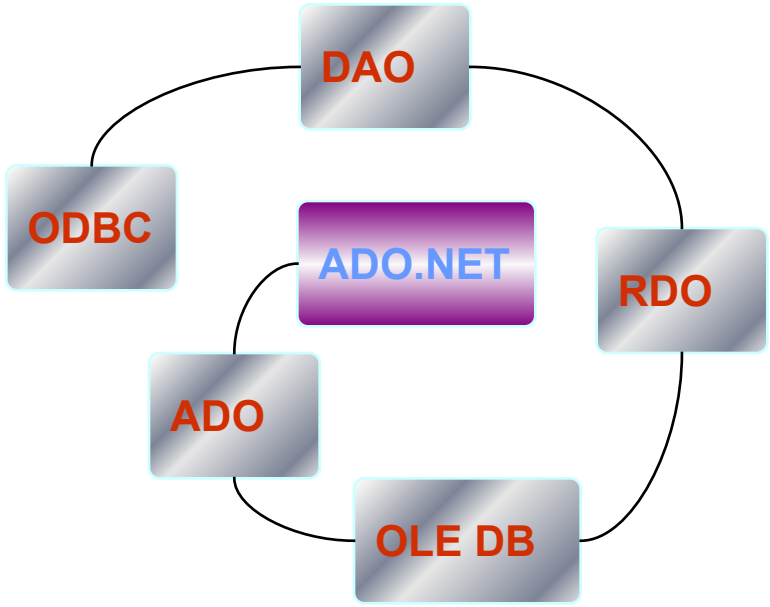
Лекция №10

Розділ 3. Архітектура та проектування компонентних систем

Основи технології ADO.NET

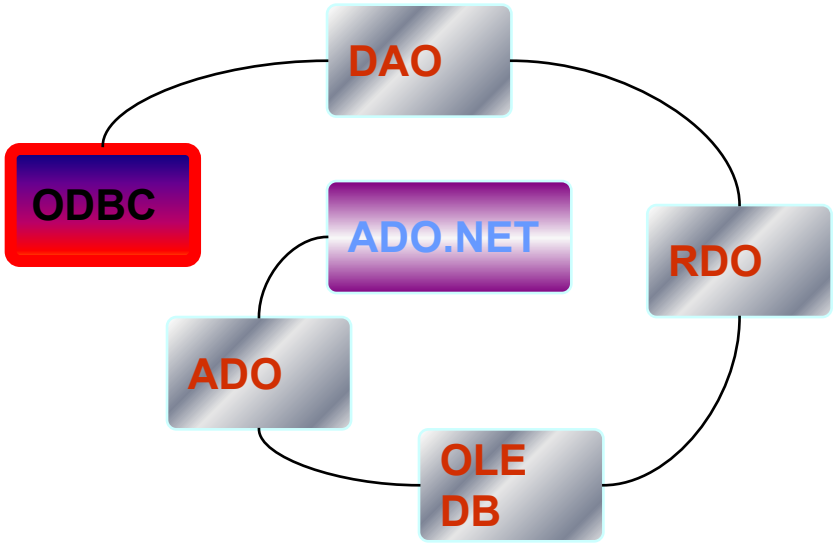
Технології Microsoft для роботи з БД

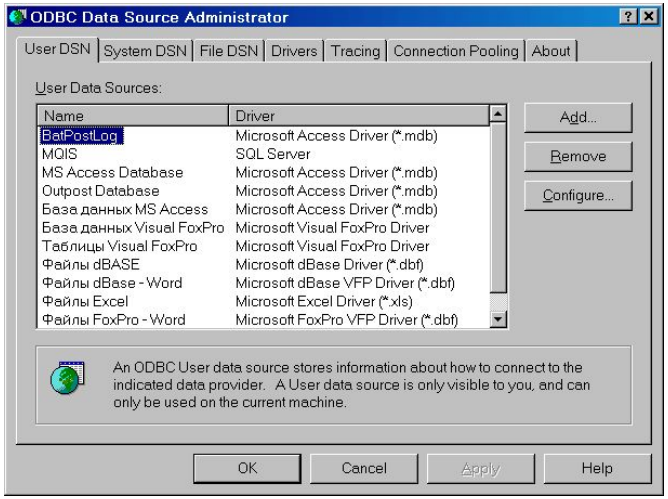
- ODBC API (Open Database Connectivity);
- RDO (Remote Data Objects);
- DAO (Data Access Objects);
- OLE DB;
- ADO (ActiveX Data Objects);
- ADO.NET;



ODBC (Open Database Connectivity)

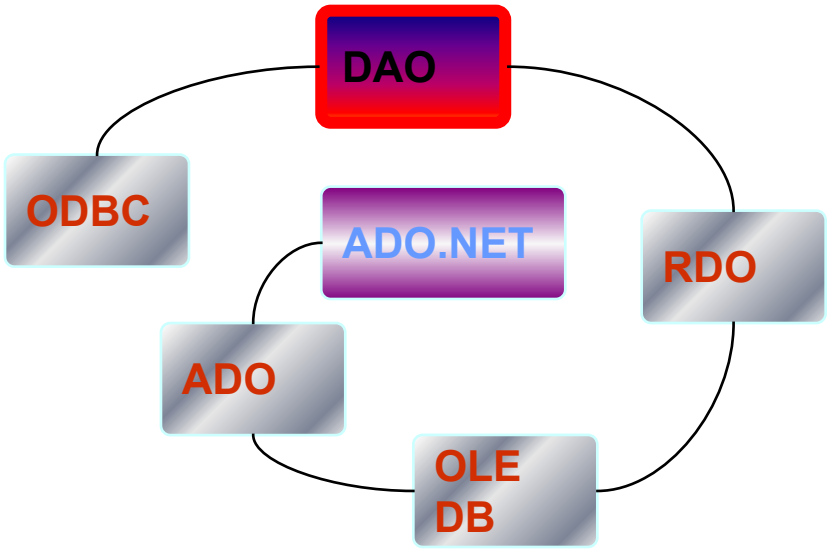
- API доступу до БД
- Ціль – один код для різних БД
- Використовує SQL у якості робочої мови
- Побудований на наборі драйверів, що забезпечують доступ до конкретних СКБД





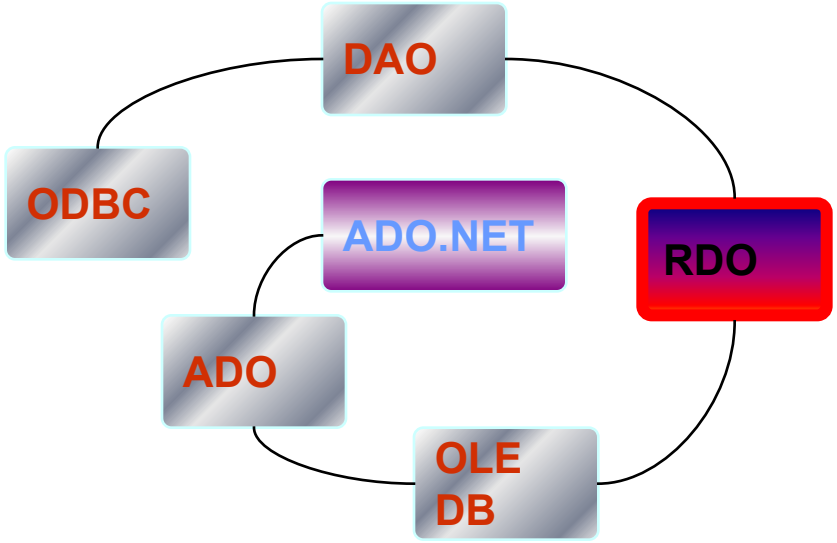
DAO (Data Access Objects)

- DAO надає модель об’єктів для доступу до локальних БД (ISAM джерела FoxPro, Paradox, Lotus 1-2-3 тощо) або до баз даних SQL через Jet (напряму або через ODBC)
- Тільки реляційні БД



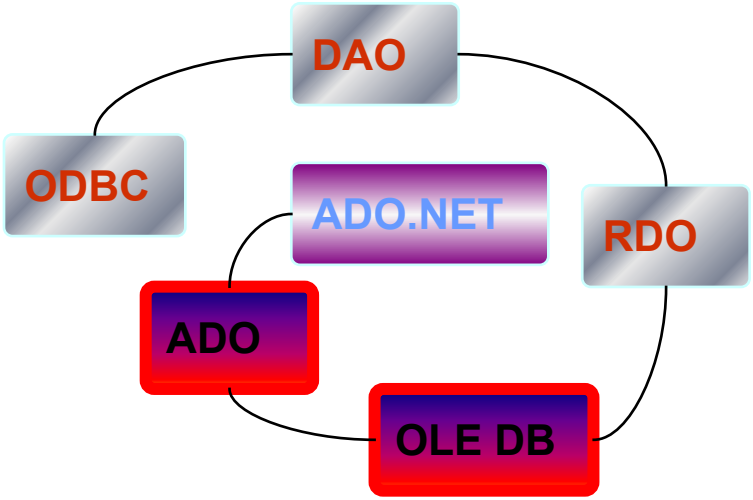
RDO (Remote Data Objects)

- RDO надає доступ до реляційних БД через ODBC
- Створювався для того, щоб можна было обходитися без ODBC API
- RDO – об’єктний COM-інтерфейс до ODBC API
- Як і DAO, тільки реляційні БД



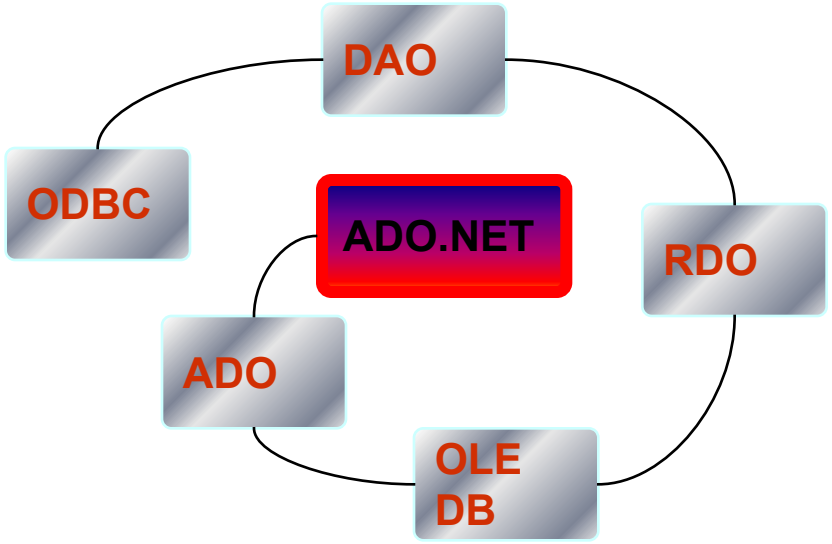
OLE DB та ADO

- OLE DB – технологія з використанням COM-компонент – провайдерів БД;
- ADO – технологія з використанням COM-компонент – провайдерів БД та класу DataSet (відрізняється від DataSet в ADO.NET);

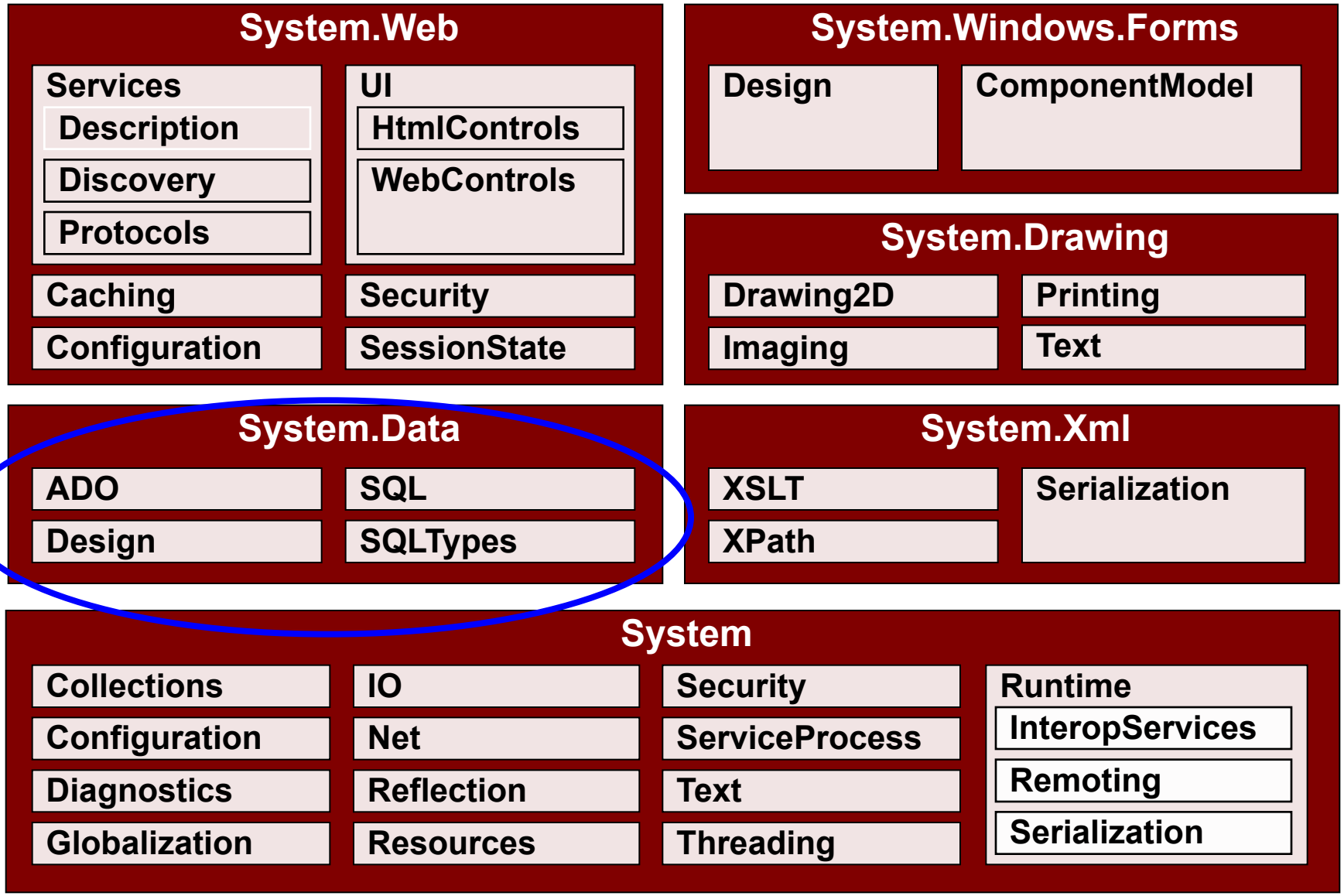


Технологія ADO.NET

- ADO.NET – набір класів, інтерфейсів, структур та перерахувань у бібліотеці .NET, які надають доступ до реляційних джерел даних.
- Основна відмінність ADO .NET від ADO – використання моделі постачальників даних замість загального набору об’єктів, що не залежать від джерел даних.



Простори імен FCL



Провайдер данных

Провайдер даних *(data provider)* – це набір класів ADO.NET, які дозволяють отримувати доступ до визначеної БД, виконувати команди SQL і витягувати дані.

Будь-який провайдер складається з наступного набору класів:

- Connection – забезпечує підключення до БД;
- Command – виконує команди SQL або збережені процедури;
- DataReader – надає доступний лише для однонаправле-ного читання набір записів, подключений до БД;
- DataAdapter – заповнює від'єднаний об'єкт DataSet або DataTable та оновлює його вміст.

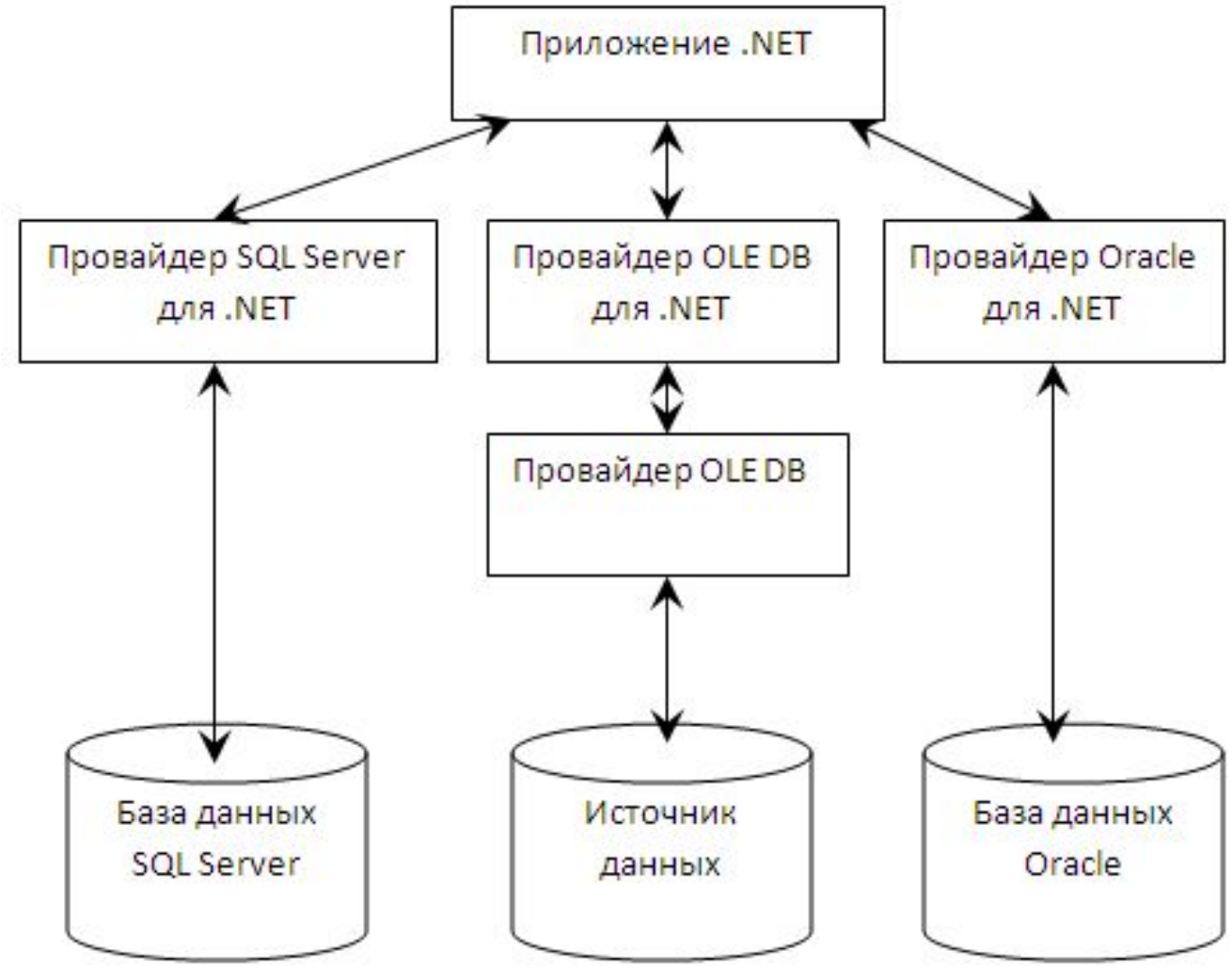
Назви класів провайдера включають префікс перед назвою типу класу. Наприклад:

- OleDb<ім'яКласу> - для провайдера OleDb;
- Sql<ім'яКласу> - для провайдера SqlClient.

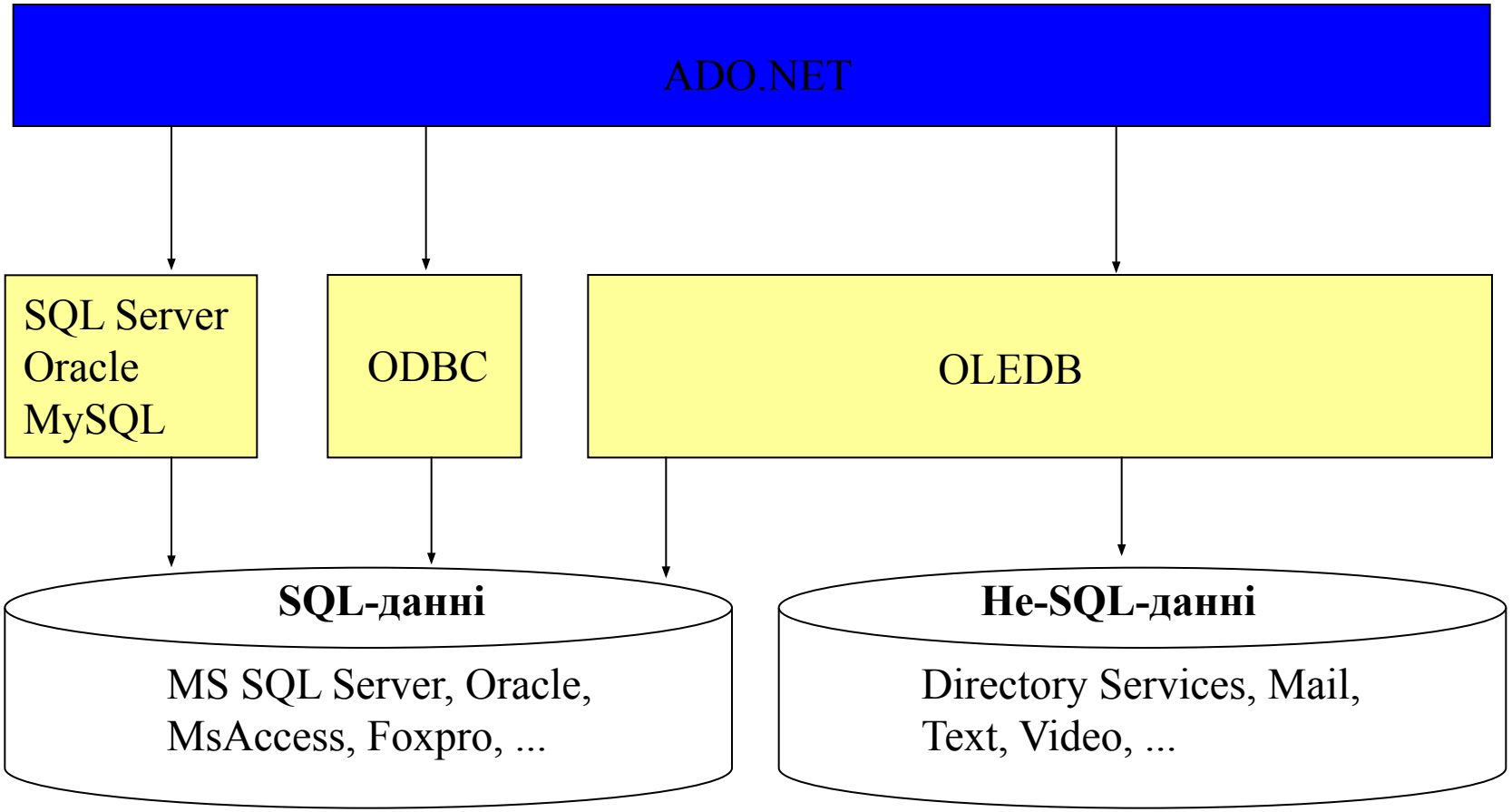
Провайдери даних Microsoft ADO.NET

Провайдери даних	Простір імен	Опис
ODBC	System.Data.Odbc	Класи для підключення до більшості драйверів ODBC, зокрема **OdbcCommand**, **OdbcConnection** та **OdbcDataAdapter**.
OLE DB	System.Data.OleDb	Класи для підключення до провайдера OLE DB, зокрема **OleDbCommand**, **OleDbConnection** та **OleDbDataAdapter**.
Microsoft SQL Server	System.Data.SqlClient	Класи для підключення до БД Microsoft SQL Server, у тому числі **SqlDbCommand**, **SqlDbConnection** та **SqlDbDataAdapter**.
Oracle	System.Data.OracleClient	Класи для підключення до БД Oracle, зокрема **OracleCommand**, **OracleConnection** та **OracleDataAdapter**.

Архітектура ADO.NET



Використання провайдерів даних для роботи з БД



Типи об’єктів ADO.NET

- *Об'єкти, засновані на з'єднанні* – такі об'єкти провайдера даних, як **Connection**, **Command** та **DataReader**.
- *Об'єкти, засновані на вмісті* – ці об'єкти у дійсності лише "упаковують" дані – **DataSet**, **DataColumn**, **DataRow**, **DataRelation** тощо.

Способи роботи з БД

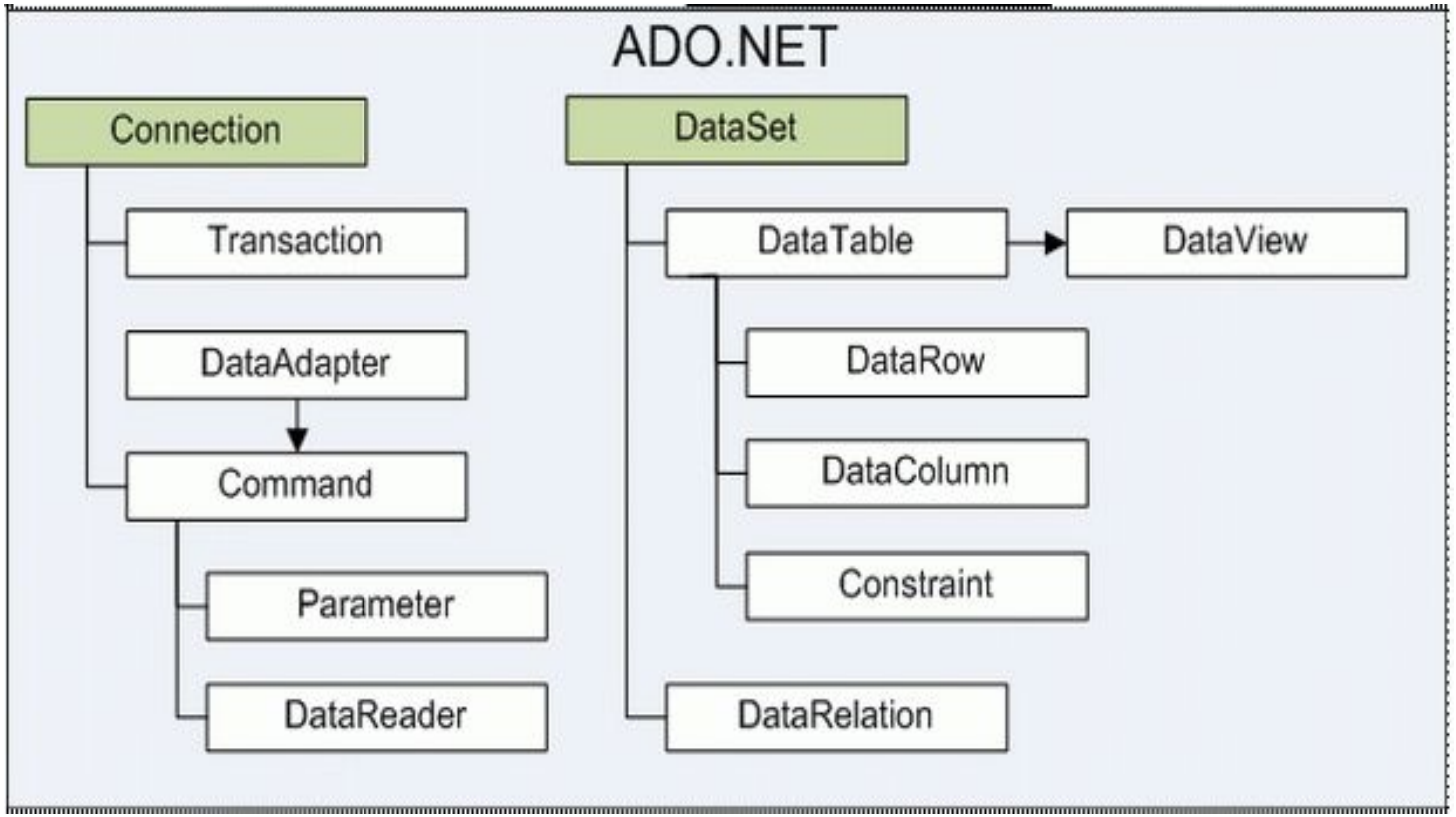
Приєднаний або *з підтримкою з'єднання (Connected,* Forward-only, read-only*)*:

- Програма робить запит, читає результати і обробляє їх
- Використовується курсор "Firehose" (брандспойт)
- Використовується об'єкт **DataReader**

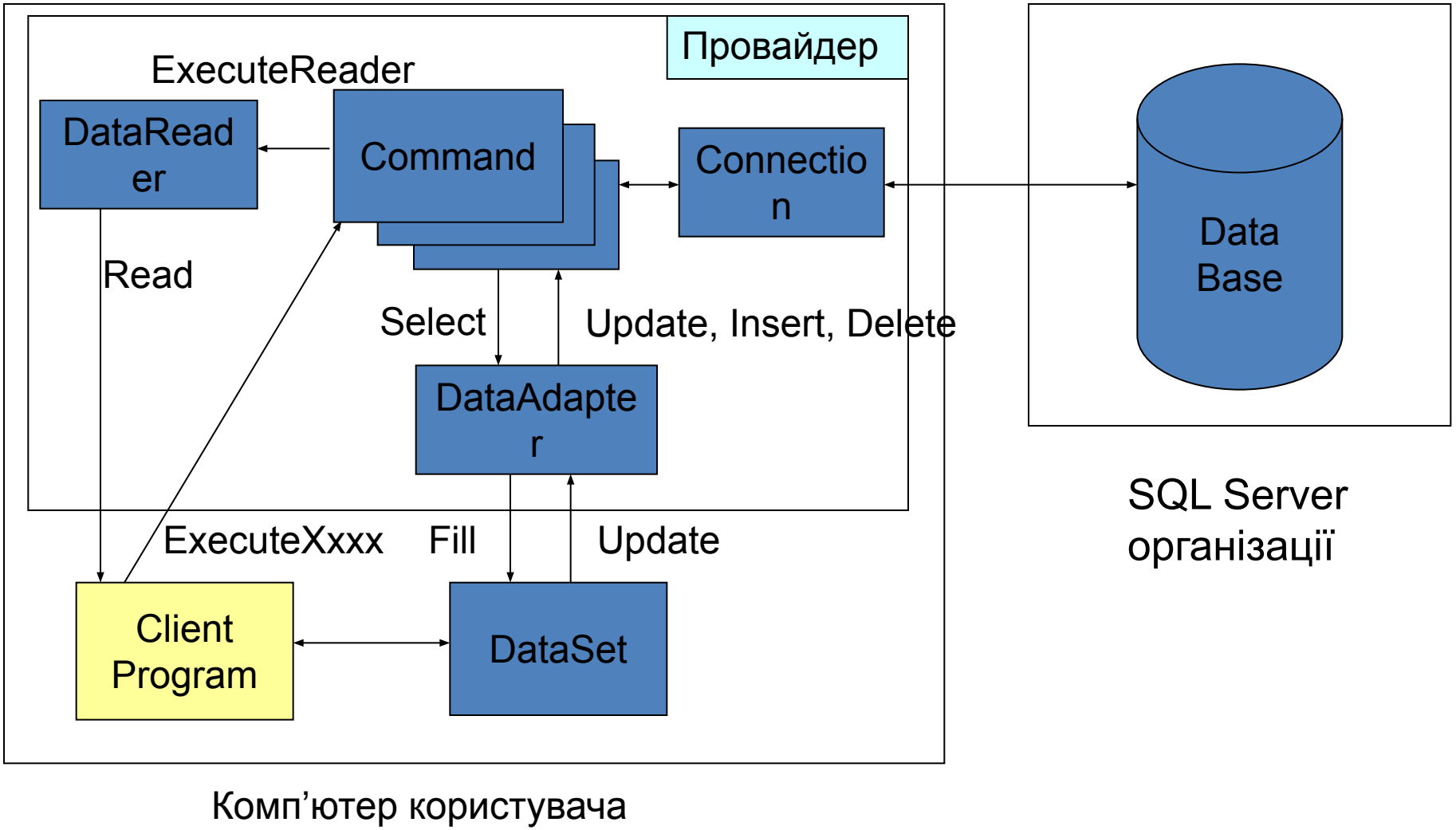
Від'єднаний або *з розривом з'єднання (Disconnected)*:

- Програма робить запит, читає і зберігає результати для обробки, від'єднується від БД
- Виконується робота з даними (додавання, зміна, видалення)
- Мінімізується час з'єднання з базою даних
- Використовується об'єкт **DataSet**

Об'єктна модель ADO.NET



Використання класів ADO.NET



Послідовність роботи з даними у приєднаному режимі

- Установити з‘єднання з БД.
- Виконати запит до БД.
  - Створити та виконати команди
- Отримати результати команди.
- Закрити з‘єднання з БД.

Шаблон роботи з БД у приєднаному режимі

```
1.) Оголосити з'єднання
try {
        1.) Відкрити з БД
```

```
        2.) Створити та виконати команди
```

```
        3.) Обробити результати
```

```
        4.) Звільнити ресурси
} catch ( Exception ) {
        Handle exception
} finally {
        try {
                4.) Закрити з'єднання
        } catch (Exception)
                { Handle exception }
}
```

Клас Connection

- виконує реальний обмін даними між БД та застосуванням
- є частиною Data Provider
- властивості
  - ConnectionString
  - ConnectionTimeout
  - DataBase
- методи
  - Open() – відкрити з‘єднання
  - Close() – закрити з‘єднання

Рядок з’єднання

Рядок з'єднання – це серія пар "ім'я-значення", розділених крапкою з комою (;). Всі разом вони специфікують базову інформацію, необхідну для встановлення з'єднання.

Формат рядка з'єднання:

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

- param – ім’я параметра рядка з'єднання
- val – значення параметра

Основні параметри рядка з’єднання

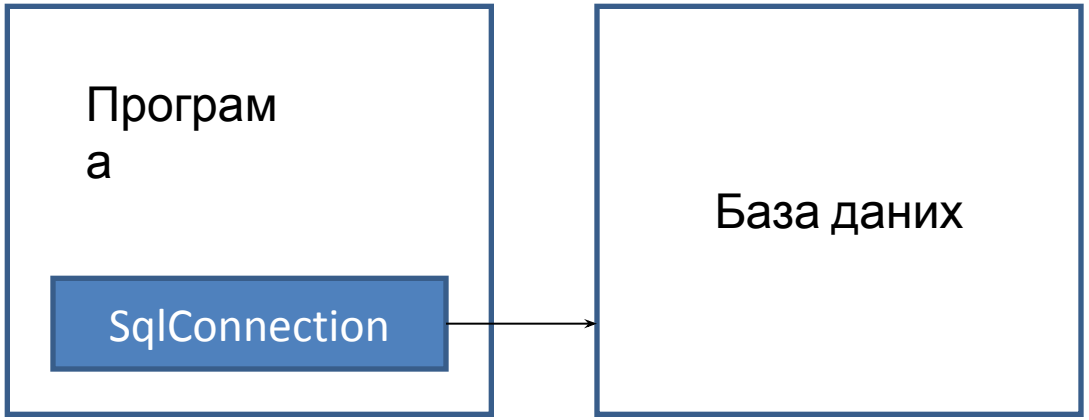
- Data Source=(local)\SQLEXPRESS;
  - (local)
  - localhost
  - . (просто точка)
- Initial Catalog = <ім’я БД>;
- uid=<ідентифікатор>;
- pwd=<пароль>;
- IntegratedSecurity =True;
  - True
  - SSPI
  - yes
- Provider= … (для ODBC та OLEDB)
- ….

Приклади рядків з’єднання

```
string connectionString =
"Data Source=localhost;Initial Catalog=Northwind;" +
"Integrated Security=SSPI";

string connectionString =
"Data Source=localhost;Initial Catalog=Northwind;" +
"user id=sa; password=opensesame";

string connectionString = "Data Source=localhost;Initial Catalog=Sales;" +
"user id=sa;password=;Provider=MSDAORA";

string connectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" +
@"Data Source=C:\DataSources\Northwind.mdb";
```

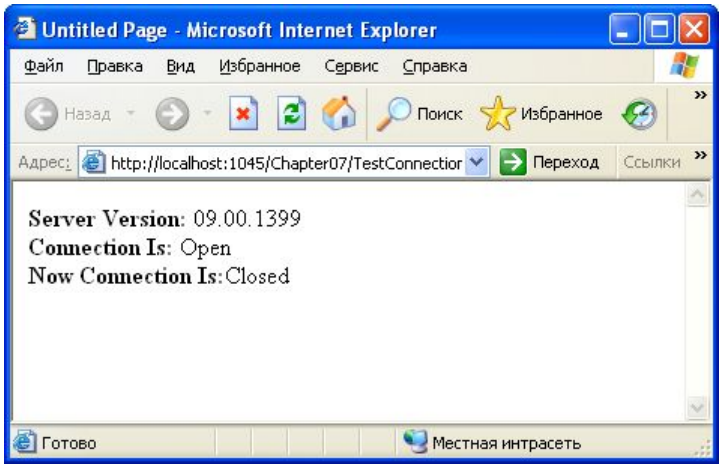
Метод SqlConnection

```
SqlConnection con = new SqlConnection(
@"Data Source=.\SQLEXPRESS2012;Initial" + "Catalog=kurs;Integrated
Security=SSPI");

con.Open();
…
con.Close();
```



Приклад використання об’єкта Connection

```
// Створити об’єкт Connection.
string connectionString =
WebConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
SqlConnection con = new SqlConnection(connectionString);
try
{       // Відкрити з’єднання.
con.Open();
lblInfo.Text = "<b>Server Version:</b> " + con.ServerVersion;
lblInfo.Text += "<br /><b>Connection Is:</b> " + con.State.ToString();
}
catch (Exception err)
{       // Обробка помилки з відображенням інформації.
lblInfo.Text = "Error reading the database. ";
lblInfo.Text += err.Message;
}
finally
{
con.Close();
lblInfo.Text += "<br /><b>Now Connection Is:</b>";
lblInfo.Text += con.State.ToString();
}
```



Клас Command

- Клас Command дозволяє виконувати дії з БД (вибірку, оновлення, доповнення, видалення, тощо).
- Властивості:
  - CommandType:
    - CommandType.Text - оператори SQL;
    - CommandType.TableDirect – робота з конкретною таблицею;
    - CommandType.StoredProcedure – виклик збереженої у БД процедури.
  - CommandText містить:
    - текст оператора SQL (для типу CommandType.Text);
    - ім’я таблиці (для CommandType.TableDirect);
    - ім’я збереженої процедури з параметрами (для CommandType.StoredProcedure);
  - Connection – посилання на відкрите з’єднання (об’єкт Connection);
  - Parameters – колекція параметрів запиту.

Основні методи виконання Command

- ExecuteReader() – виконує оператор SELECT, створює та повертає посилання на об’єкт DataReader, який містить результат виконання запиту.
- ExecuteNonQuery() – виконує оператори INSERT, DELETE, UPDATE на мові SQL (повертає кількість оброблених записів)
- ExecuteScalar() – повертає перший рядок першого стовбця у результуючому наборі (використовуючи функції COUNT, AVG, MIN, MAX, SUM);

Клас SqlCommand

```
SqlCommand cmd = con.CreateCommand();
cmd.CommandText = “INSERT INTO Students(FirstName, LastName)” +
“VALUES(‘Іван’, ‘Петров’)”;
cmd.ExecuteNonQuery();
```

FirstName	LastName
Сергій	Шаргунов
Марія	Шапіро
Василь	Стус
Іван	Петров

```
SqlCommand cmd = new SqlCommand();
cmd.Connection = con;
cmd.CommandType = CommandType.Text;
cmd.CommandText = "SELECT * FROM Employees";

SqlCommand cmd = new SqlCommand("SELECT * FROM Employees", con);

SqlCommand cmd = new SqlCommand("GetEmployees", con);
cmd.CommandType = CommandType.StoredProcedure;
```

Метод ExecuteReader()

- Створює об’єкт DataReader та повертає посилання на нього.
- Текст команди повинен містити оператор Select або виклик збереженої процедури.

Приклад виклику методу ExecuteReader()

```
string connectionString =
WebConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
SqlConnection con = new SqlConnection(connectionString);
string sql = "SELECT * FROM Employees";
SqlCommand cmd = new SqlCommand(sql, con);
con.Open();

SqlDataReader reader = cmd.ExecuteReader();
StringBuilder htmlStr = new StringBuilder("");
while (reader.Read())
{
        htmlStr.Append("<li>");
        htmlStr.Append(reader["TitleOfCourtesy"]);
        htmlStr.Append(" <b>");
        htmlStr.Append(reader.GetString(1));
        htmlStr.Append("</b>, ");
        htmlStr.Append(reader.GetString(2));
        htmlStr.Append(" - employee from ");
        htmlStr.Append(reader.GetDateTime(6).ToString("d"));
        htmlStr.Append("</li>");
}
reader.Close();
con.Close();
HtmlContent.Text = htmlStr.ToString();
```

Клас DataReader

- Об’єкти даного класу дозволяють виконувати лише читання даних з БД, отриманих за допомогою об’єкта Command, тільки в одному напрямку (від початка до кінця).
- Одночасно об’єкт RataReader надає доступ тільки до одного запису вибірки.
- Можна визначити значення поля у записі, використовуючи индексатор
  - dr[n] або dr[“ім’я поля”]

Об’єкт DataReader

- Для переходу до наступного запису вибірки використовується метод

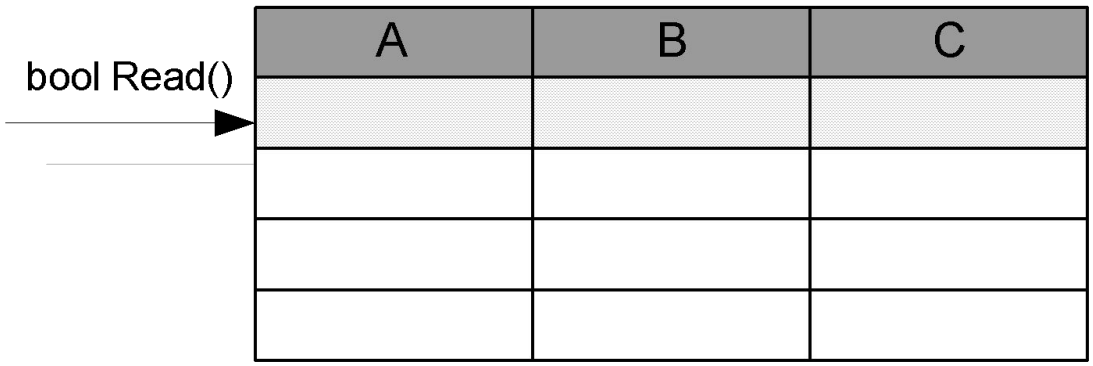
  bool Read() : читає поточний запис та переміщує курсор на наступний запис.
- Якщо метод Read повертає **true**, то наступний запис прочитано, інакше повертається **false**.
- Для закінчення роботи з об’єктом необхідно виконати виклик методу:
  - Close: закінчення роботи з даними в DataReader.

Об’єкт DataReader

- Метод ExecuteReader() повертає посилання на об’єкт DataReader

```
DataReader ExecuteReader()
```

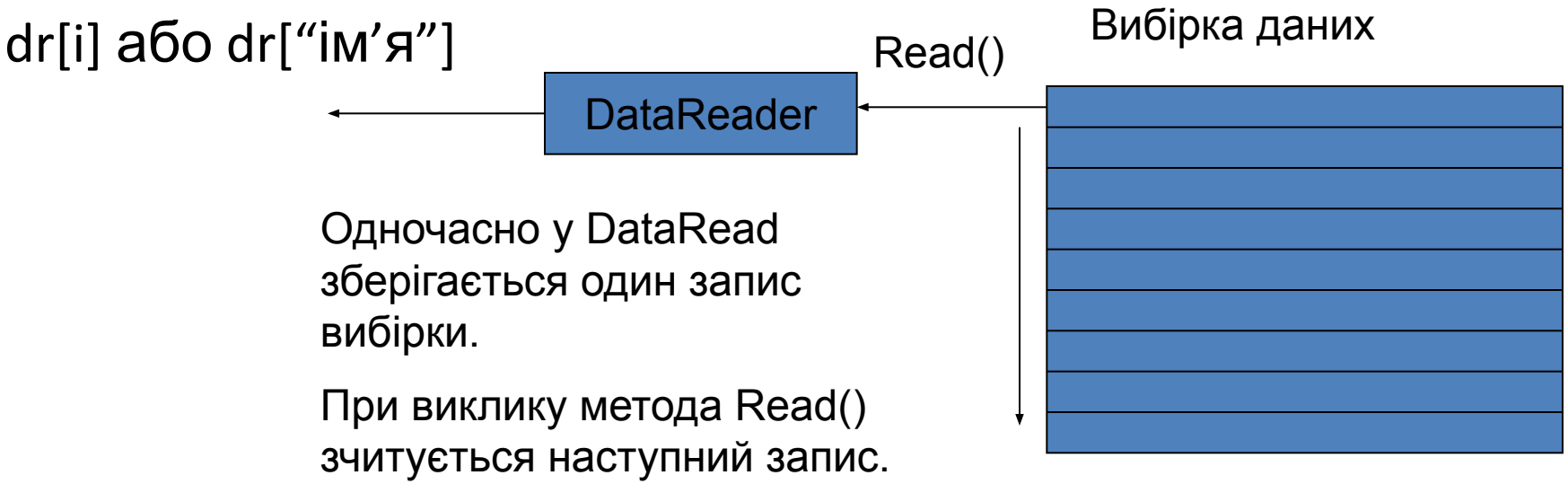
- Об’єкт DataReader дозволяє послідовно читати записи з отриманої вибірки (запис за записом)



Result table of a SELECT statement

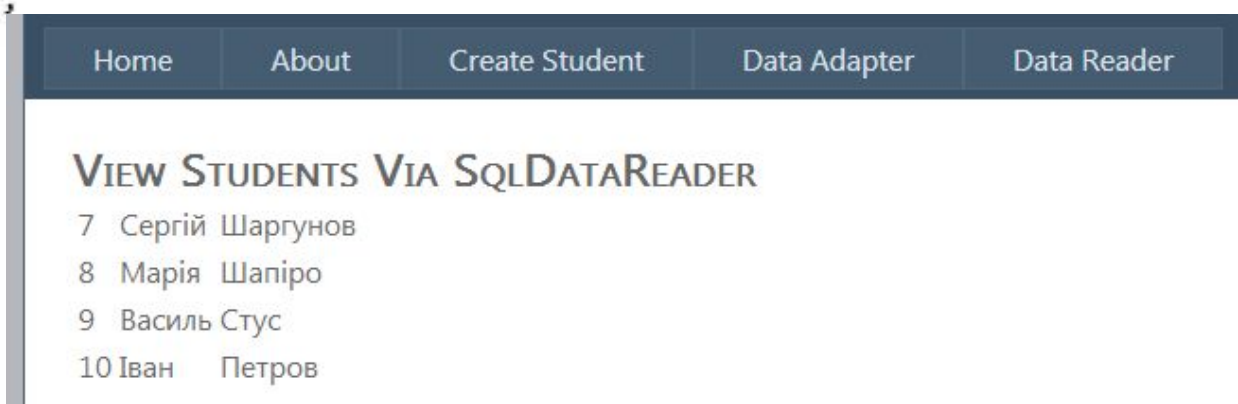
Result table of a SELECT statement

Отримання даних вибірки



Приклад застосування SqlDataReader

```
SqlCommand cmd = con.CreateCommand();
cmd.CommandText = @"SELECT Id, FirstName, LastName FROM Student";
using (SqlDataReader r = cmd.ExecuteReader())
{
    while (r.Read())
    {
        TableRow tr = new TableRow();
        TableCell cell;

        cell = new TableCell();
        cell.Text = r[0].ToString();
        tr.Cells.Add(cell);

        cell = new TableCell();
        cell.Text = r[1].ToString();
        tr.Cells.Add(cell);

        cell = new TableCell();
        cell.Text = r[2].ToString();
        tr.Cells.Add(cell);

        tableStudents.Rows.Add(tr);
    }
}
```



Читання записів за допомогою DataReader

Поля (стовбці) поточного запису можна прочитати двома способами:

1. dtReader[0]
2. dtReader[“ChildId”] // ChildId – ім’я поля запису

Приклад:

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

Метод ExecuteNonQuery()

Дозволяє виконати такі команди:

- команди корегування (повертає кількість змінених записів)
  - 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)
- інші команди, які не повертають значень (результат -1)
  - CREATE DATABASE
  - CREATE TABLE

Приклад виклику 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);
}
```

Метод SqlCommand.ExecuteScalar()

```
cmd.CommandText = “SELECT COUNT(Id) FROM Students”;
int count = (int)cmd.ExecuteScalar();
Console.WriteLine(“Count is: ” + count);
```

Параметри запиту

- У SQL запиті у Command.Text можна задавати змінні – параметри.
- Параметри дозволяють міняти SQL запит без переписування його тексту.
- Параметри використовуються при виклику збереженої процедури для передачі вхідних даних та отримання результатів.
- Для Odbc поля параметра задаються символами «?»

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

- Для OleDbCommand та SqlCommand використовуються іменовані поля параметрів – @Xxxxx

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

Додавання параметрів

- Клас xxxParameter для опису параметрів запиту
  - властивість ParameterName;
  - властивість xxxType (наприклад, SqlDbType);
  - властивість Direction (ParameterDirection.Input; ParameterDirection.Output);
  - властивість Value.

- В об’єкті Command є колекція параметрів (об’єктів **Parameter**) **Parameters**.

- Для використання параметра необхідно створити об’єкт **Parameter** та зберегти його у колекції **Parameters**.

- Методи додавання
  - Add(parameter);
  - AddWithValue(name, value);

Приклад опису параметра

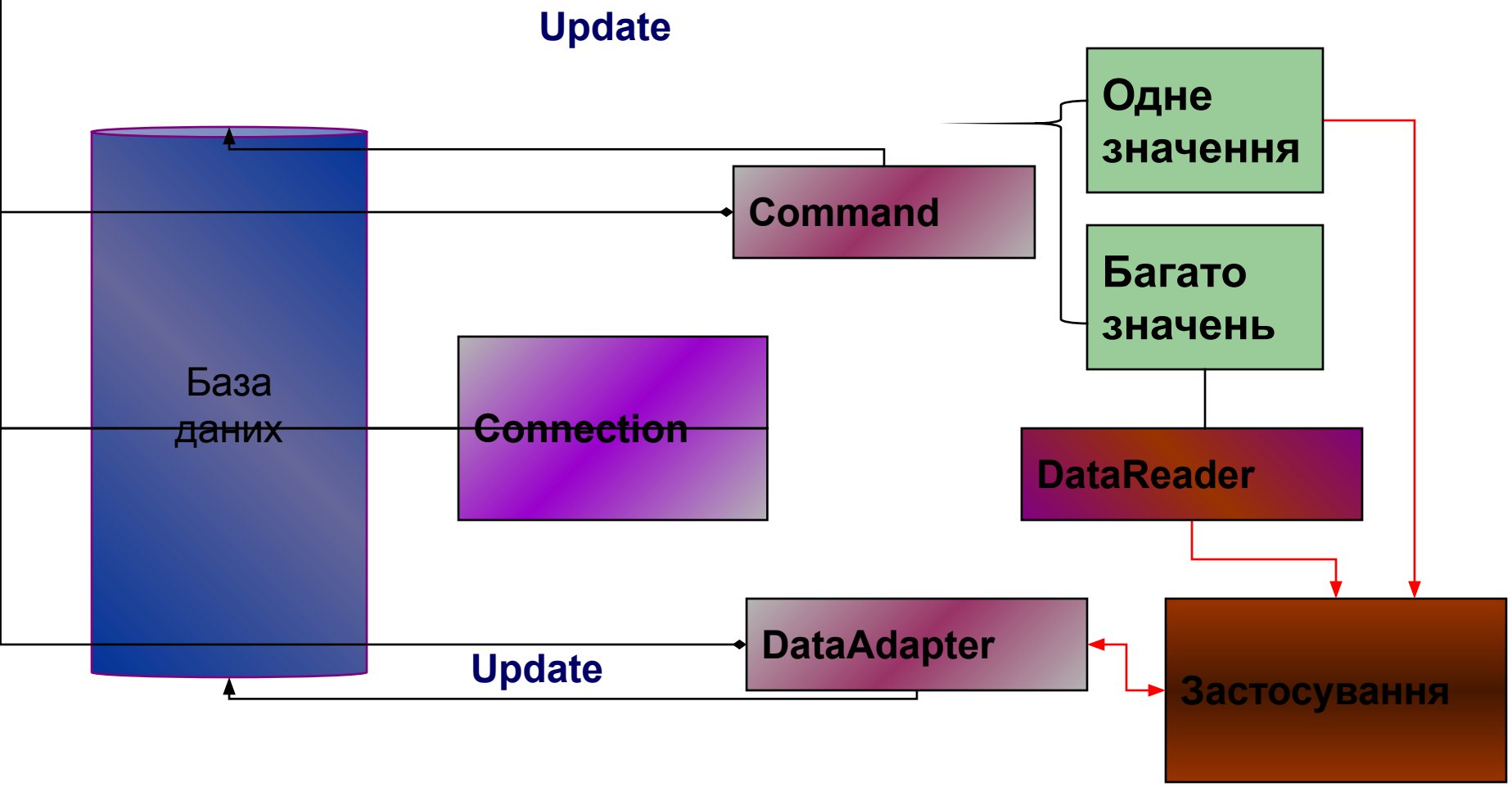
```
string connectionString =
 WebConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
SqlConnection con = new SqlConnection(connectionString);
string sql =
 "SELECT Orders.CustomerID, Orders.OrderID, COUNT(UnitPrice) AS Items,”+
 "SUM(UnitPrice * Quantity) AS Total FROM Orders " +
 "INNER JOIN [Order Details] " +
 "ON Orders.OrderID = [Order Details].OrderID " +
 "WHERE Orders.CustomerID = @CustID " +
 "GROUP BY Orders.OrderID, Orders .CustomerID";
SqlCommand cmd = new SqlCommand(sql, con);
cmd.Parameters.Add("@CustID", txtID.Text);
con.Open();
SqlDataReader reader = cmd.ExecuteReader();
GridView1.DataSource = reader;
GridView1.DataBind();
reader.Close();
con.Close();
```

Лекция №11

Основи технології ADO.NET (частина 2)

Автономні дані *(disconnected)*

Схема доступу до даних



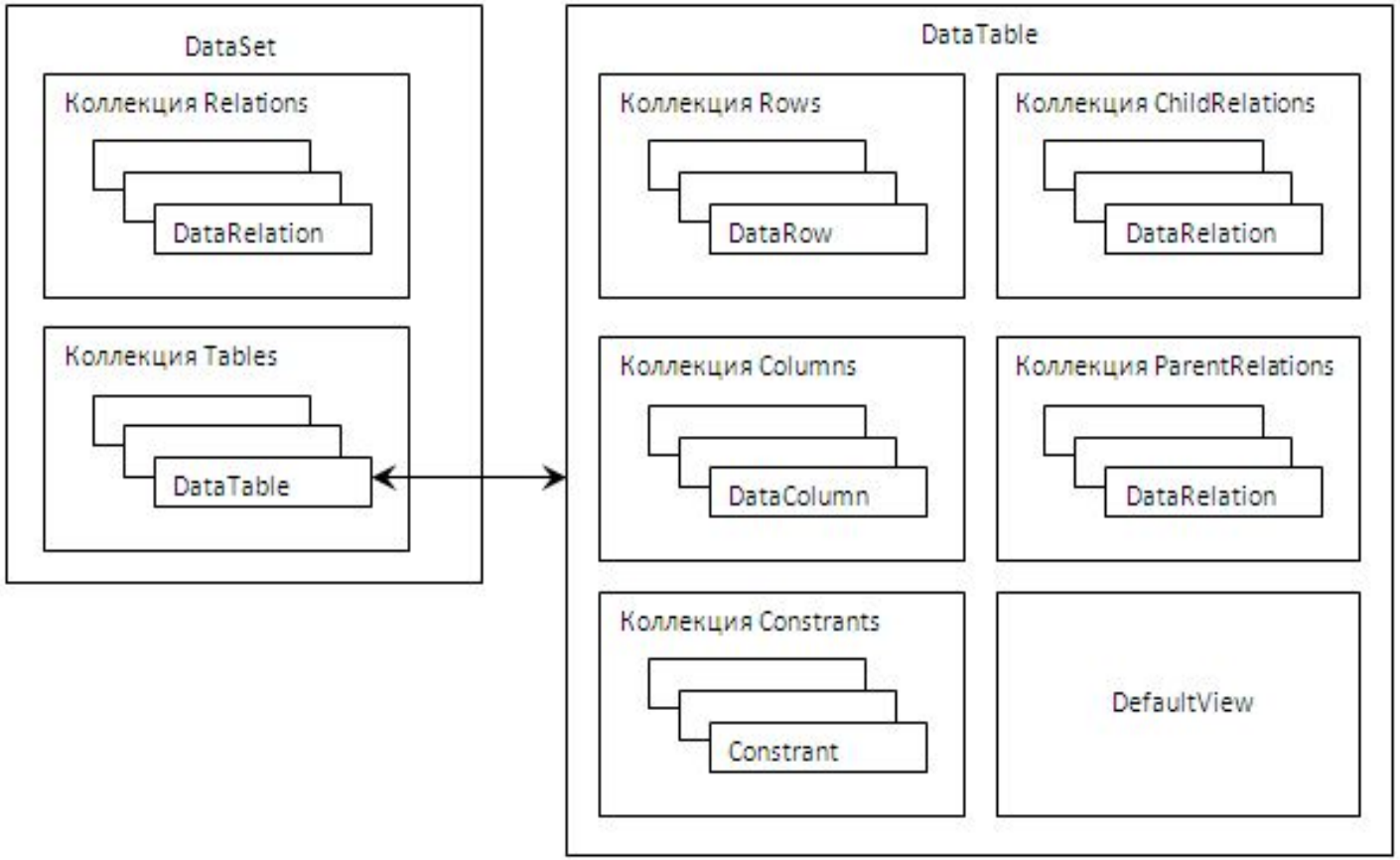
Сценарії, в яких **DataSet** використовувати легше, ніж **DataReader**:

- потрібен зручний пакет для відправки даних іншому компоненту;
- потрібен зручний формат файлу для серіалізації даних на диск;
- потрібно організувати навігацію у двох напрямках по великому об'єму даних;
- потрібно виконувати навігацію по декількох різних таблицях;
- потрібно використовувати прив'язку даних до елементів керування користувацького інтерфейсу;
- необхідно маніпулювати даними як XML;
- необхідно виконувати пакетні оновлення через веб-службу.

Класи **DataSet**

DataSet містить дві важливі складові:

- колекцію з нуля або більше таблиць (властивість **Tables**);
- колекцію з нуля або більше відношень, які можна застосовувати для зв'язування таблиць між собою (властивість **Relationships**).



Методи **DataSet**

Метод	Опис
GetXml() та GetXmlSchema()	Повертають рядок даних (у розмітці XML) або інформацію схеми для **DataSet**.
WriteXml() та WriteXmlSchema()	Зберігають дані та схеми, представлені **DataSet** у файлі або потоці формату XML.
ReadXml() та ReadXmlSchema()	Створюють таблиці в **DataSet** на основі існуючого XML-документа або документа схеми XML.
Clear()	Очищає всі дані таблиць.
Copy()	Повертає точний дублікат **DataSet** з тим же набором таблиць, відношень і даних.
Clone()	Повертає **DataSet** з тією ж структурою (таблицями та відношеннями), але без даних.
Merge()	Приймає інший **DataSet**, як введення та об'єднує його з поточним **DataSet**, додаючи нові таблиці і об'єднуючи дані в існуючих.

Типізований **DataSet**

- ADO.NET підтримує типізований **DataSet**.
- Перетворення типів при доступі до результатів для нього не потрібні:

```
string myBook = dsBooks.Books[0].Name;
```

- Насправді це зовсім інший клас, похідний від **DataSet**, структура якого визначається XSD-файлом (схема XML), в якому, зокрема, описані імена таблиць і стовпців.
- Структура даних повинна бути відома на період компіляції.

Модифікація та оновлення

- Обєкт **DataSet** можна редагувати на клієнтській машині: редагувати записи, додавати або видаляти **DataRow**
- Ці зміни потраплять до БД після застосування методу **DataAdapter.Update**
- **RowState** – стан рядка даних Unchanged/Modified/Added/Deleted/Detached/

Об’єкт **DataRelation**

- Об’єкт **DataRelation** представляє зв'язок між двома полями різних таблиць
- Набір **Relation** – властивість об'єкта **DataSet**
- Є можливість отримувати пов'язані записи за допомогою методів **GetChildRows** та **GetParentRows**

Клас **DataAdapter**

DataAdapter служить мостом між одним **DataTable** в **DataSet** та джерелом даних, включає всі доступні команди для виконання запитів та оновлення джерела даних.

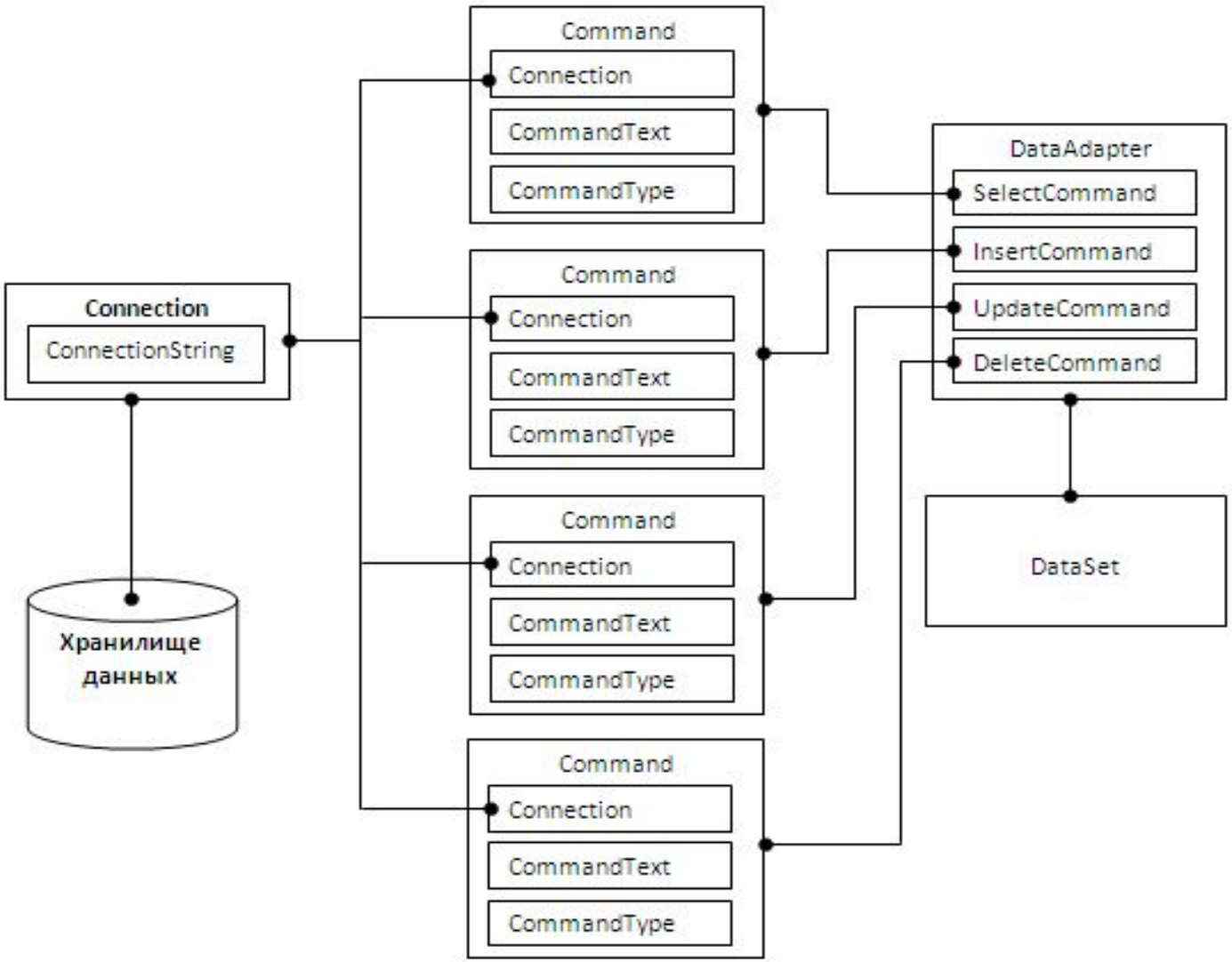
Ключові методи:

Метод	Опис
Fill()	Додає **DataTable** до **DataSet** за рахунок виконання запиту в **SelectCommand**.
FillSchema()	Додає **DataTable** до **DataSet** за рахунок виконання запиту в **SelectCommand** і витягання тільки інформації про схему.
Update()	Перевіряє всі зміни в окремій **DataTable** і застосовує пакет цих змін до джерела даних за допомогою виконання відповідних операцій **InsertCommand**, **UpdateCommand** та **DeleteCommand**.

DataReader vs. DataAdapter

- DataReader допускає швидке та ефективне односпрямова-не читання даних
- Менш гнучкий, ніж DataAdapter (не можна редагувати дані, не можна повернутися до прочитаного раніше запису, вимагає монопольного доступу до активного з'єднання)

Взаємодія **DataAdapter** з джерелом даних

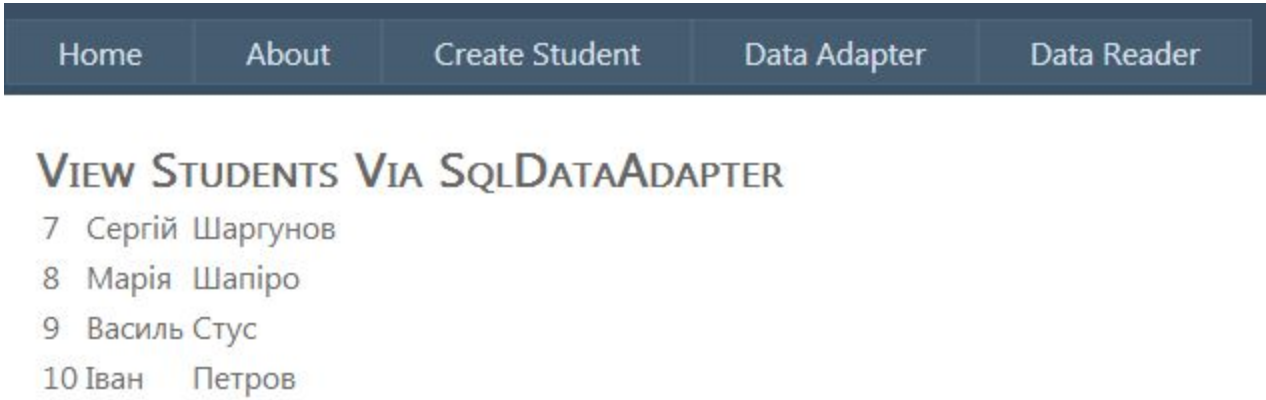


Приклад. Наповнення **DataSet**

```
// Створення з’єднання та визначення тексту запиту
string connectionString =
WebConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
SqlConnection con = new SqlConnection(connectionString);
string sql = "SELECT * FROM Employees";

// Створення екземпляру класу DataAdapter
SqlDataAdapter da = new SqlDataAdapter(sql, con);

// Створення пустого DataSet та виконання методу Fill()
DataSet ds = new DataSet();
da.Fill(ds, "Employees");

// Відображення вмісту DataSet
StringBuilder htmlStr = new StringBuilder("");
foreach (DataRow dr in ds.Tables["Employees"].Rows)
{
htmlStr.Append("<li>");
htmlStr.Append(dr["TitleOfCourtesy"].ToString());
htmlStr.Append(" <b>");
htmlStr.Append(dr["LastName"].ToString());
htmlStr.Append("</b>, ");
htmlStr.Append (dr["FirstName"] .ToString());
htmlStr.Append("</li>");
}
```

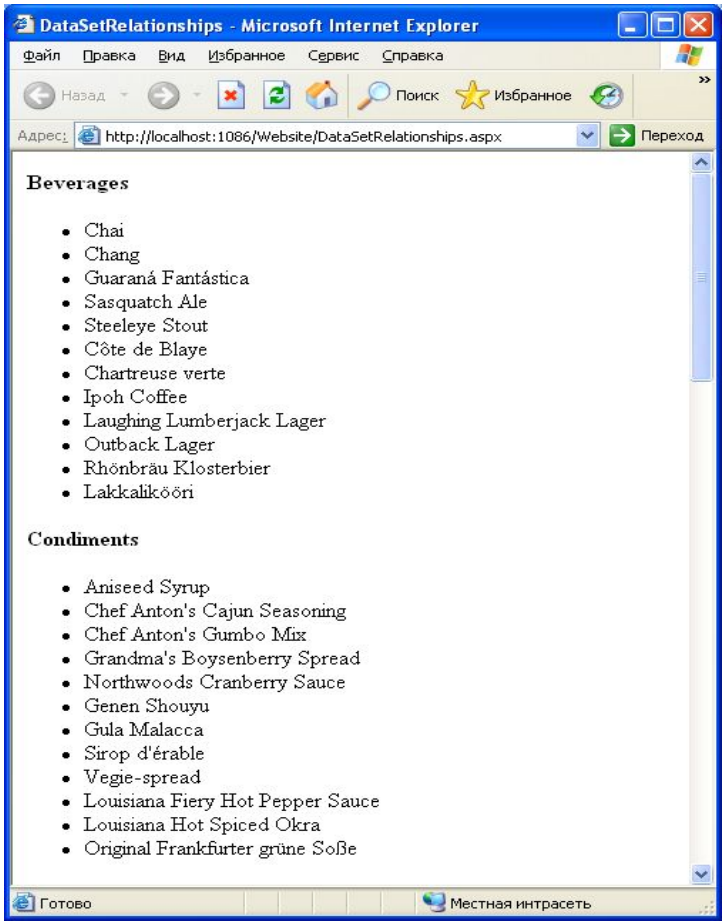
Приклад. Наповнення **DataSet**

```
// Створення з'єднання та визначення тексту запиту
SqlConnection con = new SqlConnection(@"Data Source=.\SQLEXPRESS2008R2;Initial Catalog=kurs;Integrated Security=SSPI");
con.Open();
SqlCommand cmd = con.CreateCommand();
cmd.CommandText = @"SELECT Id, FirstName, LastName FROM Student";

// Створення екземпляру класу DataAdapter
SqlDataAdapter da = new SqlDataAdapter(cmd);
// Створення пустого DataSet та виконання методу Fill()
DataSet ds = new DataSet();
da.Fill(ds);
// Закрити з'єднання та DataAdapter
da.Dispose();
cmd.Dispose();
con.Close();
foreach (DataRow row in ds.Tables[0].Rows)
{
    TableRow tr = new TableRow();
    TableCell cell;

    cell = new TableCell();
    cell.Text = row[0].ToString();
    tr.Cells.Add(cell);

    cell = new TableCell();
    cell.Text = row[1].ToString();
    tr.Cells.Add(cell);

    cell = new TableCell();
    cell.Text = row[2].ToString();
    tr.Cells.Add(cell);

    tableStudents.Rows.Add(tr);
}
```



Робота з множинними таблицями та відношеннями

```
string connectionString =
WebConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
SqlConnection con = new SqlConnection(connectionString);
string sqlCat = "SELECT CategoryID, CategoryName FROM Categories";
string sqlProd = "SELECT ProductName, CategoryID FROM Products";
SqlDataAdapter da = new SqlDataAdapter(sqlCat, con);
DataSet ds = new DataSet();
try
{       con.Open();
// Наповнити DataSet даними з таблиці Categories
da.Fill(ds, "Categories");
// Змінити текст команди та отримати дані
// таблиці Products.
da.SelectCommand.CommandText = sqlProd;
da.Fill(ds, "Products");
}
finally
{       con.Close();
}
// Продовження див. на наступному слайді
```



Робота з множинними таблицями та відношеннями (2)

```
// Визначити відношення між Categories та Products.
DataRelation relat = new DataRelation("CatProds",
ds.Tables["Categories"].Columns["CategoryID"],
ds.Tables["Products"].Columns["CategoryID"]);
// Додати відношення до DataSet.
ds.Relations.Add(relat);
StringBuilder htmlStr = new StringBuilder("");
// Пройти у цикліпо всіх записах категоріях та побудувати строку HTML.
foreach (DataRow row in ds.Tables["Categories"].Rows)
{       htmlStr.Append("<b>");
htmlStr.Append(row["CategoryName"].ToString());
htmlStr.Append("</b><ul>");
// Получить дочерние (products) записи для родителя (category).
DataRow[] childRows = row.GetChildRows(relat);
// Пройти по всем продуктам данной категории.
foreach (DataRow childRow in childRows)
{
htmlStr.Append("<li>");
htmlStr.Append(childRow["ProductName"].ToString());
htmlStr.Append("</1i>");
}
htmlStr.Append("</ul>");
}
HtmlContent.Text = htmlStr.ToString();
```

Пошук визначених рядків

Метод **Select()** класу **DataTable** дозволяє отримувати масив об'єктів **DataRow** на основі SQL-виразу.

Приклад:

```
// Отримати записи з таблиці Products
DataRow[] matchRows = DataSet.Tables["Products"].Select("Discontinued = 0");

// Пройти по усіх продуктах, що мають знижки та сгенерувати перелік
htmlStr.Append("<ul>");
foreach (DataRow row in matchRows)
{
	htmlStr.Append("<li>");
	htmlStr.Append(row["ProductName"].ToString());
	htmlStr.Append("</li>");
}
htmlStr.Append("</ul>");
```

Прив'язка даних

Ключова ідея прив'язки даних полягає у асоціюванні зв'язку між об'єктом даних та елементом керування, а про побудову відповідного виводу піклується інфраструктура прив'язки даних ASP.NET.

Приклад:

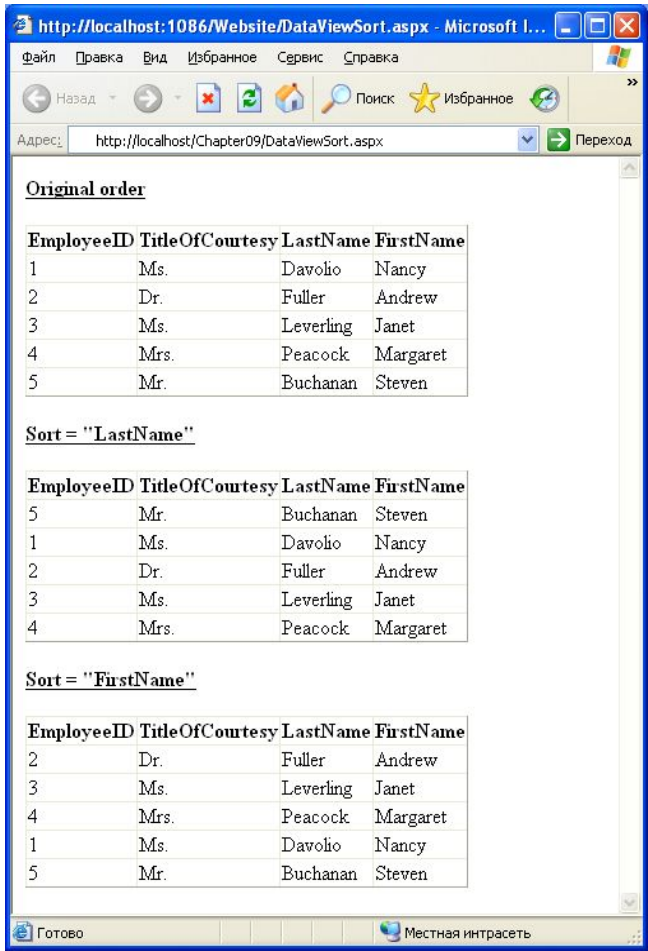
```
GridView1.DataSource = ds;
GridView1.DataMember = "Employees";
GridView1.DataBind();
```

Клас **DataView**

DataView служить

Сортування за допомогою **DataView**

```
// Створити Connection, DataAdapter та DataSet
string connectionString =
WebConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
SqlConnection con = new SqlConnection(connectionString);
String sql =
"SELECT TOP 5 EmployeeID, TitleOfCourtesy, LastName, FirsName FROM Employees";
SqlDataAdapter da = new SqlDataAdapter(sql, con);
DataSet ds = new DataSet();
// Наповнити DataSet
da.Fill(ds, "Employees");

// Прив’язати оригінальні дані до элемента №1.
grid1.DataSource = ds.Tables["Employees"];
// Сортувати за прізвищем та прив’язати до елемента №2.
DataView view2 = new DataView(ds.Tables["Employees"]);
view2.Sort = "LastName";
grid2.DataSource = view2;
// Сортувати за іменем та прив’язати до елемента №3.
DataView view3 = new DataView(ds.Tables["Employees"]);
view3.Sort = "FirstName";
grid3.DataSource = view3;

// Ініціювати процес прив'язки даних.
Page.DataBind();
```



Операції фільтрації

Операція	Опис
<, >, <= та >=	Виконують порівняння більш ніж одного значення.
<> та =	Виконують перевірку на еквівалентність.
NOT	Звертає на протилежний логічний вираз. Може використовуватися у поєднанні з будь-якої іншої конструкцією.
BETWEEN	Вказує діапазон включно. Наприклад, Units BETWEEN 5 AND 15 вибирає рядки, у яких значення стовпця Units знаходиться в межах від 5 до 15.
IS NULL	Перевіряє стовпець на null-значення.
IN(a, b, c)	Коротка форма операції OR з одним і тим же полем. Перевіряє еквівалентність значення стовпця будь-якого з перерахованих значень (a, b та c).
LIKE	Виконує перевірку відповідності рядкового значення шаблону.
+	Складає два числа або "склеює" два рядка.
-	Віднімає одне числове значення з іншого.
*	Перемножує два числових значення.
/	Ділить одне числове значення на інше
%	Обчислює модуль (залишок від ділення одного числа на інше).
AND	Комбінує більше однієї логічної конструкції.
OR	Комбінує більше однієї логічної конструкції.

Фільтрація за допомогою **DataView**

```
string connectionString =
WebConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
SqlConnection con = new SqlConnection(connectionString);
string sql = "SELECT ProductID, ProductName, UnitsInStock, UnitsOnOrder, " +
"Discontinued FROM Products";
SqlDataAdapter da = new SqlDataAdapter(sql, con);
DataSet ds = new DataSet();
da.Fill(ds, "Products");

// Фільтрувати продукт Chocolade.
DataView view1 = new DataView(ds.Tables["Products"]);
view1.RowFilter = "ProductName = 'Chocolade'";
GridViewl.DataSource = view1;

// Фільтрувати продукты, яких немає у замовленнях та на складі.
DataView view2 = new DataView(ds.Tables["Products"]);
view2.RowFilter = "UnitsInStock = 0 AND UnitsOnOrder = 0";
GridView2.DataSource = view2;

// Фільтрувати продукти, чия назва починається з букви P.
DataView view3 = new DataView(ds.Tables["Products"]);
view3.RowFilter = "ProductName LIKE 'P%'";
GridView3.DataSource = view3;

this.DataBind();
```

Розширене фільтрування з відношеннями

DataView дозволяє застосовувати деякі складні вирази фільтрації, наприклад, на основі відношень.

Щоб створити такий фільтруючий вираз, необхідно скомбінувати дві складові:

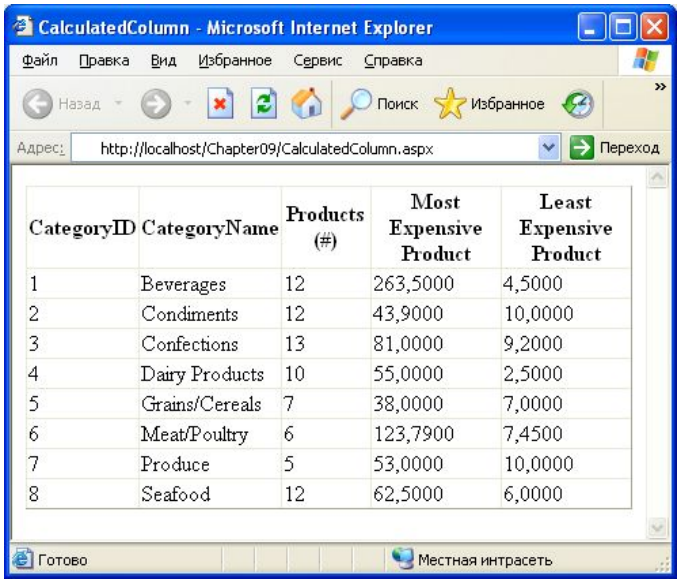
- відношення, що пов'язує дві таблиці;
- агрегатну функцію, таку як AVG(), MAX(), MIN() або COUNT(). Ця функція застосовується до даних у пов'язаних записах.

Приклад:

```
// Визначення відношення між Categories та Products.
DataRelation relat = new DataRelation ("CatProds",
ds.Tables["Categories"].Columns["CategoryID"],
ds.Tables["Products"].Columns["CategoryID"]);
// Додати відношення до DataSet.
ds.Relations.Add(relat);

// Застосування рядка з умовою до GridView
DataView view1 = new DataView(ds.Tables["Categories"));
viewl.RowFilter = "MAX(Child(CatProds).UnitPrice) > 50";
GridView1.DataSource = view1;
```

Обчислювані стовпці

```
string connectionString =
WebConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
SqlConnection con = new SqlConnection(connectionString);
string sqlCat = "SELECT CategoryID, CategoryName FROM Categories";
string sqlProd = "SELECT ProductName, CategoryID, UnitPrice FROM Products";
SqlDataAdapter da = new SqlDataAdapter(sqlCat, con);
DataSet ds = new DataSet();
...
// Визначення відношення між Categories та Products.
DataRelation relat = new DataRelation("CatProds",
ds.Tables["Categories"].Columns["CategoryID"],
ds.Tables["Products"].Columns["CategoryID"]);
// Додати відношення до DataSet.
ds.Relations.Add(relat);
// Створити обчислювані стовпці
DataColumn count = new DataColumn(
"Products (#)", typeof(int), "COUNT(Child(CatProds).CategoryID)");
DataColunm max = new DataColumn(
"Most Expensive Product", typeof(decimal), "MAX(Child(CatProds).UnitPrice)");
DataColumn min = new DataColumn(
"Least Expensive Product", typeof(decimal), "MIN(Child(CatProds).UnitPrice)");
// Додати стовбці
ds.Tables["Categories"].Columns.Add(count);
ds.Tables["Categories"].Columns.Add(max);
ds.Tables["Categories"].Columns.Add(min);
// Показати дані
GridView1.DataSource = ds.Tables["Categories"];
GridView1.DataBind();
```



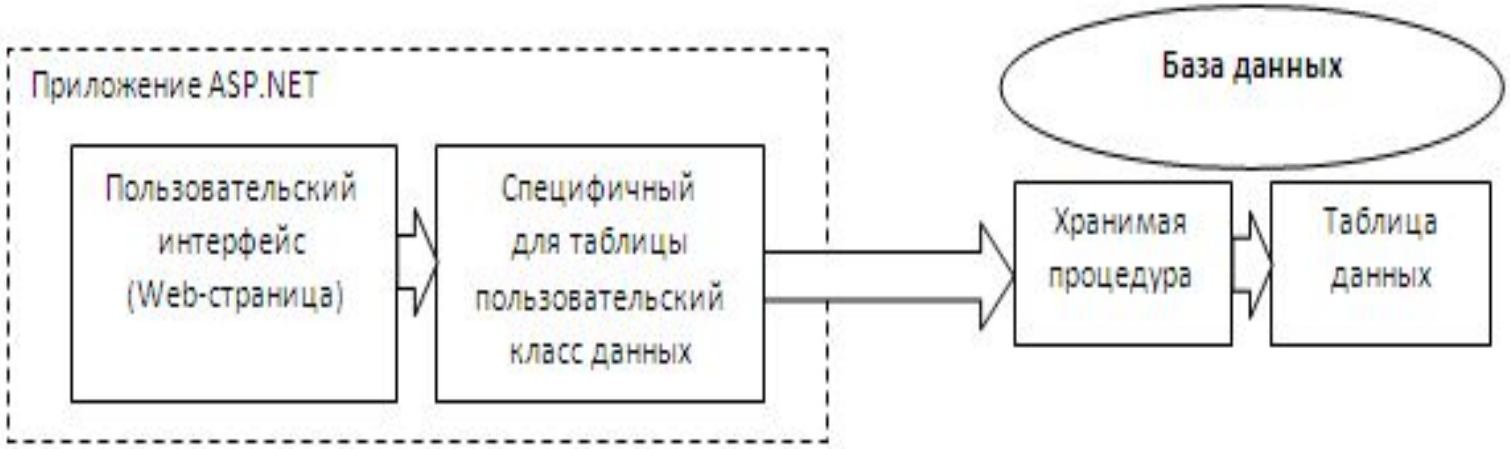
CategoryID	CategoryName	Products (#)	Most Expensive Product	Least Expensive Product
1	Beverages	12	263,5000	4,5000
2	Condiments	12	43,9000	10,0000
3	Confections	13	81,0000	9,2000
4	Dairy Products	10	55,0000	2,5000
5	Grains/Cereals	7	38,0000	7,0000
6	Meat/Poultry	6	123,7900	7,4500
7	Produce	5	53,0000	10,0000
8	Seafood	12	62,5000	6,0000

Побудова компонента доступу до даних

Основні рекомендації при створенні класу даних:

- *Відкривайте і закривайте з'єднання швидко.*
- *Реалізуйте обробку помилок.*
- *Слідуйте практиці дизайну без станів.*
- *Не дозволяйте клієнтові вказувати інформацію рядка з'єднання.*
- *Не підключайтеся під клієнтським ідентифікатором користувача.*
- *Не дозволяйте клієнтам використовувати широкі відкриті запити.*

Багатошаровий дизайн класу бази даних



Литература

1. *Мак-Дональд, Мэтью, Шпушта, Марио.* Microsoft ASP.NET 2.0 с примерами на C# 2005 для профессионалов.: Пер. с англ. – М.: ООО "И.Д. Вильямс", 2007. – 1408 с.
2. *Рихтер Дж.* CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C#. 3-е изд. – СПб.: Питер, 2012. – 928 с.

Кінець

Дякую за увагу

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
{
. . .
}
```

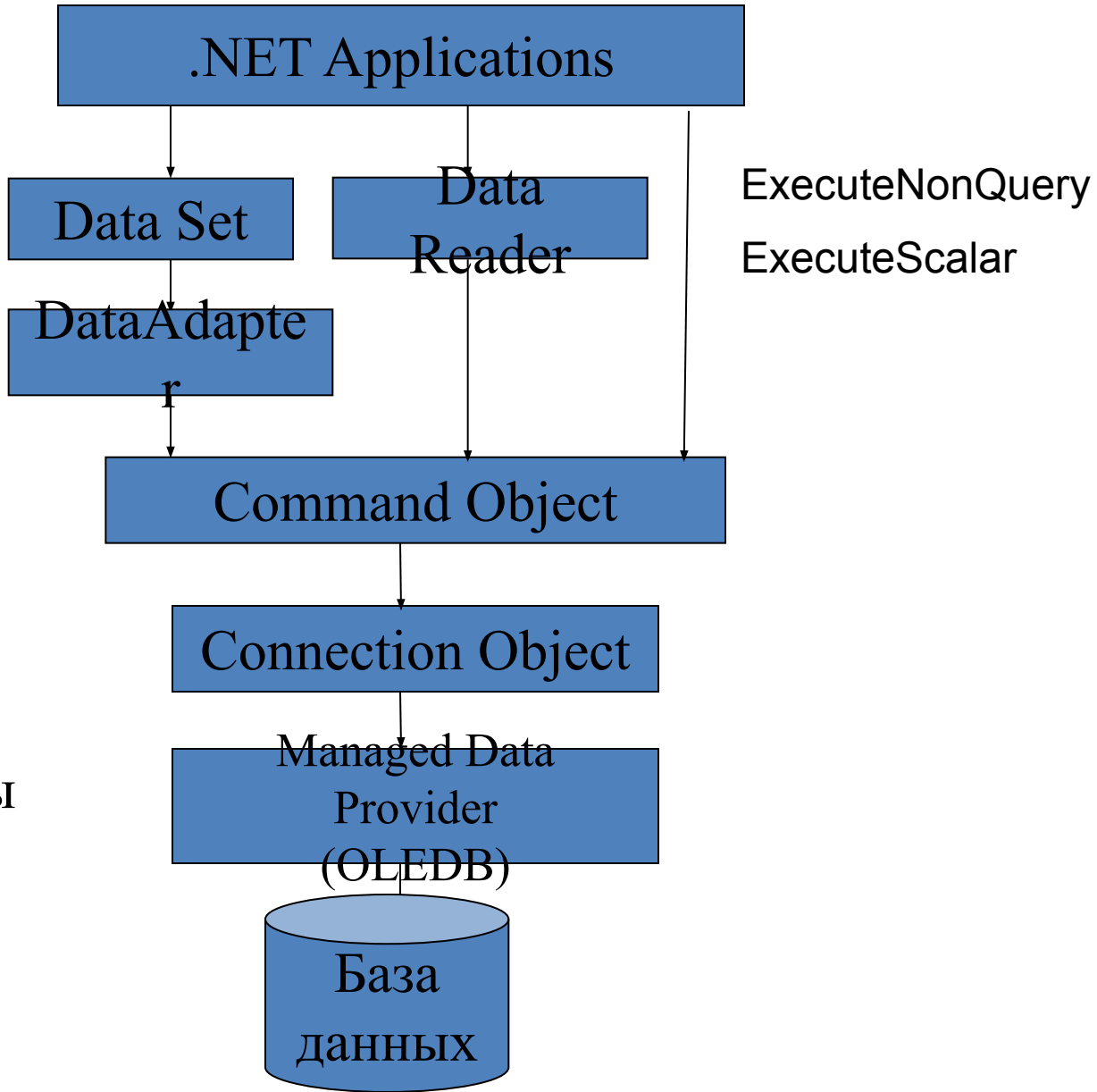
Технології Microsoft для роботи з БД

- 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.

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

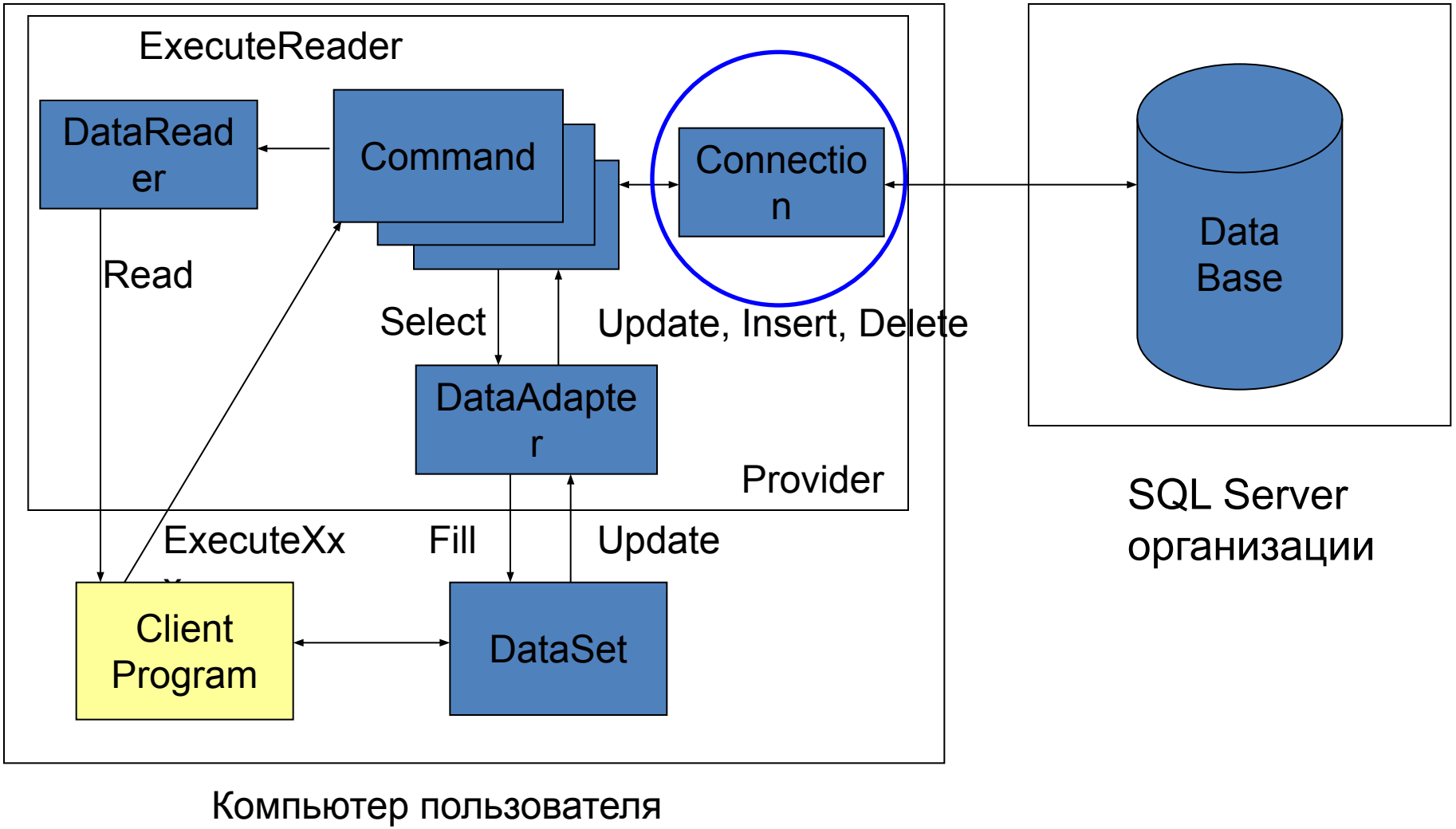
Объекты ADO.NET



Провайдер базы данных

Отсоединенный режим работы с БД

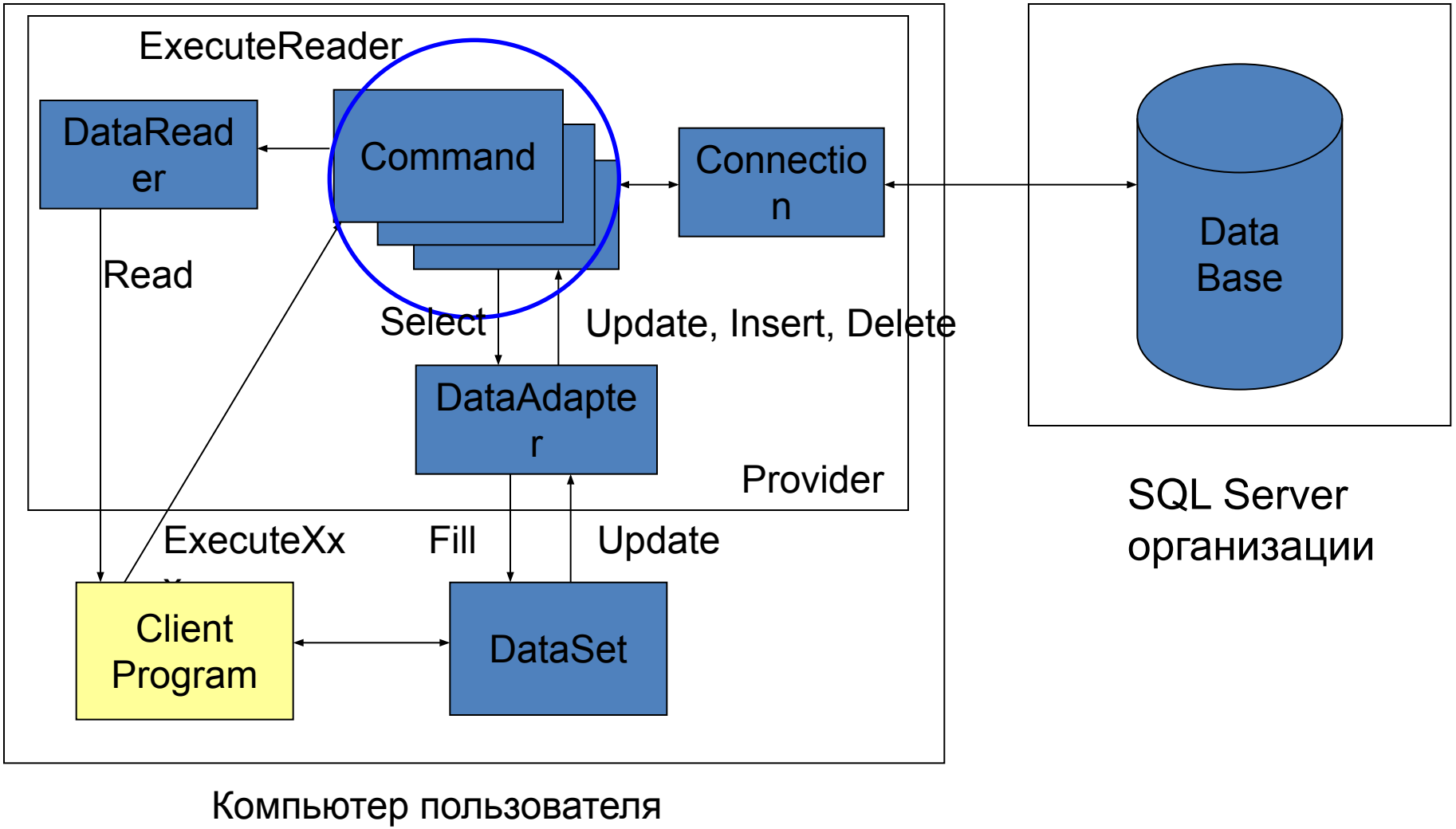
Использование классов ADO.NET



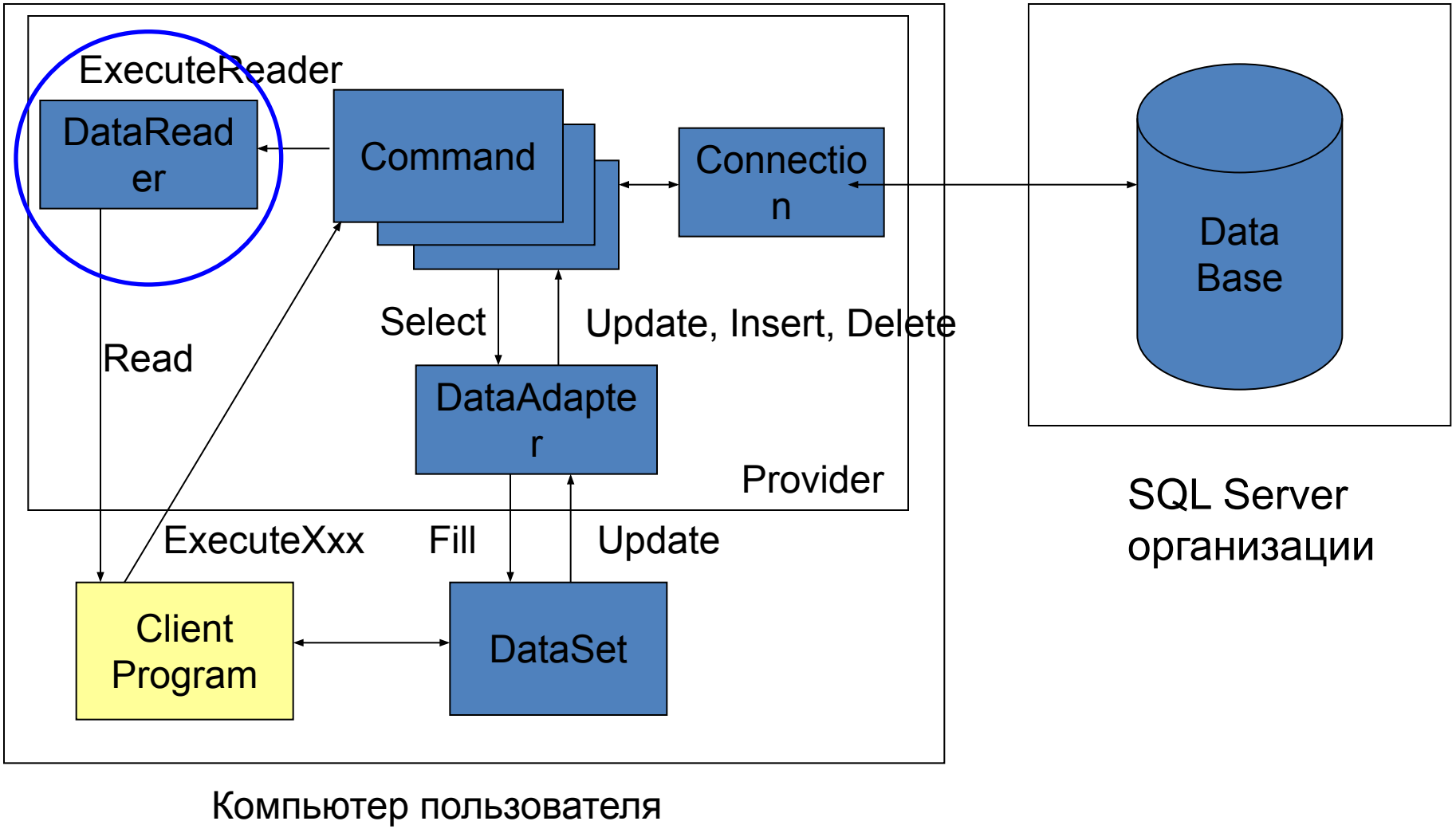
Рядок з‘єднання

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

Использование классов ADO.NET



Использование классов ADO.NET



Клас 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();
}
```

Об’єкт **DataAdapter**

- Об’єкт **DataAdapter** – основний клас ADO.NET, що забезпечує доступ до від’єднаних даних.
- DataAdapter – посередник між БД та об’єктом **DataSet**
- Методи **Fill()** (заповнення **DataSet**) та **Update()** (оновлення БД)
- Властивості **Select/Insert/Delete/UpdateCommand**

Робота з декількома наборами

У CommandText можна помістити декілька команд SQL та користуватися **DataReader.NextResults()** для читання наборів результатів.

Обмеження **Constraints**

- Ограничение – правила работы со строками в DataTable
- Два вида ограничений: UniqueConstraint и ForeignKeyConstraint
- UniqueConstraint – запрещение добавлять в таблицы дублирующиеся элементы
- ForeignKeyConstraint – правила обновления дочерних строк при изменении родительской
- Набор Constraints – свойство таблицы

ForeignKeyConstraint

- ForeignKey – внешний ключ, обычно первичный ключ в другой таблице
- Определяемые правила: UpdateRule, DeleteRule, AcceptRejectRule
- Свойства ограничения:
  - Cascase – модификации родительской строки реплицируются в дочерние
  - None – ничего не делается
  - SetDefault – значение внешнего ключа устанавливается в умолчательное
  - SetNull – значение внешнего ключа устанавливается в DBNull

Фильтрация и сортировка данных

- После заполнения объекта DataSet обычно работают с подмножеством его данных
- Для этого используют объект DataView (in-memory аналог SQL CREATE VIEW)
- DataView позволяет фильтровать и сортировать данные, а также обновлять их в связанном DataTable
- Свойство RowFilter
- Свойство RowState – фильтрация данных на основе их состояния
- Свойство Sort