Лекція 8

РОБОТА З БАЗАМИ ДАНИХ ІЗ ВИКОРИСТАННЯМ ADO.NET

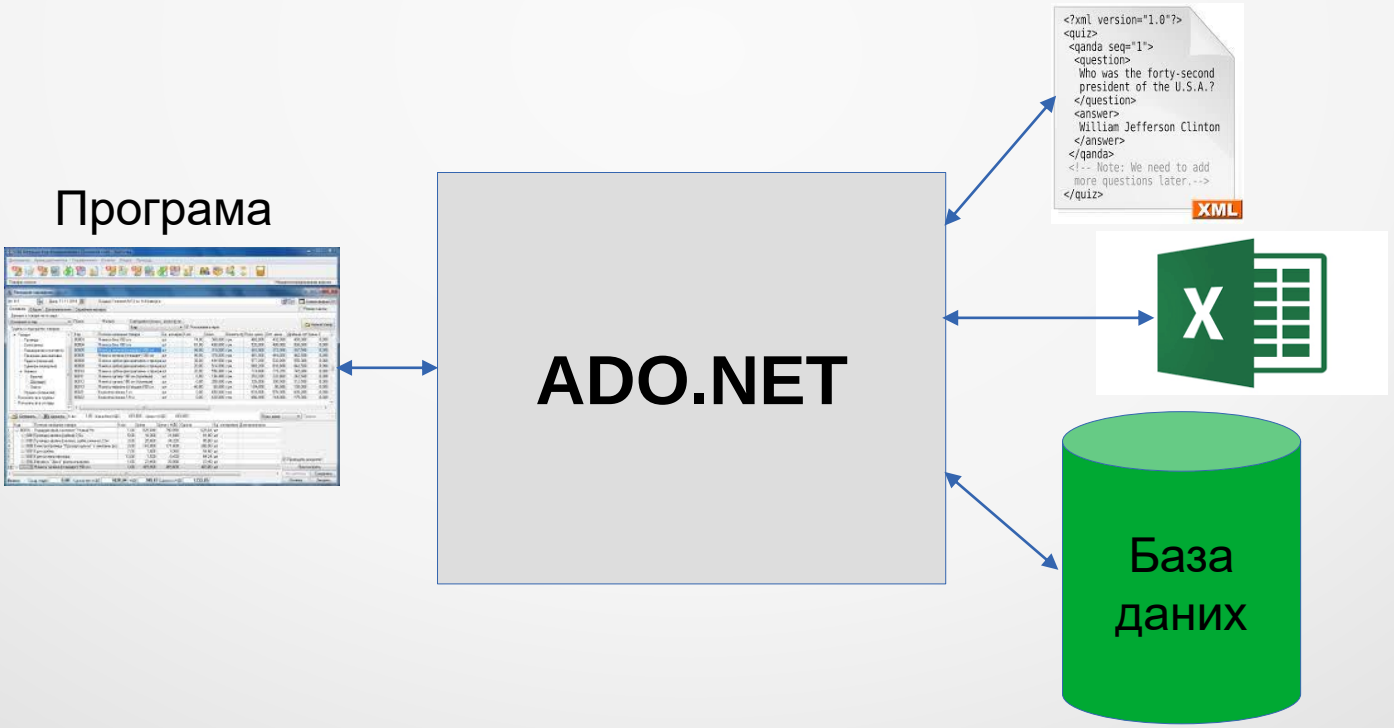
Лекція 8. Робота з базами даних із використанням ADO.NET

План

1. Концепція ADO.NET.
2. Постачальники даних ADO.NET.
3. Підключений рівень ADO.NET.
4. Транзакції.

1. Поняття ADO.NET

ADO.NET (ActiveX Data Object для .NET) – це реалізована у вигляді набору бібліотек для .NET Framework технологія доступу та управління даними, що зберігаються або в деякій базі даних (Microsoft SQL Server, Microsoft Access, Oracle, MySQL тощо), або в інших джерелах (Microsoft Excel, Microsoft Outlook, Microsoft Exchange, XML, текстові файли і т.п.).



1. Поняття ADO.NET

Технологію ADO.NET можна використовувати з допомогою трьох принципово різних методів:

1) з використанням **підключеного режиму**, коли програма явно підключається до джерела даних і працює з ним;

2) в **автономному режимі**, коли програма працює з копією даних із зовнішнього джерела (отримавши копію даних застосунок більше не витрачає мережевий трафік до моменту запису змін);

3) з використанням **технології Entity Framework** (EF), яка реалізує об'єктно-орієнтовану взаємодію з джерелом даних із застосуванням **LINQ to Entities** або **Entity SQL**.

2. Постачальники даних ADO.NET

Для доступу до різних джерел даних у платформі .NET є багато різних постачальників даних, кожен з яких орієнтований на взаємодію з певною СУБД чи типом файлу (табл. 1).

Таблиця 1. Стандартні постачальники даних Microsoft ADO.NET

Постачальник даних	Простір імен	Опис
OLE DB	System.Data.OleDb	Доступ до джерел даних, що підтримують інтерфейс OLE DB на основі COM (вважається застарілим, оскільки є досить повільним)
Microsoft SQL Server	System.Data.SqlClient	Надає прямий доступ до баз даних Microsoft SQL Server
Microsoft SQL Server Mobile	System.Data.SqlServerCe	Надає прямий доступ до баз даних Microsoft SQL Server з мобільних пристроїв
ODBC	System.Data.Odbc	Доступ до джерел даних, які підтримують інтерфейс ODBC (Open DataBase Connectivity)

2. Постачальники даних ADO.NET

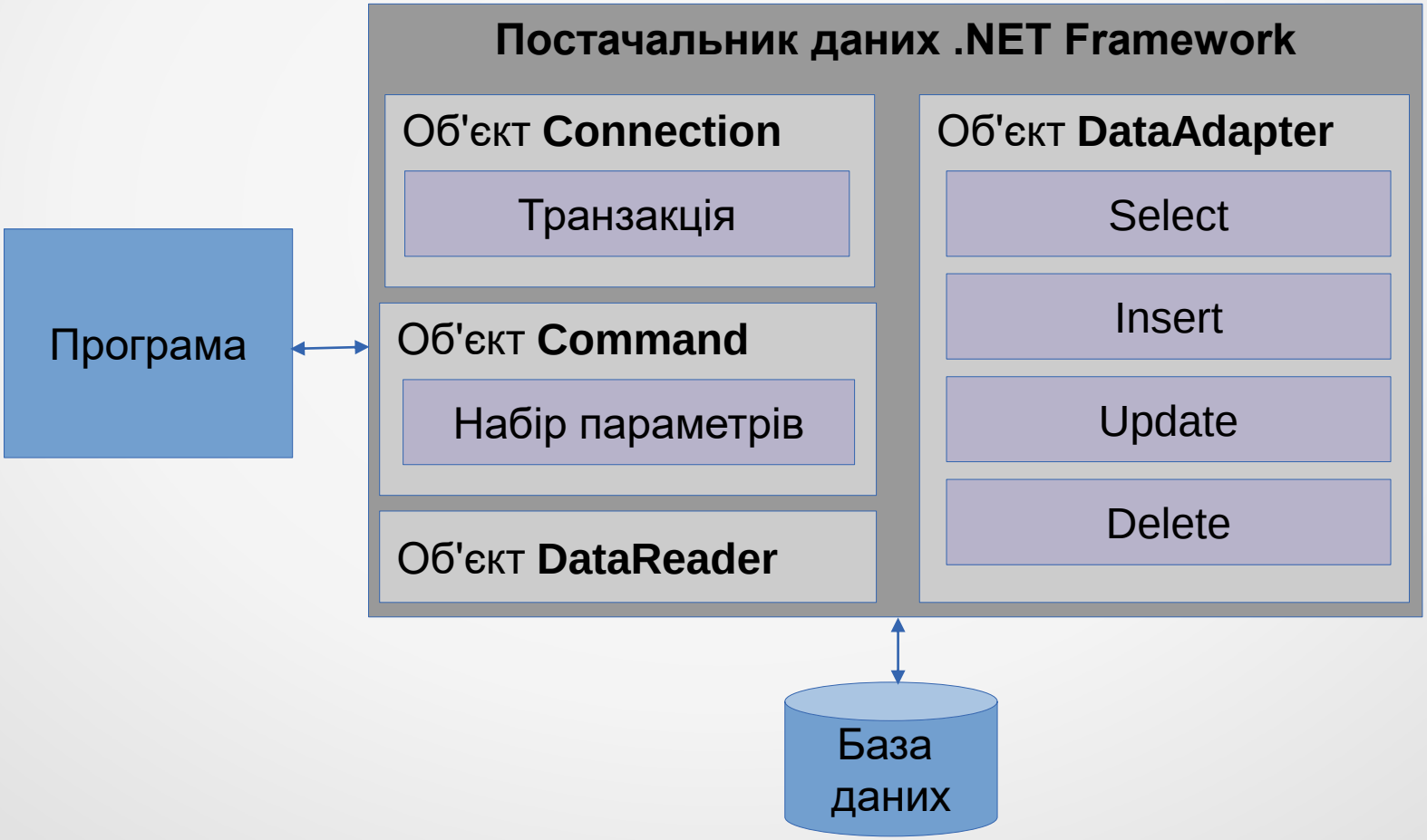
Постачальники даних містять типи даних, оптимізовані для взаємодії з конкретним джерелом даних (табл. 2).

Таблиця 2. Основні об'єкти постачальників даних ADO.NET

Тип об'єкту	Базовий клас	Опис
Connection	DbConnection	Реалізують підключення, доступ та відключення від сховища даних
Command	DbCommand	Виконання SQL-запиту або збережених процедур
DataReader	DbDataReader	Надає доступ до даних, призначених лише для читання
DataAdapter	DbDataAdapter	Передає набори даних між процесом, що викликає, і сховищем даних
Parameter	DbParameter	Описує іменований параметр у налаштованому запиті
Transaction	DbTransaction	Реалізує транзакцію бази даних

2. Постачальники даних ADO.NET

Загальна концепція використання постачальників даних ADO.NET може бути представлена таким чином:



2. Постачальники даних ADO.NET

У просторі імен **System.Data** знаходиться набір основних класів, що описують основні примітиви баз даних: таблиці, рядки, стовпці і т.п. (табл. 3).

Таблиця 3. Основні примітиви для роботи з базами даних

Клас	Опис
Constraint	Описує обмеження для певного об'єкту DataColumn
DataColumn	Описує стовпець в об'єкті DataTable
DataRelation	Реалізує відносини «батьківський-дочірній» між об'єктами DataTable
DataRow	Описує рядок в об'єкті DataTable
DataSet	Образ даних, що зберігається в пам'яті, і складається з довільної кількості взаємопов'язаних об'єктів DataTable
DataTable	Дані, що зберігаються в пам'яті у вигляді таблиці
DataTableReader	Доступ до DataTable у режимі лише читання
DataView	Спеціалізоване представлення DataTable для сортування, фільтрації, пошуку, редагування та навігації

3. Підключений рівень ADO.NET

Підключений рівень ADO.NET дозволяє працювати з базою даних за допомогою об'єктів конкретного постачальника даних, що реалізують підключення, читання даних, виконання команд і т.п.

Загалом для роботи з базою даних потрібно виконувати такі кроки:
1) створити, налаштувати та відкрити об'єкт підключення;
2) створити та налаштувати об'єкт команди (SQL-запиту), вказавши об'єкт підключення як аргумент конструктора або за допомогою властивості **Connection**;
3) викликати метод **ExecuteNonQuery()**, **ExecuteReader()** або **ExecuteScalar()** раніше налаштованого об'єкта команди;
4) у разі, якщо команда повертає **курсор** (таблично організований набір даних), обробити кожен запис за допомогою методу **Read()** об'єкта читання даних;
5) …
6) закрити об'єкт підключення.

3. Підключений рівень ADO.NET

Наприклад, **встановити з'єднання** з базою даних **SQLite** можна так:

```
using System.Data.SQLite;

// …
// Налаштування з'єднання
string dbName = "d:/data/student.db";
SQLiteConnection connect = new SQLiteConnection("Data Source =" + dbName);
// …
// Відкриття з'єднання
connect.Open();
// …
// Закриття з'єднання
connect.Close();
```

Примітка. SQLite – компактна СУБД, що вбудовується, з відкритим програмним кодом.

3. Підключений рівень ADO.NET

Виконання **SQL-запитів** у загальному вигляді здійснюється за допомогою класу **SqlCommand** та його нащадків. Наприклад, створення та налаштування SQL-запиту до СУБД SQLite може бути реалізовано так:

```
// Створення підключення
SQLiteConnection connect = new SQLiteConnection("Data Source = d:/mydb.db");
// Створення команди
SQLiteCommand command = new SQLiteCommand();
// Налаштування команди
command.Connection = connect; // Визначення з'єднання
command.CommandText = "SELECT * FROM some_table"; // Визначення команди
// …
```

Команду також можна налаштувати безпосередньо у конструкторі:

```
// …
SQLiteCommand command = new SQLiteCommand("INSERT INTO Dep (name)
VALUES ('Mathematics')", connect);
// …
```

3. Підключений рівень ADO.NET

Для виконання команди необхідно застосувати один із методів класу **SqlCommand** (або його спадкоємців, реалізованих для конкретної СУБД):

- **ExecuteNonQuery()** – виконує SQL-вираз і повертає кількість змінених записів (зазвичай використовується для виконання команд **CREATE**, **INSERT**, **UPDATE** та **DELETE**);
- **ExecuteReader()** – виконує SQL-запит і повертає результат у вигляді таблиці (найчастіше застосовується для виконання оператора **SELECT**);
- **ExecuteScalar()** – виконує SQL-вираз і повертає результат у вигляді скалярного значення, наприклад, числа (використовується для виконання SQL-запиту **SELECT** спільно з агрегатними функціями SQL: **min()**, **max()**, **sum()**, **count()**).

3. Підключений рівень ADO.NET

Наприклад, **створення таблиці** в SQLite може бути реалізовано таким чином:

```
// …
SQLiteCommand command = new SQLiteCommand();
string commandTextDep = "CREATE TABLE Dep (id INTEGER PRIMARY KEY
AUTOINCREMENT NOT NULL, name NVARCHAR(128))",
commandTextSpe = "CREATE TABLE Spe (id INTEGER PRIMARY KEY
AUTOINCREMENT NOT NULL, name NVARCHAR(128),
dep INTEGER, FOREIGN KEY (dep) REFERENCES Dep (id))";
// Відкриття раніше створеного з'єднання
connect.Open();
// Підключення команди до з'єднання
command.Connection = connect;
// Створення таблиці Dep
command.CommandText = commandTextDep;
command.ExecuteNonQuery();
// Створення таблиці Spe
command.CommandText = commandTextSpe;
command.ExecuteNonQuery();
// …
```

3. Підключений рівень ADO.NET

Для **отримання інформації** з таблиць бази даних разом із **SqlCommand** використовується клас **SqlDataReader** (і його нащадки).

Наприклад:

```
// …
var command = new SQLiteCommand("SELECT * FROM Dep", connect);
var res = command.ExecuteReader(); // Вилучення даних

// Висновок заголовка
Console.WriteLine("Id\tName");

// Виведення даних у вигляді таблиці
while (res.Read())
Console.WriteLine("{0}\t{1}", res[0], res[1]);

// ...
```

4. Транзакції

Транзакцією є послідовність операцій у базі даних, яка виконуються за принципом «**все чи нічого**» (мають бути або всі виконані, або жодна не виконана).

Транзакції застосовуються для забезпечення **безпеки**, **достовірності** та **узгодженості** даних у таблиці.

Усі взаємозалежні операції у транзакції СУБД виконуються як єдине ціле. У разі виникнення помилки на будь-якому етапі транзакції база даних автоматично повертається в початковий стан, який вона мала на початок транзакції.

4. Транзакції

Для створення об'єкта транзакції використовується клас **SqlTransaction** або його нащадки. Наприклад, використання транзакцій під час роботи з СУБД SQLite може бути реалізовано таким чином:

```
// …
connect = new SQLiteConnection("Data Source = " + dbName);
connect.Open();
// Запуск транзакції
SQLiteTransaction transaction = connect.BeginTransaction();
try
{
    // …
    transaction.Commit(); // Фіксація транзакції
}
catch (Exception e)
{
    transaction.Rollback(); // Відкат транзакції
    // …
}
connect.Close(); // Закриття БД
// …
```