**МІНІСТЕРСТВО ОСВІТИ І НАУКИ,
МОЛОДІ ТА СПОРТУ УКРАЇНИ**

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Лосєв М. Ю.
Тарасов О. В.
Федько В. В.

ОРГАНІЗАЦІЯ БАЗ ДАНИХ І ЗНАНЬ (ADO.NET)

Конспект лекцій

Харків. Вид. ХНЕУ, 2011

УДК 004.65(042.4)
ББК 32.973я73
Л79

Рецензент – канд. техн. наук, професор, зав. кафедри інформатики і комп'ютерної техніки Харківського національного економічного університету *Степанов В. П.*

Затверджено на засіданні кафедри інформаційних систем.
Протокол № 5 від 25.11.2010 р.

Лосєв М. Ю.

Л79 Організація баз даних і знань (ADO.NET) : конспект лекцій / М. Ю. Лосєв, О. В. Тарасов, В. В. Федько. – Х. : Вид. ХНЕУ, 2011. – 108 с. (Укр. мов.)

Наведено навчальний матеріал лекційних занять, що спрямовані на формування вмінь та навичок з використання новітніх технологій взаємодії з базами даних. Практичне застосування лекційного матеріалу сприятиме підвищенню компетенції в галузі розроблення багаторівневих застосувань.

Рекомендовано для студентів напряму підготовки "Комп'ютерні науки" денної форми навчання.

УДК 004.65(042.4)
ББК 32.973я73

Зміст

Вступ

ADO.NET – новітня технологія доступу до даних, яка є набором класів з .NET Framework і дозволяє звертатися до різноманітних джерел даних.

Вивчення навчальної дисципліни "Організація баз даних та знань (ADO.NET)" грунтується на знаннях та вміннях, які студенти отримали при вивченні навчальних дисциплін "Основи програмування і алгоритмічні мови", "Об'єктно-орієнтоване програмування" та "CASE-технології". Вона забезпечує такі дисципліни: "Публікація баз даних в Інтернеті", "Інтелектуальна обробка інформації", Технології програмування і створення програмних продуктів","Системи штучного інтелекту", "Системи підтримки прийняття рішень", "ІС в сучасному бізнесі". Вивчення навчальної дисципліни спрямовано на отримання студентами компетенції щодо здатності до розробки додатків для обробки інформації, яка зберігається в базах даних з використанням сучасних систем управління базами даних і вирішенню економічних задач у майбутній професійній діяльності.

Навчальний матеріал лекційних занять спрямован на вироблення вмінь та навиків з використання новітніх технологій взаємодії з базами даних. Основна мета конспекту лекцій: навчити студентів практичним прийомам розробки застосувань з використанням існуючих систем управління базами даних.

Студенти повинні знати принципи взаємодії прикладних програм, які виконані на мові високого рівня, з реляційними системами управління базами даних.

Конспект лекцій містить дві базові теми, які формують вміння студента щодо отримання компетенцій.

Перша тема – "З’єднання з базами даних. Виконання операцій в з’єднаному та роз’єднаному середовищі" формує вміння розробляти прикладні програми, які забезпечують пошук і оновлення інформації в реляційних базах даних.

Друга тема – "Робота в ADO.NET з використанням транзакцій" формує вміння програмувати транзакції в умовах необхідності збереження цілісності даних за допомогою відповідних програмних і технічних засобів, використовуючи програмні механізми, ізоляцію та блокування транзакцій, а також розробляти прикладні програми ведення

баз даних на платформі клієнт/сервер в умовах розподілення баз даних за допомогою комп'ютерних мереж.

Практичне використання лекційного матеріалу сприятиме підвищенню компетенції в галузі розроблення багаторівневих застосувань.

Тема № 1. З'єднання з базами даних. Виконання операцій в з'єднаному та роз'єднаному середовищі

Архітектура ADO.NET. Простір імен System Data. Провайдери даних. Створення з'єднання з джерелом даних і управління з'єднанням. Створення командних об'єктів. Запуск командних об'єктів. Особливості використання бібліотеки базових класів командних об'єктів. Створення об'єктів для обробки даних у роз'єднаному середовищі. Виконання операцій видалення, додавання і модифікації ієрархічних даних у таблицях. Особливості виконання операцій оновлення даних у таблицях з автоінкрементом. Забезпечення підтримки цілісності даних.

1.1. Лекція № 1. Введення в ADO.NET. З'єднання з базами даних. Управління з'єднанням

1.1.1. Введення в ADO.NET

Проектування інформаційних систем нерозривно зв'язане з використанням системами управління базами даних (СУБД). На сьогодні налічується близько двох десятків різних СУБД. І перед розробником стає складний вибір, яку з них використовувати. При цьому знати особливості проектування кожної СУБД – непосильне завдання навіть для цілої групи розробників. Ситуація ще ускладнюється, коли необхідно забезпечити підтримку різних джерел даних. Причому кожне з таких джерел даних може зберігати і обробляти дані по-своєму. Ще необхідно враховувати, що в різних мовах програмування різна підтримка роботи з тією або іншою СУБД. Тобто, ще виникає проблема невідповідності обробки інформації більшістю СУБД і способів обробки інформації різними мовами програмування.

Для спрощення процесу проектування інформаційних систем (ІС) необхідно мати інструмент, який володіє можливістю масштабування і адаптації до будь-якого джерела даних. Архітектура повинна бути проста

в розумінні розробниками ІС, і володіти гнучким механізмом використання ресурсів.

Останніми роками потреби доступу до даних змінилися, що спричинило за собою і зміну технології. Доступ до даних виконувався в основному за допомогою некерованого коду. Некерований код не тільки погіршує продуктивність порівняно з повністю керованим кодом, але і має ще більший недолік, несумісний з концепцією безпеки .NET. Іншим великим недоліком старих технологій є те, що вони насправді не були призначені для роботи з XML. XML був доданий в існуючу технологію доступу до даних ADO (ActiveX Data Objects – ADO). Отримані данні XML було важко прочитати людині, а також ці дані було важко передавати між різними об'єктами. Для усунення перерахованих недоліків компанією Microsoft була розроблена технологія ADO.NET і включена в нову платформу .NET Framework.

Архітектуру ADO.NET можна розділити на дві фундаментальні частини: підключену і автономну. Усі різні класи з ADO.NET можна приписати до підключеної або автономної частини. При цьому використовуються такі основні компоненти ADO.NET для доступу до даних і їх обробки:

постачальники даних .NET Framework;

DataSet.

Компонент DataSet використовується для доступу до інформації, що зберігається в базі незалежно від джерела даних (для управління даними в мережі або локальними даними в застосуванні, а також XML-даними). Компонент DataSet містить колекцію декількох об'єктів таблиць, що складаються з рядків і стовпців даних, а також первинний ключ, зовнішній ключ, обмеження і відомості про зв'язки між об'єктами таблиць. Детально DataSet буде розглянутий в п.1.3.1.

Класи .NET групуються у просторах імен. Усі функції, що відносяться до ADO.NET, знаходяться у просторі імен System.Data (табл. 1.1). Крім того, як і будь-які інші компоненти .NET, ADO.NET працює неізольовано і може взаємодіяти з різними іншими частинами .NET Framework, такими як клас System.Web.UI.WebControls.Adapters .TableAdapter і простір імен System.Transactions. За наявністю широкого вибору доступних джерел даних ADO.NET повинна мати можливість підтримувати безліч джерел даних. Кожне таке джерело даних може мати свої особливості або набір можливостей.

Таблиця 1.1

Простір імен System.Data

Тип	Опис
Constraint	Обмеження для стовпця таблиці
DataColumn	Стовпець таблиці
DataRelation	Відношення "батько–нащадок" між таблицями
DataRow	Рядок таблиці
DataSet	Об'єкт, призначений для зберігання в оперативній пам'яті будь-якого числа зв'язаних об'єктів таблиць
DataTable	Таблиця бази даних
DataTableReader	Забезпечує читання даних з об'єкта DataTable
DataView	Забезпечує уявлення для DataTable з використанням сортування, фільтрації, редагування і т. д.
IDataAdapter	Визначає базову поведінку об'єкта адаптера даних
IDataParameter	Визначає базову поведінку об'єкта параметра
IDataReader	Визначає базову поведінку об'єкта читання даних
IDbCommand	Визначає базову поведінку об'єкта команди
IDbDataAdapter	Забезпечує додаткові функціональні можливості IDataAdapter
IDbTransaction	Визначає базову поведінку об'єкта транзакції

Тому ADO.NET підтримує модель постачальників (provider). Постачальником для конкретного джерела даних в ADO.NET можна визначити набір класів у просторі імен, створених спеціально для роботи з цим конкретним джерелом даних.

Підключені об'єкти розділяються в ADO.NET для різних СУБД. Тобто, наприклад, для підключення до бази даних Microsoft SQL Server є спеціальний клас SqlConnection. Насправді всі класи, що відносяться до SQL Server, знаходяться в одному і тому ж просторі імен System.Data.SqlClient. Аналогічно, всі класи, що відносяться до Oracle, знаходяться у просторі імен System.Data.OracleClient. Ці окремі реалізації для конкретних СУБД називаються постачальниками даних .NET (.NET data provider). Уся концепція представлена на рис. 1.1.

Є безліч популярних СУБД, які можуть бути серверними або файловими. Серверні СУБД працюють стабільніше і краще підтримують одночасну роботу декількох користувачів, а файлові СУБД легше упроваджувати і управляти ними після інсталяції застосування клієнтів.

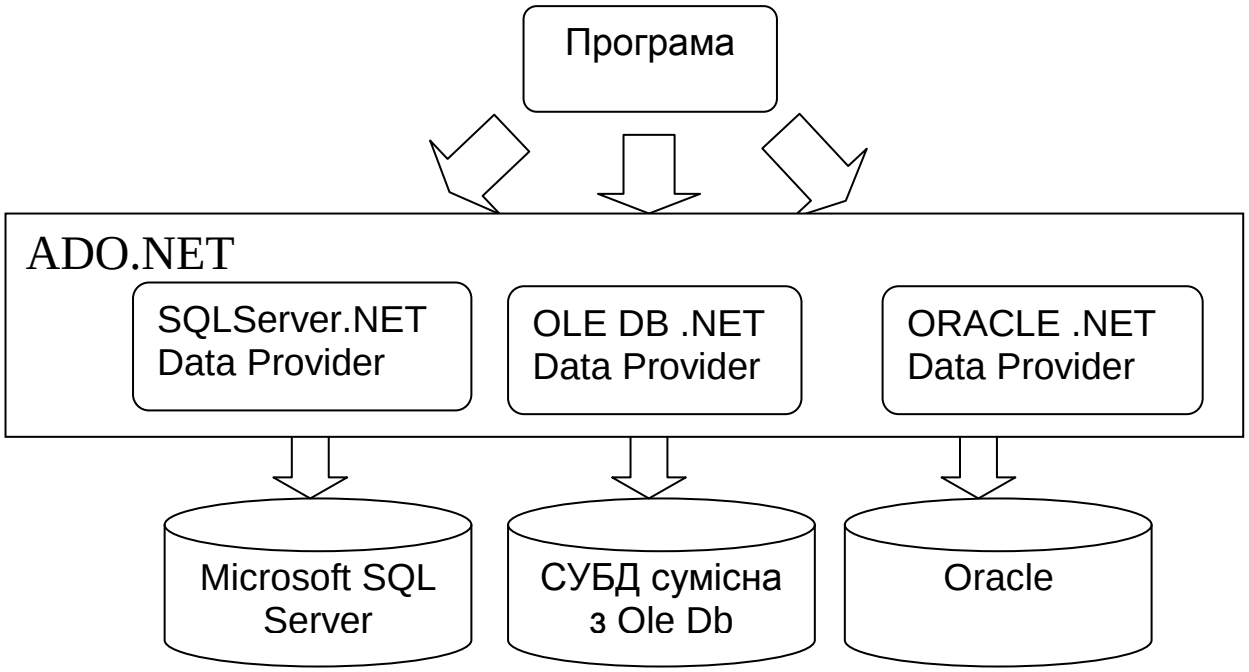

Рис. 1.1. **Постачальники даних в ADO.NET**

Проте це лише загальний принцип: наприклад, Microsoft SQL Server має всі можливості повноцінної серверної СУБД, але разом з цим дозволяє легко взаємодіяти і з файловою базою даних.

Тому для кожного конкретного джерела даних потрібен спеціальний постачальник даних .NET. Ця відмінність дещо розмита у разі OleDb і ODBC, оскільки вони за своєю суттю створені для роботи з будь-якою базою даних, сумісною з OleDb або ODBC, але навіть їх спеціальні реалізації знаходяться в окремому постачальнику даних, створеному спеціально для них. Програма може користуватися будь-якими об'єктами з великого прямокутника на рис. 1.1. Тепер при побудові застосування можна використовувати автономні об'єкти, адаптери даних, підключені об'єкти або їх комбінацію.

Найчастіше використовуються такі постачальники даних:

SQL Server .NET Data Provider – цей провайдер призначений для роботи з базами даних Microsoft SQL Server;

OLE DB.NET Data Provider – цей провайдер призначений для роботи з джерелами даних SQL Server, Oracle, Access;

ODBC.NET Data Provider.– цей провайдер забезпечує доступ до джерел даних через їх ODBC-драйвера;

ORACLE.NET Data Provider – цей провайдер забезпечує доступ до баз даних Oracle.

За угодою постачальники, що входять до складу .NET Framework, містяться у власному просторі імен, який знаходиться у просторі імен System.Data. У табл. 1.2 перераховані простори імен, що входять до складу .NET 2.0 Framework.

Таблиця 1.2

Постачальники даних і простори імен

Джерело даних	Простір імен постачальника
Microsoft SQL Server 7.0 і вище	System.Data.SqlClient
Oracle 8.1.6 і вище	System.Data.OracleClient
Підтримка SqlXml в SQL Server	System.Data.SqlXml
Будь-яке джерело даних ODBC	System.Data.ODBC
Будь-яке джерело даних OleDb	System.Data.OleDb

Постачальник даних містить чотири компоненти: Connection; Command; DataReader; DataAdapter.

Connection – клас, що дозволяє встановлювати підключення до джерела даних. Залежно від реально задіяного постачальника даних, об'єкти підключення автоматично буферизують за вас фізичні підключення до баз даних. Приклади класів підключення – OleDbConnection, SqlConnection, OracleConnection і т. д. Вони будуть детально розглянуті нижче.

Command – цей клас надає команду, яка виконується у джерелі даних. Команда може повернути який-небудь результат, а може і не повертати. Такі команди дають можливість маніпулювати існуючими даними, запрошувати існуючі данні, а також оновлювати і навіть видаляти існуючі данні. Крім того, ці команди дозволяють працювати із структурами базових таблиць. Приклади класів команд – SqlCommand, OracleCommand і т. д. Докладніше цей клас буде розглянутий в п. 1.2.

DataReader – це еквівалент конвейєрного курсора з можливістю тільки читання даних, який дозволяє вибирати дані з бази з дуже великою швидкістю, але тільки в режимі послідовного читаня. Цей клас буде докладніший розглянутий в п. 1.2

DataAdapter – це своєрідний шлюз між автономними і підключеними об'єктами ADO.NET. Він встановлює за вас підключення або, якщо підключення вже встановлене, містить досить інформації, щоб сприймати дані автономних об'єктів і взаємодіяти з базою даних. Приклади використання класів DataAdapter будуть розглянуті в п. 1.3.

1.1.2. З’єднання з базами даних

Створення застосування з використанням ADO.NET виконується протягом декількох етапів:

створення підключення до бази даних;

створення команди, що містить SQL-запит;

відкриття підключення;

виконання команди;

закриття підключення;

виведення результатів.

Застосування починається із створення підключення до бази даних. ADO.NET дозволяє організувати з'єднання з нею за допомогою об'єкта підключення, який входить до складу відповідного постачальника даних.

Сам об'єкт підключення джерела даних успадковується від класу DbConnection і одержує вже готову логіку, реалізовану в базових класах провайдерів (OleDbConnection, SqlConnection, OracleConnection і т. д.). Для організації з'єднання необхідно створити об'єкт Connection, який виконує ці функції. Наприклад, створимо декілька об'єктів для різних постачальників даних:

```
OleDbConnection oledbconn = newOleDbConnection();
SqlConnection sqlconn = new SqlConnection();
OdbcConnection odbcconn = new OdbcConnection();
OracleConnection oracleconn = new OracleConnection();
```

Об'єкти підключення (SqlConnection, OleDbConnection та ін.) мають безліч методів, які використовуються для ефективної роботи з'єднання (табл. 1.3).

Таблиця 1.3

Основні методи об'єкта SqlConnection

Метод	Опис
1	2
BeginTransaction	Починає транзакцію для з'єднання
ChangeDatabase	При відкритому з'єднанні перемикає на вказану базу даних
ClearAllPools	Очищує вільні з'єднання в усіх пулах SqlConnection
ClearPool	Очищує вільні з'єднання в пулі з'єднань з відповідним об'єктом SqlConnection
Close	Закриває з'єднання

Закінчення табл. 1.6

1	2
Create Command	Створює об'єкт SqlCommand для поточного з'єднання
EnlistDistributedTransaction	Вручну додає з'єднання в розподілену транзакцію
EnlistTransaction	Вручну додає з'єднання у транзакцію
GetSchema	Одержує інформацію про схему бази даних для з'єднання
Open	Відкриває з'єднання
ResetSatistics	Скидає статистику для поточного з'єднання
RetreiveStatistics	Повертає статистику для поточного з'єднання

Приклади використання цих методів будуть приведені по мірі розгляду відповідної теми навчального матеріалу.

Клас SqlConnection включає дві події: InfoMessage і StateChange (табл. 1.4).

Таблиця 1.4

Події класу SqlConnection

Події	Опис
InfoMessage	Наступає, коли з'єднання одержує інформаційне повідомлення від джерела даних
StateChange	Наступає при зміні властивості *State* з'єднання

Деякі бази даних підтримують інформаційні повідомлення. SQL Server дозволяє посилати повідомлення клієнту за допомогою команди PRINT. Такого роду повідомлення не містять інформації про помилки і результати запитів.

Для відстежування таких повідомлень можна скористатися подією InfoMessage класу SqlConnection. Можна також примусити об'єкт SqlConnection реагувати на помилки в запитах (наприклад, на спробу запиту неіснуючої таблиці) за допомогою події InfoMessage, не генеруючи виняткової ситуації. Для цього потрібно привласнити властивості FireInfoMessageEventOnUserErrors об'єкта SqlConnection значення True.

Подія StateChange наступає після зміни значення властивості State об'єкта Connection. Воно буде особливо корисним під час відображення поточного стану з'єднання (наприклад, в інформаційній панелі застосування).

Для того, щоб відкрити підключення необхідно вказати, яка інформація необхідна для виконання цього завдання, наприклад, ім'я сервера, ідентифікатор користувача, пароль і т. д. Оскільки кожному цільовому джерелу підключення може знадобитися особливий набір інформації, що дозволяє ADO.NET підключиться до джерела даних, обирають гнучкий механізм вказівки всіх параметрів через рядок підключення ConnectionString. Рядок підключення – це набір розділених крапкою з комою пар атрибут-значень, що визначають, як слід встановлювати підключення до джерела даних. Хоча рядки підключення мають тенденцію виглядати однаково, можливі та обов'язкові атрибути розрізняються залежно від постачальника даних і джерела даних. Нижче приведений перелік атрибутів рядків підключення для встановлення з'єднань:

Provider – им’я провайдера (тільки для OLE DB);

Driver – ODBC-драйвер (тільки для ODBC);

Connection Timeout – час у секундах для спроб установки з'єднання (за замовчуванням 15 сек.);

InitialCatalog – им’я бази даних в SQLServer;

Data Source – ім'я сервера або шлях до бази даних Access;

Password – пароль;

User ID – логін;

Integrated Security – True, якщо вибирається Windows Authentication, False - якщо SQL Authentication.

Наприклад, рядок з'єднання при використанні провайдера OLE DB і підключенні до бази даних lab.mdb матиме такий вигляд:

string str="Provider=Microsoft.Jet.OLEDB.4.0;DataSource=lab.mdb";

Тепер властивості ConnectionString об'єкта підключення oledbconn можна задати необхідне значення:

oledbconn.ConnectionString = str;

Ці дії можна виконати за допомогою одного рядка, наприклад:

oledbconn.ConnectionString="Provider=Microsoft.Jet.OLEDB.4.0;
DataSource=1ab.mdb";

Як видно з приведених прикладів не всі елементи рядка підключення можуть бути в ній записані. Для організації з'єднання з

джерелом даних SQL Server рядок підключення буде змінений, наприклад:

```
SqlConnection sqlconn = new SqlConnection();
string str ="Data Source=(local)/Initial Catalog=Rtest;
IntegratedSecurity=SSPI";
sqlconn.ConnectionString = str;
```

Цей код готує об'єкт підключення до відкриття з'єднання до SQL Server на локальній машині за допомогою Windows-аутентифікації і до підключення до бази даних з ім'ям Rtest.

При програмуванні застосувань потрібно вибрати як найкраще місце для зберігання рядків підключення, з тим щоб досягти більшої зручності обслуговування, полегшити внесення змін у майбутньому і позбавитися необхідності перекомпілювати застосування при його модифікації. Зберігати рядки підключення можна різними способами:

жорстко програмувати в коді застосування;

у конфігураційному файлі застосування;

у реєстрі Windows;

представляти за допомогою UDL-файла (файл універсальної вказівки даних – universal data link);

поміщати в окремий файл.

Перший спосіб зберігання рядків підключення – це жорстко програмувати їх у застосуванні. Хоча такий підхід дає якнайкращу продуктивність, гнучкість його залишає бажати кращого: якщо з якоїсь причини потрібно буде змінити рядок підключення, застосування необхідне буде потім перекомпілювати. Невисокий і рівень захисту: код можна дизасемблювати і витягнути з нього рядок підключення. У багатьох випадках переважним підходом є зовнішнє зберігання рядка підключення унаслідок його більшої гнучкості, безпеки і простоти в настройці.

Конфігураційний файл застосування – це текстовий файл у форматі XML, де зберігаються настройки, використовувані у процесі роботи. Він розташовується в тому ж каталозі, що і файл, який виконується. Конфігураційному файлу застосування Windows Forms дає ім'я app.config – так прийнято за замовчуванням. На етапі компіляції цей файл буде автоматично скопійований Visual Studio .NET у стартовий каталог. Рядок підключення до бази даних розташовується в елементі <connectionStrings>:

```
<connectionStrings>
<add name="lab"
connectionString= "Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=|DataDirectory|\lab.mdb"
providerName="System.Data.OleDb" />
</connectionStrings>
```

Зберігання налаштувань у конфігураційних файлах спрощує розгортання застосувань, оскільки ці файли просто встановлюються поряд з іншими файлами. Недолік цього способу полягає в тому, що він за природою не є безпечним: інформація зберігається відкритим текстом у файлі, доступному через файлову систему. Для усунення цього недоліку рядок підключення та іншу конфіденційну інформацію в конфігураційному файлі можна зашифрувати.

Постачальник даних .NET для OLE DB підтримує імена UDL-файлів у рядку підключення. UDL-файл – це спеціальний файл, в якому міститься інформація про підключення. Його необхідно захистити правами доступу NTFS, щоб відомості про підключення не можна було прочитати або змінити. Постачальник даних .NET для SQL Server не підтримує UDL-файли в рядку підключення.

Рядки підключення можна зберігати в системному реєстрі Windows як додаткові ключи усередині ключа HKEY_LOCAL_MACHINE \SOFTWARE. Даний спосіб характеризується простотою в застосуванні завдяки програмній підтримці доступу до системного реєстру в класах .NET Registry і RegistryKey, імен Microsoft, що належать простору Win32. Зберігати рядки підключення в системному реєстрі звичайно не рекомендується через проблеми, пов'язані з розгортанням при установці застосування необхідно записувати налаштування в реєстр, крім того, доступ додатків до системного реєстру може бути обмежений, що ще більше ускладнює розгортання.

Налаштування, які використовуються у процесі роботи, можуть зберігатися у спеціальному файлі. Зберігання інформації про підключення у спеціальному файлі не має яких-небудь особливих переваг, тому використовувати даний спосіб не рекомендується. Він більш трудомісткий у програмуванні і примушує явним чином мати справу з проблемами паралельної обробки даних. Місце зберігання рядка підключення багато в чому визначає спосіб програмування

з'єднання застосування з базою даних. Розглянемо декілька способів організації з'єднання з базою даних.

При зберіганні рядка підключення у програмі для виконання з'єднання з базою даних можна використовувати такий код:

```
OleDbConnection oledbconn = new OleDbConnection();
oledbconn.ConnectionString="Provider=Microsoft.Jet.OLEDB.4.0;
DataSource=lab.mdb";
try
 {
oledbconn.Open();
MessageBox.Show(  "З'єднання відчинено" );
// Виконання корисних дій
        oledbconn.Close();
        MessageBox.Show(  " З'єднання зачинено " )
 }
catch(System.Data.OleDb.OleDbException e1)
// Виникнення OleDbException
{
MessageBox.Show("З'єднання відчинено "+e1.Message);
}
catch (System.InvalidOperationException e2)
// Помилка відкриття бази даних
{
MessageBox.Show("З'єднання не відчинено " + e2.Message);
 }
}
```

У рядках підключення можна зазначити множину різних параметрів. Для успішного встановлення підключення до бази даних кожний із цих параметрів потрібно правильно назвати і вказати для нього значення. Але якщо відсутній легкий інтуїтивний спосіб конструювання рядка підключення, то не тільки важко запам'ятати точне ім'я кожного параметра, але й легко пропустити різні можливості конфігурування для об'єктів підключення кожного постачальника даних.

У ADO.NET ця проблема вирішується за допомогою класу DbConnectionStringBuilder. Об'єкт DbConnectionStringBuilder суворо типізує різні значення, що складають рядок підключення. Він дозволяє уникнути тривіальних програмістських помилок, а також полегшує

керування інформацією в рядку підключення. Приведемо приклад використання цього об'єкта:

```
SqlConnectionStringBuilder conBuilder =
                        new SqlConnectionStringBuilder ();
conBuilder.DataSource = "(local)";
conBuilder.InitialCatalog = "Rtest";
conBuilder.IntegratedSecurity = true;
using (SqlConnection connect =
new SqlConnection(conBuilder.ToString ())) {
try
{
   connect.Open();
   Console.WriteLine ("Підключення успішно відкрите"-);
}
catch (Exception)
{
   Console.WriteLine("Неможливо відкрити підключення");}}
   Console.WriteLine ("Для продовження натисніть будь-яку клавішу");
   Console.Read();
}}
```

У тому випадку, якщо рядок підключення зберігається в конфігураційному файлі з'єднання з базою даних може здійснюватися за допомогою класу ConfigurationManager і його властивості Connection Strings. При виконанні з'єднання таким способом слід обов'яз-ково додати посилання на компонент System.Configuration. Нижче приведений приклад використання класу ConfigurationManager:

```
using System.Configuration;
using System.Data.OleDb;
……
OleDbConnection connection = new OleDbConnection();
  string str =
  ConfigurationManager.ConnectionStrings["lab"].ToString();
  connection.ConnectionString = str;
  try
  {
     connection.Open();
  }
```

```
catch (Exception ex)
{
   MessageBox.Show(
"Помилка під час створення підключення\n" + ex.ToString());
}
```

Докладний приклад використання підключення за допомогою конфігураційного файла приведений в [1].

У .NET кожний постачальник даних пропонує спеціальний клас System.Data.Common.DbProviderFactory. Для одержання типу, що є похідним від DbProviderFactory і відповідним для постачальника даних, простір імен System.Data.Common пропонує додатковий тип класу DbProviderFactories. Методом GetFactory() можна одержати унікальний об'єкт цього класу для заданого постачальника даних. Нижче наведено приклад підключення за допомогою зазначених класів:

```
DbConnection connection = null;
    DbProviderFactory factory =
DbProviderFactories.GetFactory("System.Data.OleDb");
//Одержання джерела постачальника даних SQL
//          DbProviderFactory sqlfactory =
//DbProviderFactories.GetFactory("System.Data.SqlClient");
```

Докладний приклад використання класу DbProviderFactory приведений в [1].

1.1.3. Управління з’єднанням

Ефективність використання з'єднань істотно залежить від управління процесом підключення до баз даних. При цьому можна використовувати поточний стан з'єднання, який зберігається у властивості State. Ця властивість приймає такі значення типа ConnestionState:

Broken – з'єднання з джерелом даних розірване, подібне може трапитися тільки після того, як з'єднання було встановлено, у цьому випадку з'єднання може бути або закрито, або повторно відкрито;

Closed – з'єднання закрите;

Connecting – триває процес підключення (зарезервовано);

Executing – з'єднання знаходиться у процесі виконання команди (зарезервовано);

Fetching – об'єкт з'єднання зайнятий вибіркою даних (зарезервовано);

Open – з'єднання відкрито.

Наприклад, наступний код закриває з'єднання якщо воно відкрито:

```
if (connect.State == ConnectionState.Open)
connect.Close();
```

Застосування, яке мінімізує число одночасно відкритих з'єднань (а значить і навантаження на сервер), здатне поліпшити продуктивність бази даних. ADO.NET передбачає наявність засобу управління з'єднаннями – так званого пулу з'єднань. Організація пулу з'єднань – це процес управління спільними ресурсами, які виділяються з пулу недавно використаних з'єднань. При організації з'єднань враховується, що для більшості застосувань потрібна установка з'єднання на дуже невелику тривалість часу, тоді як створення і звільнення об'єкта з'єднання є досить коштовной операцією. Пул з'єднань – це спосіб їх повторного використання. Якщо застосування запитає повторну установку з'єднання, пул надасть йому вже відкрите з'єднання, зекономивши час на звільнення і створення нового об'єкта з'єднання. Реалізація пулу з'єднань відрізняється в кожному з керованих постачальників.

Пулом підключень можна управляти за допомогою атрибутів рядка підключення (табл. 1.5).

Аналогічні атрибути використовуються для управління пулом в СУБД Oracle.

Таблиця 1.5

Атрибути рядка підключення в SQL Server

Атрибут рядка	Опис
Connection Lifetime	Тривалість інтервалу часу в секундах, при перевищенні якого з'єднання з базою даних руйнується
Connection Reset	Указує, чи слід скидати підключення при видаленні його з пулу. За замовчуванням дорівнює true
Enlist	Значення дорівнює true, яке призводить до того, що диспетчер пулу автоматично заносить підключення в поточний контекст транзакції
Max Pool Size	Максимальна кількість підключень у пулі. За замовчуванням дорівнює 100
Min Pool Size	Мінімальна кількість підключень у пулі. За замовчуванням дорівнює 0
Pooling	Використовується для включення або відключення пулу. За замовчуванням має значення true

Провайдер даних .NET для OLE DB здійснює управління пулом за допомогою параметра OLE DB Services рядка підключення до бази даних. Наприклад, для відключення пулу в рядку з'єднання необхідно вказати:

OLEDBServices= -4; **.**

При використанні об'єкта SqlConnection, в рядок підключення слід додати такий атрибут:

Pooling=False; **.**

У середовищі .NET при використанні дефіцитних ресурсів, таких як з'єднання з базами даних, правила хорошого тону в програмуванні вимагають, щоб кожен такий ресурс був негайно закритий після його використання.

Закривати з'єднання можна за допомогою конструкції try...catch...finally. Наприклад:

```
try
{    // Відкрити з'єднання
conn.Open() ;
// Зробити що-небудь корисне    }
catch (SqlExceptionex )
{  // Зробити що-небудь таке, що викличе виключення  }
finally          { // Переконатися в тому, що з'єднання звільнене
conn.Close () ;  }
```

Об'єкт підключення ADO.NET для закриття з'єднання може використовувати метод Dispose. Окрім зачистки звільнених ресурсів, метод Dispose викликає метод Close об'єкта підключення і готує його до повторного використання з пулу підключень. Окрім виклику Close, метод Dispose очищає внутрішні набори від різних параметрів, наприклад, від рядків підключення для об'єктів підключення, і таким чином дозволяє складальнику сміття повторно використовувати пам'ять, зайняту конкретним об'єктом підключення. Приведемо приклад закриття з'єднання методом Dispose з використанням оператора блоку:

```
using (SqlConnection conn = new SqlConnection ( source ))
{
// Відкрити з'єднання
conn.Open() ;
// Зробити що-небудь корисне
}
```

У приведеному прикладі конструкція using гарантує, що з'єднання з базою даних буде закрито. У наступному прикладі приведений код, який дозволяє звільняти ресурс відразу після його використання:

```
try
{
using (SqlConnection conn = new SqlConnection ( source ))
{
// Відкрити з'єднання
conn.Open () ;
// Зробити що-небудь корисне. Закрити з'єднання  самостійно
        conn.Close () ;
    }
}
catch (SqlExceptione)
{
// Зробити що-небудь відносно виключення...
}
```

Цей приклад забезпечує мінімальну затримку використання ресурсу з'єднання.

Висновки. У лекції розглянута архітектура ADO.NET, а також простір імен System Data, які надають широкі можливості користувачу для роботи з базами даних. Наведені особливості використання провайдерів даних. Розглянуті способи створення з'єднання з джерелом даних і управління з'єднанням.

Тести для самодіагностики

Тест № 1. Ви розробляєте застосування, яке використовує базу даних SQLServer. Код вашої програми повинен дістати найшвидший доступ до бази. Ви повинні досягти цієї мети з мінімальною кількістю коду. Що необхідно використовувати при проектуванні застосування:

1) класи простору імен System.Data.OleDb.OleDb namespace;

2) класи простору імен System.Data.SqlClient namespace;

3) використовувати remoting, щоб з'єднатися з базою SQLServer?

Тест № 2. Ви розробляєте застосування, в якому при закритті з'єднання метод .Dispose() повинен викликатися автоматично. Який з перерахованих засобів ви використовуєте у своєму коді.

1) метод Close();

2) метод IDisposable.Dispose();

3) Using (SQLConnection conn=new SQLConnection(connString)){};

4) використовувати конфігураційний файл?

Контрольні завдання

Завдання № 1. Створити застосування, яке визначає, чи дійсно з'єднання закрите і чи поміщено воно в пул.

Завдання № 2. Визначити, чи є тільки що створене з'єднання новим або використовується повторно з пулу з'єднань.

Література: [2, с. 5–20; 3, с. 46–91].

1.2. Лекція № 2. Виконання операцій в з’єднаному середовищі

1.2.1. Виконання операцій обробки даних з використанням класу SqlCommand

У сполученому середовищі застосування вибирає і обробляє інформацію, залишаючись підключеним до джерела даних. Постачальник даних містить декілька компонентів, серед яких є клас Command, який надає команди для роботи з джерелом даних. Залежно від провайдера даних, який використовується, необхідно вибрати відповідний клас та об'єкт для передачі команд джерелу даних, наприклад System.Data.SqlClient.SqlCommand – призначений для роботи з SQL Server7.0 або вище.

Команди дають можливість маніпулювати існуючими даними, запрошуючи, оновлюючи і видаляя інформацію з бази даних. Команда у простій формі є рядком тексту, що містить оператори SQL. У табл. 1.6 приведені властивості класу Command, які найбільш часто використовуються.

Таблиця 1.6

Властивості класу Command

Властивість	Опис
1	2
(Name)	Ім'я командного об'єкта
Connection	Ім'я з'єднання, через яке командний об'єкт працюватиме з джерелом даних
CommandType	Тип командного об'єкта
Parameters	Колекція для зберігання можливих параметрів

Закінчення табл. 1.6

1	2
Transaction	Указує на транзакцію, яка використовується при виконанні запиту
UpdateRowSource	Визначає, яким чином результати виконання запиту впливають на поточний рядок
CommandTimeout	Час очікування результату в секундах при виконанні командного об'єкта, за замовчуванням складає 30 секунд

Командний об'єкт може бути створений за допомогою конструктора класу Command. ADO.NET надає декілька конструкторів, які можна використовувати для створення об'єктів команд:

конструктор класу **SqlCommand** (**OleDbCommand**) без параметрів;

конструктор класу **SqlCommand** (**OleDbCommand**), в якому як параметр використовується рядок підключення;

конструктор класу **SqlCommand**, який як параметри приймає рядок підключення і текст команди;

конструктор класу **SqlCommand**, який як параметри приймає рядок підключення, текст команди і об'єкт транзакції.

У табл. 1.7 приведені значення властивості CommandType.

Таблиця 1.7

Значення властивості CommandType

Значення	Опис
CommandText	Текст запиту, який необхідно виконати
StoredProcedure	Указує на те, що властивість CommandText містить ім'я процедури бази даних, що зберігається
TableDirect	Означає, що властивість CommandText містить ім'я таблиці в базі даних, з якої будуть вибрані дані у разі виклику методу Execute. Використовується тільки провайдером OleDb

Приведемо приклади створення командних об'єктів:

```
stringcommandText = "SELECT * FROM Товари";
SqlConnectionconn = newSqlConnection(connectionString);
// перший конструктор
SqlCommand myCommand1 = new SqlCommand();
```

```
myCommand.Connection = conn;
myCommand.CommandText = commandText;
// другий конструктор
SqlCommand myCommand2 = new SqlCommand(commandText);
// третій конструктор
SqlCommand myCommand3 = new SqlCommand(commandText, conn);
// четвертий конструктор з об'єктом транзакції
SqlTransaction trans = conn.BeginTransaction();
SqlCommand myCommand4 =
                    new SqlCommand(commandText, conn, trans);
```

Командний об'єкт можна створити за допомогою методу **CreateCommand()** объекта **Connection**:

```
SqlCommand myCommand = conn.CreateCommand();
myCommand.CommandText = commandText;
```

Кожен об'єкт **Command** має безліч методів. У табл. 1.8 приведені найбільш часто використовувані методи класу SqlCommand.

Таблиця 1.8

Методи класу SqlCommand

Метод	Опис
ExecuteNonQuery	Виконує запити і повертає тільки число оброблених записів
ExecuteReader	Виконує запит і додає його результати в об'єкт SqlDataReader
ExecuteScalar	Призначений для виконання запитів, результат яких містить одне число (наприклад, обчислення агрегатних функцій)
BeginExecuteNonQuery BeginExecuteReader BeginExecuteXmlReader	Починають асинхронне виконання запиту
Cancel	Відміняє виконання запиту
Clone	Повертає копію об'єкта SqlCommand
CreateParameter	Створює новий параметр для запиту
EndExecuteNonQuery EndExecuteReader EndExecuteXmlReade	Завершують асинхронне виконання запиту
Prepare	Створює у сховищі даних версію запиту
ResetCommandTimeout	Задає властивості CommandTimeout, його значення за замовчуванням - 30 секунд

Розглянемо приклади використання деяких методів класу SqlCommand.

Методом ExecuteNonQuery() дуже зручно виконувати оновлення, додавання і видалення рядків у таблицях бази даних, а також використовується для виконання запитів, що відносяться до категорії DDL мови SQL.

Наприклад, у приведеному нижче коді виконується оновлення стовпця **Value** таблиці **Table1** і додавання нового стовпця **Data** в цю таблицю:

```
connection.Open();
SqlCommand Upd = connection.CreateCommand();
Upd.CommandText = "UPDATE Table1 SET Value=12 WHERE ID=3";
Upd.ExecuteNonQuery();
Upd.CommandText = "ALTER TABLE Table1 ADD Data datetime";
Upd.ExecuteNonQuery();
connection.Close();
```

При виконанні запитів до бази даних, в яких необхідно одержати лише одне число зручно застосовувати метод ExecuteScalar(). Наприклад, за допомогою цього методу можна обчислити кількість рядків у таблиці "Товари":

```
connection.Open();
SqlCommand myCommand = connection.CreateCommand();
command.CommandText = "SELECT COUNT (*) FROM Товари";
int kilkistTovariv = (int) command.ExecuteScalar();
connection.Close();
```

1.2.2. Виконання операцій читання даних з використанням класу SqlDataReader.

Для виконання запитів, що вибирають набір даних, використовується метод ExecuteReader, який додає результати в об'єкт DataReader. Властивості і методи об'єкта DataReader допомагають проглядати результати запиту. Приведений нижче фрагмент коду ілюструє, як проглянути результати простого запиту за допомогою об'єкта SqlDataReader:

```
//створюємо командний об’єкт
SqlCommand command =
new SqlCommand("SELECT * FROM Товари", connection);
```

```
//відкриваємо Connection
connection.Open();
//виконуємо ExecuteReader
SqlDataReader reader=command.ExecuteReader();
//друкуємо таблицю
while (reader.Read()) {
Console.WriteLine("{0}\t{l}\t{2}",
reader["IDТовару"] .ToString(), reader ["Ціна"].ToString());
//закриваємо DataReader
reader.Close();
//закриваємо Connection
connection.Close ();
```

Слід зазначити, що об'єкт DataReader необхідно обов'язково закривати, оскільки повторне звернення до цього об'єкта приведе до помилки InvalidOperationException.

Властивості і методи класу SqlDataReader, які найбільш часто використовуються, приведені відповідно в табл. 1.9 і табл. 1.10.

Таблиця 1.9

Властивості класу SqlDataReader

Властивість	Опис
FieldCount	Повертає кількість полів в об'єкті DataReader
HasRows	Доступно тільки для читання. Указує, чи повернув запит SqlCommand записи
IsClosed	Указує, чи закритий об'єкт DataReader. Доступно тільки для читання
Item	Повертає вміст вказаного поля поточного запису. Доступно тільки для читання
RecordsAffected	Указує кількість записів, зачеплених виконаними запитами. Доступно тільки для читання

Визначення .NET-типа даних, які використовуються для зберігання вмісту певного поля, виконується методом GetFieldType класу SqlDataReader. Він, подібно до методу GetName, приймає ціле число, яке відповідає порядковому номеру поля і повертає тип його даних. Якщо необхідно встановити тип даних поля, то використовується метод

GetDataTypeName класу SqlDataReader. Він приймає порядковий номер як ціле число і повертає рядок з назвою типу даних поля в базі даних.

Таблиця 1.10

Методи класу SqlDataReader

Метод	Опис
1	2
Close	Закриває об'єкт SqlDataReader
Get<DataType>	Повертає вміст поля поточного рядка по порядковому номеру поля
GetByte	Одержує з вказаного поля поточного запису масив байтів
GetChars	Одержує з вказаного поля поточного запису масив символів
GetDataTypeName	Повертає ім'я типу даних поля по порядковому номеру поля
GetFieldType	Повертає тип даних поля по порядковому номеру поля
GetName	Повертає ім'я поля по його порядковому номеру
GetOrdinal	Повертає порядковий номер поля по його імені
GetProviderSpecificFieldType	Повертає SqlType поля по його порядковому номеру
GetProviderSpecificValue	Повертає значення поля по його порядковому номеру
GetProviderSpecific Values	Повертає вміст поточного стовпця SqlType-об'єктів
GetSchemaTable	Повертає інформацію схеми (імена і типи даних полів) у вигляді об'єкта DataTable.
GetSqlValue	Повертає значення поля типу SqlType по його порядковому номеру
GetValue	Повертає значення поля по його порядковому номеру, як тип даних .NET
GetValues	Повертає вміст поточного стовпця, як тип даних .NET
IsDBNull	Указує чи містить поле значення Null

Закінчення табл. 1.10

1	2
NextResult	Забезпечує перехід до наступного результату
Read	Перехід до наступного запису

У приведеному нижче фрагменті коду застосовується переобтяжений метод ExecuteReader для повернення даних про рядкі таблиці. При цьому використовуються методи GetName, GetFieldType і GetDataTypeName, щоб відобразити інформацію про схему таблиці MyAnimal, наприклад:

```
string strConn, strSQL;
strConn = @"Data Source=.\SQLEXPRESS;AttachDbFilename=
Animals.mdf;Integrated Security=True;Connect Timeout=30;User
Instance=True";
        strSQL = "SELECT * FROM MyAnimal";
SqlConnection cn = newSqlConnection(strConn);
cn.Open();
SqlCommand cmd = newSqlCommand(strSQL, cn);
SqlDataReader rdr =
cmd.ExecuteReader(CommandBehavior.SchemaOnly);
for (int field = 0; field < rdr.FieldCount; field++)
      {   textBox1.Text += field;
          textBox1.Text += "   Name  " + rdr.GetName(field);
          textBox1.Text += "   .NET data type: " +
rdr.GetFieldType(field).Name;
          textBox1.Text += " Data type Database: " +
rdr.GetDataTypeName(field) + "\r\n";             }
         cn.Close();
```

Метод ExecuteReader переобтяжений і може приймати значення з переліку CommandBehavior. У табл. 1.11 приведені можливі значення цього переліку. Результат виконання вказаного вище коду представлений на рис. 1.2.

Метод GetSchemaTable об'єкта SqlDataReader аналогічний методу FillSchema об'єкта DataAdapter. Обидва створюють об'єкт DataTable з вкладеними об'єктами DataColumn. Метод GetSchemaTable не приймає яких-небудь параметрів і повертає новий об'єкт DataTable.

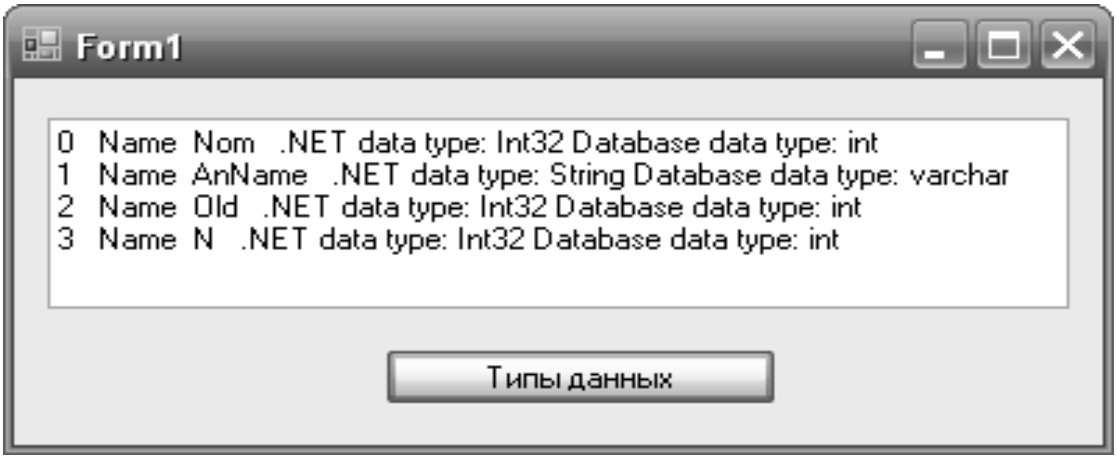

Рис. 1.2. **Результат виконання завдання визначення типу даних**

Можливо спочатку важко розібратися в даних, що повертаються методом GetSchemaTable, оскільки він повертає об'єкт DataTable із зумовленою структурою.

Таблиця 1.11

Елементи переліку CommandBehavior

Константа	Опис
CloseConnection	При закритті об'єкта SqlDataReader закривається з'єднання
KeyInfo	Об'єкт SqlDataReader одержує відомості про первинний ключ для стовпців, що входять в набір результатів
SchemaOnly	Об'єкт SqlDataReader містить тільки інформацію про стовпці, запит фактично не виконується
SequentialAccess	Значення стовпців доступні тільки в послідовному порядку. Наприклад, проглянувши вміст третього стовпця, ви вже не зможете проглянути вміст першого і другого стовпців
SingleResult	Об'єкт SqlDataReader містить результати тільки першого запиту, що повертає записи
SingleRow	Об'єкт SqlDataReader містить тільки перший запис, повернений запитом

Вкладені об'єкти DataRow поверненого методом об'єкта DataTable відповідають стовпцям набору результатів запиту, а об'єкти DataColumn – властивостям або атрибутам цих стовпців. Наступний фрагмент коду виводить для кожного поверненого запитом стовпця його ім'я і тип даних.

Найпростіший спосіб зрозуміти данні, що повертаються методом GetSchemaTable, – показати їх в Windows Forms DataGrid:

```
DataTable tableClient;
      OleDbConnection conn = new OleDbConnection();
      OleDbCommand comm;
conn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;
DataSource= E:\\ADO\\ ClientBaze \\bin\\Debug\\lr2.mdb";
        tableKlient = new DataTable("Client");
        comm = new OleDbCommand("SELECT * FROM Client", conn);
        conn.Open();
OleDbDataReader res =
                comm.ExecuteReader(CommandBehavior.KeyInfo);
        tableClient = res.GetSchemaTable();
        dataGridView1.DataSource = tableClient;
```

Цей фрагмент коду вирішує задачу визначення схеми таблиці, результати приведені на рис. 1.3. В об'єкті DataTable, що повертається методом GetSchemaTable, будуть доступні додаткові схеми, якщо виклику Sql Command.ExecuteDataReader буде переданий рядок CommandBehavior.KeyInfo.

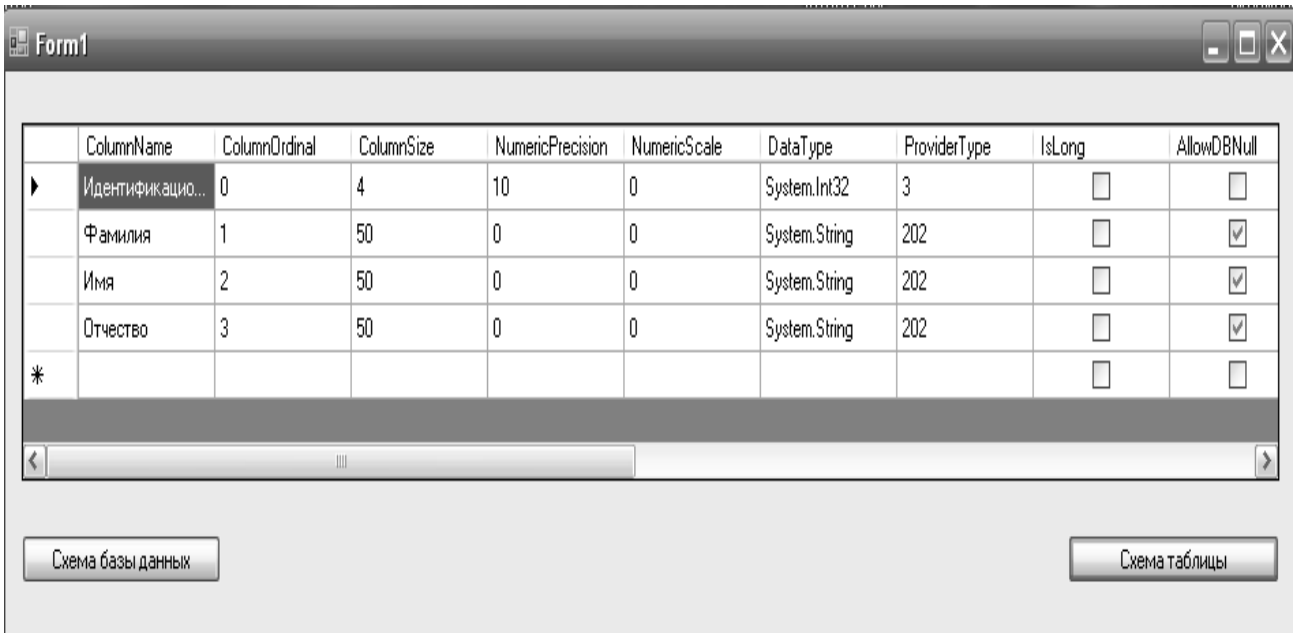

Рис. 1.3. **Результат рішення задачі визначення схеми таблиці**

Клас SqlDataReader надає методи Get для багатьох базових типів даних .NET: GetString GetInt32, GetDateTime та ін. Звичайно їх називають функціями Get, що строго типізуються. Використовуючи відповідний метод Get класу SqlDataReader можна добитися зниження

продуктивності через необхідність проводити додаткові перетворення типів даних. Наприклад, стовпець AnName містить дані типу String. Вміст стовпця можна одержати за допомогою методу GetValue об'єкта SqlDataReader, а потім привести дані до типу string:

```
SqlDataReader readerTable = cmd.ExecuteReader();
int  i = 1; // Номер стовпця в таблиці
string  str;
while (readerTable.Read())
{
    str =  (string) readerTable.GetValue (i);
    Console.WriteLine("{0}", readerTable.GetString(i));
}
readerTable.Close();
```

Скориставшись методом GetString об'єкта SqlDataReader, можна одержати вміст цього стовпця у вигляді рядка, як показано в наступному фрагменті коду. Хоча результат виходить той же, даний код працює швидше, ніж попередній:

```
SqlDataReader   readerTable = cmd.ExecuteReader();
int  i = 1; // Номер стовпця в таблиці
while (readerTable.Read())
Console.WriteLine("{0}", readerTable.GetString(i));
readerTable.Close();
```

Необхідно завжди використовувати відповідну функцію Get, що строго типізується, яка повертає стовпець з набору результатів. Виконання запитів з використанням невідповідної типу даних функції Get, що строго типізується, приводить до виняткової ситуації InvalidCastException. Тому заздалегідь слід визначати тип даних у базі, наприклад за допомогою методу GetFieldType.

Метод GetValues дозволяє помістити запис у масив. Якщо необхідно максимально швидко одержати вміст кожного поля, то цей метод ефективніший, ніж читання значень окремих полів, наприклад:

```
SqlDataReader readerTable = cmd.ExecuteReader();
object[ ] aData = new object[readerTable.FieldCount];
while (readerTable.Read())
{
readerTable.GetValues(aData);
Console.WriteLine(aData[0].ToString());     }
```

Методи GetSqlValues і GetProviderSpecificValues аналогічні методу GetValues, але працюють з типами даних SqlTypes, а не базовими типами даних .NET.

У процесі проектування застосувань може виникнути необхідність виконувати декілька запитів як єдиний пакет. Деякі СУБД, наприклад SQLServer, дозволяють виконувати такий пакет запитів, що повертає декілька наборів результатів. Припустимо, що необхідно виконати такий запит до бази даних:

```
SELECT * FROM  Animals;
SELECT * FROM  MyAnimal;
```

При виконанні запиту об'єкт DataReader, циклічно проглядають результати запиту до тих пір, поки метод Read не повертає значення False, тому без застосування спеціальних засобів код проглядатиме тільки перший набір результатів, тобто виконувати перший запит. Застосувавши об'єкт DataReader до тільки що приведеного пакетного запиту, ми одержимо данні тільки з таблиці Animals. Якщо скористатися методом NextResult об'єкта DataReader, то можна одержати результати наступного запиту, який повертає записи з таблиці MyAnimal. Цей метод повертає логічне значення, яке вказує чи є ще невиконані запити.

Наступний фрагмент коду вибирає результати пакетного запиту за допомогою методу NextResult:

```
cn.Open();
string strSQL;
strSQL = " SELECT * FROM Animals;" +
" SELECT * FROM MyAnimal;";
SqlCommand cmd = new SqlCommand(strSQL, cn);
SqlDataReader rdr = cmd.ExecuteReader();
do
{
while (rdr.Read())
Console.WriteLine("{0} - {1}", rdr[0], rdr[1]);
Console.WriteLine();
}
while (rdr.NextResult());
```

В об'єкті DataReader передбачені методи для всіх типів даних .NET Framework: GetByte, GetChar, GetDateTime і т. п. Клас SqlDataReader містить також методи, за допомогою яких можна повернути типи даних у таких просторах імен SqlTypes, як GetSqlDecimal і GetSqlString.

1.2.3. Виконання операцій оновлення даних з використанням параметрів

Дотепер розглядалися запити, в яких параметри рядка запиту відомі наперед. Наприклад, для методу ExecuteNonQuery рядок commandText може мати такий вигляд:

myCommand.CommandText = "UPDATE Туристи SET Прізвище = 'Петренко' WHERE Кодклієнта = 6";

Якщо ми створимо застосування, де користувач вводитиме прізвище і код клієнта, то ми не можемо у процесі виконання програми вказати, які це будуть значення. Логічно запит можна представити приблизно так:

myCommand.CommandText = "UPDATE Клієнти
SET Прізвище = 'Якесь прізвище, яке введе користувач'
WHERE Кодклієнта = Код, який введе користувач";

Для вирішення таких завдань, які виникли ще на самому початку розробки мови SQL, використовувалися параметрізовані запити. У них невідомі значення замінюються параметрами. Наприклад:

myCommand.CommandText = "UPDATE Клієнти
SET Прізвище = @Family WHERE Кодклієнта = @ClientID";

Тут @Family (зверніть увагу, пишеться без лапок) – параметр для невідомого значення прізвища, @ClientID –- параметр для невідомого значення коду клієнта. Тепер ми можемо прив'язувати параметри до тексту, що вводиться користувачем.

Щоб виконати запит, що параметризується, в об'єктній моделі ADO.NET, необхідно додати об'єкт Parameter в колекцію Parameters об'єкта Command. Якщо ви використовуєте постачальник даних SQLClient, то як клас Parameter потрібно застосовувати SqlParameter. Властивості об'єкта SqlParameter приведені в табл. 1.12.

Таблиця 1.12

Властивості об'єкта SqlParameter

Властивість	Опис
DbType	Указує тип даних параметра
Direction	Указує тип параметра: параметр введення (Input), виведення (Output), введення – виведення (Input-Output)
IsNullable	Указує, чи може параметр приймати значення null
ParameterName	Указує ім'я параметра
Precision	Указує точність параметра
Scale	Указує числову шкалу параметра
Size	Указує розмір параметра
SourceColumn	Указує ім'я стовпця в об'єкті DataSet, на який посилається даний параметр
SourceColumnNullMapping	Використовується для управління значеннями null
SourceVersion	Указує версію стовпця (поточну або оригінальну) в об'єкті DataSet, на який посилається
SqlDbType	Указує Sql-тип даних параметра
SqlValue	Указує значення параметра, який використовує SqlType
Value	Указує значення параметра
XmlSchemaCollectionDatabase XmlSchemaCollectionOwning Schema XmlSchemaCollectionName	Властивості використовуються для роботи з XML-значеннями параметрів

Додавання об'єкта Parameter в колекцію Parameters об'єкта Command можна виконати декількома способами, наприклад:

```
SqlCommand cmd = new SqlCommand (sql,cn);
SqlParameter par = new SqlParameter();
par.ParameterName = "@ClientID";
par.Value = id;
par.SqlDbType = SqlDbType.Int;
cmd.Parameters . Add ( par );
par = new SqlParameter();
par.ParameterName = "@Family";
```

```
par.Value = fam;
par.SqlDbType = SqlDbType. NVarChar;
par.Size = 20;
cmd.Parameters .Add( par );
```

Простіший спосіб створення об'єкта класу SqlParameter показаний в наступному коді:

```
cmd.Parameters.Add("@ClientID", SqlDbType.Int, 4);
cmd.Parameters.Add("@Family", SqlDbType.NVarChar,20);
```

Ще один простий спосіб створення класу SqlParameter – виклик методу AddWithValue властивості Parameters об'єкта SqlCommand, як приведено в наступному коді:

```
cmd.Parameters.AddWithValue("@Family ", "Петренко");
```

У останньому прикладі об'єкт SqlParameter визначив тип даних, виходячи з вмісту властивості Value. Оскільки властивості Value привласнений рядок, який в .NET є послідовністю знаків Unicode, об'єкт SqlParameter неявно привласнює властивості SqlDbType значення SqlDbType.NVarChar, а властивості Size – значення 8.

Розглянемо приклад, в якому інформація про код клієнта і прізвища клієнта вводиться за допомогою компонентів textBox1 і textBox2 відповідно.

```
// Створюємо змінну Family, в яку поміщаємо значення
 // введене користувачем у поле textBox1:
    string Family = Convert.ToString(textBox1.Text);
// Створюємо змінну ClientID, в яку поміщаємо значення
 // введене користувачем у поле textBox2:
    Int ClientID = int.Parse(textBox2.Text);
    conn = new SqlConnection();
conn.ConnectionString = "…….рядок з'єднання……";
        conn.Open();
        SqlCommand myCommand = conn.CreateCommand();
myCommand.CommandText = "UPDATE Клієнти SET Прізвище =
                                    @Family WHERE ClientID = @ClientID";
// Додаємо параметр @Family в колекцію параметрів об'єкта
// myCommand
myCommand.Parameters.Add("@Family",  SqlDbType.NVarChar, 20);
// Встановлюємо значення параметра @Family
 // рівним значенню змінної Family
        myCommand.Parameters["@Family"].Value = Family;
```

// Додаємо параметр @ClientID у колекцію параметрів об'єкта
// myCommand

myCommand.Parameters.Add("@V", SqlDbType.Int, 4);

// Встановлюємо значення параметра @ClientID
// рівним значенню змінної ClientID

myCommand.Parameters["@ClientID"].Value = ClientID;

int zmineno = myCommand.ExecuteNonQuery();

У попередніх прикладах у запиті використовувався параметр для того, щоб передати дані в базу. Це приклад параметра введення. Параметри можна використовувати і для отримання даних з бази даних. За допомогою властивості Direction указуються типи параметрів, які приведені в табл. 1.13.

Таблиця 1.13

Елементи переліку ParameterDirection

Константа	Опис
Input	Параметр використовується тільки для введення
Output	Параметр використовується тільки для виведення
InputOutput	Параметр використовується тільки для введення – виведення
Return	Параметр що повертається містить значення процедури, що зберігається

Припустимо, необхідно одержати єдиний запис з бази даних, використовуючи параметри виведення, а не перевіряти запис за допомогою об'єкта SqlDataReader. Повернення цих даних за допомогою параметрів виведення відбувається швидше, тому що для них потрібно менше ресурсів.

Наступний запит, за допомогою параметра введення, шукає у таблиці **Clients** рядок за значенням стовпця **ClientID,** а за допомогою параметрів виведення повертає значення стовпців **ClientID** і **Family**:

strSQL = "SELECT @ClientID = ClientID, @Family = Family " +
"FROM Clients WHERE ClientID = @ ClientID";

SqlCommand cmd = new SqlCommand(strS0L, connection);

SqlParameter paramClient, paramFamily;

paramClient = cmd.Parameters.Add("@ClientID", SqlDbType.Int);
paramClient.Direction = ParameterDirection.Output;
paramFamily = cmd.Parameters.Add("@Family",
SqlDbType.NVarChar, 20);

```
paramFamily.Direction = ParameterDirection.Output;
    paramClient.Value = 4;
    cmd.ExecuteNonQuery();
```

Якщо жоден рядок у таблиці **Clients** не відповідатиме критерію, вказаному у виразі WHERE, то запит успішно виконується, але властивості Value для всіх параметрів виведення привласнюється значення DBNull.Value. Так, щоб визначити, чи існує в таблиці **Clients** задовольняючий критерію рядок, потрібно включити в код перевірку після виклику методу ExecuteNonQuery, як це показано нижче:

```
int UspeshnoeIzmenenie = myCommand.ExecuteNonQuery();
if (UspeshnoeIzmenenie !=0)
{    MessageBox.Show( " Зміни внесені  " );   }
else
{   MessageBox.Show( " Не вдалося внести зміни",  }
```

Висновки. У лекції розглянуто виконання операцій обробки даних з використанням класів SqlCommand і SqlDataReader. Приведені приклади створення командних об'єктів і запуску командних об'єктів. Розглянуті особливості використання бібліотеки базових класів командних об’єктів, які надають широкі можливості користувачу для роботи з базами даних. Особлива увага приділяється виконанню операцій оновлення даних з використанням параметрів.

Тести для самодіагностики

Тест № 1. Ви виконуєте запит до бази даних з допомогою об'єкта OleDbCommand. Запит використовує функцію для повернення значення, що представляє середню ціну продуктів. Ви хочете оптимізувати роботу запиту. Який метод ви повинні використовувати:

1) ExecuteNonQuery;
2) ExecuteScalar;
3) ToString;
4) ExecuteReader?

Тест № 2. Ви повинні створити об'єкт OleDbCommand, щоб одержати інформацію про поштові коди для списку адресатів. Ви повинні встановити властивості CommandText і Connection. Які два можливі сегменти коду:

1) OleDbCommand comm = new OleDbCommand();
comm.CommandText="SELECT * FROM Regions";comm.Connection=conn;

2) OleDbCommand comm = new OleDbCommand("sp_GetRegions", conn); comm.CommandType = CommandType.Text;

3) OleDbCommand comm = new OleDbCommand("SELECT * FROM Regions", conn); comm.CommandType = CommandType.Text;

4) OleDbCommand comm = new OleDbCommand("sp_GetRegions", conn); comm.CommandType = CommandType.TableDirect;

5) OleDbCommand comm = new OleDbCommand(); comm.CommandType=SELECT* FROM Regions";comm.Connection=conn;?

Контрольні завдання

Завдання № 1. Визначити кількість оброблених записів у кожній таблиці під час виконання пакетного запиту.

Завдання № 2. Застосування виконує запит до бази даних, результати якого можуть отримуватися із затримкою. Необхідно передбачити забезпечення виконання яких-небудь дій на головній формі застосування протягом цієї затримки.

Література: [2, с. 21–28; 3, с. 96–155].

1.3. Лекція № 3. Виконання операцій в роз’єднаному середовищі

1.3.1. Клас DataSet

У процесі виконання операцій у сполученому середовищі весь час роботи необхідно тримати відкритим зв'язок з джерелом даних. Такий метод роботи сильно завантажує мережеві ресурси. Для зниження завантаження мережевих ресурсів є режим роботи в роз'єднаному середовищі, який дозволяє використовувати ресурси сервера лише тоді, коли програма читає або записує дані у сховищі.

ADO.NET пропонує декілька класів для забезпечення роботи застосування в роз'єднаному середовищі. Одним з таких класів є клас DataSet, який розроблений автономним сховищем даних і дозволяє виконувати такі особливості:

після отримання результатів запиту в об'єкті DataSet з’єднання між базою даних і об'єктом DataSet припиняється;

зміни вмісту об'єкта DataSet не позначаються на вмісті бази даних;

якщо інші користувачі змінять дані в базі даних, то ці зміни не вплинуть на вміст об'єкта DataSet;

об'єкт DataSet дає можливість проглядати результати запиту різними способами (дані у ньому дозволено сортувати по окремому полю або групі полів, можна шукати дані згідно з простим критерієм пошуку або відображати тільки записи, що задовольняють заданим критеріям);

об'єкти DataSet призначені для роботи з ієрархічно організованими даними, можна визначити відносини між таблицями даних, що зберігаються в об'єкті;

об'єкт DataSet дозволяє кешувати зміни запису даних і потім передавати ці зміни в базу даних;

об'єкт DataSet розрахований на роботу з XML, його вміст можна завантажувати і зберігати у вигляді XML-документів, крім того, об'єкт DataSet дає можливість виділити інформацію схеми (відомості про таблиці, стовпці і обмеження) у файл XML-схеми.

Клас DataSet складається з набору таблиць, кожна з яких містить безліч рядків і стовпців даних, при цьому можуть бути визначені зв'язки між таблицями. Можна сказати, що цей клас дозволяє створювати копію бази даних в оперативній пам'яті. У табл. 1.14 приведені властивості, які складають структуру даних DataSet: Tables, Relations і ExtendedProperties.

Таблиця 1.14

Клас DataSet і його основні складові

Властивість	Опис
1	2
Tables	Об'єкт типу System. Data. DataTableCollection, який може містити об'єкти System.Data.DataTable. У кожного DataTable є колекції з іменами Columns і Rows, які містять об'єкти DataColumn або DataRow
Relations	Об'єкт типу System.Data.DataRelationCollection містить об'єкти DataRelation, які визначають відношення предок / нащадок між двома об'єктами DataTable
ExtendedProperties	Об'єкт типу System.Data.PropertyCollection, який може містити властивості, визначені користувачем. Колекцію можна використовувати для зберігання даних, таких як час створення об'єкта
CaseSensitive	Визначає чи розрізняється регістр символів при порівнянні рядків
DataSetName	Визначає ім'я об'єкта DataSet

Закінчення табл. 1.14

1	2
DesignMode	Указує, чи знаходиться об'єкт DataSet в режимі проектування
EnforceConstraints	Визначає чи забезпечує об'єкт DataSet виконання визначених на ньому обмежень
HasErrors	Указує чи містить об'єкт DataSet помилки
IsInitialized	Указує чи був об'єкт DataSet ініціалізований
Locale	Визначає регіональні параметри, які використовуються об'єктом DataSet при порівнянні рядків
Namespace	Містить простір імен, який використовується при записі вмісту об'єкта DataSet в XML-файл або при завантаженні XML-даних в об'єкт DataSet
Prefix	Містить префікс простору імен, який вживається при записі вмісту об'єкта DataSet в XML-файл або при завантаженні XML-даних в об'єкт DataSet.
RemotingFormat	Визначає формат серіалізації
SchemaSerialization Mode	Визначає чи буде включена схема DataSet в процесі серіалізації
DesignMode	Указує, чи знаходиться об'єкт DataSet в режимі проектування
EnforceConstraints	Визначає чи забезпечує об'єкт DataSet виконання визначених на ньому обмежень
HasErrors	Указує чи містить об'єкт DataSet помилки
IsInitialized	Указує чи був об'єкт DataSet ініціалізований
Locale	Визначає регіональні параметри, які використовуються об'єктом DataSet при порівнянні рядків
Namespace	Містить простір імен, який використовується при записі вмісту об'єкта DataSet в XML-файл або при завантаженні XML-даних в об'єкт DataSet
Prefix	Містить префікс простору імен, який вживається при записі вмісту об'єкта DataSet в XML-файл або при завантаженні XML-даних в об'єкт DataSet.
RemotingFormat	Визначає формат серіалізації
SchemaSerialization Mode	Визначає чи буде включена схема DataSet в процесі серіалізації

Для створення об'єктів типу DataSet можна скористатися одним з чотирьох способів:

створення програмним шляхом;

використовуючи графічну утиліту з пакета Microsoft Visual Studio .NET;

генерувати при створенні об'єкта DataAdapter;

завантажувати структуру і дані в DataSet з XML-файлів.

У цьому розділі розглядатимуться способи створення об'єктів тільки програмним шляхом.

Створити об'єкт типу DataSet можна за допомогою двох конструкторів. Перший конструктор приймає як параметр рядок, який містить ім'я об'єкта, другий конструктор дозволяє створювати об'єкт DataSet, не приймаючи параметрів, в цьому випадку ім'я нового об'єкта буде NewDataSet, наприклад:

```
DataSet data=new DataSet("База_даних");
DataSet data=new DataSet();
```

Ім'я для об'єкта типу DataSet необхідне, щоб гарантувати наявність імені в документі XML.

Об'єкт DataSet володіє широкими можливостями завдяки безлічі властивостей і методів, які приведені в табл. 1.14 та табл. 1.15, їх використання буде розглянуто по мірі викладення навчального матеріалу.

Таблиця 1.15

Методи класу DataSet

Метод	Опис
1	2
AcceptChanges	Підтверджує всі відкладені зміни в об'єкті DataSet
Clear	Видаляє з об'єкта DataSet всі об'єкти DataRow
Clone	Створює новий об'єкт DataSet з ідентичною схемою, але без об'єктів DataRow
Copy	Створює новий об'єкт DataSet з такою ж схемою і об'єктами DataRow
CreateDataReader	Повертає об'єкт DataTableReader, який містить дані з об'єктів DataTable, які заявлені у виклику методу
GetChanges	Повертає новий DataSet з ідентичною схемою, який містить змінені записи оригінального об'єкта DataSet
GetXml	Повертає вміст об'єкта DataSet у вигляді XML-рядка

Закінчення табл. 1.15

1	2
GetXmlSchema	Повертає схему об'єкта DataSet у вигляді XML-рядка
HasChanges	Повертає логічне значення, вказуюче, чи містять об'єкти DataRow зі складу DataSet відкладені зміни
InferXmlSchema	Завантажує інформацію з XML-схеми, дозволяє задати список просторів імен, елементи яких треба виключити з схеми об'єкта DataSet
Load	Завантажує дані з об'єкта DataReader в об'єкт DataTable
Merge	Виконує злиття даних з іншим об'єктом DataSet, DataTable або масивом об'єктів DataRow
ReadXml	Завантажує XML-дані в об'єкт DataSet з файла, об'єкта Stream, TextReader або XmlReader
ReadXmlSchema	Завантажує інформацію XML-схеми в об'єкт DataSet з файла, об'єкта Stream, TextReader або XmlReader
RejectChanges	Відміняє всі відкладені зміни в об'єкті DataSet
Reset	Відновлює початковий стан об'єкта DataSet, в якому він знаходився до ініціалізації
WriteXml	Записує вміст об'єкта DataSet в XML-форматі у файл, об'єкт Stream, TextReader або XmlReader
WriteXmlSchema	Записує схему об'єкта DataSet в XML-форматі у файл, об'єкт Stream, Text Reader або XmlReader

1.3.2. Клас DataTable

Об'єкт DataTable створюється так само, як і об'єкт DataSet, а його додатковий конструктор дає можливість задати значення властивості TableName цього об'єкта, наприклад:

```
DataTable tablica = new DataTable();
DataTable tablica = new DataTable("Products");
```

Створений об'єкт DataTable можна за допомогою методу Add класу DataTableCollection включити в колекцію Tables наявного об'єкта DataSet, наприклад:

```
DataSet ds = new DataSet();
DataTable tablica = new DataTable("Products");
ds.Tables.Add(tablica);
```

Створити новий об'єкт DataTable і додати його в колекцію Tables об'єкта DataSet можна за допомогою одного рядка, наприклад:

DataSet ds = new DataSet();

DataTable tablica = ds.Tables.Add("Products");

Клас DataTable реалізує всі стандартні інтерфейси, які реалізовані в DataSet, тому він може виконувати ті ж функції, що і DataSet. Клас DataTable містить колекцію Columns, колекцію Rows і колекцію Constraints. Колекції Columns і Constraints разом утворюють схему для DataTable, а колекція Rows містить самі дані. Ці три колекції описані в табл. 1.16.

Таблиця 1.16

Колекції класу DataTable

Колекція	Опис
Columns	Екземпляр класу System.Data.DataColumnCollection, який може містити об'єкти DataColumn. Об'єкти DataColumn визначають властивості кожного стовпця з DataTable ім'я, тип даних, що зберігається, і первинний ключ
Rows	Екземпляр класу System.Data.DataRowCollection, який може містити об'єкти DataRow
Constraints	Екземпляр класу System.Data.ConstraintCollection, який містить об'єкти System.Data.ForeignKeyConstrant або System.Data.UniqueConstraint. Перші визначають дію, що виконується над стовпцем у відношенні первинний ключ – зовнішній ключ при зміні або видаленні рядка, другі – для забезпечення унікальності значень у даному стовпці

Клас DataTable має безліч властивостей і методів, які дозволяють вирішувати широкий круг завдань. У справжньому конспекті лекцій перелік властивостей і методів класу DataTable не приводиться з огляду на те, що багато хто з них співпадає з властивостями і методами класу DataSet. Як найповніший перелік цих характеристик приведений [2].

1.3.3. Клас DataColumn.

Новий об'єкт DataColumn можна створити або за допомогою конструктора DataColumn, або викликавши метод Add () властивості-колекції DataTable.Columns:

// Додавання стовпця за допомогою конструктора

DataColumn myColumn = new DataColumn("ID", typeof(System.Int32));
// Додавання стовпця за допомогою DataTable
productsTable.Columns.Add("ID", Type.GetType("System.Int32"));

Використана тут версія методу DataTable.Columns.Add() приймає два аргументи: ім'я нового об'єкта DataColumn і об'єкт Type (для другого можно використовувати методи typeof або GetType). Властивість Columns має тип DataColumnCollection. Взагалі є чотири інших переобтяжених версії цього методу, доступні для DataColumnCollection, нижче описано їх використання:

Add () – створює новий об'єкт DataColumn і додає його в DataColumnCollection (а, отже, і додає новий стовпець у таблицю), якщо нічого не треба указувати, то новому об'єкту DataColumn привласнюється стандартне ім'я ("Column1", "Column2" і т. д.):

dataTable.Columns.Add();

Add("Им'я_Стовбця") – створює і додає у таблицю новий об'єкт DataColumn з вказаним ім'ям, якщо тип даних не вказаний, то за замовчуванням приймається тип System.String, наприклад:

dataTable.Columns.Add("ProductID");

Add(myDataColumn) – додає в DataColumnCollection вказаний існуючий об'єкт DataColumn:

DataColumn coll=new DataColumn();

dataTable.Columns.Add(coll);

Add("SubTotal",Type.GetType("System.Single"),"Sum(Price)") – створює і додає в таблицю об'єкт DataColumn з вказаним ім'ям, типом даних і властивістю Expression (табл. 1.17), вираз використовується для фільтрації рядків, обчислення значень у стовпцях або (як в даному випадку) для створення агрегованого стовпця, наприклад:

dataTable.Columns.Add(("ProductCode",
System.Type.GetType("System.Int32"), "ProductID*2"));

Властивості об'єкта DataColumn приведені в табл. 1.17.

Таблиця 1.17

Властивості об'єкта DataColumn

Властивість	Опис
1	2
AllowDBNull	Визначає, чи допустимі у стовпці значення Null
AutoIncrement	Визначає, чи генеруються нові значення автоінкремента у стовпці

Закінчення табл. 1.17

1	2
AutoIncrementSeed	Визначає початкове значення автоінкремента
AutoIncrementStep	Визначає значення, яке буде використане для генерування подальших значень автоінкремента
Caption	Містить заголовок стовпця
ColumnMapping	Визначає, як записується вміст стовпця в XML-документ
ColumnName	Містить ім'я об'єкта DataColumn
Datatype	Визначає тип даних стовпця
DateTimeMode	Визначає формат серіалізації для полів DateTime
DefaultValue	Визначає значення за замовчуванням, яке використовується при заповненні поля в нових записах
Expression	Управляє генерацією значень полів, заснованих на виразі
Maxlength	Задає максимально допустиму довжину рядка
Ordinal	Повертає порядковий номер об'єкта DataColumn у колекції Columns об'єкта DataTable
Prefix	Містить префікс простору імен, який використовується при записі вмісту DataSet в XML-файл або завантаженню XML-даних в об'єкт DataSet
ReadOnly	Визначає чи доступний вміст стовпця тільки для читання
Table	Повертає об'єкт DataTable, до складу якого входить DataColumn
Unique	Визначає, чи повинні бути значення цього стовпця унікальними в межах об'єкта DataTable

У наступному фрагменті коду створюється новий об'єкт DataTable за допомогою одного з трьох його конструкторів (інші два дозволяють створити таблицю із стандартним ім'ям і задати як ім'я таблиці, так і її простір імен). Після створення об'єкта визначається схема таблиці за допомогою створення в колекції Columns трьох нових стовпців:

```
// Створення нового DataTable
    DataTable productsTable = new DataTable("Products");
// Створення схеми Products
```

```
productsTable.Columns.Add("ID", typeof(System.Int32));
productsTable.Columns.Add("Name", typeof(System.String));
products Table. Columns .Add ("Category", typeof (System. Int 32) ) ;
```

Для того, щоб створити схему нової таблиці, можна викликати метод Add() колекції DataTable.Columns для кожного стовпця, який потрібно додати в DataTable. Для кожного DataColumn можна передати через аргументи значення властивостей ColumnName і DataType. Результатом буде об'єкт DataTable з ім'ям Products, складений з трьох стовпців з іменами "ID", "Name" і "Category" і типами Int32, String і Int32 відповідно.

У об'єкті DataTable повинен бути присутнім первинний ключ. У загальному випадку він визначається як масив об'єктів DataColumn, який забезпечує унікальність ідентифікатора DataRow. Для створення первинного ключа необхідно у властивості **PrimaryKey** об'єкта DataTable вказати масив об'єктів DataColumn, наприклад:

```
productsTable.PrimaryKey = new DataColumn[ ]
                                { productsTable.Columns["ID"]};
```

При такому визначенні первінного ключа на масив об'єктів DataColumn накладається обмеження UniqueConstraint.

1.3.4. Клас DataRow

Після створення об’єкта DataTable і визначення стовпців можна приступити до заповнення таблиці даними. При цьому в колекцію DataTable.Rows типу DataRowCollection додаються нові об'єкти DataRow. Щоб створити в DataTable новий рядок, необхідно спочатку викликати метод DataTable.NewRow (), який повертає об'єкт DataRow, що задовольняє поточній схемі DataTable, наприклад:

```
DataRow tempRow = productsTable.NewRow();
// Установка значень стовпців
tempRow.Item["ID"] = 1;
tempRow.Item["Name"] = " Хліб ";
tempRow.Item["Category"] = 1;
// Додавання DataRow в DataTable
productsTable.Rows.Add(tempRow);
```

У даному прикладі до об'єкта productsTable додається один рядок. Спочатку створюється новий об'єкт DataRow (tempRow) за допомогою схеми для DataTable і виклику методу productsTable.NewRow (). Потім, з

допомогою властивості Item (табл. 1.18), задаються значення для кожного із стовпців, які визначені в колекції productsTable.Columns. І, нарешті, викликається метод productsTable.Rows.Add(), який додає новий об'єкт DataRow в колекцію productsTable.Rows. Властивості об'єкта DataRow приведені в табл. 1.18.

Таблиця 1.18

Властивості об'єкта DataRow

Властивості	Опис
HasErrors	Указує, чи містить поточний запис помилки
Item	Повертає або задає вміст поля
ItemArray	Повертає або задає вміст запису
RowError	Повертає або задає відомості про наявність у записі помилок
RowState	Повертає стан запису
Table	Повертає об'єкт DataTable, до складу якого входить запис

Зі всіх властивостей об'єкта DataRow (табл. 1.18) найчастіше використовується властивість RowState. Проглянувши її значення, можна визначити чи був запис змінений, дабавлений або видалений. Перелік можливих значень властивості RowState приведений в табл. 1.19.

Таблиця 1.19

Значення властивості RowState

Значення	Опис
Added	Рядок був добалений в таблицю
Deleted	Рядок помічений для видалення
Detached	Рядок був створений, але для цього рядка ще не був викликаний метод Add
Modified	Рядок був модифікований
Unchanged	Рядок не був змінений

Об'єкт DataRow має метод AcceptChanges, який підтверджує всі відкладені зміни даних і неявно задає значення властивості RowState. Виклик методу AcceptChanges міняє значення властивості RowState. Якщо рядок був доданий (Added) або змінений (Modified), то після використання методу AcceptChanges він буде помічений Unchanged.

Якщо рядок був видалений (Deleted), то після використання методу AcceptChanges він буде помічений Detached. Приведемо приклад змін властивості RowState об'єкта DataRow:

```
        DataRow myRow;
// Створення нового об'єкта DataRow.
        myRow = myTable.NewRow();
        Console.WriteLine(myRow.RowState.ToString());
// Додавання рядка в таблицю.
        myTable.Rows.Add(myRow);
        Console.WriteLine(myRow.RowState.ToString());
// Підтвердження змін.
        myTable.AcceptChanges();
        Console.WriteLine(myRow.RowState.ToString());
// Модифікація даних рядка.
        myRow["Category"] = 100;
        Console.WriteLine(myRow.RowState.ToString());
// Видалення рядка.
        myRow.Delete();
        Console.WriteLine(myRow.RowState.ToString());
```

Нижче приведений результат виконання цього фрагмента програми:

Зміна властивості RowState

Detached

Added

Unchanged

Modified

Deleted

У другому рядку програми об'єкт DataRow, створений методом NewRow, ще не приєднаний до DataTable. Іншими словами, його властивість RowState дорівнює Detached. Цей рядок не входить в таблицю, поки вона не буде додана в колекцію Rows за допомогою виклику Rows.Add.

1.3.5. Клас DataAdapter

В об'єктній моделі ADO.NET дуже важливе місце займає клас DataAdapter. Він забезпечує зв'язок між автономними об'єктами класів DataSet, DataTable та іншими з підключеними до бази даних об'єктами DbCommand. DataAdapter не тільки дозволяє заповнити об'єкт DataSet або DataTable з джерела даних, але і надає зручний механізм внесення змін у базу даних.

Для створення об'єкта DataAdapter можна використовувати чотири конструктори, приведені в табл. 1.20.

Таблиця 1.20

Конструктори об'єкта DataAdapter

№	Вид	Опис
1	public DataAdapter ()	Конструктор за замовчуванням
2	public DataAdapter (string commandText, string connectionString)	Перший параметр є властивість CommandText об'єкта Command. У свою чергу, об'єкт Command представляється властивістю SelectCommand об'єкта DataAdapter. Другий параметр – рядок підключення до бази даних ConnectionString
3	public DataAdapter (string commandText, Connection connection)	Перший параметр – властивість CommandText об'єкта Command, другий – об'єкт класу Connection
4	public DataAdapter (Command command)	Як параметр передається об'єкт класу Command

У подальших прикладах вважатимемо, що об'єкти **commandText** і **connectionString** визначені таким чином:

```
string commandText = "SELECT * FROM Clients";
string connectionString = @"Data Source=.\SQLExpress;" +
    "Initial Catalog=BAZA; Integrated Security=True;";
```

При застосуванні першого конструктора об'єкта DataAdapter можна використовувати такий код:

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = connectionString;
SqlCommand myCommand = new SqlCommand();
myCommand.Connection = conn;
myCommand.CommandText = commandText;
SqlDataAdapter dataAdapter = new SqlDataAdapter();
dataAdapter.SelectCommand = myCommand;
```

Другий варіант конструктора об'єкта DataAdapter виглядатиме так:

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = connectionString;
SqlDataAdapter dataAdapter = new SqlDataAdapter(commandText,
connectionString);
```

У даному прикладі ми відмовилися від застосування екземпляра myCommand класу SqlCommand. Застосовуючи конструктор об'єкта

dataAdapter, в якому задаються обидва рядки commandText і connectionString, ми робимо непотрібним введення об'єкта myCommand. Для інших завдань, наприклад, для запуску процедури, що зберігається, об'єкт myCommand може знадобитися.

Значення рядків commandText і connectionString можна задавати прямо в конструкторі, тоді їх не потрібно оголошувати у класі форми, наприклад:

```
SqlDataAdapter dataAdapter = new SqlDataAdapter(
"SELECT ClientID, ClientName FROM Clients",
@"Data Source=.\SQLExpress;
Initial Catalog=BAZA; Integrated Security= True;"; );
```

Якнайкращим способом управління даними в ланцюжку "декілька об'єктів DataTable об'єкта DataSet – декілька таблиць в базі даних" є створення окремих об'єктів DataAdapter для кожного екземпляра DataTable. Уявимо собі, що DataSet складається з десяти таблиць (які представлені відповідними об'єктами DataTable). Нам знадобиться десять екземплярів DataAdapter для управління даними. Якщо ці десять екземплярів ми створюватимемо другим конструктором, куди подаються рядки запиту і підключення, то в результаті з'являться десять незалежних об'єктів класу Connection, однакових за своєю суттю. Продуктивність такого застосування буде низкою.

Наступний фрагмент коду є прикладом застосування третього варіанту конструктора для створення адаптера, що приймає рядок запиту і об'єкт Connection. При цьому створюється два об'єкти DataAdapter, які використовують один об'єкт Connection:

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = connectionString;
SqlDataAdapter dataAdapter1 = new
SqlDataAdapter(commandText1, conn);
// Створюємо другий екземпляр dataAdapter2 класу DataAdapter
SqlDataAdapter dataAdapter2 = new
SqlDataAdapter(commandText2, conn);
conn.Open();
// Використовуємо  dataAdapter1 і dataAdapter2 для заповнення даними
conn.Close();
```

У рядках commandText1 і commandText2 застосовується початковий рядок commandText, в якому розділені вибирані поля навпіл:

```
string commandText1 = "SELECT ClientID, ClientName
                       FROM Clients";
string commandText2 = "SELECT Address, City, Phone
                       FROM Clients";
```

У конструкторі можна передати значення цих рядків безпосередньо, наприклад:

```
SqlDataAdapter dataAdapter1 = new SqlDataAdapter(
"SELECT ClientID, ClientName  FROM Clients", conn);
// Створюємо другий екземпляр dataAdapter2 класу DataAdapter
SqlDataAdapter dataAdapter2 =
        new SqlDataAdapter("SELECT Address, City, Phone
        FROM Clients",conn);
```

Четвертий спосіб створення об'єкта **DataAdapter** заснований на використанні конструктора, який включає об'єкт Command, наприклад:

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = connectionString;
SqlCommand myCommand = new SqlCommand();
myCommand.Connection = conn;
myCommand.CommandText = commandText;
SqlDataAdapter dataAdapter = new SqlDataAdapter(myCommand);
```

Який з цих способів краще всього використовувати? Важко дати універсальну рекомендацію – у конкретному випадку найбільш відповідним може бути один з цих способів.

Об'єкт DataAdapter має безліч властивостей, які дозволяють ефективно управляти процесами читання і оновлення даних. Властивості об'єкта SqlDataAdapter приведені в табл. 1.21.

Таблиця 1.21

Властивості об'єкта SqlDataAdapter

Властивість	Опис
1	2
AcceptChangesDuringFill	Управляє значенням властивості RowState записів, що повертаються об'єктом SqlDataAdapter (значення – True, False)
AcceptChangesDuring Update	Визначає чи викличе об'єкт SqlDataAdapter метод AcceptChanges після передачі оновлень в DataRow (значення за замовчуванням – True)

Закінчення табл. 1.21

1	2
ContinueUpdateOnError	Визначає чи продовжить об'єкт SqlDataAdapter передавати зміни при виникненні помилки
DeleteCommand InsertCommand UpdateCommand SelectCommand	Використовується для передачі відкладених видалень, вставок, оновлень, виконання запитів до бази даних
FillLoadOption	Визначає, як об'єкт DataTable обробляє завантаження записів, які вже існують у ньому
TableMapping	Колекція, яка використовується для відображення імен таблиць
MissingMappingAction	Визначає дії об'єкта SqlDataAdapter при виявленні стовпців, яких немає в колекції TableMappings
MissingSchemaAction	Визначає дії об'єкта SqlDataAdapter при виявленні стовпців, яких немає в колекції Columns об'єкта DataTable
UpdateBatchSize	Визначає кількість записів, які оновлюються в одному пакеті об'єктом SqlDataAdapter

Застосування властивості SelectCommand було розглянуто в попередніх прикладах. Використання інших властивостей об'єкта **DataAdapter** буде викладено у міру розгляду його методів. Методи об'єкта DataAdapter приведені в табл. 1.22.

Таблиця 1.22

Методи об’єкта SqlDataAdapter

Метод	Опис
1	2
Fill	Виконує SelectCommand запит і поміщає його результати до об'єкта DataTable
FillSchema	Одержує інформацію схеми для запиту, що зберігається у властивості SelectCommand
GetFillParameters	Повертає масив з параметрами для властивості SelectCommand
Update	Передає в базу даних зміни, що зберігаються в об'єкті DataSet (DataTable або DataRows)

Метод Fill об'єкта DataAdapter використовується для заповнення даними DataSet, при цьому він є переобтяженим. У табл. 1.23 приводится опис різних конструкторів цього методу.

Таблиця 1.23

Конструктори методу Fill

№	Конструктор	Параметри конструктора
1	DbDataAdapter.Fill(System.Data.DataSet dataSet, string startRecord, string maxRecords, string TableName)	Об'єкт DataSet, початковий запис, кількість записів, таблиця-джерело
2	DbDataAdapter.Fill(System.Data.DataSet dataSet, string TableName)	Об'єкт DataSet, таблиця-джерело
3	DbDataAdapter.Fill(System.Data.DataSet dataSet)	Об'єкт DataSet
4	DbDataAdapter.Fill(System.Data.DataTable dataTable)	Об'єкт DataTable

Нижче приведений приклад використання конструкторів методу Fill:

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = connectionString;
SqlCommand myCommand = new SqlCommand();
myCommand.Connection = conn;
myCommand.CommandText = commandText;
SqlDataAdapter dataAdapter = new SqlDataAdapter();
dataAdapter.SelectCommand = myCommand;
DataSet dataSet = new DataSet();
  conn.Open();
        dataAdapter.Fill(dataSet, 4, 12," Clients ");
// або   dataAdapter.Fill(dataSet, "Clients");
// або   dataAdapter.Fill(dataSet);
// або   DataTable datatableClients = dataset.Tables.Add("Clients");
//        dataAdapter.Fill(datatableClients);
  conn.Close();
```

У процесі виконання методу Fill встановлюватиметься з'єднання з базою даних на час витягання даних. Тут методи Open() і Close() об'єкта

Connection є необов'язковими – іншими словами, метод Fill сам відкриє і закриє з'єднання, коли йому буде потрібно. SqlDataAdapter – вельми акуратний об'єкт. Він завжди повертає об'єкт SqlConnection, вказаний у властивості SelectCommand, в його початковий стан. Якщо ви відкриєте об'єкт SqlConnection до виклику методу Fill, після закінчення виклику з'єднання залишиться відкритим. Наприклад (як виконується відкриття і закриття з'єднання?):

```
string strConn, strSQL;
strConn = @"Data Source=.\SQLExpress;" +
                    "Initial Catalog=BAZA;Integrated Security=True;";
SqlConnection con = new SqlConnection(strConn);
SqlDataAdapter daClients, daDoc;
strSQL = "SELECT ClientID, ClientName FROM Clients";
daClients = new SqlDataAdapter(strSQL, con);
strSQL = "SELECT DocID, DocDate FROM Document";
daDoc = new SqlDataAdapter(strSQL, con);
DataSet ds = new DataSet();
daClients.Fill(dataSet, "Clients");
daDoc.Fill(dataSet, " Document ");
```

У приведеному прикладі, викликаючи кожного разу метод Fill об'єкта SqlDataAdapter, двічі відкривається і закривається об'єкт SqlConnection. Це може істотно уповільнити роботу застосування, тому перед викликом методу Fill об'єктів SqlDataAdapter краще викликати метод Open() об'єкта SqlConnection, а після виконання всіх операцій потрібно закрити з'єднання методом Close().

Що потрібно робити, якщо вам потрібно відновити данні об'єкта DataSet? Припустимо, після запуску застосування об'єкт SqlDataAdapter вибирає вміст таблиці, і ви хочете проглянути точні дані на даний момент часу, наприклад:

```
string strConn, strSQL;
strConn = @"Data Source=.\SQLExpress; Initial Catalog= BAZA;
Integrated Security=True;";
strSQL = "SELECT ClientID, ClientName FROM Clients ";
SqlDataAdapter da = new SqlDataAdapter(strSQL,strConn);
DataSet ds = new DataSet();
da.Fill(ds, " Clients ");
da.Fill(ds, " Clients ");
```

Який результат виконання програми? Якщо перед повторним викликом методу Fill об'єкта SqlDataAdapter визначити первинний ключ для таблиці Customers об'єкта DataSet, то об'єкт SqlDataAdapter виявить ідентичні записи і відкине старі значення.

Метод Fill використовується під час створення веб-застосувань, в яких необхідно посторінково розбивати дані. Припустимо, що ви хочете надати користувачам каталог у вигляді сторінок, що містять по десять найменувань товарів. Вкажіть номер початкового запису і кількість записів, яку повинен одержати метод Fill об'єкта SqlDataAdapter таким чином:

```
string strConn, strSQL;
strConn = @"Data Source=.\SQLExpress;" +
"Initial Catalog=BAZA; Integrated Security=True;";
strSQL = "SELECT TovarID, TovarName FROM Products";
SqlDataAdapter da = new SqlDataAdapter(strSQL, strConn);
DataSet ds = new DataSet();
int intStartRow = 10;
int intRowsShag * 10;
int intRows = da.Fill(ds,  intStartRow, intRowsShag, " Products ");
Console.WriteLine("{0}  row(s)  ",  intRows);
foreach (DataRow row in ds.Tables[ " Products "].Rows)
Console.WriteLine( "{0} - {1}",  row["TovarID"],
row["TovarName"]);
```

Об'єкт SqlDataAdapter буде передавати запит, якщо ви повідомите, що вибірку даних слід почати з десятого запису (як в даному прикладі), при цьому SqlDataAdapter просто відкине перші десять записів і вибере вказану кількість записів. Якщо запит не поверне потрібну вам кількість записів, об'єкт SqlDataAdapter вибере записи, що залишилися, не генеруючи при цьому повідомлення про виняткову ситуацію. Чи є спосіб посторінкового розбиття результатів запиту найбільш оптимальним?

Складніший і ефективніший спосіб досягти тієї ж функціональності – зберігати значення ключа останнього запису з попередньої сторінки. Припустимо, що вибираються перші 10 записів за допомогою такого запиту:

```
SELECT TOP 10 TovarID,  TovarName,  ClientName,  PhoneClient
ORDER BY CustomerID
```

SELECT TOP 10 TovarID, TovarName, ClientName, PhoneClient FROM Products ORDER BY TovarID WHERE TovarID > 18

1.3.6. Робота з реляційними даними

Дуже рідко можна зустріти базу даних, яка складається з сукупності незалежних таблиць. У процесі створення застосування розробник обов'язково зіткнеться з необхідністю обробки даних, розташованих у зв'язаних таблицях. Одержувати дані з декількох таблиць можна за допомогою одного запиту. Нижче приведено як запит вибирає дані з таблиць Customers, Orders і Order Details бази даних Northwind [5]:

```
SELECT C.CustomerID, C.CompanyName, C.ContactName, C
Phone, O.OrderID, O.EmployeeID, O.OrderDate, D.ProductID,
D.Quantity, D.UnitPrice
FROM Customers C, Orders O, [Order Details] D
WHERE C.CustomerID = O.CustomerID AND O.OrderID = D.OrderID
```

Основні переваги таких запитів полягають у наступному:

запит повертає дані у вигляді єдиної структури;

запит легко фільтрує результати.

Проте запити мають і недоліки:

повертають надмірні дані;

ускладнено оновлення результатів запитів;

ускладнена синхронізація результатів запитів.

Для організації відносин між об'єктами DataTable, з метою обробки реляційних даних, модель ADO.NET використовує об'єкти DataRelation, які володіють наступними перевагами:

об'єкти DataRelation повертають менше даних, ніж запити;

об'єкти DataRelation спрощують пошук зв'язаних даних;

об'єкти DataRelation не вимагають складного синхронізуючого коду;

об'єкти DataRelation розраховані на складні випадки оновлення;

об'єкти DataRelation є динамічними, їх можна програмно створювати, змінювати і видаляти як до, так і після запиту до зв'язаних таблиць бази даних;

об'єкти DataRelation підтримують каскадні оновлення;

об'єкти DataRelation використовуються для різних джерел даних.

Недолік у об'єктів DataRelation тільки один – ускладнена фільтрація даних. Властивості об'єкта DataRelation приведені в табл. 1.24.

Таблиця 1.24

Властивості об'єкта DataRelation

Властивості	Опис
ChildColumns	Указує дочірні стовпці, що визначають відношення; доступно тільки для читання
ChildKeyConstraint	Указує обмеження ForeignKeyConstraint в дочірній таблиці; доступно тільки для читання
ChildTable	Указує дочірню таблицю у відношенні; доступно тільки для читання
DataSet	Указує об'єкт DataSet, в якому знаходиться об'єкт DataRetation; доступно тільки для читання
ExtendedProperties	Указує колекцію динамічних властивостей
Nested	Указує, чи потрібно перетворювати дочірні записи в дочірні об'єкти при записі вмісту об'єкта DataSet в XML-файлі
ParentColumns	Указує батьківські стовпці, що визначають відношення, доступно тільки для читання
ParentKeyConstraint	Указує обмеження UniqueConstraint, що бере участь відносно батьківської таблиці; доступно тільки для читання
ParentTable	Указує батьківську таблицю у відношенні, доступно тільки для читання
RelationName	Указує ім'я відношення

За допомогою об'єктів DataRelation є можливість використовувати власний код для переміщення по декількох таблицях, для перевірки і агрегації даних.

У об'єкта DataRelation є декілька важливих властивостей, значення яких можна задавати за допомогою конструкторів. Створюючи об'єкт DataRelation, слід вказати його ім'я, щоб об'єкт можна було знайти в колекції; крім того, необхідно вказати батьківський і дочірній стовпці, на яких буде засноване відношення, наприклад:

```
DataSet ds = new DataSet();
DataTable tablica;
tablica = ds.Tables.Add( "Products");
tablica.Columns.Add("TovarID", typeof(string));
tablica.Columns.Add("TovarName", typeof(string));
```

```
tablica = ds.Tables.Add("Prodage");
tablica.Columns.Add("ProdagelD",  typeof(int));
tablica.Columns.Add("TovarlD",   typeof(string));
tablica.Columns.Add("ProdageDate", typeof(DateTime));
//  Додаємо об'єкт DataRelation,  що зв'язує дві таблиці
DataRelation relat;
relat = new DataRelation("Products_Orders",
ds.Tables["Products" ].Columns["TovarlD"],
ds. Tables[ " Prodage "]. Columns[ "TovarlD"]);
ds.Relations.Add(relat);
```

Як і у випадку з новими об'єктами DataTable і DataColumn, можна створити об'єкт DataRelation і додати його в колекцію Relations об'єкта DataSet за допомогою одного виклику:

```
//Додаємо об'єкт DataRelation, що зв'язує дві таблиці
ds.Relations.Add( " Products_Prodage",
ds.Tables[ " Products"].Columns[ "TovarlD"],
ds.Tables[ " Prodage "].Columns[ "TovarlD"]);
```

Приведемо приклад визначення відношення, заснованого на декількох стовпцях (конструктор DataRelation, що приймає масиви об'єктів DataColumn):

```
// Створюємо новий об'єкт DataSet і додаємо об'єкти DataTable і
// DataColumn
DataSet ds = new DataSet();
DataTable tblParent, tblChild;
tblParent = ds.Tables("ParentTable");
tblParent.Columns.Add( "ParentColumn1", typeof(string));
tblParent.Columns.Add( "ParentColumn2", typeof(string));
tblChild = ds.Tables( "ChildTable");
tblChild.Columns.Add( "ChildColumn1", typeof(string));
tblChild.Columns.Add( "ChildColumn2", typeof(string));
// Створюємо масиви об'єктів DataColumn, на яких
// буде заснований новий об'єкт DataRelation
DataColumn[ ] colsParent, colsChild;
colsParent = new DataColumn[ ]
            {   tblParent.Columns[ "ParentColumn1"],
                tblParent.Columns[ "ParentColumn2"]     };
colsChild = new DataColumn[ ]
            {   tblChild.Columns["ChildColumn1"],
```

```
                tblChild.Columns["ChildColumn2"]        };
    // Створюємо новий об’єкт
    DataRelation relat;
relat = new DataRelation("Columns_Relation", colsParent, colsChild);
    ds.Relations.Add(relat);
```

1.3.7. Відображення реляційних даних.

У процесі створення застосування розробник обов'язково зіткнеться з ситуацією, коли буде потрібно вивести або програмно звернутися до даних зі зв'язаних таблиць бази даних. Користувачу необхідно забезпечити зручне переміщення між різними таблицями з інформацією. Крім того, їм необхідно редагувати дані. Застосуванням найчастіше доводиться збирати агрегатні відомості, а при зміні одного запису виконувати зміни в інших, пов'язаних з ним записах (наприклад, коли вилучено замовлення, виникає потреба у видаленні всіх пов'язаних з ним товарів).

Для пошуку дочірніх записів, що пов'язані з батьківським записом в іншому об'єкті DataTable, достатньо викликати метод GetChildRows об'єкта DataRow і передати йому ім'я об'єкта DataRelation, що визначає відношення між об'єктами DataTable. Замість імені можна також передати сам об'єкт DataRelation. Метод GetChildRows повертає зв'язані дані у вигляді масиву об'єктів DataRow. Нижче наведено стандартний приклад виведення вмісту записів з батьківської таблиці (таблиця Tovar) й, пов'язаних з ними записів з дочірньої таблиці (таблиця Prodage).

```
//Додаємо об'єкт DataRelation, що  зв'язує дві таблиці;
     rel = ds.Relations.Add("Tovar_Prodage".
     ds.Tables["Tovar "].Columns["Tovar lD"],
     ds.Tables["Prodage"].Columns["ProdagelD"]);
//Заповнюємо об'єкт DataSet
     string strConn, strSQL ;
     str.Conn = @"DataSource=.\SQLExpress;"+
     "Initial Catalog=BAZA ;Integrated Security=True;";
     strSQL = "SELECT Tovar lD,  TovarName FROM Tovar " + "WHERE
     TovarlD = 12;" +  "SELECT ProdagelD, Cilkist FROM Prodage ";
     SqlDataAdapter da = new SqlDataAdapter(strSQL,strConn);
     da.TableMappings.Add("Table",   "Tovar");
     da.TableMappings.Add( "Table1",   " Prodage ");
```

В останніх двох рядках встановлюємо відповідність між таблицями "Table" бази даних і таблицями Tovar, Prodage об'єкта DataSet. Чому таблиця Tovar, що ми витягаємо, тут називається "Table"? Справа в тому, що в об'єкта DataAdapter немає можливості визначити, як називається таблиця в базі, і він підставляє можливу назву "Table".

```
da.Fill(ds);
// Переглядаємо записи про товари
    foreach (DataRow rowTovar in
    ds.Tables[ " Tovar "].Rows)
    { Console.WriteLine("{0}  -  {1}",   rowTovar ["TovarlD"],
    rowTovar [ "TovarName"]);
// Переглядаємо продажі товарів, що зроблені
foreach (DataRow rowProdage in rowTovar.GetChildRows(rel))
    Console.WriteLine(" {0} {1 }", rowProdage ["ProdagelD"],
    row Prodage ["Cilkist"]);
    Console.WriteLine();
```

Для пошуку батьківськіх записів, що пов'язані з дочірнім записом в іншому об'єкті DataTable, достатньо викликати метод GetParentRows об'єкта DataRow і передати йому ім'я об'єкта DataRelation, що визначає відношення між об'єктами DataTable. Замість імені можна також передати сам об'єкт DataRelation. Метод GetParentRows повертає зв'язані дані у вигляді масиву об'єктів DataRow. Нижче наведено приклад виведення вмісту записів з дочірньої таблиці (таблиця Prodage) й, пов'язаних з ними записів батьківської таблиці (таблиця Tovar):

```
    DataRow rowTovar;
// Переглядаємо продажі товарів
    foreach (DataRow rowProdage in ds.Tables[ "Prodage"].Rows)
  {
// Переглядаємо батьківські записи про товари
    rowTovar = rowOrder.GetParentRow( "Tovar_Prodage");
    Console.WriteLine( "{0}  {1 } ", rowProdage [ "ProdageID"],
    rowTovar [ "TovarName"]);
}
```

1.3.8. Відображення даних за допомогою об'єкта DataView

Об'єкти DataView дають можливість фільтрувати і сортувати вміст об'єктів DataTable, а також вести пошук, при цьому вони не є SQL-запи-

тами. За допомогою об'єкта DataView не можна об'єднати данні двох об'єктів DataTable, і проглянути окремі стовпці об'єкта DataTable.

Для того, щоб за допомогою об'єкта DataView проглянути дані об'єкта DataTable, його слід пов'язати з цим об'єктом. Вказати об'єкт DataTable, який використовується об'єктом DataView, можна двома способами:

за допомогою властивості Table об'єкта DataView (створюємо об'єкт DataView і пов'язуємо його з об'єктом DataTable):

```
DataTable tablica = new DataTable("TableName");
DataView viu = new DataView();
viu.Table = tablica;
```

за допомогою конструктора об'єкта DataView:

```
viu = new DataView(tablica);
```

Усі властивості об'єкта DataView приведені в табл. 1.25.

У процесі оновлення даних найчастіше використовується властивість RowStateFilter. Ця властивість приймає значення з переліку DataViewRowState (табл. 1.26).

Таблиця 1.25

Властивості об'єкта DataView

Властивість	Опис
AttowDelete	Указує чи можна видаляти записи об'єкта Data-View
AllowEdit	Указує, чи можна змінювати записи об'єкта Data-View
AllowNew	Указує, чи можна додавати записи в об'єкт DataView
ApplyDefaultSort	Указує, чи використовується порядок сортування за замовчуванням (значення за замовчуванням – False)
Count	Повертає кількість записів в об'єкті DataView
DataViewManager	Повертає посилання на контейнер DataViewManager об'єкта DataView
IsInitialized	Указує, чи ініціалізований об'єкт Data-View
Item	Повертає об'єкт DataRowView, який містить запис даних, доступний через об'єкт Data-View
RowFilter	Містить фільтр, вказуючий, які записи об'єкта DataTable доступні через об'єкт DataView
RowStateFilter	Указує, які записи доступні через об'єкт DataView
Sort	Указує порядок сортування записів
Table	Повертає об'єкт DataTable, з яким пов'язаний об'єкт Data View

Таблиця 1.26

Елементи переліку DataViewRowState

Елементи	Опис
Added	Відображаються додані записи
CurrentRows	Відображаються записи, які не були видалені
Deleted	Відображаються видалені записи
ModifiedCurrent	Відображаються змінені записи з їх поточними значеннями
ModifiedOriginal	Відображаються змінені записи з їх початковими значеннями
None	Записи не відображаються
OriginalRows	Відображаються видалені, змінені та незмінені записи з їх початковими значеннями
Unchanged	Відображаються незмінені записи

Приведемо декілька прикладів фільтрації даних за допомогою об'єкта DataView:

```
// Створюємо екземпляри dataAdapter і ds об'єктів DataAdapter і DataSet
// відповідно
    SqlDataAdapter dataAdapter = new
    SqlDataAdapter(CommandText, ConnectionString);
    DataSet ds = new DataSet();
// Створюємо екземпляр datatab об'єкта DataTable
    DataTable datatab = ds.Tables.Add("Client");
    dataAdapter.Fill(ds, " Client ");
    dataGrid1.DataSource = ds.Tables["Client "].DefaultView;
// Створюємо екземпляр myDataView об'єкта DataView,
 // передаємо йому об'єкт datatab
    DataView myDataView = new DataView(datatab);
// Встановлюємо фільтр для видалених записів
    myDataView.RowStateFilter = DataViewRowState.Deleted;
// Для другого елементу DataGrid указуємо myDataView як джерело даних
    dataGrid2.DataSource = myDataView;
// Для виведення нових рядків (DataViewRowState.Added) або змінених
 // (DataViewRowState.ModifiedCurrent) фільтр виглядатиме так:
    myDataView.RowStateFilter =
    ((DataViewRowState)((DataViewRowState.Added |
```

DataViewRowState.ModifiedCurrent)));

// Для виведення фільтрів по властивості RowFilter використовуємо
// наступні фрагменти коду:
// Виводимо всі записи із значенням 'Полтава' поля City:

myDataView.RowFilter = "Місто = 'Полтава'";

// Виводимо всі записи із значенням 'Антоненко' поля Name:

myDataView.RowFilter = "Прізвище = 'Антоненко'";

// Сортуємо записи по полю "City" у порядку зростання:

myDataView.Sort = " Місто ASC";

// Сортуємо записи по полю Name у порядку убування:

myDataView.Sort = " Прізвище DESC";

Розглянемо приклад виведення зв'язаних таблиць в один елемент DataGrid на основі бази даних, схема якої приведена на рис. 1.4.

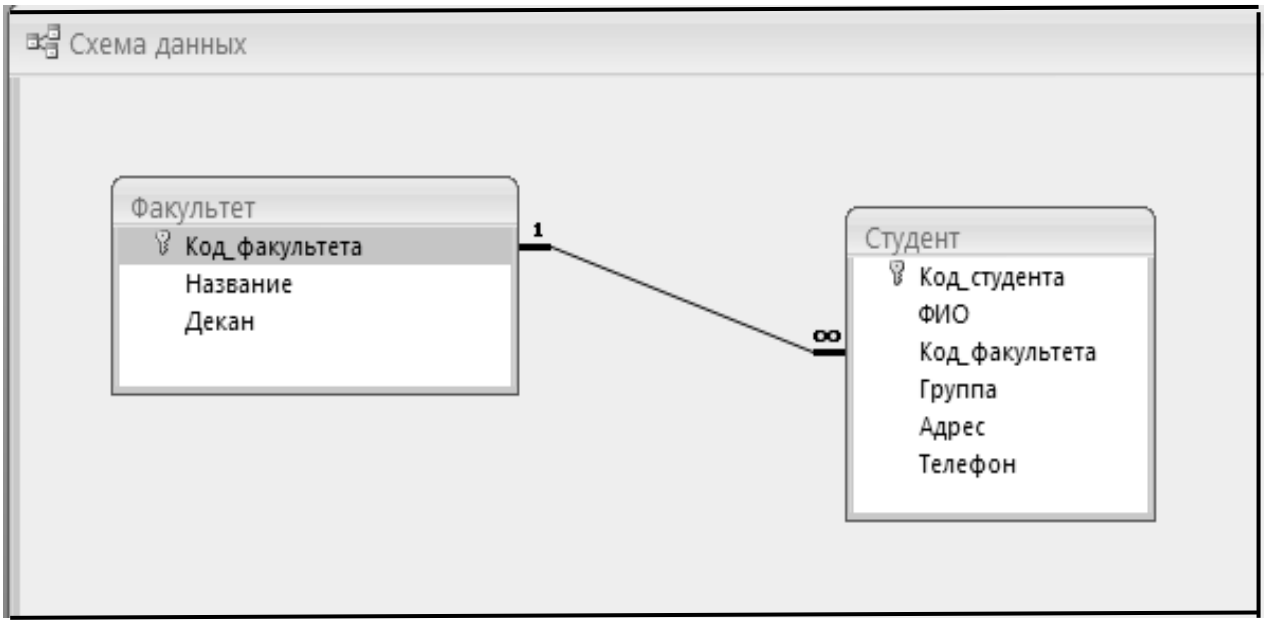

Рис. 1.4. **Схема бази даних**

У конструкторі форми створюємо з'єднання, об'єкт OleDbCommand і визначаємо рядок CommandText:

OleDbConnection conn = new OleDbConnection(connectionString);

OleDbCommand myCommand = new OleDbCommand();

myCommand.Connection = conn;

myCommand.CommandText = commandText;

Підключаємося до файла бази даних Faculty.mdb, указуємо відповідні параметри рядків connectionString і commandText:

string commandText = "SELECT * FROM Факультет ";

string connectionString = "Provider=Microsoft.Jet.OLEDB.4.0;

Data Source= \\Faculity.mdb;";

Створюємо об'єкт DataAdapter, у властивості SelectCommand встановлюємо значення myCommand і відкриваємо з'єднання:

OleDbDataAdapter dataAdapter = new OleDbDataAdapter();

dataAdapter.SelectCommand = myCommand;

conn.Open();

Створюємо об'єкт DataSet:

DataSet ds = new DataSet();

У об'єкті DataSet зберігатимуться дві таблиці – головна і дочірня, яка пов'язана з нею. Тому скористаємося властивістю TableMappings об'єкта DataAdapter для занесення в нього першої таблиці " Факультет ":

DataSet ds = new DataSet();

dataAdapter.TableMappings.Add("Table", "Факультет");

dataAdapter.Fill(ds);

Властивості DataSource об'єкта dataGrid1 указуємо таблицю "Факультет" об'єкта ds. Зверніть увагу на синтаксис – властивість Tables має на увазі наявність декількох таблиць в об'єкті DataSet:

dataGrid1.DataSource = ds.Tables[" Факультет "].DefaultView;

Закриваємо з'єднання:

conn.Close();

Тепер необхідно додати об'єкти OleDbDataAdapter і OleDbCommand для таблиці "Студент":

OleDbCommand myCommand2 = new OleDbCommand();

myCommand2.Connection = conn;

myCommand2.CommandText = commandText2;

OleDbDataAdapter dataAdapter2 = new OleDbDataAdapter();

Слід звернути увагу на те, що dataAdapter2 використовує те ж саме підключення conn, що і dataAdapter. Рядок commandText2 визначимо таким чином:

string commandText2 = "SELECT * FROM Студент";

Тепер пов'яжемо другий об'єкт OleDbDataAdapter з тільки що створеною командою і відобразимо "Інформацію про клієнтів" на його таблицю. Потім можна заповнити об'єкт DataSet даними з другої таблиці:

dataAdapter2.SelectCommand = myCommand2;

dataAdapter2.TableMappings.Add("Table", "Информация о студентах");

dataAdapter2.Fill(ds);

У результаті вийшов об'єкт DataSet з двома таблицями. Тепер можна виводити одну з цих таблиць на форму. Але зв'язок між таблицями ще не створений. Для конфігурації відношення по полю "Код_факультета" створюємо два об'єкти DataColumn:

```
DataColumn facultID =
ds.Tables["Факультет"].Columns["Код_факультета"];
DataColumn dcInfofacultID = ds.Tables[ "Информация о
студентах"].Columns[ " Код_ факультета "];
```

Створюємо об'єкт DataRelation, в його конструкторі передаємо назву відношення між таблицями і два об'єкти DataColumn:

```
DataRelation dataRelation = new DataRelation("Дополнительная
информация", facultID, dcInfofacultID);
```

Додаємо створений об'єкт відношення до об'єкта DataSet:

```
ds.Relations.Add(dataRelation);
```

Створюємо об'єкт DataViewManager, що відповідає за відображення DataSet в об'єкті DataGrid:

```
DataViewManager dsview = ds.DefaultViewManager;
```

Привласнюємо властивості DataSource об'єкта DataGrid створений об'єкт DataViewManager:

```
dataGrid1.DataSource = dsview;
```

Останнє, що залишилося зробити, – повідомити об'єкт DataGrid, яку таблицю вважати головною (батьківською) і, відповідно, відображати на формі:

```
dataGrid1.DataMember = "Факультет";
```

Закриваємо з'єднання:

```
conn.Close();
```

На рис. 1.5–1.7 приведені результати роботи програми виведення зв'язаних таблиць в один елемент DataGrid. На рис. 1.5 відображається вміст батьківської таблиці "Факультет". Для отримання дочірніх рядків з таблиці "Студент" будь-якому із записів таблиці "Факультет" необхідно зробити клацання на значку "+", який знаходиться в лівому стовпці таблиці, що відображається.

Після цього слід зробити клацання на повідомленні "Студенты данного факультета" (рис. 1.6). У результаті цих дій буде одержане вікно (рис.1.7), в якому у верхній частині відображається батьківський рядок, а в нижній частині – дочірні рядки.

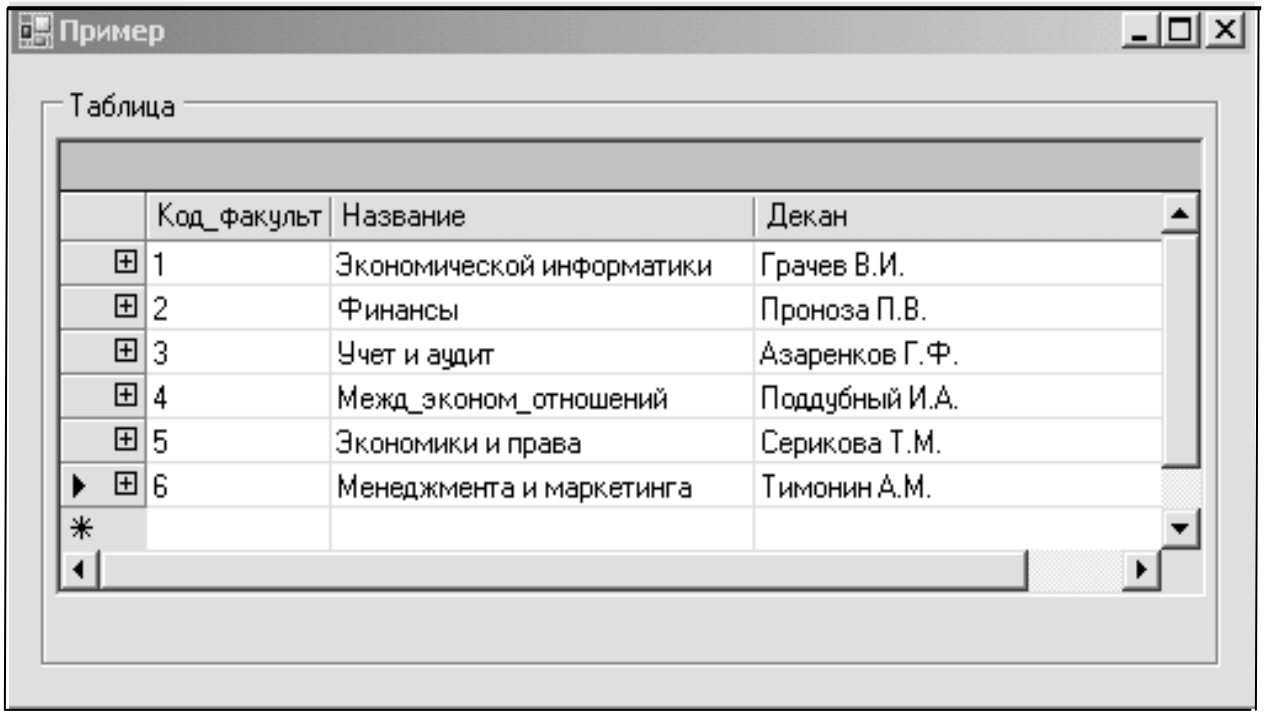

Рис. 1.5. **Відображення таблиці "Факультет"**

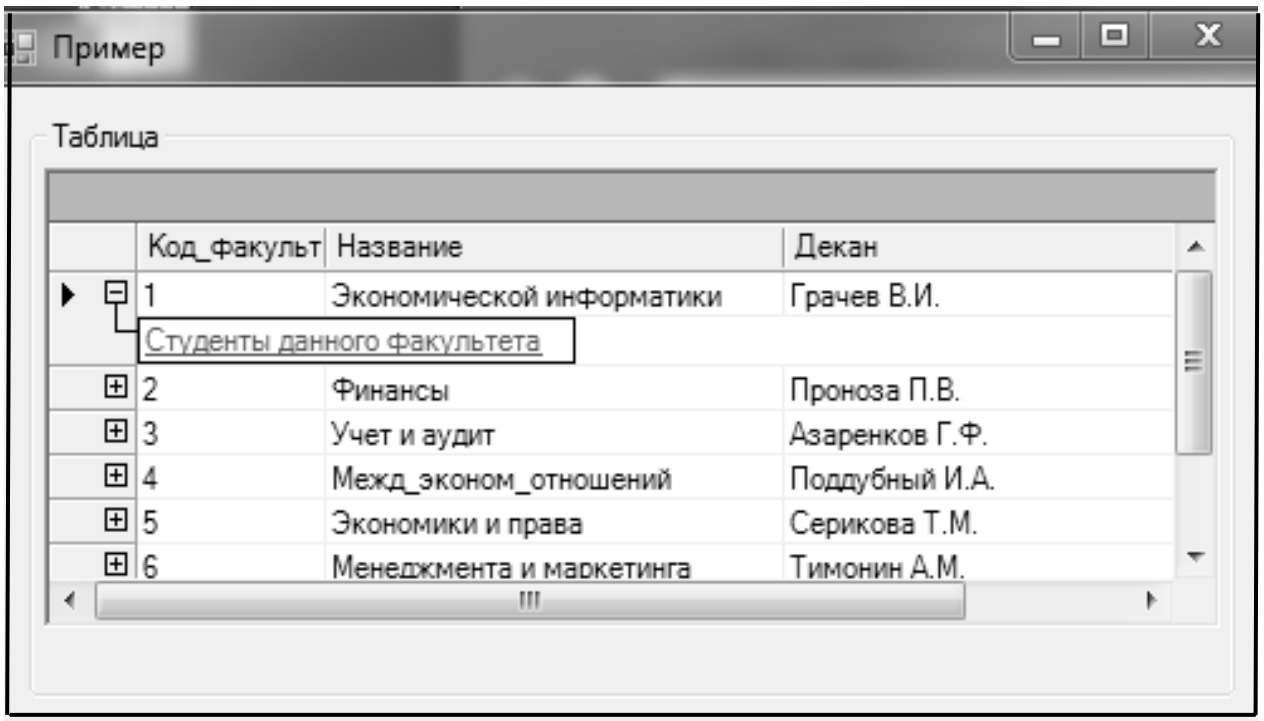

Рис. 1.6. **Відображення дочірніх рядків таблиці "Студент"**

У цьому вікні, за допомогою піктограми , можна відображати або приховувати батьківські рядки, а піктограма дозволяє виконати повернення до батьківських рядків.

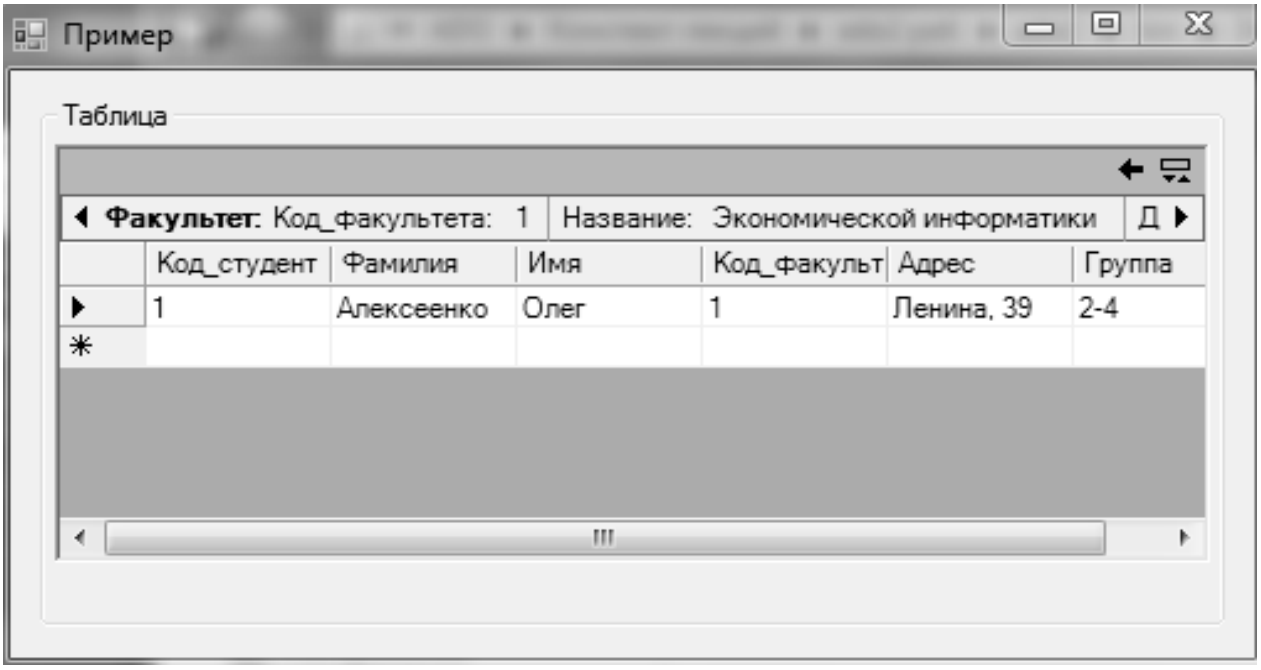

Рис. 1.7. **Відображення дочірніх рядків таблиці "Студент"**

Висновки. У лекції розглянуто виконання операцій обробки даних з використанням класів DataSet, DataTable, DataColumn, DataRow і DataAdapter. Докладно розглянуті особливості використання методів базового класу DataAdapter, які надають широкі можливості користувачу для роботи з базами даних. Особлива увага приділяється роботі з реляційними даними, а також їх відображенню за допомогою об'єкта DataView.

Тести для самодіагностики

Тест № 1. Ви розробляєте застосування для внесення змін в базу даних в автономному режимі. Які чотири дії ви повинні виконати? (кожна правильна відповідь представляє частину рішення):

створити об'єкт OleDbDataAdapter і визначити вміст SelectCommand;

створити об'єкт OleDbCommand і використати метод ExecuteScalar;

створити об'єкт DataTable як контейнер для даних;

створити об'єкт DataSet як контейнер для даних;

викликати DataAdapter.Fill метод, щоб заповнити об'єкт набору даних;

викликати DataAdapter.Update метод, щоб заповнити об'єкт DataSet;

викликати DataAdapter.Update метод, щоб зберегти зміни в базі;

викликати DataSet.Accept метод, щоб зберегти зміни у базі даних?

Тест № 2. Ви розробляєте Windows-застосування, яке використовує об'єкт DataSet з ім'ям customDataSet. Після редагування даних перевірка коректності даних повинна виконатися компонентом з ім'ям myComponent. Вам треба переконатися, що застосування пересилає тільки відредаговані рядки даних з customDataSet в myComponent. Який фрагмент коду треба використовувати:

1) DataSet changeDataSet = new DataSet();
if (customDataSet.HasChanges) { myComponent.Validate(changeDataSet); }
2) DataSet changeDataSet = new DataSet();
if (customDataSet.HasChanges) { myComponent.Validate(customDataSet); }
3) DataSet changeDataSet = customDataSet.GetChanges();
myComponent.Validate(changeDataSet);
4) DataSet changeDataSet = customDataSet.GetChanges();
myComponent.Validate(customDataSet);?

Контрольні завдання.

Завдання № 1. У процесі роботи застосування користувач редагує дані таблиць. Для ведення протоколу роботи користувача потрібно відобразити всі зміни даних у таблицях.

Завдання № 2. У процесі налагодження застосування потрібно забезпечити перегляд, як поточних значень змінених рядків таблиці, так і їхніх початкових значень.

Література: [2, с. 29–43; 3, с. 158–195].

1.4. Лекція № 4. Модифікація ієрархічних даних

1.4.1. Передача змін у базу даних і збереження цілісності даних

ADO.NET відрізняється від попередньої архітектури доступу до даних тим, що дозволяє витягувати абсолютно автономні записи за допомогою класу DataSet. DataSet схожий на базу даних, але це не зовсім база даних. Він дозволяє здійснювати пошук, сортування і фільтрацію записів, що містяться у ньому, працює з різними відносинами і для полегшення оновлення його вмісту використовує можливості як об'єктного уявлення, так і XML-уявлення.

У підключеному середовищі можна легко оновлювати, вставляти або видаляти різні споріднені дані; проте при роботі з автономними

даними рутинні завдання з витягання останніх ключів, що згенерували, і управлінню питаннями паралельної обробки інформації в ієрархічно реляційних даних можуть виявитися достатньо складними. Тому спочатку розглянемо простіші питання оновлення інформації в одній таблиці бази даних. Наприклад, перед розробником стоїть завдання створити код для внесення змін у базу даних "Факультет". Це завдання найпростіше можна вирішувати використовуючи підключене середовище. Наприклад, додавання запису в таблицю "Студент" виконується за допомогою вікна, яке приведене на рис. 1.8.

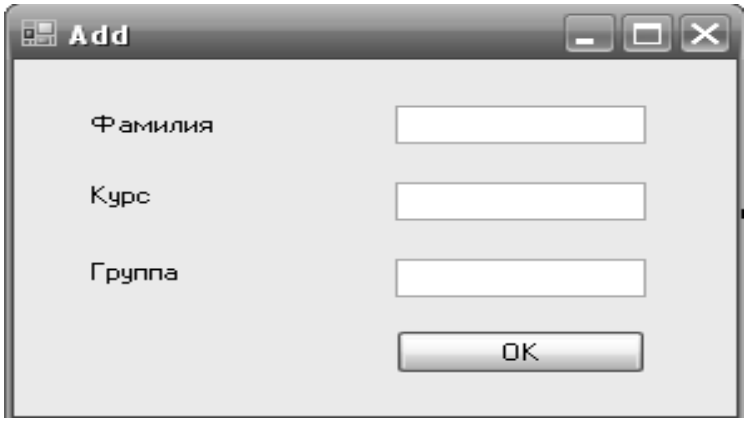

Рис. 1.8. **Вікно додавання рядка в таблицю**

Код, що реалізовує операцію додавання рядка в таблицю, приведений нижче:

```
        string Name = NameTxt.Text;
        int Course = Convert.ToInt32(CourseTxt.Text);
        int Groupe = Convert.ToInt32(GroupeTxt.Text);
// Створення обєкту команди додавання запису в таблицю
        SqlCommand myCommand = connect.CreateCommand();
        connect.Open();
// Створення параметрів команди
        myCommand.Parameters.Add("@Name", SqlDbType.NChar, 50);
        myCommand.Parameters["@Name"].Value = Name;
        myCommand.Parameters.Add("@Course", SqlDbType.Int, 4);
        myCommand.Parameters["@Course"].Value = Course;
        myCommand.Parameters.Add("@Groupe", SqlDbType.Int, 4);
        myCommand.Parameters["@Groupe"].Value = Groupe;
// Завдання тексту команди додавання запису в таблицю Student
        myCommand.CommandText = "INSERT INTO Student
        (Name,Course,Groupe) VALUES(@Name,@Course,@Groupe)";
```

// Виконання команди додавання запису в таблицю Student

myCommand.ExecuteNonQuery();

connect.Close();

Якщо дані таблиці були відображені в елементі **dataGridView1,** то код, що реалізує операцію видалення рядка з таблиці може бути таким:

int id = Convert.ToInt32(dataGridView1.CurrentRow.Cells["id"].Value);

connect.Open();

// Створення обєкта команди видалення запису з таблиці та її параметрів

SqlCommand myCommand = connect.CreateCommand();

myCommand.Parameters.Add("@id", SqlDbType.Int, 5);

myCommand.Parameters["@id"].Value = id;

// Завдання тексту команди видалення запису з таблиці

myCommand.CommandText = "DELETE FROM Student

WHERE id = @id";

// Виконання команди видалення запису з таблиці

myCommand.ExecuteNonQuery();

connect.Close();

Як видно з приведених прикладів опис параметрів команд може бути досить рутинним процесом. Чи можна взагалі уникнути опису параметрів для передачі змін, якщо вже дані прочитані? Відповідь – можна, для цього ми повинні застосовувати роз'єднаний режим і об'єкт CommandBuilder, який надасть можливість зробити це, якщо виконуються такі умови:

запит повертає дані тільки з однієї таблиці;

таблиця містить первинний ключ;

запит, що повертається, містить первинний ключ.

У коді об'єкт CommandBuilder зв'язується з вже наявним екземпляром dataAdapter так:

SqlCommandBuilder commandBuilder = new

SqlCommandBuilder(dataAdapter);

Створимо фрагмент застосування, який виконує оновлення даних за допомогою об'єкта CommandBuilder. Для цього у класі форми створюємо рядок підключення, а також об'єкти DataSet і dataAdapter:

string connectionString = " рядок підключення";

DataSet ds;

SqlDataAdapter dataAdapter;

У конструкторі форми ми створюємо найзвичайніший набір об'єктів ADO .NET і додаємо одним рядком об'єкт CommandBuilder:

```
        public Form1()
         {
        InitializeComponent();
         SqlConnection conn = new SqlConnection();
         conn.ConnectionString = connectionString;
        SqlCommand mySelectCommand = conn.CreateCommand();
mySelectCommand.CommandText = "SELECT * FROM MyAnimal ";
        dataAdapter = new SqlDataAdapter();
         dataAdapter.SelectCommand = mySelectCommand;
         ds = new DataSet();
         dataAdapter.Fill(ds);
         dataGrid1.DataSource = ds.Tables[0].DefaultView;
        SqlCommandBuilder commandBuilder = new
        SqlCommandBuilder(dataAdapter);
         }
```

У обробнику події Closing форми передаємо зміни в базу даних:

```
private void Form1_Closing(object sender,
                    System.ComponentModel.CancelEventArgs e)
        {
         try          {
        if (ds.HasChanges())
        {       dataAdapter.Update(ds);   }   }
        catch(Exception ex)
        {       MessageBox.Show(ex.ToString());          }   }
```

Об'єкт SqlCommandBuilder вибирає метадані, необхідні для генерації логіки оновлення, за допомогою властивості SelectCommand об'єкта SqlDataAdapter. Цей об'єкт звертається до бази даних за іменами базової таблиці та стовпців, а також за відомостями про ключові стовпці. Первинний ключ гарантує, що об'єкт SqlCommandBuilder відновить не більше одного запису. Метод ExecuteReader об'єкта SqlCommand дозволяє одержати ці метадані разом з результатами запиту:

```
private void button_ComBuild (object sender, EventArgs e)
{     string strConn2;
      strConn2 = @"Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\Animals.mdf
; Integrated Security=True;Connect Timeout=30;User Instance=True";
          SqlConnection conn = new SqlConnection();
```

```
        conn.ConnectionString = strConn2;
        SqlCommand mySelectCommand = conn.CreateCommand();
    mySelectCommand.CommandText = "SELECT * FROM MyAnimal";
        SqlDataAdapter dataAdapter = new SqlDataAdapter();
        dataAdapter.SelectCommand = mySelectCommand;
        DataSet ds = new DataSet();
        dataAdapter.Fill(ds);
        dataGridView1.DataSource = ds.Tables[0].DefaultView;
SqlCommandBuilder commandBuilder =
                        new SqlCommandBuilder(dataAdapter);
textBox1.Text +="" +
commandBuilder.GetInsertCommand().CommandText;
textBox1.Text += "" +
commandBuilder.GetDeleteCommand().CommandText;
textBox1.Text += "" +
commandBuilder.GetUpdateCommand().CommandText;      }
```

На рис. 1.9 приведені тексти команд додавання, видалення і оновлення даних, які сформовані об'єктом **commandBuilder.**

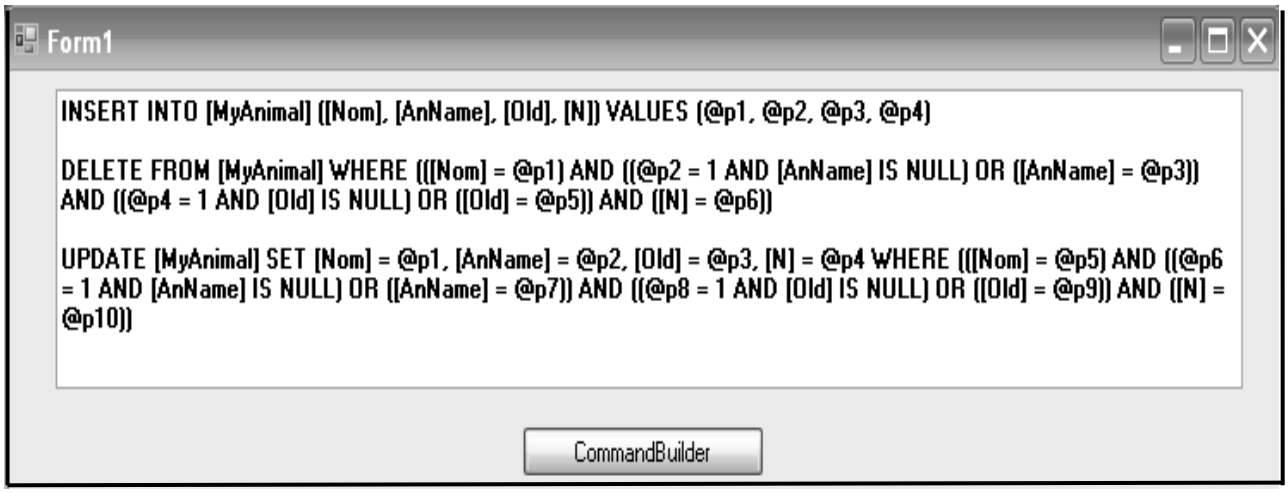

Рис. 1.9. **Команди, створені об'єктом commandBuilder**

Переваги використання об'єкта SqlCommandBuilder:
виходить менший код;

за допомогою об'єкта SqlCommandBuilder вдається створювати логіку оновлення, маючи навіть поверхневе уявлення про SQL-синтаксис запитів UPDATE, DELETE і INSERT;

об'єкт SqlCommandBuilder корисний, коли у вас виникли проблеми з генерацією власної логіки оновлення, якщо він успішно згенерує необхідну логіку, прогляньте значення властивості CommandText

створених їм об'єктів SqlCommand або значення властивостей об'єктів SqlParameter;

об'єкт SqlCommandBuilder вельми корисний у застосуваннях, в яких повинна бути підтримка оновлення даних і ви не хочете проглядати структуру запитів в період розробки.

Недоліки використання об'єкта SqlCommandBuilder:

оскільки об'єкт SqlCommandBuilder генерує логіку оновлення під час виконання запитів, то його продуктивність не максимальна;

об'єкт SqlCommandBuilder не дозволяє управляти створенням логіки оновлення;

не можна виконувати оновлення засобами процедур, що зберігаються.

1.4.2. Приклад модифікації ієрархічних даних

Оновлення ієрархічних даних – це одна з найскладніших частин будь-якого застосування. Запис інформації у джерело даних вимагає від користувача ухвалення безлічі рішень з метою забезпечення високої продуктивності застосування і цілісності інформації в базі даних. Розглянемо процес створення застосування, оновлюючого інформацію в базі даних "Продукция", схема якої приведена на рис. 1.10.

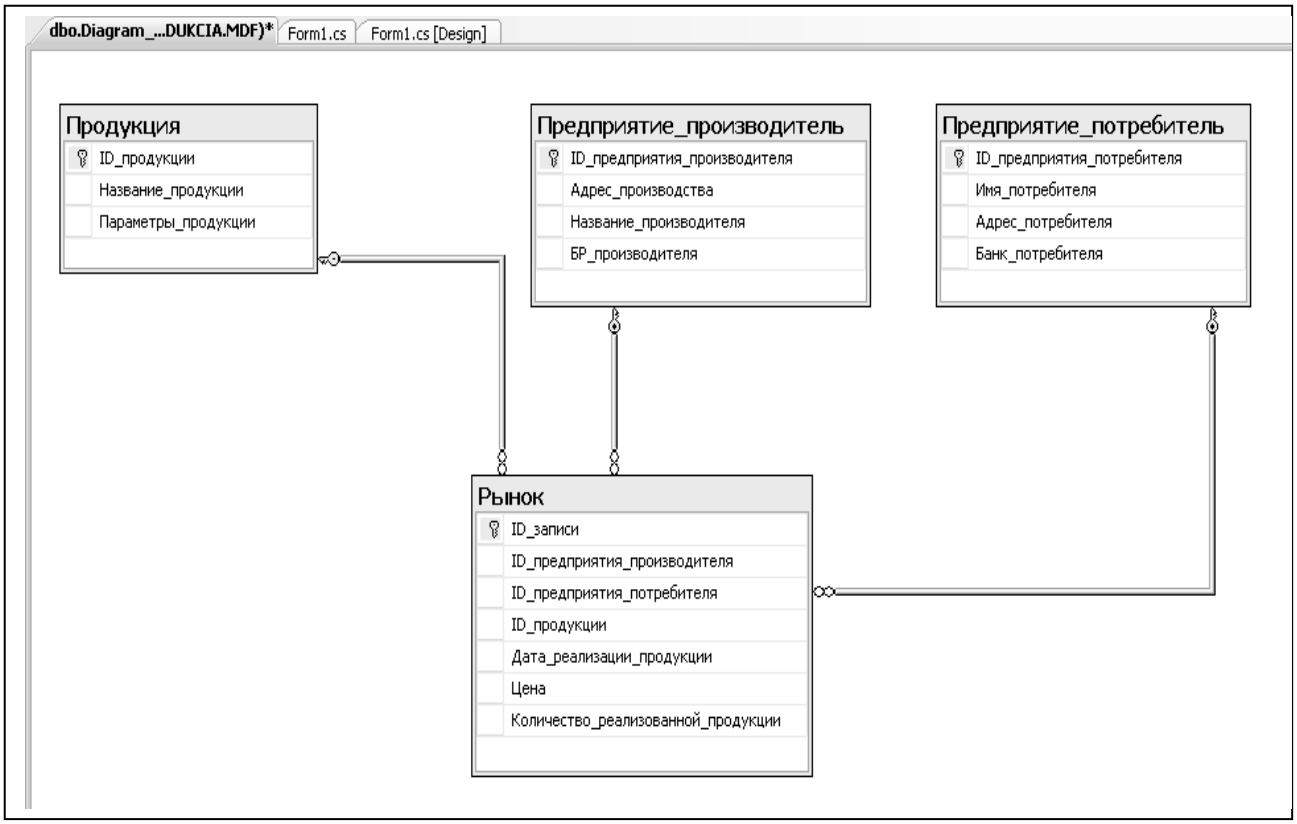

Рис. 1.10. **Схема бази даних "Продукция"**

Розробка застосування для оновлення даних може здійснюватись на основі алгоритму, який приведений на рис. 1.11.

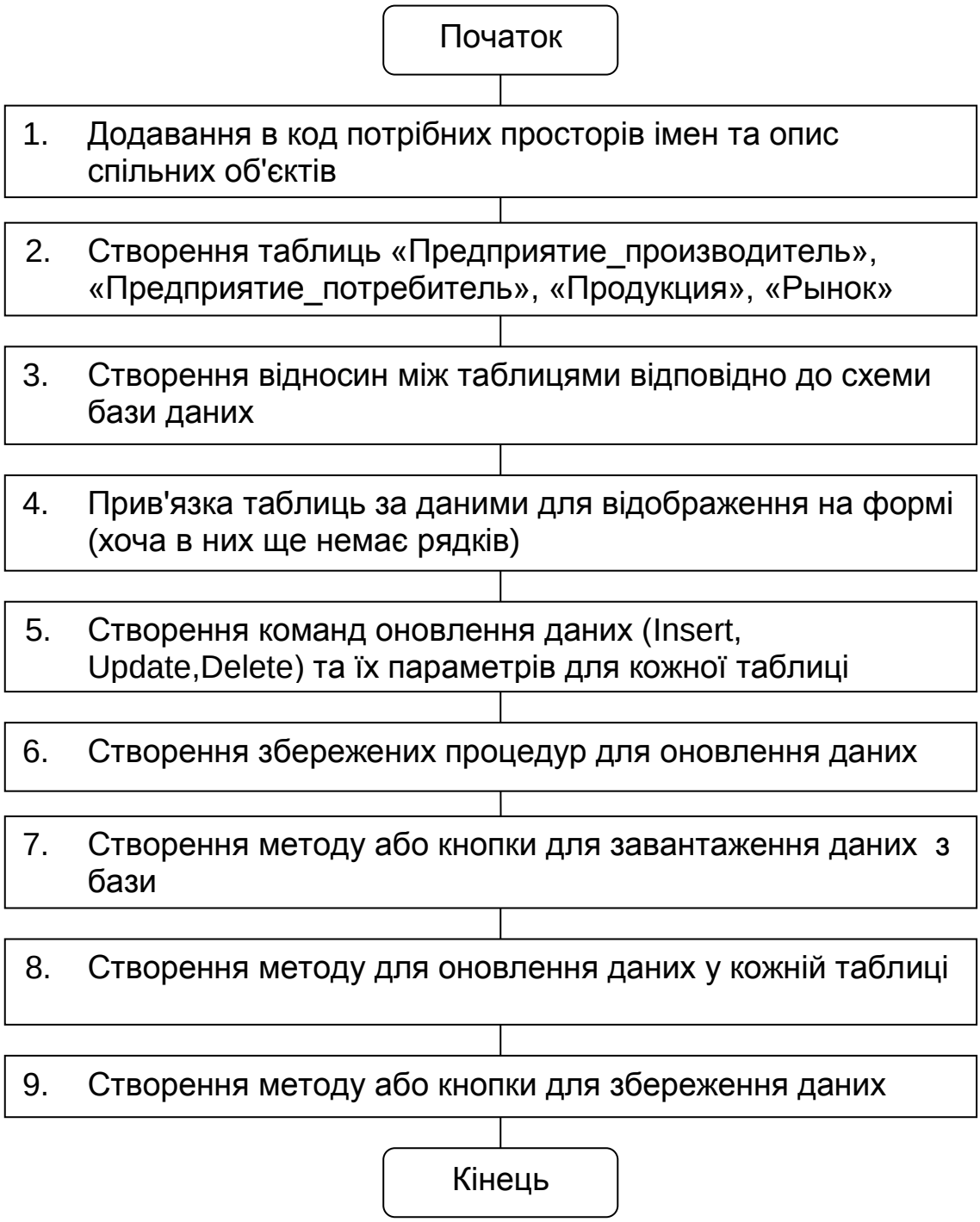

Рис. 1.11. **Структурна схема алгоритму створення додатку для оновлення даних**

Виконання всіх операцій з даними будемо здійснювати з використанням збережених процедур. Вибір такого способу взаимодії з даними пояснюється насамперед тим, що при виконанні команди Insert

застосуванню потрібно повернути значення первинного ключа, справжнє його значення відомо, в першу чергу, базі даних. Розглянемо докладніше вміст кожного з елементів алгоритму оновлення даних (рис. 1.11).

Блок № 1. Додавання в код потрібних просторів імен та опис спільних об'єктів.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Text;
using System.Windows.Forms;
using System.Data.Sql;
using System.Data.SqlClient;
namespace Obnovlenie
{   public partial class Form1 : Form
   {
string connstring = @"Data Source=
.\SQLEXPRESS;AttachDbFilename= |DataDirectory|\Produkcia.mdf;
Integrated Security=True;Connect Timeout=30;User Instance=True";
      string comtext;
      DataSet ds = new DataSet();
      SqlDataAdapter da;
      SqlDataAdapter da1;
      SqlDataAdapter da2;
      SqlDataAdapter da3;
      DataTable table_Predpr_proizv;
      DataTable table_Predpr_potr;
      DataTable table_Prodykcia;
      DataTable table_Market;
```

У цій частині коду описується рядок з'єднання і об'єкти DataSet, DataTable, SqlDataAdapter, причому останні два види об'єктів – для кожної таблиці бази даних.

Блок № 2. Створення таблиць "Предприятие_производитель", "Предприятие_потребитель", "Продукция", "Рынок".

```
// Створення таблиці "Предприятие_производитель"
        table_Predpr_proizv = new
                        DataTable("Предприятие_производитель");
```

```
        DataColumn dc = new
                        DataColumn("ID_предприятия_производителя");
        dc.Unique = true;
        dc.AutoIncrement = true;
        dc.AutoIncrementSeed = -1;
        dc.AutoIncrementStep = -1;
        dc.ReadOnly = true;
        dc.DataType = typeof(System.Int32);
        table_Predpr_proizv.Columns.Add(dc);
        dc = new DataColumn("Адрес_производства");
        table_Predpr_proizv.Columns.Add(dc);
        dc = new DataColumn("Название_производителя");
        table_Predpr_proizv.Columns.Add(dc);
        dc = new DataColumn("БР_производителя");
        table_Predpr_proizv.Columns.Add(dc);
        ds.Tables.Add(table_Predpr_proizv);
// Створення таблиці "Предприятие_потребитель"
// Створення таблиці "Продукция"
// Створення таблиці "Рынок"
```

При створенні таблиць описуються всі стовпці, що входять в кожну таблицю та їх типи. Якщо тип даних стовпця не задається, то встановлюється тип даних String. У таблиці "Предприятие_ производитель" міститься чотири стовпці, з яких один стовпець (тип даних Int32) є первинним ключем, а інші мають тип даних String.

Процес оновлення даних у зв'язаних таблицях істотно ускладнюється, якщо вони використовуються декількома застосуваннями при наявності ключового поля типу AutoIncrement в Sql Server або "Лічильник" в Access. У базі даних цей стовпчик вважається стовпчиком ідентифікації. Тому під час додавання нового рядка, його заповнює СУБД, а не автономне застосування. Автономний користувач не може точно визначити наступне значення ідентифікаційного номера, тому що інший користувач може використати значення, яке ви намагаєтеся занести в DataTable. Тому в застосуваннях треба генерувати фіктивне значення первинного ключа під час оновлення запису. Негативні значення гарантують відмінність ключа в застосуванні від його значення в таблиці бази даних:

dc.AutoIncrement = true;

dc.AutoIncrementSeed = -1;

dc.AutoIncrementStep = -1;

Після завершення оновлення необхідно формувати справжнє значення первинного ключа.

Блок № 3. Створення відносин між таблицями відповідно до схеми бази даних.

```
ds.Relations.Add(table_Predpr_proizv.Columns[0],
                                    table_Market.Columns[1]);
ds.Relations.Add(table_Predpr_potr.Columns[0],
                                    table_Market.Columns[2]);
ds.Relations.Add(table_Prodykcia.Columns[0],
                                    table_Market.Columns[3]);
```

У даному фрагменті програми встановлені три зв'язки:

зв'язок між таблицями "Продукция" і "Рынок" по стовпцю "ID_продукции", який в таблиці "Продукция" має індекс 0, а в таблиці "Рынок" індекс 3;

зв'язок між таблицями "Предприятие_производитель" і "Рынок" по стовпцю "ID_предприятия_производителя", який в таблиці "Предприятие_производитель" має індекс 0, а в таблиці "Рынок" індекс 2;

зв'язок між таблицями "Предприятие_потребитель" і "Рынок" по стовпцю "ID_предприятия_потребителя", який в таблиці "Предприятие_потребитель" має індекс 0, а в таблиці "Рынок" індекс 1.

Блок № 4. Прив'язка таблиць за даними для відображення на формі

Створимо на формі чотири компоненти DataGridView з іменами: dataGridView1, dataGridView2, dataGridView3, dataGridView4, і виведемо на форму вміст таблиць бази даних:

dataGridView1.DataSource = table_Predpr_proizv;

dataGridView2.DataSource = table_Predpr_potr;

dataGridView3.DataSource = table_Prodykcia;

dataGridView4.DataSource = table_Market;

Нові дані будуть додаватися в компонентах DataGridView і передавати їх для уведення в таблиці бази даних краще використовуючи параметри.

Блок № 5. Створення команд оновлення даних (Insert, Update, Delete) та їх параметрів для кожної таблиці будемо виконувати з

допомогою методу CreateComParam(). Нижче наведений фрагмент методу CreateComParam():

```
public void CreateComParam()
      {
//Створення команди додавання даних у таблицю"
//"Предприятие_производитель"
          SqlCommand insertCommand = new SqlCommand();
          insertCommand.Connection = con;
          insertCommand.CommandType =
CommandType.StoredProcedure;
        insertCommand.CommandText = "pr_proizv_insert";
//Створення параметрів команди додавання даних у таблицю
              SqlParameter param = null;
          param = new SqlParameter("@ID_pr_proizv", SqlDbType.Int);
          param.Direction = ParameterDirection.Input;
          param.SourceColumn = "ID_предприятия_производителя";
          insertCommand.Parameters.Add(param);
          param = new SqlParameter("@adr_proizv", SqlDbType.Text);
          param.Direction = ParameterDirection.Input;
          param.SourceColumn = "Адрес_производства";
          insertCommand.Parameters.Add(param);
          param = new SqlParameter("@nazv_proizv", SqlDbType.Text);
          param.Direction = ParameterDirection.Input;
          param.SourceColumn = "Название_производителя";
          insertCommand.Parameters.Add(param);
          param = new SqlParameter("@BR_proizv", SqlDbType.Text);
          param.Direction = ParameterDirection.Input;
          param.SourceColumn = "БР_производителя";
          insertCommand.Parameters.Add(param);
//Створюємо об'єкт SqlDataAdapter
              da = new SqlDataAdapter(comtext, con);
          insertCommand.Connection = new SqlConnection(connstring);
          insertCommand.UpdatedRowSource = UpdateRowSource.Both;
          da.InsertCommand = insertCommand;
```

У наведеному вище фрагменті програми треба звернути увагу на такий рядок:

```
insertCommand.UpdatedRowSource = UpdateRowSource.Both;
```

Цей рядок визначає процес оновлення джерела даних після виконання команди **Insert.** Те, як результати виклику методу **Update()** об'єкта **DataAdapter** будуть застосовуватися до об'єкта DataRow, визначає властивість **UpdatedRowSource**. Можливі значення властивості **UpdateRowSource**:

Both – вставленому або оновленому рядку в наборі даних співставляються дані як з першого повернутого рядка, так і з вихідного параметра;

FirstReturnedRecord – вставленому або оновленому рядку в наборі даних співставляються дані з першого повернутого рядка;

None – повертають значення і параметри ігноруються. Це значення встановлено за замовчуванням, коли дана команда генерується об'єктом **CommandBuilder**;

OutputParameters – вставленому або оновленому рядку в наборі даних співставляються дані з вихідного параметра.

У такий спосіб для об'єкта DataAdapter визначається команда вставлення з параметрами. Тепер у джерело даних можна додавати нові рядки, а значення стовпчику "ID_предприятия_производителя", що згенерований джерелом даних для цього рядка, можна було б одержувати через вихідний параметр або через перший запис, який повертає збережена процедура pr_proizv_insert. Слід зазначити, що параметр "ID_предприятия_производителя" у процедурі обов'язково повинен описуватися із службовим словом **output.**

Аналогично створюються команди оновлення і видалення даних у таблиці "Предприятие_производитель":

```
//Створення  команди оновлення даних у таблиці
//"Предприятие_производитель"
        SqlCommand updateCommand = new SqlCommand();
        updateCommand.Connection = con;
  updateCommand.CommandType = CommandType.StoredProcedure;
        updateCommand.CommandText = "pr_proizv_update";
  //Створення параметрів команди оновлення даних у таблиці
        param = null;
        param = new SqlParameter("@ID_pr_proizv", SqlDbType.Int);
        param.Direction = ParameterDirection.Input;
        param.SourceColumn = "ID_предприятия_производителя";
        updateCommand.Parameters.Add(param);
```

```
        param = new SqlParameter("@adr_proizv", SqlDbType.Text);
        param.Direction = ParameterDirection.Input;
        param.SourceColumn = "Адрес_производства";
        updateCommand.Parameters.Add(param);
        param = new SqlParameter("@nazv_proizv", SqlDbType.Text);
        param.Direction = ParameterDirection.Input;
        param.SourceColumn = "Название_производителя";
        updateCommand.Parameters.Add(param);
        param = new SqlParameter("@BR_proizv", SqlDbType.Text);
        param.Direction = ParameterDirection.Input;
        param.SourceColumn = "БР_производителя";
        updateCommand.Parameters.Add(param);
        updateCommand.Connection = new SqlConnection(connstring);
        updateCommand.UpdatedRowSource = UpdateRowSource.Both;
        da.UpdateCommand = updateCommand;
//Створення команди видалення даних із таблиці
// "Предприятие_производитель"
        SqlCommand udalCommand = new SqlCommand();
        udalCommand.Connection = con;
    udalCommand.CommandType = CommandType.StoredProcedure;
        udalCommand.CommandText = "pr_proizv_udal";
  //Створення параметрів команди видалення даних із таблиці
        param = null;
        param = new SqlParameter("@ID_pr_proizv", SqlDbType.Int);
        param.SourceColumn = "ID_предприятия_производителя";
        udalCommand.Parameters.Add(param);
        udalCommand.Connection = new SqlConnection(connstring);
        udalCommand.UpdatedRowSource = UpdateRowSource.Both;
        da.DeleteCommand = udalCommand;
```

Команди оновлення і видалення даних із таблиці використовують процедури, що зберігаються pr_proizv_update і pr_proizv_udal відповідно. Команди додавання, оновлення і видалення даних у таблицях "Предприятие_потребитель", "Продукция", "Рынок" створюються аналогічно.

Блок № 6. Створення збережених процедур для оновлення даних

Виконання всіх операцій з даними будемо здійснювати з використанням збережених процедур. Вибір такого способу взаимодії з

даними пояснюється насамперед тим, що при виконанні команди Insert застосуванню потрібно повернути значення первинного ключа, справжнє його значення відомо, в першу чергу, базі даних. Перш ніж використовувати застосування, треба створити збережені процедури для додавання, видалення та редагування інформації в кожній таблиці бази даних. На рис. 1.12 наведений список збережених процедур бази даних Produkcia.

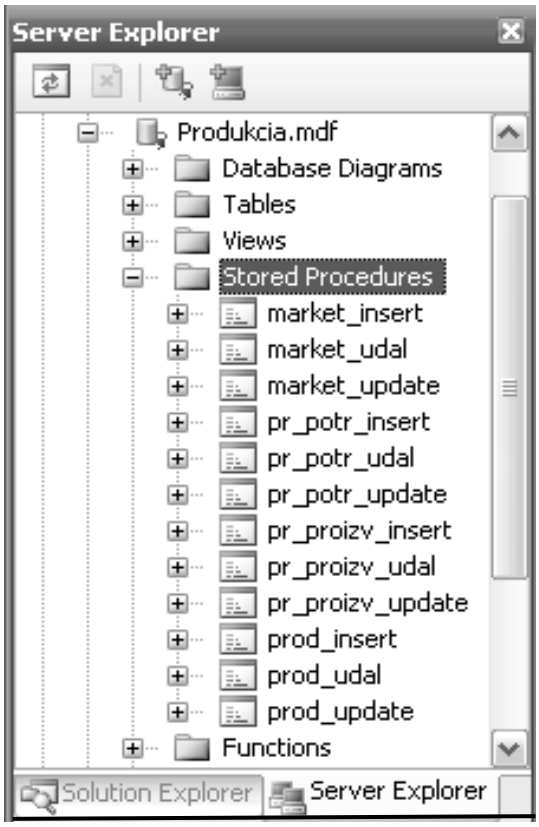

Рис. 1.12. **Список збережених процедур бази даних Produkcia**

Розглянемо вміст збережених процедур тільки для однієї таблиці. Нижче наведений вміст збереженої процедури додавання запису до таблиці "Предприятие_производитель":

```
ALTER PROCEDURE dbo.pr_proizv_insert
    @ID_pr_proizv int,
    @adr_proizv text,
    @nazv_proizv text,
    @BR_proizv text
AS
INSERT INTO Предприятие_производитель(Адрес_производства,
Название_производителя, БР_производителя)
values(@adr_proizv,@nazv_proizv,@BR_proizv)
```

if @@rowcount=0 return 1

select SCOPE_IDENTITY() ID_предприятия_производителя

RETURN 0

Збережена процедура виконує такі функції:

обчислює значення ID_предприятия_производителя, що становить наступне число;

виконує вставку нового рядка;

вибирає значення первинного ключа з тільки що уведеного рядка за допомогою функції SQL Server **SCOPE_IDENTITY()**.

Нижче наведений вміст збереженої процедури оновлення запису таблиці "Предприятие_производитель":

```
ALTER PROCEDURE dbo.pr_proizv_update
    @ID_pr_proizv int,
    @adr_proizv text,
    @nazv_proizv text,
    @BR_proizv text
AS
UPDATE Предприятие_производитель
SET Адрес_производства = @adr_proizv, Название_производителя =
@nazv_proizv, БР_производителя = @BR_proizv
WHERE (ID_предприятия_производителя = @ID_pr_proizv)
    RETURN
```

Збережена процедура виконує оновлення значень усіх полів рядка, який буде обраний в компоненті dataGridView1, за виключенням значення ключового поля.

Далі наведений вміст збереженої процедури видалення запису з таблиці "Предприятие_производитель":

```
ALTER PROCEDURE dbo.pr_proizv_udal
@ID_pr_proizv int
AS
DELETE FROM Предприятие_производитель
WHERE ID_предприятия_производителя = @ID_pr_proizv
RETURN
```

Блок № 7. Створення кнопки для завантаження даних з бази.

Завантаження даних виконується з використанням метода Fill() кожного з чотирьох об’єктів SqlDataAdapter (відповідно кількості таблиць). Нижче наведено фрагмент програми, який завантажує дані:

```
private void button1_Click(object sender, EventArgs e)
{
   SqlConnection con = new SqlConnection(connstring);
   con.Open();
   table_Market.Rows.Clear();
   table_Predpr_proizv.Rows.Clear();
   table_Predpr_potr.Rows.Clear();
   table_Prodykcia.Rows.Clear();
   comtext = "SELECT * FROM Предприятие_производитель";
   da = new SqlDataAdapter(comtext, con);
   da.Fill(table_Predpr_proizv);
   comtext = "SELECT * FROM Предприятие_потребитель";
   da = new SqlDataAdapter(comtext, con);
   da.Fill(table_Predpr_potr);
   comtext = "SELECT * FROM Продукция";
   da = new SqlDataAdapter(comtext, con);
   da.Fill(table_Prodykcia);
   comtext = "SELECT * FROM Рынок";
   da = new SqlDataAdapter(comtext, con);
   da.Fill(table_Market);
}
```

Слід відмітити, що завантаження даних також можно виконати за допомогою збережених процедур.

Блок № 8. Створення методу для оновлення даних у кожній таблиці. У фрагменті коду для оновлення даних використовується перевантажений метод Update() об'єкта SqlDataAdapter. Він додає (оновлює або видаляє) масив рядків, що відбираються за їхнім станом (DataViewRow State.Added, DataViewRowState.ModifiedCurrent, DataViewRowState. Deleted). Нижче наведено код цього методу:

```
public void Update_tables()
 {
// Додавання записів
  da.Update(ds.Tables[0].Select(null, null, DataViewRowState.Added));
  da1.Update(ds.Tables[1].Select(null, null, DataViewRowState.Added));
  da2.Update(ds.Tables[2].Select(null, null, DataViewRowState.Added));
  da3.Update(ds.Tables[3].Select(null, null, DataViewRowState.Added));
```

```
// Оновлення записів
        da.Update(ds.Tables[0].Select(null, null,
                                DataViewRowState.ModifiedCurrent));
        da1.Update(ds.Tables[1].Select(null, null,
                                DataViewRowState.ModifiedCurrent));
        da2.Update(ds.Tables[2].Select(null, null,
                                DataViewRowState.ModifiedCurrent));
        da3.Update(ds.Tables[3].Select(null, null,
                                        DataViewRowState.ModifiedCurrent));
// Видалення записів
da3.Update(ds.Tables[3].Select(null, null, DataViewRowState.Deleted));
da2.Update(ds.Tables[2].Select(null, null, DataViewRowState.Deleted));
da1.Update(ds.Tables[1].Select(null, null, DataViewRowState.Deleted));
da.Update(ds.Tables[0].Select(null, null, DataViewRowState.Deleted));
  }
```

Блок № 9. Створення кнопки для збереження даних.

У застосуванні, яке розробляється залишилося створити елемент завершення процесу оновлення даних. Для цього потрібно додати на форму кнопку "Зберегти зміни". Код, що виконує відповідну операцію, наведено нижче:

```
private void buttonSave_Click(object sender, EventArgs e)
{
 CreateComParam();
 Update_tables();
 ds.AcceptChanges();
}
```

При розробці застосування оновлення даних не обов’язково строго притримуватись алгоритму, який відображений на рис. 1.11, тобто деякі блоки алгоритму можуть бути поміняні місцями. Але для успішної роботи застосування, всі дії, які викладені в даному прикладі повинні бути виконані обов’язково.

Висновки. У лекції розглянуто виконання операцій додавання, видалення та оновлення ієрархічних даних. Подробно розглянуті особливості оновлення ієрархічних даних з використанням збережених процедур, які надають широкі можливості користувачу для роботи з базами даних. Особлива увага приділяється збереженню цілосності

даних, а також роботі з ключовими полями типу AutoIncrement. Приведений детальний приклад модифікації ієрархічних даних.

Тести для самодіагностики

Тест № 1. Застосування розробляється для виконання оновлення інформації в базі даних SQL Server. Вам необхідно набути останнього значення ключового поля з автоінкрементом, що згенерувало в даному з'єднанні. Який запит необхідно використовувати:

1) SELECT @Param=SCOPE_IDENTITY();
2) SELECT @Param=@@IDENTITY;
3) da.InsertCommand.UpdatedRowSource =
   UpdateRowSource.OutputParameters;
4) da.InsertCommand.UpdatedRowSource = UpdateRowSource.Both;?

Тест № 2. Ви розробляєте застосування BrainD, що використовує об'єкт DataSet, який містить два об'єкти DataTable. Одна таблиця містить інформацію клієнта, яка повинна бути показана в компоненті List. Інша – містить інформацію замовлення, показану в компоненті DataGrid. Вам необхідно змінити BrainD, щоб забезпечити взаємозв'язок таблиць. Що ви повинні зробити:

1) використати DataSet.Merge метод;
2) визначити первинні ключі для об'єктів Data Table;
3) створити обмеження зовнішнього ключа на об'єкта DataSet;
4) додати DataRelation в колекцію відносин об'єкта DataSet?

Тест № 3. Застосування використовує об'єкт набору даних на ім'я CurrentOrders. До CurrentOrders ви додаєте об'єкти DataTable – Orders і OrderDetails. OrderDetails містить дані про кожен пункт, включений в замовлення. Якщо замовлення не містить записів в OrderDetails, то воно видаляється з Orders. Користувачі не повинні видаляти замовлення, у яких є відповідні записи в OrderDetails. Що ви повинні зробити для того, щоб таких видалень не було:

1) додати об'єкт UniqueConstraint в CurrentOrders;
2) додати об'єкт ForeignKeyConstraint в CurrentOrders;
3) додати об'єкт DataRelation в CurrentOrders і встановити властивість ChildKeyConstraint;
4) додати об'єкт DataRelation в CurrentOrders і встановити властивість ParentKeyConstraint у відповідну колонку?

Контрольні завдання

1. Розробити застосування, що забезпечує відображення таблиць бази даних на формі в наочному вигляді (схема бази даних видається викладачем). При виділенні запису в батьківській таблиці на формі повинні відображатися всі залежні записи дочірньої таблиці.

2. Застосування має виконувати оновлення, додавання й видалення даних з таблиць. Забезпечити виконання цих операцій з використанням параметрів і збережених процедур.

3. Виконати тестування, використовуючи кілька екземплярів застосування для паралельної роботи при оновленні рядків у базі даних.

Література: [2, с. 44–54; 3, с. 418–454].

Тема № 2 Робота в ADO.NET з використанням транзакцій

Поняття транзакції. Локальні, розподілені, ручні і автоматичні транзакції. Основи використання транзакцій в ADO.NET. Використання транзакцій за класами DataSet і DataAdapter. Проміжні точки збереження і вкладені транзакції. Рівні ізоляції транзакцій. Розподілені транзакції і принципи управління їх виконанням. Диспетчер ресурсів і координатор розподілених транзакцій. Особливості розробки програм взаємодії з базами даних з використанням розподілених транзакцій і принципів паралелізму.

2.1. Лекція № 5. Взаємодія прикладних програм з базами даних з використанням транзакцій

2.1.1. Основи використання транзакцій в ADO.NET

Транзакції широко використовуються при роботі з базами даних. Все або нічого – в цьому головний сенс транзакції. При збереженні декількох записів або всі повинні бути записані у базі даних, або вся операція повинна бути відмінена. Якщо відбувається єдиний збій при внесенні одного запису, то все, що було виконано до цього моменту в межах даної транзакції, відкатується.

Транзакцією називається виконання послідовності команд в базі даних, яка або фіксується при успішному виконанні кожної команди, або відміняється при невдалому виконанні хоч би однієї команди.

До транзакції пред'являються спеціальні вимоги. У результаті її роботи повинно бути отримано коректний стан системи, який є необхідною умовою для нормального функціонування програми, оскільки у процесі виконання транзакції можливі різні збої в роботі системи, наприклад, збій живлення на сервері. Характеристики транзакції можуть бути визначені терміном ACID. ACID – це абревіатура чотирьох слів, що означають атомарність (atomicity), узгодженість (consistency), ізоляцію (isolation) і стійкість (durability).

Атомарність представляє одиницю роботи. Відносно транзакції це означає, що або вся одиниця роботи буде успішно виконана, або нічого не буде виконано.

Узгодженість – перед стартом транзакції і після її завершення стан системи повинні бути коректним.

Ізоляція означає, що транзакції, що виконуються одночасно, ізольовані від стану, який змінюється під час транзакції. Транзакція А не може бачити проміжного стану транзакції В до тих пір, поки вона не буде завершена.

Стійкість – після завершення транзакції її результат повинен бути незворотним (навіть при відмові системи). Це значить, що якщо відбудеться збій мережі або сервера, то стан повинен бути відновлений.

Види транзакцій:

локальна транзакція – використовує підтримуюче транзакції джерело даних (наприклад, SQL Sever) і не виходить за рамки однієї транзакції або однієї бази даних;

розподілена транзакція – охоплює декілька підтримуючих транзакції джерел даних, і їй можливо буде читати повідомлення з черги повідомлень сервера, вибирати дані з бази даних SQL Server і записувати їх в інші бази даних;

ручна транзакція – використовує явні інструкції початка і кінця транзакцій, дозволяє почати нову транзакцію з іншої активної транзакції (тобто підтримує вкладені транзакції);

автоматична транзакція – якщо у транзакцію залучено багато диспетчерів ресурсів (таких як SQL Server), то кращим вибором буде автоматична транзакція, всю роботу з координації виконує System. Transaction або COM+, це створює додаткове навантаження на систему, але сильно спрощує структуру застосування і зменшує вимоги до кодування.

У ADO.NET є набір класів: SqlConnection, SqlTransaction і т. д. призначених для створення підключення до джерела даних, початку, фіксації або відкочування транзакції, а потім ручного закриття підключення. У ADO.NET реалізований могутній механізм підтримки транзакцій. Можна задіювати простір імен System.Transactions для виконання транзакцій з декількома базами даних або диспетчерами ресурсів.

Простір імен System.Transactions додав нову транзакційну модель у застосування .NET. Деякі класи цієї моделі перераховані в табл. 2.1.

Таблиця 2.1

Класи транзакційної моделі .NET

Клас	Опис
1	2
Transaction	Базовий клас для всіх транзакційних класів, який визначає властивості, методи і події, доступні у всіх транзакційних класах
CommittableTransaction	Застосовується для організації розподілених транзакцій
DependentTransaction	Використовується з транзакціями, залежними від інших транзакцій
SubordinateTransaction	Застосовується в поєднанні з координатором розподілених транзакцій (Distributed Transaction Coordinator – DTC)
TransactionScope	Застосовується для організації декількох з'єднань з базами даних у рамках однієї транзакції

У табл. 2.2 і табл. 2.3 наведені властивості і методи об'єкта Transaction відповідно.

Таблиця 2.2

Властивості об'єкта Transaction

Властивість	Опис
Current	Повертає поточну транзакцію, якщо така існує
IsolationLevel	Визначає тип доступу інших транзакцій до проміжних результатів транзакції
TransactionInformation	Тримає інформацію про поточний стан транзакції, час створення транзакції і її ідентифікатори

Таблиця 2.3

Методи об'єкта Transaction

Метод	Опис
Clon	Створити клон транзакції
Commit	Підтвердити виконання транзакції
Dispose	Завершує область дії об’єкта транзакції
DependetClon	Створити транзакцію, залежну від поточної транзакції
GetType	Повертає тип поточного екземпляра транзакції
Rollback	Перериває транзакцію і відміняє все, що було зроблено в її рамках, повертаючи все у стан, що був спочатку транзакції
Save	Створює "точку збереження" для відкочування транзакції
EnlistVolatile(), EnlistDurable(), PromotableSinglePhase()	Методами класу CommittableTransaction можна задіювати диспетчери ресурсів, що приймають участь у транзакції

ADO.NET забезпечує як підключений, так і автономний доступи до даних і підтримує транзакції в обох цих режимах. У підключеному режимі типова послідовність дій у транзакції така:

відкривається підключення до бази даних за допомогою методу **Open()** об'єкта підключення;

починається транзакція за допомогою методу **BeginTransaction()** об'єкта підключення, цей метод повертає об'єкт транзакції, який надалі використовується для фіксації або відкочування транзакції, всі зміни, виконані будь-якими запитами до виклику методу **BeginTransaction()**, фіксуються в базі даних відразу ж після їх виконання;

за допомогою об'єкта команди виконується команда SQL, для цієї мети можна використовувати більше одного об'єкта команди, але тільки якщо у властивості Transaction всіх цих об'єктів вказаний допустимий об'єкт транзакції;

транзакція фіксується або відкочується за допомогою методу **Commit()** або **Rollback()** об'єкта транзакції;

підключення до бази даних закривається.

Нижче наведений приклад використання транзакції:

```
try
{
 connection.Open();
 // Створюється об'єкт транзакції
        transaction = connection.BeginTransaction();
 // Транзакція фіксується в командах
        command1.Transaction = transaction;
        command2.Transaction = transaction;
 // Виконання команд
         command1.ExecuteNonQuery();
        command2.ExecuteNonQuery();
 // Якщо ВСЕ ДОБРЕ – ПРИЙНЯТИ ТРАНЗАКЦІЮ!
         transaction.Commit();
}
catch
{
 // Якщо виникли ускладнення – відміняємо транзакцію
         transaction. Rollback();
}
finally
{
 // З’єднання закривається
         connection.Close();
}
```

В автономному режимі дані (звичайно одна або більш таблиць) вибираються в об'єкт DataSet. У цьому режимі звичайна послідовність дій така:

відкриття підключення до бази даних;

вибірка потрібних даних в об'єкт DataSet;

закриття підключення;

обробка даних в об'єкті DataSet;

знову відкриття підключення до бази даних;

початок транзакції;

призначення об'єкта транзакції відповідним командам адаптера даних;

занесення в базу даних змін з DataSet;

закриття підключення.

Приведемо приклад, в якому виконується додавання записів у батьківську і дочірню таблиці бази даних з використанням транзакції в автономному режимі. Для виконання оновлення даних необхідно створити такі об'єкти:

```
SqlConnection MyConn = new SqlConnection();
SqlTransaction transaction = null;
SqlDataAdapter sqlDA1 = new SqlDataAdapter();
SqlDataAdapter sqlDA0 = new SqlDataAdapter();
SqlCommand insertCommand = new SqlCommand();
SqlCommand insertCommand1 = new SqlCommand();
```

Далі вважатимемо, що створення команд оновлення даних **insertCommand** і **insertCommand1** та їх параметрів для кожної таблиці виконується як у блоці № 5 (рис. 1.11) методом CreateComParam(). Саме додавання даних здійснюється за допомогою збережених процедур (блок № 6, рис. 1.11). Нижче наведений метод **UpdateWithTransaction()**, який виконує додавання даних у батьківську і дочірню таблиці:

```
public void UpdateWithTransaction()
{
  MyConn.Open();
// Створюємо транзакцію
transaction = MyConn.BeginTransaction();
// Включення команд до транзакції
    insertCommand.Transaction = transaction;
    insertCommand1.Transaction = transaction;
    sqlDA.InsertCommand = insertCommand;
    sqlDA1.InsertCommand = insertCommand1;
 try
 {
sqlDA.Update(ds.Tables[0].Select(null, null, DataViewRowState.Added));
sqlDA1.Update(ds.Tables[1].Select(null,null,DataViewRowState.Added));
//Підтверджуємо транзакцію
 transaction.Commit();
 }
 catch (Exception ex)
 {
//Відхиляємо транзакцію
 transaction.Rollback();
```

```
MessageBox.Show("Не можливо виконати!  Відкат транзакції");
MessageBox.Show(ex.Message);
}
finally
{ MyConn.Close(); }
   }
```

2.1.2. Рівні ізоляції транзакцій. Проміжні точки збереження і вкладені транзакції

Основні правила, яких необхідно дотримуватися при розробці застосувань з використанням транзакцій:

після початку транзакції по якому-небудь підключенню всі запити, що використовують це підключення, повинні відноситися до цієї транзакції, наприклад, усередині транзакції можуть бути виконані два запити **INSERT** і один **UPDATE**. І якщо у середині транзакції спробувати виконати запит **SELECT** без транзакції, то ви одержите повідомлення про помилку, вказуюче, що транзакція ще не завершена.

у одного об'єкта підключення у будь-який момент часу може бути тільки одна чекаюча завершення транзакція, після виклику методу **BeginTransaction** об'єкта підключення неможливо ще раз викликати **BeginTransaction**, поки для цієї транзакції не буде виконана фіксація або відкіт, у таких ситуаціях ви одержите повідомлення про те, що паралельні транзакції не підтримуються, щоб позбавиться цієї помилки, для виконання запиту потрібно використовувати інше підключення.

При виконанні транзакцій декількома користувачами однієї бази даних можуть виникати такі проблеми:

Dirty reads (брудне читання) – при брудному читанні інша транзакція може читати записи, змінені (але незафіксовані) усередині поточної транзакції, такі помилки можна усувати блокуванням змінних записів;

On-repeatable reads (невідтворне читання) –- невідтворне читання трапляється, коли дані читаються усередині транзакції, і поки ця транзакція виконується, інша транзакція змінює ті ж самі записи, якщо запис читається повторно усередині транзакції, буде одержаний результат, що відрізняється, – невідтворний, цьому можна запобігти за допомогою блокування читання записів;

Phantom reads (читання фантомів) – фантомне читання трапляється при читанні або оновленні діапазону даних, наприклад, з використанням конструкції WHERE, інша транзакція може додати новий запис, який потрапляє в діапазон тих, що підлягають читанню в транзакції, цієї проблеми можна уникнути, якщо застосувати блокування діапазону рядків.

Для вирішення цих проблем розроблені декілька рівнів ізоляції транзакції:

Read uncommitted – транзакція може прочитувати дані, з якими працюють інші транзакції, застосування цього рівня ізоляції може привести до всіх перерахованих проблем;

Read committed – транзакція не може прочитувати дані, з якими працюють інші транзакції, застосування цього рівня ізоляції виключає проблему "брудного" читання;

Repeatable read – транзакція не може прочитувати дані, з якими працюють інші транзакції, інші транзакції також не можуть прочитувати дані, з якими працює ця транзакція, застосування цього рівня ізоляції виключає всі проблеми, окрім читання фантомів;

Serializable – транзакція повністю ізольована від інших транзакцій застосування цього рівня ізоляції повністю виключає всі проблеми;

Snapshot – цей рівень ізоляції зменшує вірогідність установки блокування рядків, зберігаючи копію даних, які одне застосування може читати, тоді як інше модифікує ці ж дані, іншими словами, якщо транзакція A модифікує дані, то транзакція B не побачить виконані зміни, але важливо те, що транзакція B не буде заблокована і читатиме знімок даних, зроблений перед початком транзакції A, в SQL Server 2005 ізоляція **Snapshot** перед використанням повинна бути дозволена на рівні самої бази даних;

Chaos – транзакція не може перезаписати інші неприйняті транзакції з великим рівнем ізоляції, але може перезаписати зміни, внесені без використання транзакцій. Дані, з якими працює поточна транзакція, не блокуються;

Unspecified – транзакція з цим рівнем може застосовуватися для завдання власного рівня ізоляції.

У ADO .NET рівень ізоляції можна встановити при створенні транзакції, наприклад:

```
myCommand.Transaction =
conn.BeginTransaction (System.Data.IsolationLevel.Serializable);
transaction =
conn.BeginTransaction(System.Data.IsolationLevel.ReadCommitted);
transaction =
conn.BeginTransaction(System.Data.IsolationLevel.ReadUncommitted);
transaction =
conn.BeginTransaction(System.Data.IsolationLevel.RepeatableRead);
```

Використання найбільшого рівня ізоляції (Serializable) означає найбільшу безпеку і разом з тим найменшу продуктивність – всі транзакції виконуються у вигляді серії, подальша вимушена чекати завершення попередньої. Застосування найменшого рівня (Read uncommitted) означає максимальну продуктивність і повну відсутність безпеки. Втім, не можна дати універсальних рекомендацій з застосування цих рівнів – у кожній конкретній ситуації рішення залежатиме від структури бази даних і характеру виконуваних запитів.

При відкочуванні транзакції відбувається анулювання кожної операції транзакції. У деяких випадках не потрібно відміняти повністю всі операції –- значить потрібен механізм відкоту тільки частини транзакції. Це можна реалізувати за допомогою точок збереження.

Точки збереження (savepoint) – це маркери, що виконують роль закладок. Під час виконання транзакції можна помітити яку-небудь точку, і потім виконати відкочування до цієї точки замість повного відкочування всієї транзакції. Для цього призначений метод Save об'єкта транзакції (табл. 2.3). Метод Save є тільки у класі SqlTransaction, але не у класах OleDbTransaction і OracleTransaction.

Приведемо приклад оновлення даних у батьківській та дочірній таблицях, в якому використовується точки збереження. У прикладі вважатимемо, що створення команд додавання і видалення даних та їх параметрів для кожної таблиці виконується як у блоці № 5 (рис. 1.12) методом CreateComParam(). Саме додавання даних здійснюється за допомогою збережених процедур (блок № 6, рис. 1.12). Нижче наведений код методу **UpdateWithSaveTransaction()**, який виконує додавання і видалення даних з батьківської і дочірньої таблиць:

```
public void UpdateWithSaveTransaction ()
{ conn.Open();
```

```
// Створюємо транзакцію і задаємо рівень ізоляції ReadCommitted
transaction =
conn.BeginTransaction(System.Data.IsolationLevel.ReadCommitted);
// Включення команд до транзакції
        insertCommand.Transaction = transaction;
        insertCommand1.Transaction = transaction;
        deleteCommand.Transaction = transaction;
        deleteCommand1.Transaction = transaction;
        sqlDA.InsertCommand = insertCommand;
        sqlDA1.InsertCommand = insertCommand1;
        sqlDA.DeleteCommand = deleteCommand;
        sqlDA1.DeleteCommand = deleteCommand1;
   try
   {
// Додавання даних у батьківську таблицю
sqlDA.Update(ds.Tables[0].Select(null, null, DataViewRowState.Added));
       if (MessageBox.Show("Створити  точку зберігання??",
"Точка зберігання...", MessageBoxButtons.YesNo,
MessageBoxIcon.Question) == DialogResult.Yes)
    { transaction.Save("MySave");
    MessageBox.Show("Створена точка зберігання!!!"); }
// Додавання даних у дочірню таблицю
sqlDA1.Update(ds.Tables[1].Select(null, null,
                                    DataViewRowState.Added));
// Видалення даних з батьківської таблиці
sqlDA1.Update(ds.Tables[1].Select(null, null,
                                    DataViewRowState.Deleted));
// Видалення даних з дочірньої таблиці
sqlDA.Update(ds.Tables[0].Select(null, null,
                                    DataViewRowState.Deleted));
if (MessageBox.Show("Виконати відкочування транзакції в точку
зберігання??", "Точка зберігання...", MessageBoxButtons.YesNo,
MessageBoxIcon.Question) == DialogResult.Yes)
 {
//Частково відхиляємо транзакцію до точки зберігання
    transaction.Rollback("MySave");
MessageBox.Show("Виконано відкат транзакції в точку зберігання ");
```

```
 }  //Підтверджуємо транзакцію
    transaction.Commit();
    MessageBox.Show("Оновлення даних виконано!");
    }
   catch (Exception ex)
   {
//Відхиляємо транзакцію
   transaction.Rollback();
MessageBox.Show("Виконано відкочування транзакції !");
   MessageBox.Show(ex.Message);
   }
   finally
   { conn.Close(); }
  }
```

Метод виконує в цілому чотири запити для додавання і видалення рядків в батьківській і дочірній таблицях. Після вставки рядків у батьківську таблицю, метод в діалозі з користувачем, встановлює точку збереження за допомогою наступного оператора:

```
transaction.Rollback("MySave");
```

Метод **UpdateWithSaveTransaction()** може додати ще рядки в дочірню таблицю, а також видалити рядки з дочірньої і батьківської таблиць. Крім того, можна виконати відкочування до точки збереження **MySave**. Зверніть увагу, як той же метод **Rollback** використовується з ім'ям точки збереження як параметр (щоб відкочувати транзакцію повністю, параметр не потрібен). Після цього застосування остаточно фіксує транзакцію.

Поширені помилки при роботі з точками збереження:

розробники часто забувають викликати метод **Commit** або **Rollback** після відкочування до однієї з точок збереження, точки збереження краще розглядати як закладки – все одно потрібно викликати **Commit** або **Rollback**;

після виконання відкочування до однієї з точок збереження всі точки збереження, встановлені за цією точкою, втрачаються, якщо вони потрібні, їх доведеться встановити наново.

Точки збереження дозволяють організувати транзакцію у вигляді послідовності дій, для кожного з яких можна виконати індивідуальне відкочування. А вкладення транзакцій дає можливість організувати

ієрархію таких дій. У разі вкладених транзакцій одна транзакція може містити одну або декілька інших транзакцій. Для запуску таких вкладених транзакцій використовується метод **Begin** класу транзакції. Цей метод доступний тільки для класу **OleDbTransaction**, але не для SqlTransaction або **OracleClientTransaction**. За допомогою класу **OleDbTransaction** вкладену транзакцію можна створити таким чином:

transaction = connection.BeginTransaction ();

anothertransaction = transaction.Begin ();

Метод **Begin** повертає екземпляр іншого об'єкта транзакції, який можна використовувати так само, як і первинний об'єкт транзакції. Але відкіт цієї транзакції просто відкатує поточну транзакцію, але не всю в цілому. Вкладення транзакцій можливо і для **SqlTransaction** при використанні класів, що реалізовують розподілені транзакції, наприклад клас **TransactionScope.**

2.1.3. Взаємодія прикладних програм з базами даних з використанням розподілених транзакцій

Транзакція, що охоплює декілька ресурсів, називається розподіленою транзакцією. Наприклад, фінансові перекази між рахунками банків є розподіленою транзакцією.

Найбільш важливі учасники в розподілених транзакціях – це:

диспетчера ресурсів (resource manager – RM), що безпосередньо виконують всю роботу і що повідомляють про успішне або невдале її завершення;

диспетчер транзакцій або координатор розподілених транзакцій (Distributed Transaction Coordinator – DTC).

Координатор транзакцій, який поставляється з Windows, називається координатором розподілених транзакцій Microsoft (Microsoft Distributed Transaction Coordinator – MSDTC) є служба Windows, яка виконує координацію транзакцій для розподілених застосувань.

У будь-якому випадку, координатор транзакції координує роботу всіх RM, щоб забезпечити їх успішне завершення або всі вони відмінили результати своєї роботи.

Структурна схема взаємодії учасників розподіленої транзакції при переказі грошей з одного рахунку на інший приведена на рис. 2.1.

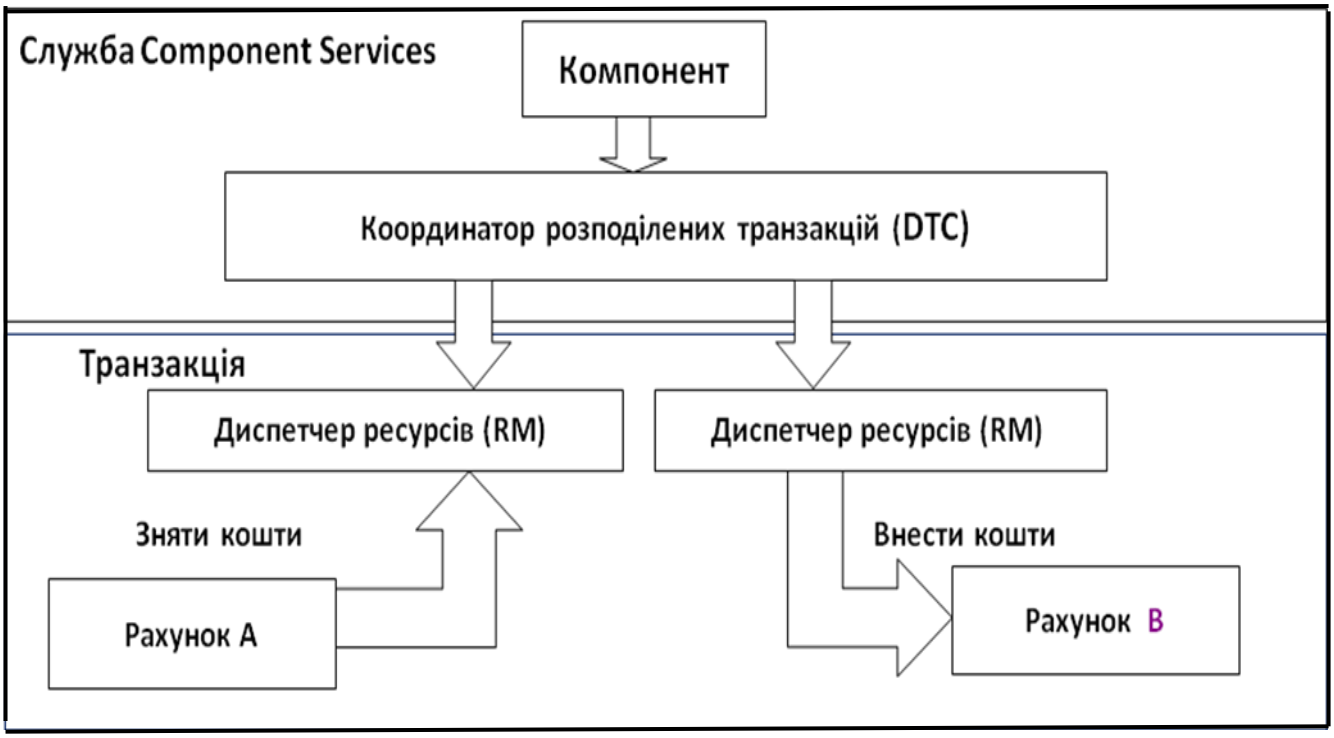

Рис. 2.1. **Структурна схема взаємодії учасників розподіленої транзакції**

Стандартний алгоритм виконання розподіленої транзакції:

застосування починає транзакцію, запрошуючи її у MSDTC, це застосування звичайно називається ініціатором;

потім застосування пропонує RM виконувати свою роботу у складі цієї транзакції, і диспетчери ресурсів реєструються в диспетчері транзакцій як частина цієї ж транзакції, звичайно це називається включенням або занесенням (enlisting) у транзакцію;

якщо все нормально, то застосування фіксує транзакцію, якщо щось не так, кожен крок може виконати відкочування.

Простір імен System.EnterpriseServices має безліч корисних служб. Одна з них – автоматичні транзакції. Використання транзакції з System.EnterpriseServices володіє тією перевагою, що не доводиться мати справу з транзакціями безпосередньо; транзакції автоматично створюються виконуючою системою. Недоліком є те, що потрібна модель хостингу COM +, і те, що клас, що використовує засоби цієї технології, повинен успадковуватися від класу ServicedComponent.

Простір імен System.Transactions доступний, починаючи з версії .NET 2.0, привніс нову транзакційну модель у застосування .NET. Складовими частинами цієї моделі є ціла група класів: CommittableTransaction; DependentTransaction; TransactionScope (табл. 2.1).

2.1.4. Розробка застосувань з використанням розподілених транзакцій.

Дуже зручним інструментом для роботи з розподіленими транзакціями є клас TransactionScope простору імен System.Transactions. Наприклад, відкриття з'єднання в рамках класу TransactionScope спричинить за собою автоматичну вказівку цього з'єднання у транзакції.

Виклик методу Commit класу TransactionScope підтверджує виконувану транзакцією дію. При виклику методу Dispose цього класу (неявно в кінці блоку Using або явно) виконувана у транзакції непідтверджена дія автоматично відміняється.

Як працювати з безліччю з'єднань у класі TransactionScope і підтверджувати зміни розглянемо на прикладі додавання рядків у різні таблиці, причому додавання кожного рядка виконується за окремим підключенням до бази даних. Код прикладу приведений нижче:

```
public void transact_scope()
  {
// Створюємо транзакцію
   using (TransactionScope cStransaction = new TransactionScope())
   {
    try
    {
// Виконуємо першу операцію
using (SqlConnection connection1 = new
                                   SqlConnection(connectionString))
     {
      Connection1.Open();
      SqlCommand myCommand = connection1.CreateCommand();
      myCommand.CommandText =
                          "Insert into Кредит(Сума_кредиту) Values (100) ";
      myCommand.ExecuteNonQuery();
      transInformation(Transaction.Current.TransactionInformation);
     }
// Перша операція транзакції завершена, переходимо до другої
using (SqlConnection connection2 = new
                                   SqlConnection(connectionString2))
     {
      Connection2.Open();
```

```
      SqlCommand myCommand = connection2.CreateCommand();
      myCommand.CommandText =
                        " Insert into Дебіт(Сума_дебіту) Values (100) ";
      transInformation(Transaction.Current.TransactionInformation);
      myCommand.ExecuteNonQuery();
     }
// Підтверджуємо транзакцію
     cStransaction.Complete();
   }
   catch (System.Exception ex)
    {
     MessageBox.Show(ex.Message);
     transInformation(Transaction.Current.TransactionInformation);
     cStransaction.Dispose();
    }
   } }
```

У приведеному прикладі транзакція оформлюється за допомогою блоку using. Застосування створює новий екземпляр TransactionScope, який визначає частину коду, що додається у транзакцію. І весь код між конструктором **TransactionScope** і викликом **Dispose** для цього екземпляра **TransactionScope** заноситься у створювану транзакцію.

У прикладі є виклик методу **transInformation()**, який призначений для виведення інформації про стан транзакції. Об'єкт класу **TransactionScope** не має властивості **TransactionInformation**, проте ця властивість доступна з поточної транзакції (**Transaction.Current**). Код методу **transInformation()** приведений нижче:

```
public void transInformation(TransactionInformation information)
 {
  string str = " Стан транзакції:   " +
            Convert.ToString ((object) (information.Status));
  string str1 =
"Ідентифікатор транзакції : " +information.LocalIdentifier +
"Розподілений ідентифікатор: " + information.DistributedIdentifier;
  string str2 = " Час створення транзакції  " +
            Convert.ToString((object)(information.CreationTime));
  str += str1+str2;
  MessageBox.Show(str1);  }
```

У результаті використання методу **transInformation()** буде одержане повідомлення про виконання транзакції (рис. 2.2.). У цьому повідомленні визначений ідентифікатор транзакції, але ідентифікатор розподіленої транзакції дорівнює 0, що свідчує про те, що транзакція ще не є розподіленою.

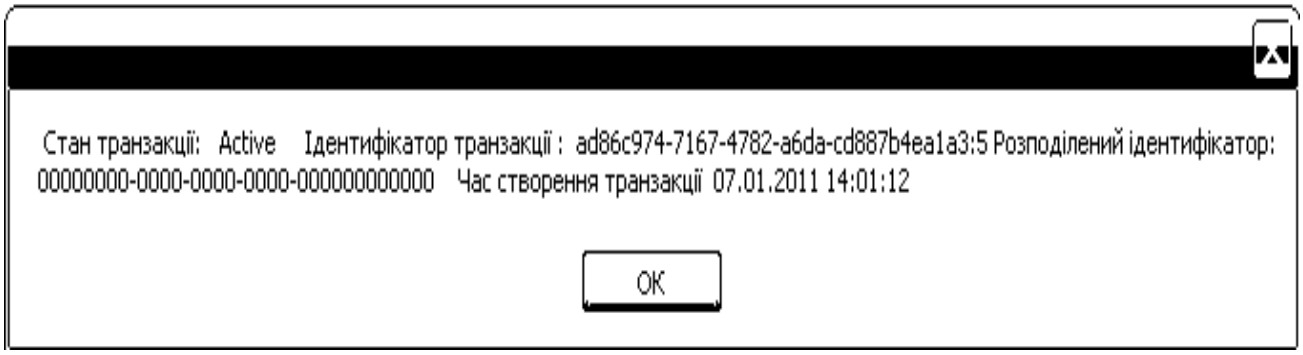

Рис. 2.2. **Повідомлення про виконання транзакції**

Один з конструкторів TransactionScope дозволяє запропонувати рівень ізоляції для окремих RM з транзакції. Саме запропонувати, але не визначити, оскільки за необхідністю RM може і проігнорувати цю пропозицію і реалізувати вищий рівень ізоляції. Потім за допомогою блоку using починають роботу різні RM – в нашому випадку два об'єкти SqlConnection, що підключаються до двох різних баз даних. У такий спосіб можна створити декілька компонентів, які включають себе у транзакцію MSDTC. Кожний з них викликає метод SetComplete при успішному завершенні або SetAbort – при невдалому. Виклик SetComplete повідомляє координатора транзакцій, що зміни повністю готові та чекають інших. Якщо хто-небудь в ланцюжку викличе SetAbort, то всі оновлення відміняються, і жодна зміна не фіксується.

Після компіляції та запуску застосування можна встановити точку переривання в кінець виконання команди першого підключення ExecuteNonQuery. Якщо в цій точці перейти в SQL Server Management Studio і виконати команду SQL:

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
BEGIN TRANSACTION
SELECT Сума_кредиту FROM Кредит
COMMIT ,

то SQL Server 2005 повідомить, що поки вставлено тільки один рядок. Врахуйте, що ця вставка залежить від другої вставки, тобто ви бачите результат "брудного" читання.

Інформацію про виконання транзакції можно отримати якщо відкрити аплет Control Panels /Administrative Tools /Component Services (Панель управління /Средства адміністрування /Служба компонентів) і відкрити на лівій панелі список транзакцій, то під рядком координатора розподілених транзакцій (Distributed Transaction Coordinator) не буде нічого, поки не почне працювати підключення connection2. Список розподілених транзакції, які виконуються під керівництвом DTC приведений на рис. 2.3.

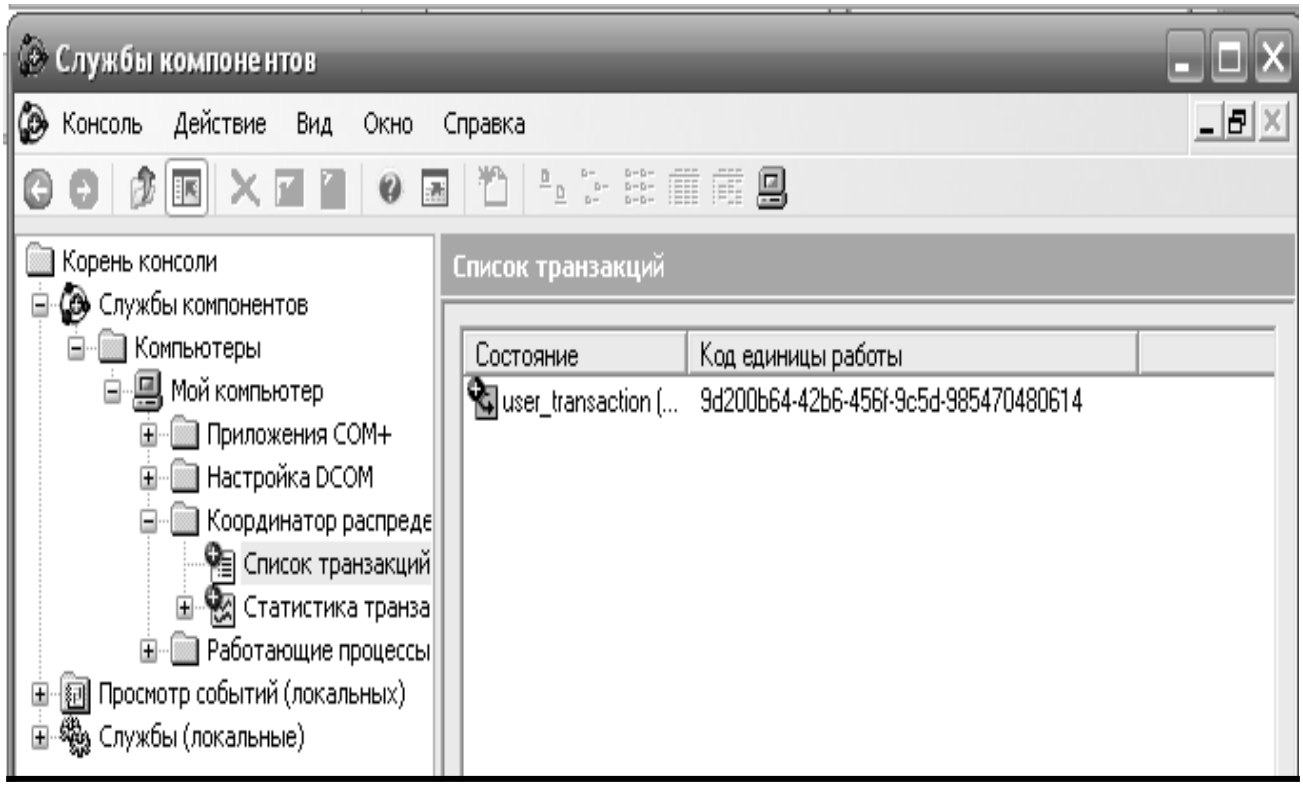

Рис. 2.3. **Розподілені транзакції, які виконуються під керівництвом DTC**

При покроковому виконанні коду в режимі відладки, як тільки буде пройдений рядок connection2.0pen, у списку транзакцій DTC з'явиться транзакція. Це важливе поняття, яке звичайно називається підвищуване занесення транзакції або розповсюдження транзакції (promotable enlistment). Повідомлення про виконання розподіленої транзакції буде отримане за допомогою методу **transInformation()** (рис. 2.4).

У цьому повідомленні визначений ідентифікатор розподіленої транзакції (9d200b64-42b6-456f-9c5d-985470480614), що також свідчить про підвищуване занесення її до рівня DTC.

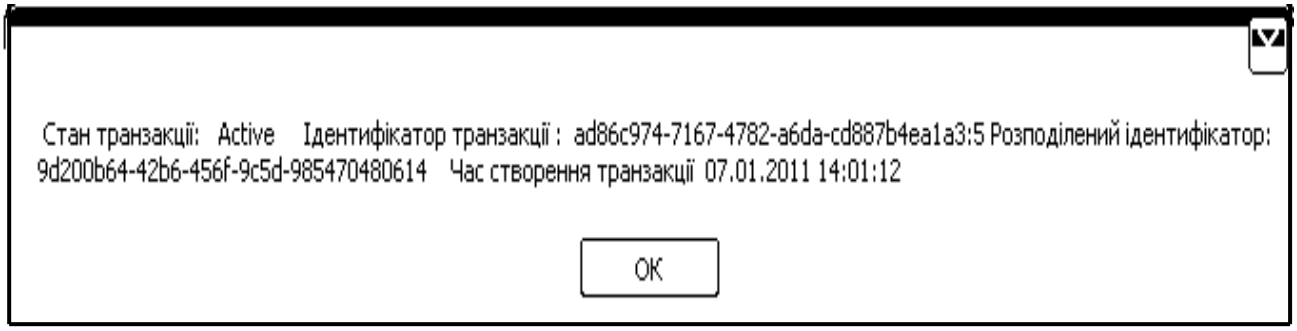

Рис. 2.4. **Повідомлення про виконання розподіленої транзакції**

При використанні класу CommittableTransaction необхідно явно вказати з'єднання, які повинні брати участь у виконанні транзакції. Для вказівки з'єднання у клас CommittableTransaction потрібно передати об'єкт транзакції методу з'єднання EnlistTransaction.

При явному або неявному виклику методу Dispose класу CommittableTransaction автоматично відміняється будь-яка непідтверджена дія, що виконується у цьому класі. Нижче приведений код рішення попередньої задачі з використанням классу **CommittableTransaction:**

```
public void transact_committable()
{
SqlConnection connection1 = new SqlConnection(connectionString);
SqlConnection connection2 = new SqlConnection(connectionString1);
using (CommittableTransaction comTransaction =
                                        new CommittableTransaction())
 {
  try
  {
// Виконуємо першу операцію
   connection1.Open();
   connection1.EnlistTransaction(comTransaction);
   SqlCommand myCommand = connection1.CreateCommand();
   myCommand.CommandText =
                    "Insert into Кредит(Сума_кредиту) Values (100) ";
   myCommand.ExecuteNonQuery();
   transInformation(comTransaction.TransactionInformation);
// Перша операція транзакції завершена, переходимо до другої
   connection2.Open();
   connection2.EnlistTransaction(comTransaction);
   SqlCommand myCommand1 = connection2.CreateCommand();
myCommand1.CommandText =
                        " Insert into Дебіт(Сума_дебіту) Values (100)" ;
   transInformation(comTransaction.TransactionInformation);
   myCommand1.ExecuteNonQuery();     }
```

```
  catch (System.Exception ex)
  {
MessageBox.Show(ex. Message);
   comTransaction.Rollback();
   MessageBox.Show("Не можливо виконати!  ");
   transInformation(comTransaction.TransactionInformation);
  }
  finally
  {
   transInformation(comTransaction.TransactionInformation);
   cn1. EnlistTransaction(null);
   cn2. EnlistTransaction(null);
  connection1.Close();    connection2.Close();
  }
  } }
```

У даному прикладі клас **CommittableTransaction** дозволяє виконувати підвищуване занесення транзакції. Крім того, оскільки клас **CommittableTransaction** має властивість **TransactionInformation**, то виклик методу **transInformation()** дещо спрощується за допомогою параметра **comTransaction.TransactionInformation**.

Для того, щоб зафіксувати розподілену транзакцію необхідно дотримуватися таких правил:

тільки творець розподіленої транзакції може її зафіксувати;

копії CommittableTransaction, одержані за допомогою виклику System.Transactions. ransaction.Clone, не можуть бути зафіксовані.

Приклади застосування System.Transactions працюватимуть і з іншими базами даних, такими як Oracle. Проте підвищуване занесення працює тільки для SqlConnection для підключень до SQL Server 2005.

Рекомендації по створенню транзакцій:

робіть транзакції якомога коротше;

уникайте повернення даних за допомогою оператора SELECT у середині транзакції, крім випадків, коли логіка залежить від даних, що повертаються;

при використанні оператора SELECT вибирайте тільки потрібні рядки, щоб не блокувати дуже багато ресурсів і підтримувати максимальну продуктивність;

намагайтесь писати транзакції повністю або на T-SQL, або за допомогою ADO, змішування того і іншого приведе до плутанини, можливі ситуації, коли доведеться написати транзакцію повністю на T-SQL – це нормально, якщо це дійсно потрібне, а ось чого необхідно уникати, так це починати транзакцію за допомогою SqlTransaction, відкочувати або фіксувати її з процедури, що зберігається, або якимсь іншим способом;

уникайте транзакцій, в яких об'єднуються декілька незалежних пакетів робіт, помістіть кожен такий пакет в окрему транзакцію;

по можливості уникайте великих оновлень, якщо ви не можете уникнути цього – просто не збільшуйте без необхідності розмір транзакції, оскільки це приведе до блокування безлічі ресурсів;

якщо транзакція явно не фіксується, то вона відкочується, така стандартна поведінка дозволяє виконати відкочування, але все таки краще явно викликати метод Rollback, це не тільки зніме всі блокування даних, але і зробить код зрозумілішим і менш схильним до помилок;

слід уникати ситуацій, коли користувач виконує якісь дії у середині транзакції.

Висновки. У лекції розглянуті такі питання:

поняття транзакції, локальні, розподілені, ручні і автоматичні транзакції, основи використання транзакцій в ADO .NET;

використання транзакцій з класами DataSet і DataAdapter, проміжні точки збереження і вкладені транзакції, рівні ізоляції транзакцій; розподілені транзакції і принципи управління їх виконанням;

диспетчер ресурсів і координатор розподілених транзакцій, особливості розробки програм взаємодії з базами даних з використанням розподілених транзакцій і принципів паралелізму.

Тести для самодіагностики

Тест № 1. Ви розробляєте застосування, в якому виконується транзакція, що містить команду відновити інформацію у таблиці бази даних SQL Server, яка містить трігер для цієї ж таблиці, що виконує команду Select. Який результат слід чекати при оновленні таблиці:

1) дані таблиці будуть оновлені і виконається команда Select;
2) дані таблиці будуть оновлені і не виконається команда Select;
3) відбудеться блокування (клінч, deadlock);
4) SQL Server забезпечить вихід з блокування і транзакція відкочується;
5) транзакція зробить відкочування?

Тест № 2. Який рівень ізоляції може бути дозволений тільки на рівні бази даних:

1) Repeatable read;
2) Read uncommitted;
3) Read committed;
4) Serializable;
5) Snapshot;
6) Chaos?

Тест № 3. Ви розробляєте застосування, використовуючи постачальник даних .NET для SQL Server. Вам необхідно створити вкладену транзакцію. Який код для цього слід використовувати:

1) SQLTransaction tran = conn.BeginTransaction();
   myanothertransaction = tran.Begin();
2) SqlTransaction tran = conn.BeginTransaction();
   tran.Save("SavePoint");
3) SqlTransaction tran = conn.BeginTransaction();
   SqlTransaction tran1 = conn1.BeginTransaction();
4) SqlTransaction tran = conn.BeginTransaction();
   SqlTransaction tran1 = BeginTransaction.conn(); ?

Контрольні завдання

1. Розробити застосування, яке демонструє дію рівнів ізоляції транзакцій.

2. Розробити застосування, яке використовує вкладену транзакцію за допомогою класу TransactionScope при роботі з СУБД SQL Server.

Глосарій

Термін	Значення
1	2
ADO (ActiveX Data Objects)	Набір компонентів ActiveX, які використовуються для доступу до баз даних
Автоінкремент	Властивість ключового поля таблиці, яке має цілий тип даних і автоматично змінюється в базі даних з визначеним кроком
База даних	Сукупність даних, об'єднаних за певними ознаками і правилами
DTC (Distributed Transaction Coordinator)	Координатор розподілених транзакцій. Виконує керування ресурсами розподілених транзакцій
Дані	Окремі факти, що характеризують об'єкти, процеси і явища наочної області, а також їх властивості
Джерело даних	Об'єкт, який містить і поставляє дані в заданий момент часу
З’єднане середовище	Середовище, в якому застосування знаходиться у постійному з’єднанні з базою даних
IS (information system)	Інформаційна система, яка забезпечує збирання, зберігання та доступ користувачів до даних
Клас	Основна одиниця організації даних в .NET
Командний об'єкт	Обєкт, який призначений для виконання операцій з даними
Метод класу	Функція, яка є членом класу

1	2
Об’єкт (object)	Базове поняття об’єктно-орієнтованого програмування, яке об’єднує дані та операції над ними
Об’єктний код (object code)	Код програми, отриманий як результат обробки її початкового тексту компілятором
Провайдер даних	Набір класів у просторі імен, створених спеціально для роботи з цим конкретним джерелом даних
Простір імен	Сукупність класів бібліотекі Fraimework
Пул	Область пам'яті, що тимчасово зберігає контекст з'єднання з базою даних
RM (resource manager)	Диспетчер ресурсів, що безпосередньо виконує роботу з базою даних і що повідомляє про успішне або невдале її завершення
Рівень ізоляції	Здатність застосування виконувати роботу з даними ізольовано від інших застосувань
Роз’єднане середовище	Середовище, в якому запрошувані дані переносяться в локальну пам'ять, де і відбувається робота з ними
Розподілена транзакція	Транзакція, що охоплює декілька об'єктів
Транзакція	Здійснення закінчених дій стосовно визначеного об'єкта, що переводить цей об'єкт з одного постійного стану в інший
XML (Extensible Markup Language)	Розширювальна мова розмітки – метамова форматування документів, яка дає змогу враховувати специфіку документів певної предметної сфери

Рекомендована література

1. Байдачный С.С. .NET Framework. Секреты создания Windows-приложений / С. С. Байдачный. – М. : СОЛОН-Пресс, 2004. – 496 с.

2. Методичні рекомендації до виконання лабораторних робіт з навчальної дисципліни "Організація баз даних і знань (ADO.NET)" для студентів напряму підготовки "Комп‘ютерні науки" денної форми навчання укл. М. Ю. Лосєв, О. В. Тарасов, В. В. Федько. – Харків : Вид. ХНЕУ, 2010. – 86 с.

3. Сеппа Д. Программирование на Microsoft ADO.NET. Мастер-класс / Д. Сеппа ; пер. с англ. – СПб. : Питер, 2007. – 764 с.

4. Троелсен Э. Язык программирования C# 2005 и платформа .Net 2.0 / Э. Троелсен ; пер. с англ. – М. : Издательский дом "Вильямс", 2007. – 1168 с.

5. Троелсен Э C# и платформа .Net / Э. Троелсен ; пер. с англ. – СПб. : Питер, 2007. – 796 с.

НАВЧАЛЬНЕ ВИДАННЯ

Лосєв Михайло Юрійович
Тарасов Олександр Васильович
Федько Віктор Васильович

ОРГАНІЗАЦІЯ БАЗ ДАНИХ І ЗНАНЬ (ADO.NET)

Конспект лекцій
для студентів напряму підготовки "Комп'ютерні науки"
денної форми навчання

Відповідальний за випуск **Пономаренко В. С.**

Відповідальний редактор **Сєдова Л. М.**

Редактор **Носач О. С.**

Коректор **Бриль В. О.**

План 2011 р. Поз. № 98-К.
Підп. до друку Формат 60 × 90 1/16. Папір MultiCopy. Друк Riso.
Ум.-друк. арк. 6,75. Обл.-вид. арк. 8,44. Тираж прим. Зам. №

Видавець і виготівник – видавництво ХНЕУ, 61001, м. Харків, пр. Леніна, 9а

Свідоцтво про внесення до Державного реєстру суб'єктів видавничої справи
Дк № 481 від 13.06.2001 р.

Лосєв М. Ю.
Тарасов О. В.
Федько В. В.

ОРГАНІЗАЦІЯ БАЗ ДАНИХ І ЗНАНЬ (ADO.NET)

Конспект лекцій