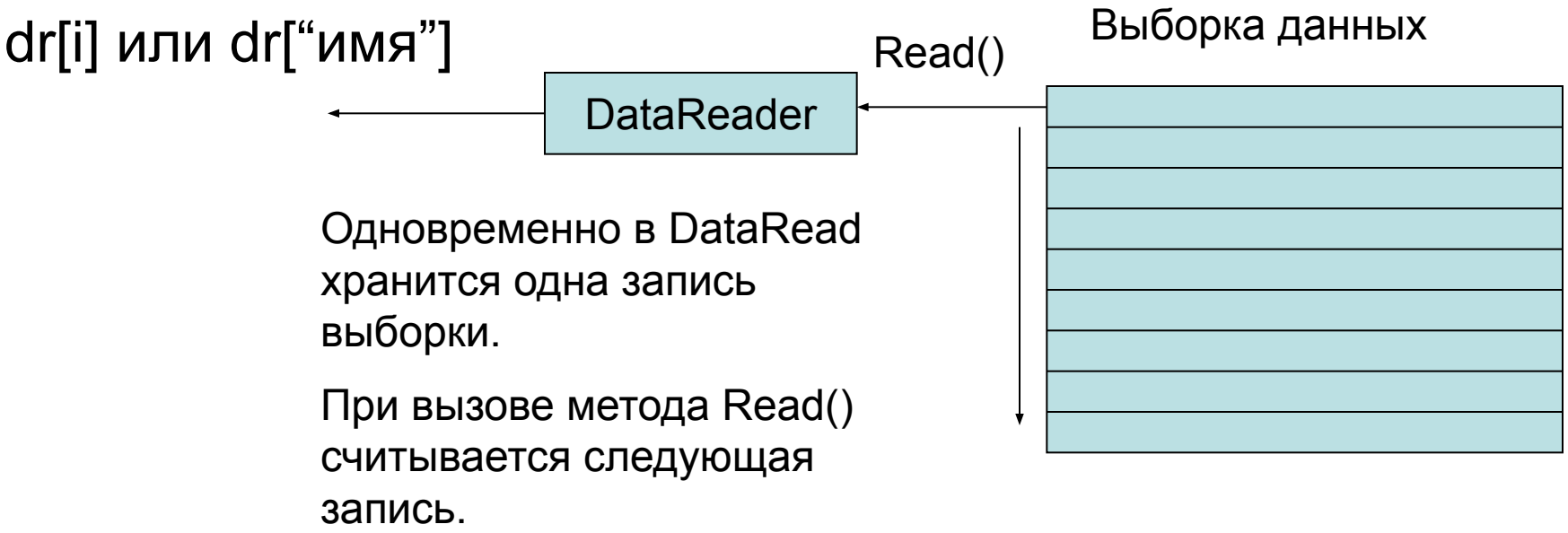
```
DataReader ExecuteReader()
```

- Объект DataReader позволяет последовательно читать записи из полученной выборки (запись за записью)



Result table of a SELECT statement

Получение данных выборки



Чтение записей с помощью DataReader

Поля (столбцы) текущей записи можно прочитать двумя способами

1. dtReader[0]
2. dtReader[“ChildId”] // ChildId – имя поля записи

Пример:

```
string Results;
while (dtReader.Read() == true)
{
    Console.WriteLine(dtReader[“ChildId”] +  “ “  +
        tReader[“name”];
}
Textbox1.text=Results;
```

Класс DataTableReader

```
DataTableReader dtr = tbl.CreateDataReader();

while(dtr.Read())
{
    for(int i=0; I < dtr.FieldCount; i++)
    {
     Console.Write("{0} = {1}",
          dtr.GetName(i),
          dtr.GetValue(i).ToString().Trim());
    }
    Console.WriteLine();
    dtr.Close();
}
```

Метод ExecuteNonQuery()

Позволяет выполнить такие команды,

- команды корректировки (возвращает количество измененных записей)
  - INSERT

  (INSERT INTO tbl (f1, f2, f3) VALUES ('xxx', 1986, 'yyy'))
  - UPDATE

  (UPDATE childs SET id = 27 WHERE year = 1997)
  - DELETE

  (DELETE FROM childs WHERE ID = 5)
- другие команды, которые не возвращают значений (результат -1)
  - CREATE DATABASE
  - CREATE TABLE

Пример вызова метода ExecuteNonQuery()

```
OleDbCommand Comm = new OleDbCommand();
Comm.Connection = Conn;
Comm.CommandType = CommandType.Text;
Comm.CommandText = "INSERT into Books(id, [year], author, name) " +
                   "VALUES (33, 2006, ‘John’, ‘Programming’)";
try
{
    int rc = (int)Comm.ExecuteNonQuery();
}
catch (OleDbException ex)
{
    System.Console.WriteLine(ex.Message);
}
```

Параметры запроса

- 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.
- 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

- Класс xxxParameter для описания параметров запроса.
  - свойство ParameterName;
  - свойство xxxType (например, SqlDbType);
  - свойство Direction (ParameterDirection.Input; ParameterDirection.Output);
  - свойство Value.

- В объекте Command имеется коллекция параметров (объектов Parameter) Parameters.

- Для использования параметра нужно создать объект Parameter и сохранить его в коллекции Parameters.

- Методы добавления
  - Add(parameter);
  - AddWithValue(name, value);

Пример описания параметра

```
SqlParameter parameter = new SqlParameter();
parameter.ParameterName = "@CategoryName";
parameter.SqlDbType = SqlDbType.NVarChar;
parameter.Direction = ParameterDirection.Input;
```

Метод AddWithValue()

- Коллекции Parameters имеет метод AddWithValue() с двумя входными параметрами:
  - name – название параметра;
  - value – значение параметра.
- Тип параметра не задается, а выводится из типа данных.

```
cmd.Parameters.AddWithValue("@LastName",
  lastName);
```

- Этот метод нельзя использовать для выходных параметров хранимой процедуры.

Передача параметров в хранимую процедуру

```
// Create the command and set its properties.
SqlCommand command = new SqlCommand();
command.Connection = connection;
command.CommandText = "SalesByCategory";
command.CommandType = CommandType.StoredProcedure;

// Add the input parameter and set its properties.
SqlParameter parameter = new SqlParameter();
parameter.ParameterName = "@CategoryName";
parameter.SqlDbType = SqlDbType.NVarChar;
parameter.Direction = ParameterDirection.Input;

// Add the parameter to the Parameters collection.
command.Parameters.Add(parameter);

// задаем значение параметра
command.Parameters[“@CategoryName”.Value = categoryName;

// Открывем соединение и выполняем работу с объектом DataReader
connection.Open();
SqlDataReader reader = command.ExecuteReader();
```

Пример вызова хранимой процедуры

- В БД есть хранимая процедура

```
CREATE PROCEDURE GetPetName
@carID int, @petName char(10) output AS
SELECT @petName = PetName from Inventory where CarID = @carID
```

- Вызов процедуры

```
SqlCommand cmd = new SqlCommand("GetPetName", this.sqlCn);
cmd.CommandType = CommandType.StoredProcedure;
// Input param.
SqlParameter param = new SqlParameter(); param.ParameterName = "@carID";
param.SqlDbType = SqlDbType.Int; param.Value = carID; param.Direction = ParameterDirection.Input;
cmd.Parameters.Add(param);

// Output param.
param = new SqlParameter(); param.ParameterName = "@petName";
param.SqlDbType = SqlDbType.Char; param.Size = 10; param.Direction = ParameterDirection.Output;
cmd.Parameters.Add(param);

// Execute the stored proc.
cmd.ExecuteNonQuery();

// Return output param.
carPetName = ((string) cmd.Parameters["@petName"].Value).Trim();
```

Пример использования метода AddWithValue

```
Comm.CommandText = "INSERT into “ + "
    "Books(id,[year],author,name) " +
   "VALUES (@id,@year,@au,@nm)";
//
Comm.Parameters.AddWithValue(“@id”, 112);
Comm.Parameters.AddWithValue (“@year”, 200);
Comm.Parameters.AddWithValue (“@au”, “Иванов С.П.”);
Comm.Parameters.AddWithValue (“@nm”, “История России”);

Comm.ExecuteNonQuery();
```

Метод ExecuteScalar()

- возвращает первую строку первого столбца в результирующем наборе (используя функции COUNT, AVG, MIN, MAX, SUM);
- используются функции SQL
  - COUNT – рассчитать количество
  - AVG – рассчитать среднее значени е
  - MIN – определить минимальное значение
  - MAX – определить максимальное значение

- Пример:

```
OleDbCommand Comm = new OleDbCommand();
Comm.Connection = Conn;
Comm.CommandType = CommandType. Text;
Comm.CommandText = "Select max(year) From Books"; // Text

int yr = (int)Comm.ExecuteScalar();
```