МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

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

СУЧАСНІ ЗАСОБИ ДОСТУПУ ДО ДАНИХ

Навчальний посібник
для самостійної роботи студентів
з навчальної дисципліни
"ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ"

Харків. Вид. ХНЕУ ім. С. Кузнеця, 2014

УДК 004.65(076)
ББК 32.973.26-018.2я73
Ф35

Рецензенти: докт. техн. наук, професор кафедри штучного інтелекту Харківського національного університету радіоелектроніки *Філатов В. О.;* канд. техн. наук, доцент кафедри вищої математики та інформатики, ст. наук. співробітник Харківського торговельно-економічного інституту КНТЕУ *Алісейко О. В.*

Рекомендовано до видання рішенням вченої ради Харківського національного економічного університету імені Семена Кузнеця.
Протокол № 9 від 22.04.2014 р.

Авторський колектив: канд. фіз.-мат. наук, доцент Федько В. В. – вступ, розділи 5, 8; канд. техн. наук, доцент Тарасов О. В. – розділи 6, 7; канд. техн. наук, доцент Лосєв М. Ю. – розділ 9.

Федько В. В.

Ф35 Сучасні засоби доступу до даних : навчальний посібник для самостійної роботи студентів з навчальної дисципліни "Організація баз даних та знань" для студентів напряму підготовки 6.050101 "Комп'ютерні науки" / В. В. Федько, О. В. Тарасов, М. Ю. Лосєв. – Х. : Вид. ХНЕУ ім. С. Кузнеця, 2014. – 328 с. (Укр. мов.)

Наведено навчальний матеріал для самостійної підготовки студентів у ході вивчення сучасних технологій доступу до даних ADO.NET. Розглянуто засоби побудови застосувань з базами даних на основі типізованих наборів даних, використання технології LINQ to DataSet у задачах аналізу даних, розроблення застосувань на платформі Entity Framework, використання збережених процедур у ADO.NET.

Рекомендовано для студентів напряму підготовки 6.050101 "Комп'ютерні науки".

ISBN 978-966-676-531-7

УДК 004.065(076)
ББК 32.973.26-018.2я73

Вступ

Характерною рисою переважної більшості програмних продуктів є використання даних, що зберігаються у базах даних. Тому технології доступу до даних стали важливою частиною розробки застосувань і є невід'ємним напрямом підготовки сучасних фахівців у галузі комп'ютерних наук. Ураховуючи те, що ефективність праці розробника значною мірою залежить від технологічності засобів, які використовуються під час створення застосувань, проблема опанування сучасних засобів доступу до даних набула актуальності протягом останніх років. Достатньо зазначити те, що за період підготовки даного навчального посібника двічі змінювалися версії платформи Entity Framework (від четвертої до шостої). Тому оволодіння сучасними та перспективними технологіями є важливою складовою у підготовці фахівців із розробки програмного забезпечення.

Даний навчальний посібник призначено, у першу чергу, для студентів напряму підготовки 6.050101 "Комп'ютерні науки" з метою вивчення та закріплення теоретичного матеріалу зі змістового модуля "Сучасні засоби доступу до даних" навчальної дисципліни "Організація баз даних та знань".

Навчальний посібник призначено для формування у студентів компетентностей, які забезпечуються отриманням концептуальних сучасних знань і вмінь розв'язувати складні непередбачувані задачі, створюванням у тих, хто навчається, спроможності донесення до фахівців власного досвіду, а також відповідальності за прийняття рішень у різноманітних умовах.

Навчальний посібник розроблено відповідно до програми навчальної дисципліни "Організація баз даних та знань" [4]. Він охоплює такі теми:

типізовані набори даних;

технологія LINQ to DataSet;

платформа Entity Framework;

технологія Code First;

використання збережених процедур у ADO.NET.

Опанування цих тем дозволяє студентам оволодіти вміннями в рамках компетентностей, визначених Галузевим стандартом вищої освіти України з напряму підготовки 6.050101 "Комп'ютерні науки" [2].

Компетентності соціально-особистісні:

вміння знаходити нові, нешаблонні рішення і засоби їх здійснення, діяти протягом тривалого часу, незважаючи на труднощі, проявляти гнучкість у подоланні перешкод;

здатність до генерації нових ідей і варіантів розв'язання задач, до комбінування та експериментування, до оригінальності, конструктивності, економічності та простих рішень;

вміння виявляти недоліки і помилки та виправляти їх, розв'язувати протиріччя.

Загальнонаукові компетентності:

вміння застосовувати базові знання стандартів у галузі інформаційних технологій під час розробки та впровадження інформаційних систем і технологій.

Інструментальні компетентності:

вміння обробляти отримані результати, аналізувати, осмислювати та подавати їх, обґрунтовувати запропоновані рішення на сучасному науково-технічному рівні.

Загальнопрофесійні компетентності:

підготовленість до розробки нових математичних методів, ефективних алгоритмів та методів реалізації функцій інформаційних систем і технологій у прикладних галузях;

здатність до програмної реалізації алгоритмів розв'язання задач, розробки прикладного програмного забезпечення інформаційних систем.

Спеціалізовано-професійні компетентності:

вміння використовувати основні поняття, ідеї та методи фундаментальної математики під час розв'язання конкретних задач у галузі комп'ютерних наук;

вміння розробляти та застосовувати ефективні алгоритми для розв'язання професійних завдань у галузі комп'ютерних наук;

вміння проектувати логічні та фізичні моделі баз даних, запити до них та використовувати різноманітні системи керування базами даних;

вміння моделювати системи та процеси, стани та поведінки складних об'єктів інформатизації в процесі розроблення інформаційних систем і технологій.

Навчальний посібник складається з 5 розділів і є продовженням матеріалу, який викладено у навчально-практичному посібнику [10]. Останній складається з 4 розділів. Тому нумерація розділів у даному посібнику продовжується, починаючи з п'ятого.

П'ятий розділ присвячено ознайомленню з перевагами типізованих наборів даних, способами їх створення, роботі з таблицями у вікні конструктора наборів даних, методам типізованих наборів даних, адаптерам таблиць, зв'язуванню з інтерфейсом користувача.

У кінці розділу подано опис лабораторної роботи, виконання якої сприятиме набуттю студентами початкових вмінь зі створення типізованих наборів даних, виробленню вмінь зміни властивостей об'єктів у типізованих наборах даних, набуттю практичних навичок щодо використання адаптерів таблиць та відображення даних у застосуваннях на основі типізованих наборів даних.

У шостому розділі розглянуто технологію LINQ to DataSet, призначення й переваги технології LINQ, її види, зокрема технологію LINQ to DataSet і запити, поняття виразу запиту та речення запиту, особливості синтаксису методів, відкладені й негайні операції, застосування технології LINQ до типізованих DataSet.

У кінці розділу подано опис лабораторної роботи, в якій передбачено оволодіння засобами розробки програм на основі технології LINQ to DataSet, набуття практичних навичок формування LINQ-запитів до DataSet, вироблення вмінь та навичок групування даних і обчислення агрегованих величин у технології LINQ to DataSet, опанування операцій об'єднання даних кількох таблиць та створення підзапитів, набуття практичних навичок побудови діаграм із використанням технології LINQ to DataSet.

Сьомий розділ присвячено вивченню платформи Entity Framework, її призначенню, зв'язку моделі й основних понять у Entity Framework, сценаріям створення моделі EDM, реалізації операцій CRUD, засобам LINQ to Entities, відображення даних моделі в інтерфейсі користувача.

У кінці розділу подано опис лабораторної роботи, в якій передбачено оволодіння засобами розробки застосувань на основі платформи Entity Framework, набуття практичних навичок створення концептуальних моделей, освоєння сценаріїв Model First і DB First, оволодіння технологі-

єю LINQ to Entities, набуття практичних навичок відображення даних у застосуваннях на основі платформи Entity Framework.

У восьмому розділі розглянуто основні поняття технології Code First, її призначення, та переваги, класи сутностей, об'єкти доступу до даних, бібліотеки EntityFramework, засоби створення бази даних, домовленості в Code First, налаштування моделі даних засобами Data Annotations та Fluent API, вдосконалення моделі (міграції), побудова застосування з використанням технології Code First.

У кінці розділу подано опис лабораторної роботи, виконання якої сприятиме набуттю практичних навичок реалізації доступу до даних на основі технології Code First, побудови класів сутностей та класу DbContext, освоєнню засобів Data Annotations та Fluent API для налаштування сутностей, оволодінню засобами вдосконалення моделі на основі міграцій, набуттю практичних навичок побудови застосувань Windows Forms на основі технології Code First.

Дев'ятий розділ присвячений вивченню засобів збережених процедур у ADO.NET, створенню збережених процедур у ADO.NET та в ADO.NETEntityFramework, викликанню збережених процедур, параметрам збережених процедур, особливостям оновлення даних із використанням збережених процедур, перевагам та недоліки використання збережених процедур, особливостям використання збережених процедур для контролю конфліктів паралельної обробки даних.

У кінці розділу подано опис лабораторної роботи, в якій передбачено оволодіння засобами розробки програм взаємодії з базами даних із використанням збережених процедур, набуття практичних навичок використання концептуальних моделей і створення збережених процедур за допомогою платформи Entity Framework, оволодіння засобами обробки конфліктів паралелізму на основі платформи Entity Framework, удосконалення практичних навичок відображення даних у застосуваннях на основі платформи Entity Framework.

Вивчення матеріалу навчального посібника повинно відбуватися у такому порядку: спочатку слід прочитати матеріал, який подано у параграфі, потім спробувати відповісти на запитання і виконати завдання, що розміщені в кінці параграфа. Після ознайомлення з теоретичним матеріалом розділу виконати лабораторну роботу.

5. Типізовані набори даних

5.1. Переваги типізованих наборів даних

Після вивчення основ доступу до даних за допомогою технології ADO.NET у подальшому слід розглянути удосконалення цієї технології, що дасть змогу швидше і якісніше створювати різноманітні програми обробки даних. Першим із таких удосконалень є типізовані набори даних, які з'явилися в ADO.NET 2.0 (Visual Studio 2002) як новий рівень інкапсуляції для даних і їхніх схем. Вони, зокрема, дали можливість створювати застосування візуальними засобами майже без використання "ручного" написання коду, а в разі їхнього вживання значно полегшилося безпомилкове написання коду за рахунок використання IntelliSense.

Типізований набір даних – це сукупність класів, що успадковують і розширюють можливості класів DataSet, DataTable і DataRow. Вони надають додаткові властивості, методи й події на базі схеми класу DataSet.

Типізований DataSet відрізняється від нетипізованого в першу чергу тим, що під час його створення таблиці зразу отримують схеми. Причому схеми таблиць відомі заздалегідь – до початку виконання програми. Тому багато помилок відшукуються ще на етапі компіляції (наприклад, стовпцю присвоюють дані не підходящого типу). У свою чергу, нетипізовані набори даних мають переваги в задачах, коли схеми таблиць змінюються динамічно в процесі виконання програми.

Типізований DataSet містить крім класу, який розширює клас DataSet, ще по три класи на кожну таблицю в наборі даних. Вони розширюють класи DataTable, DataRow і DataRowChangeEvent.

Під час створення типізованого набору даних Visual Studio сама надає імена об'єктам за таким правилом: спочатку префіксом йде назва об'єкта у базі даних, а потім – його тип. Зокрема, за ім'ям бази даних створюється ім'я DataSet (наприклад, хлібСерверDataSet), а таблиці в ній – ім'я локальної таблиці (наприклад, товариTable). Причому імена об'єктів починаються з малої літери, а відповідних класів – з великої. Така сувора дисципліна іменування об'єктів значно полегшує читання і розуміння коду програми.

Найважливішими перевагами типізованого набору даних є такі:

містить у собі схему даних, що забезпечує більш високу продуктивність порівняно із завантаженням схеми з бази даних (але її можна задати програмним способом);

програмування є більш інтуїтивним, а код виявляється простішим в обслуговуванні. Імена таблиць, стовпців та інших об'єктів доступні через властивості, а не через використання індексів або рядків із роздільниками (розширює можливості IntelliSense й автодоповнення);

надає доступ до значень, що мають правильний тип під час компіляції. Тому помилки невідповідності типів виявляються вже під час компіляції коду, а не під час виконання;

швидка розробка інтерфейсу користувача методом drag-and-drop.

Приклад 5.1. Звертання до об'єктів.

```
decimal decimalЦіна;

// Нетипізований DataSet
decimalЦіна  =(decimal)ds.Tables["Товари"].Rows[0]["Ціна"];

// Типизированый DataSet
decimalЦіна = хлібСерверDataSet.Товари[0].Ціна;
```

Запитання і завдання

1. Яке призначення має типізований набір даних?
2. Які основні переваги має типізований набір даних перед нетипізованим? Перерахуйте і наведіть приклади їхнього використання.
3. У яких випадках краще користуватися нетипізованим набором даних? Обґрунтуйте відповідь.

5.2. Способи створення типізованих наборів даних

Для створення типізованого набору даних найчастіше використовують такі візуальні засоби:

майстер настроювання джерела даних;

конструктор наборів даних.

Слід розглянути докладніше ці способи.

Щоб створити типізований набір даних потрібно виконати таке:

1. Запускають майстра настроювання джерела даних вибором команди **Добавить новый источник данных**, що знаходиться в меню **Данные**.

2. Вибирають значок **База данных** у вікні **Выбор типа источника данных** (рис. 5.1).

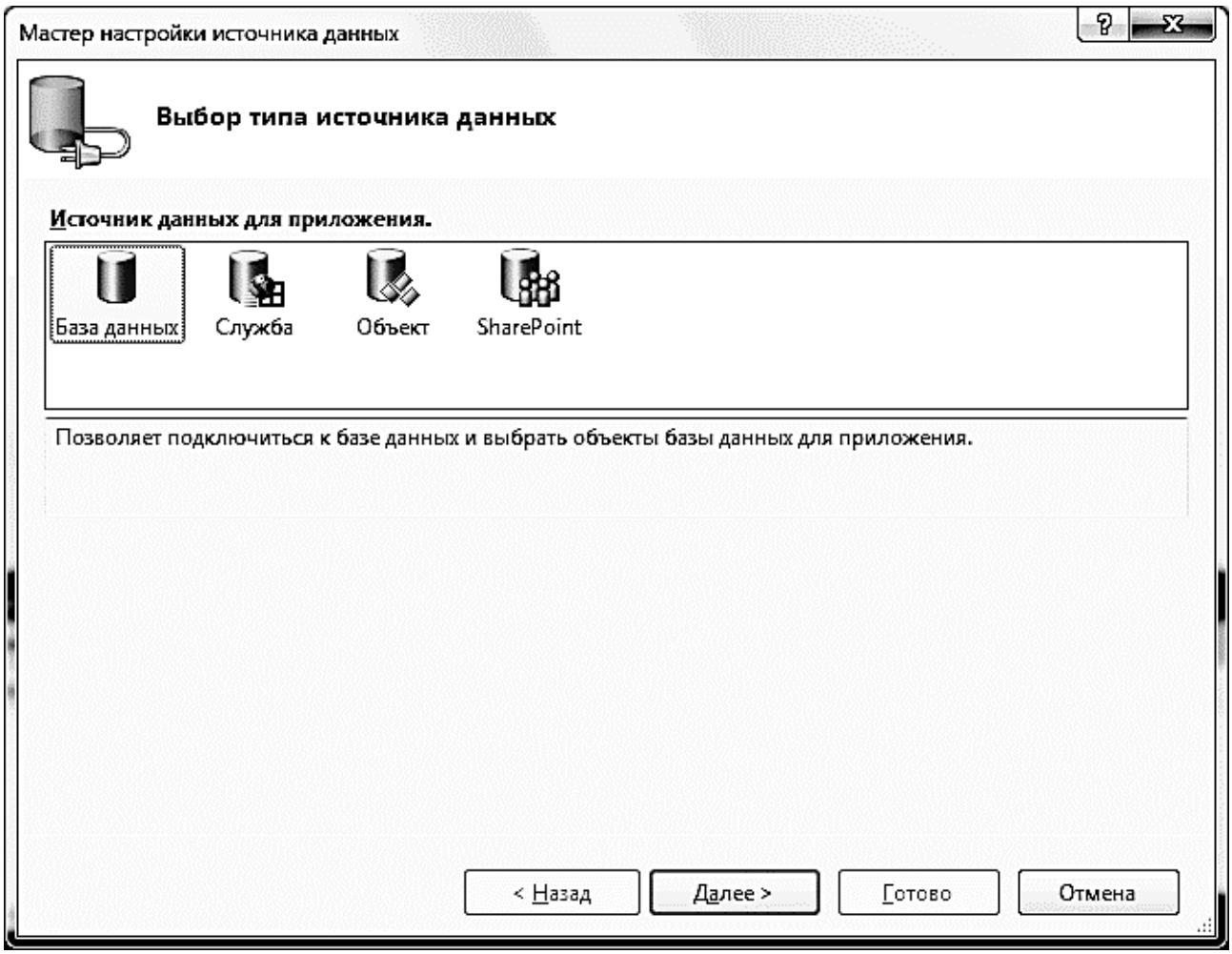


Рис. 5.1. **Вікно *Выбор типа источника данных***

3. Далі виконують ті самі дії, що й під час створення підключення до бази даних [10].

У результаті роботи майстра окрім файла ***app.config***, який розглядався в [10], у вікні оглядача розв'язків додався компонент з ім'ям ***База_данихDataSet.xsd*** (рис. 5.2). Це XML-файл, у якому зберігається схема набору даних.

Якщо відкрити файл ***База_данихDataSet.xsd*** у текстовому редакторі XML, то з'явиться вікно, що подано на рис. 5.3. У ньому можна змінювати схему набору даних, але на початковому етапі роботи з типізованими наборами даних не рекомендується це робити.

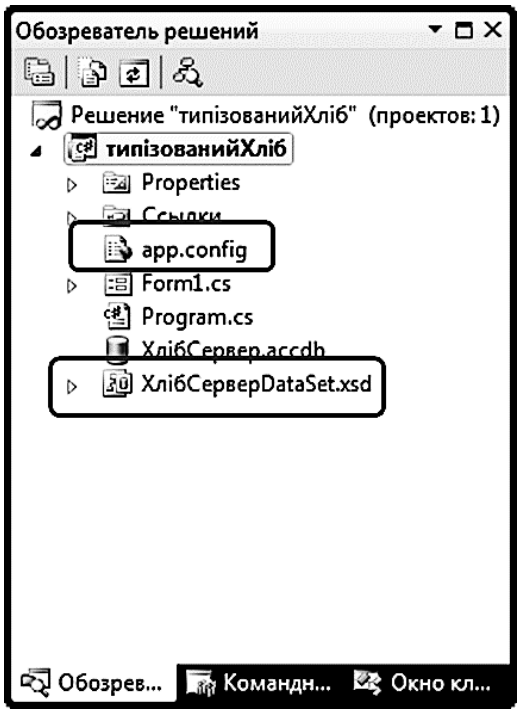


Рис. 5.2. **Компоненти, що створені майстром настроювання джерела даних**

Рис. 5.3. **Файл *ХлібСерверDataSet.xsd* у текстовому редакторі XML**

Для перегляду концептуальної схеми і її редагування краще відкрити файл ***База_данихDataSet.xsd*** подвійним клацанням миші на його значку. У цьому разі він відображається у конструкторі набору даних (рис. 5.4). У ньому можна змінити схему набору даних, додаючи або змінюючи таблиці даних і зв'язки, адаптери таблиць і запити до них.

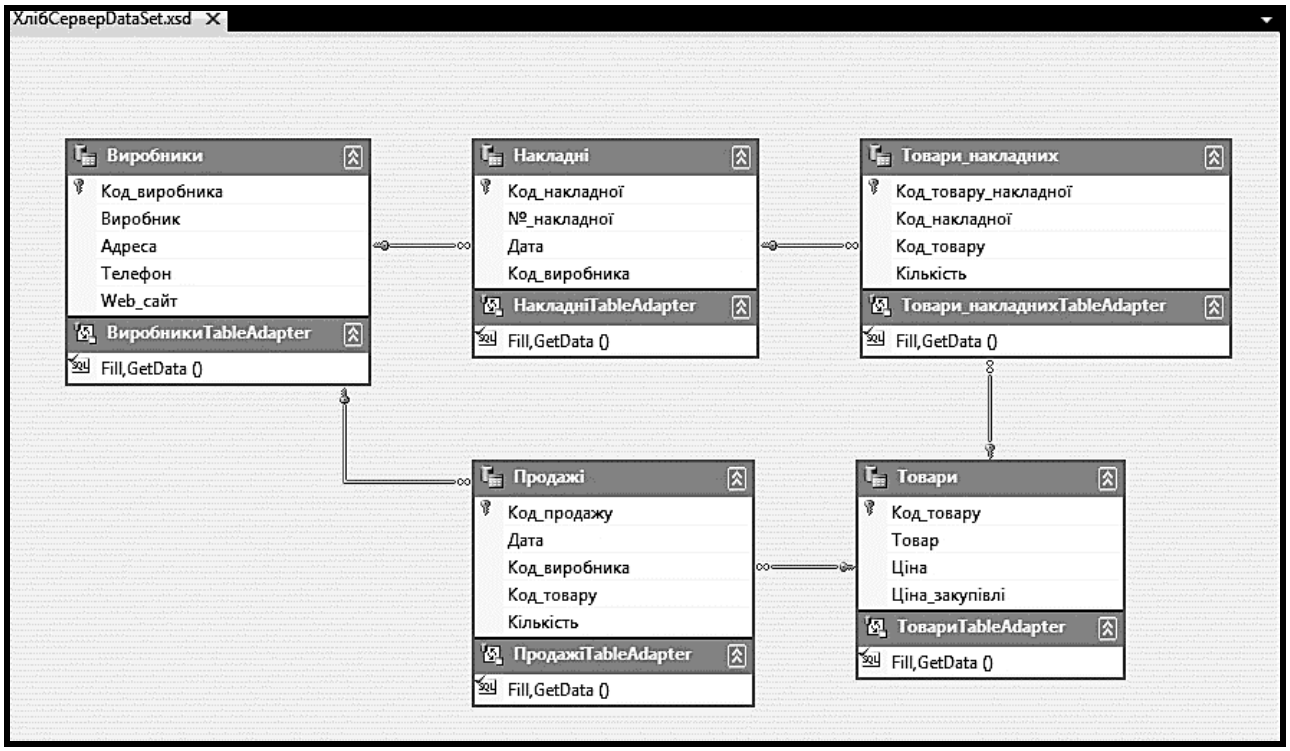


Рис. 5.4. **Схема набору даних у вікні конструктора**

Розглянутий вище метод є достатньо простим. Його рекомендують вживати на початковому рівні знайомства з технологією типізованих наборів даних. У ньому спочатку створюються локальні таблиці на основі відповідних таблиць бази даних, а потім можна виконати їхнє налаштування згідно з тими задачами, що будуть вирішуватися у застосуванні.

Більш складним вважається метод створення набору даних за допомогою однойменного конструктора. Під час його застосування відразу можна враховувати специфіку задач. Щоб побудувати типізований набір даних цим методом, потрібно виконати таке:

1. Вибирають команду **Добавить новый элемент** у меню **Проект**.
2. Вибирають значок **Набор данных**, що знаходиться в категорії **Данные** вікна **Добавление нового элемента** (рис. 5.5), а потім вводять ім'я набору даних (щоб не порушувати дисципліну надання імен, краще дотримуватися правил іменування у типізованих наборах даних).

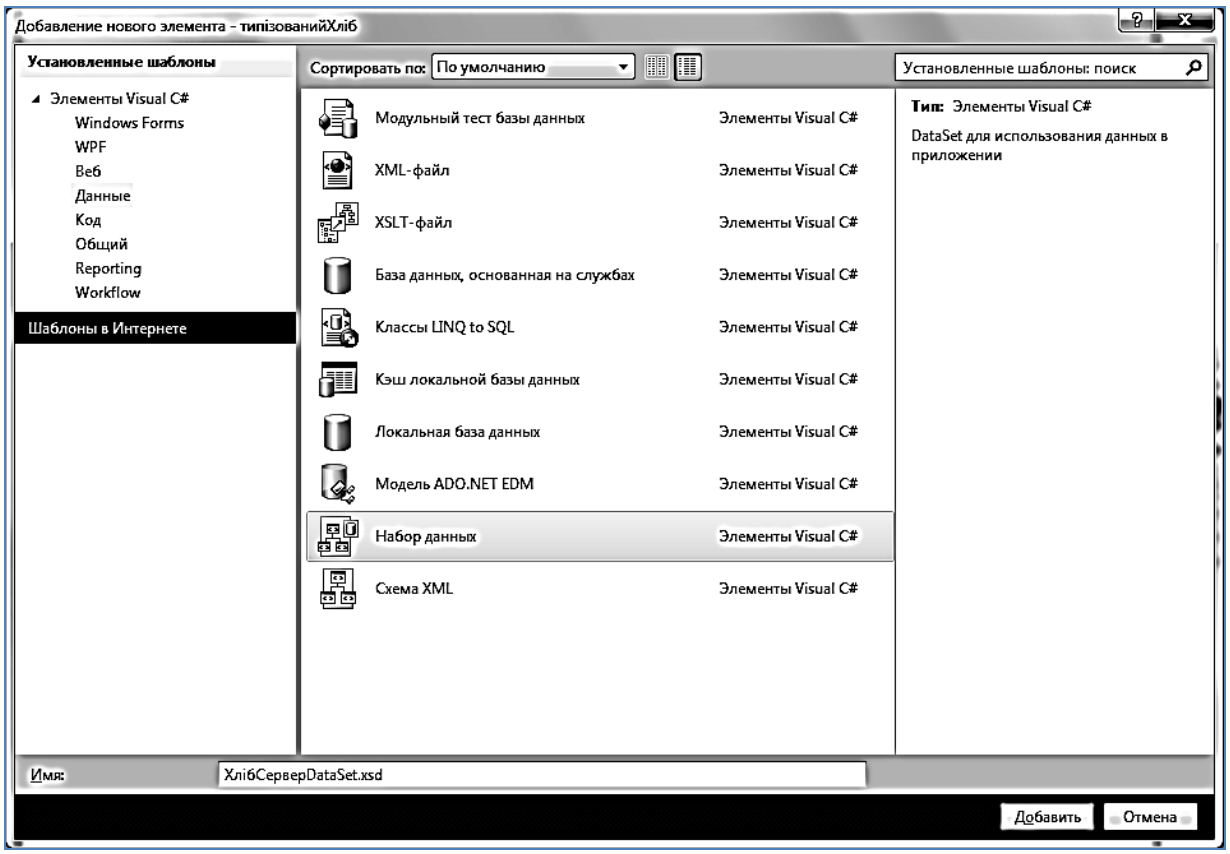


Рис. 5.5. **Вибір значка *Набор данных* і введення імені**

Після натискання кнопки **Добавить** набір даних додається в проект і з'являється порожнє вікно конструктора наборів даних (рис. 5.6).

Рис. 5.6. **Порожнє вікно конструктора наборів даних**

3. Приєднують базу даних до Visual Studio з використанням контекстового меню елемента **Подключения данных** у вікні **Обозреватель серверов** (рис. 5.7).

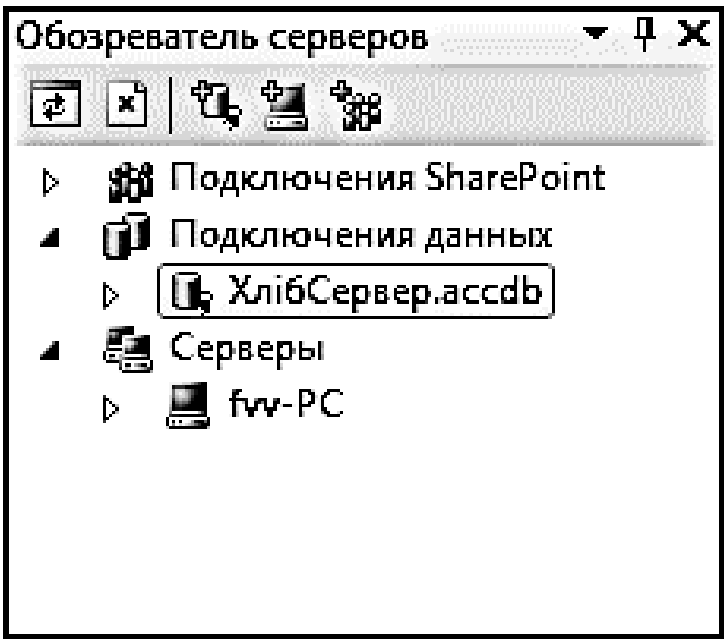


Рис. 5.7. **Приєднана база даних у вікні *Обозреватель серверов***

4. Розкривають вузол приєднаної бази даних і перетягують з нього у вікно конструктора наборів даних потрібні елементи.

Примітки. 1. Після перетягування першого елемента у вікні **Обозреватель решений** з'являється значок файла ***app.config***.

2. Якщо перетягують пов'язані таблиці, у вікні конструктора відображаються лінії їхнього зв'язку.

Запитання і завдання

1. Які способи створення типізованих наборів даних ви знаєте? Опишіть сфери застосування кожного з них.

2. Які вікна Visual Studio використовують під час створення типізованих наборів даних? Перерахуйте їх і опишіть їхнє призначення у процесі створення.

3. На яких етапах вказують ім'я типізованого набору даних у різних способах створення? Яких правил потрібно дотримуватися при цьому?

4. Які операції можна виконувати у вікні конструктора наборів даних? Перерахуйте їх.

5.3. Робота з таблицями у вікні конструктора наборів даних

Після того, як створено набір даних, у вікні конструктора можна виконувати такі операції з локальними таблицями:

додавати стовпці в об'єкт DataTable (у тому числі з автоматичною генерацією значення);

змінювати властивості для об'єкта DataColumn (ім'я, заголовок, тип даних значення за замовчуванням, обмеження унікальності значень, встановлення первинного ключа, обчислюване значення тощо);

редагувати зв'язки між таблицями.

Слід розглянути їх детальніше.

Щоб додати стовпець в об'єкт DataTable, треба виконати таке:

1. Відкрити вікно конструктора наборів даних подвійним клацанням на значку набору даних, що розташований у вікні **Обозреватель решений**.

2. Клацути правою кнопкою миші у верхній частині таблиці, до якої потрібно додати стовпець, і з контекстового меню вибрати команду **Добавить – Столбец** (рис. 5.8).

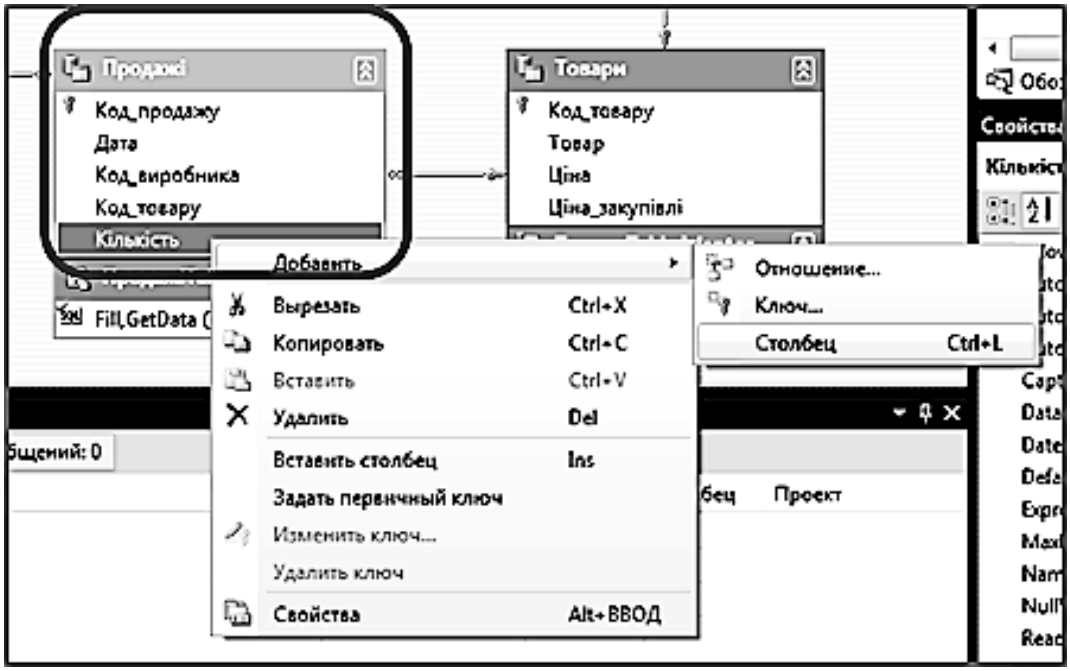


Рис. 5.8. **Додавання стовпця до таблиці *Продажі***

3. Встановити потрібні значення у вікні властивостей стовпця.

Щоб змінити властивості якогось стовпця, його виділяють в таблиці вікна конструктора набору даних і встановлюють (вводять чи вибирають

із списку) нове значення властивості у вікні властивостей. Найчастіше змінюють значення властивостей, що подані в табл. 5.1.

Таблиця 5.1

Основні властивості класу DataTable

Властивість	Опис
Name	Ім'я стовпця для посилань на нього. Задають за правилами для ідентифікаторів (заборонені пробіли)
Caption	Заголовок стовпця. Відображається в елементах керування (наприклад, в DataGridView)
DataType	Тип даних, що зберігаються у стовпці. Вибирають зі списку значень (Boolean, DateTime, Decimal, Int32, String тощо)
DefaultValue	Значення за замовчуванням. Задають константу відповідного типу. Змінюване початкове значення задають у коді
Unique	Обмеження унікальності значень. Вибирають зі списку знчень Fase чи True
Expression	Обчислюване значення. Задають у вигляді виразу. Напрклад, для обчислення вартості товару в таблиці Продажі вказують вираз Parent(ТовариПродажі).Ціна*Кількість
AutoIncrement	Автоматичне генерування номерів (як у типі Лічильник в Access). Вибирають зі списку значень Fase чи True
AutoIncrementSeed	Початкове значення для автоматичного генерування номерів, якщо AutoIncrement дорівнює True
AutoIncrementStep	Величина кроку під час автоматичного генерування номерів, якщо AutoIncrement дорівнює True

Щоб встановити первинний ключ таблиці, клацають правою кнопкою миші на потрібному полі і з контекстового меню вибирають команду **Добавить – Ключ**.

Щоб встановити зв'язок між таблицями, перетягують мишею поле первинного ключа з батьківської таблиці на відповідне поле зовнішнього кюча в дочірній таблиці. У вікні **Отношение**, що відкрилося після перетягування поля, можна встановити правила оновлення і видалення рядків у батьківській таблиці.

Властивості відношення можна змінити у вікні **Отношение**. Його викликають подвійним клацанням на лінії, що пов'язує дві таблиці, або за

допомогою контекстового меню. Зокрема, тут змінюють ім'я відношення після того, як набір даних створено майстром настроювання джерела даних, оскільки майстер надає відношенню числове ім'я, з яким важко оперувати.

Запитання і завдання

1. Які операції з локальними таблицями можна виконувати у вікні конструктора? Перерахуйте їх.

2. Яким способом можна задати поточну дату як початкове значення поля ***Дата*** у таблиці ***Продажі***? Опишіть цей спосіб і перевірте його.

3. Які властивості відношення можна змінити у вікні **Отношение**. Перерахуйте їх, відкривши це вікно.

5.4. Методи типізованих наборів даних

Оскільки типізований клас DataSet є похідним від нетипізованого класу DataSet, він успадковує всі методи, події й властивості батьківського класу. Крім того, типізований DataSet надає типізовані методи. Про їхні назви можна дізнатися за допомогою IntelliSense відповідної таблиці. Слід розглянути деякі з них.

Щоб додати запис до таблиці застосовують методи **New*Ім'яТаблиці*Row** та **Add*Ім'яТаблиці*Row**.

Приклад 5.2. Додавання нового запису до типізованої таблиці ***Товари***.

```
ХлібСерверDataSet.ТовариRow row =
    хлібСерверDataSet.Товари.NewТовариRow();
row.Товар= "Пиріг";
//… Установлення значень для інших полів
хлібСерверDataSet.Товари.AddТовариRow(row);
```

або коротше

```
хлібСерверDataSet.Товари.AddТоварыRow("Пиріг", 3, 2)
```

Для пошуку запису за ключем застосовують метод **FindBy*Ключ***.

Приклад 5.3. Пошук запису з кодом товару ***2*** в типізованій таблиці ***Товари***.

```
ХлібСерверDataSet.ТовариRow row =
    хлібСерверDataSet.Товари.NewТовариRow();
row = хлібСерверDataSet.Товари.FindByКод_товару(2);
if (row == null)
      MessageBox.Show("Товар не знайдено", "Пошук");
else
      MessageBox.Show("Товар: " + row.Товар + " Ціна:" + row.Ціна);
```

Приклад 5.4. Редагування ціни для другого запису в типізованій таблиці ***Товари***.

```
ХлібСерверDataSet.ТовариRow row = хлібСерверDataSet.Товари[1];
row.Ціна= 1.5M
```

Запитання і завдання

1. Які основні переваги мають методи типізованих наборів даних перед нетипізованими? Перерахуйте і наведіть приклади їхнього використання.

2. Які методи генерує Visual Studio для типізованих наборів даних? Дізнайтеся їхні імена за допомогою засобу IntelliSense.

5.5. Адаптери таблиць

Під час створення типізованого набору даних Visual Studio генерує для кожної таблиці свій адаптер даних. У конструкторі наборів даних він займає нижню частину кожної таблиці (рис. 5.9).

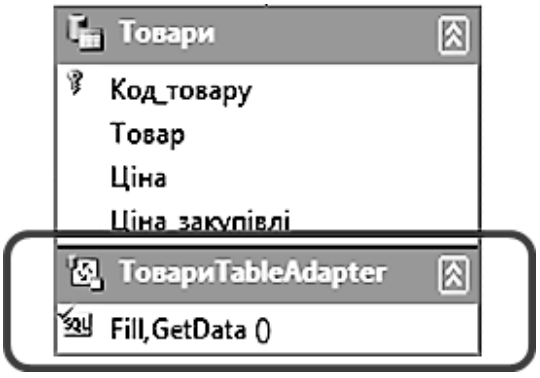


Рис. 5.9. **Адаптер таблиці *Товари***

Адаптер таблиці (TableAdapter) забезпечує зв'язок між застосуванням і базою даних. Він підключається до бази даних, виконує запити

або збережені процедури й або повертає нову заповнену таблицю даних, або заповнює існуючу локальну таблицю даними, що повертаються.

Адаптер таблиці також використовується для відправлення оновлених даних із застосування назад у базу даних.

На додаток до стандартних функціональних можливостей адаптери таблиць надають свої типізовані методи (Delete, Insert тощо). Вони інкапсулюють запити, що спільно використовують загальну схему з поєднаною типізованою таблицею. Про їхні назви можна дізнатися за допомогою IntelliSense відповідної таблиці. Найчастіше використовують такі методи адаптера таблиць:

Fill – заповнює таблицю, що пов'язана з адаптером, результатами виконання команди SELECT, які задані в адаптері;

Update – відправляє зміни назад у базу даних;

GetData – повертає нову локальну таблицю, що заповнена даними.

Методи Fill і GetData відображаються в нижній частині типізованої таблиці в конструкторі наборів даних (рис. 5.9). Тут вони подані відповідним оператором SELECT. За його допомогою відбираються дані для заповнення таблиці.

Під час створення типізованої таблиці автоматично генеруються оператори INSERT, DELETE та UPDATE для збереження змін у базі даних.

Усі ці оператори можна переглянути у вікні властивостей адаптера таблиці (рис. 5.10). Тут їх також можна змінити.

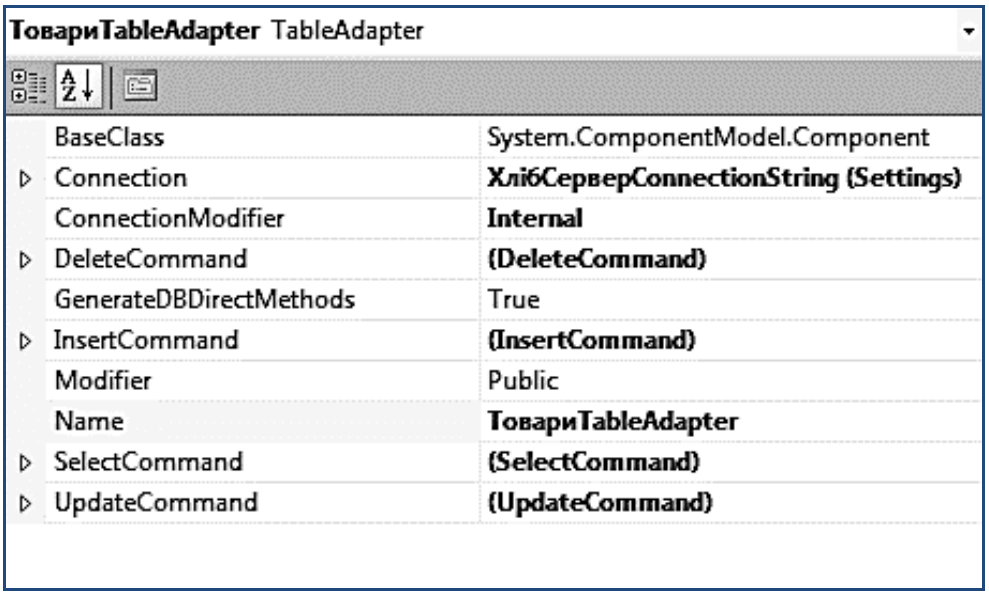


Рис. 5.10. **Вікно властивостей адаптера таблиці *Товари***

Адаптер таблиці може містити кілька запитів для заповнення пов'язаної таблиці. Кожний запит повертає дані, які відповідають тій самій схемі, що й пов'язана таблиця даних. Це дозволяє завантажувати дані, які задовольняють різним критеріям.

Наприклад, для заповнення таблиці ***Товари*** даними про товари з ціною понад ***2 грн*** створюють відповідні методи **Fill** і **GetData**:

1. У контекстовому меню адаптера вибрають команду **Добавить запрос**.

2. Вибирають перемикач **Инструкция SELECT, возвращающая строки**.

3. Створюють інструкцію SELECT, наприклад

```
SELECT Код_товару, Товар, Ціна, Ціна_закупівлі FROM Товари
WHERE Ціна > 2
```

4. Задають імена методів FillByЦіна та GetDataByЦіна.

У нижній частині адаптера таблиці з'являються імена нових методів (рис. 5.11).

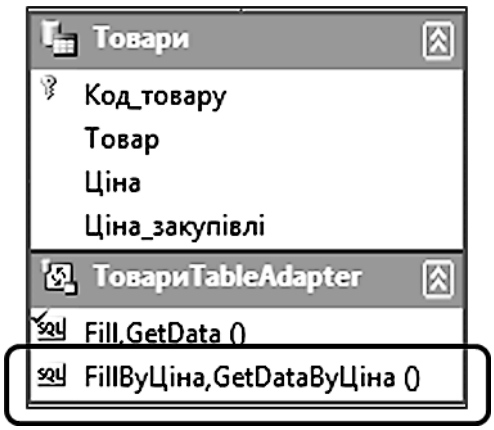


Рис. 5.11. **Імена нових методів в адаптері таблиці**

Метод викликають так:

```
товариTableAdapter.FillByЦіна(хлібСерверDataSet.Товари);
```

Запитання і завдання

1. Які основні переваги мають адаптери таблиць у типізованих наборах даних перед адаптерами даних у нетипізованих наборах даних? Перерахуйте і наведіть прилади їхнього використання.

2. Чи використовують об'єкти CommandBuilder для типізованих наборів даних? Обґрунтуйте відповідь.

3. У чому полягає різниця між методами Fill і GetData?

5.6. Зв'язування з інтерфейсом користувача

Схему локальних таблиць, що створилися під час побудови типізованого набору даних, можна переглядати не тільки у вікні конструктора наборів даних, але й у вікні **Источники данных** (рис. 5.12).

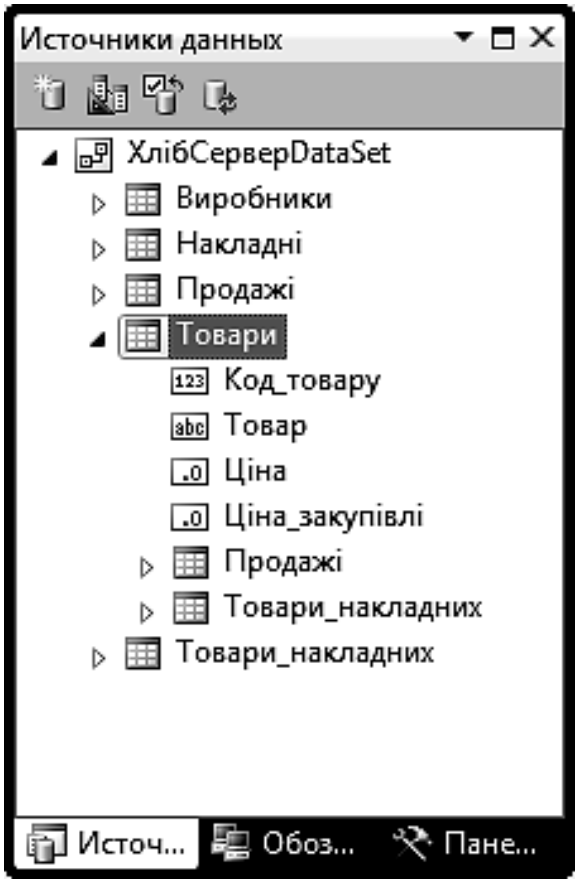


Рис. 5.12. **Вікно *Источники данных***

Кожна таблиця набору даних тут подана вузлом у вигляді значка. Елементами вузла є імена полів. Якщо таблиця має пов'язану дочірню таблицю, то остання відображається у вигляді підвузла. На рис. 5.12 у розкритому вузлі таблиці ***Товари*** відображаються її поля та дві дочірні таблиці – ***Продажі*** і ***Товари_накладних***.

Об'єкти вікна **Источники данных** найчастіше використовують для відображення відповідних даних у застосуванні. Для цього достатньо пе-

ретягнути елемент на форму. Слід розглянути більш детально побудову інтерфейсу візуальними засобами, які надає вікно **Источники данных**.

Якщо перетягнути вузол таблиці на форму, на останній з'являється елемент керування DataGridView для відображення даних відповідної таблиці (рис. 5.13). Над ним розміщається елемент керування BindingNavigator. Його використовують для переходів записами, а також виконання операцій з ними (додавання, вилучення, збереження). Кожний з цих елементів можна налаштовувати.

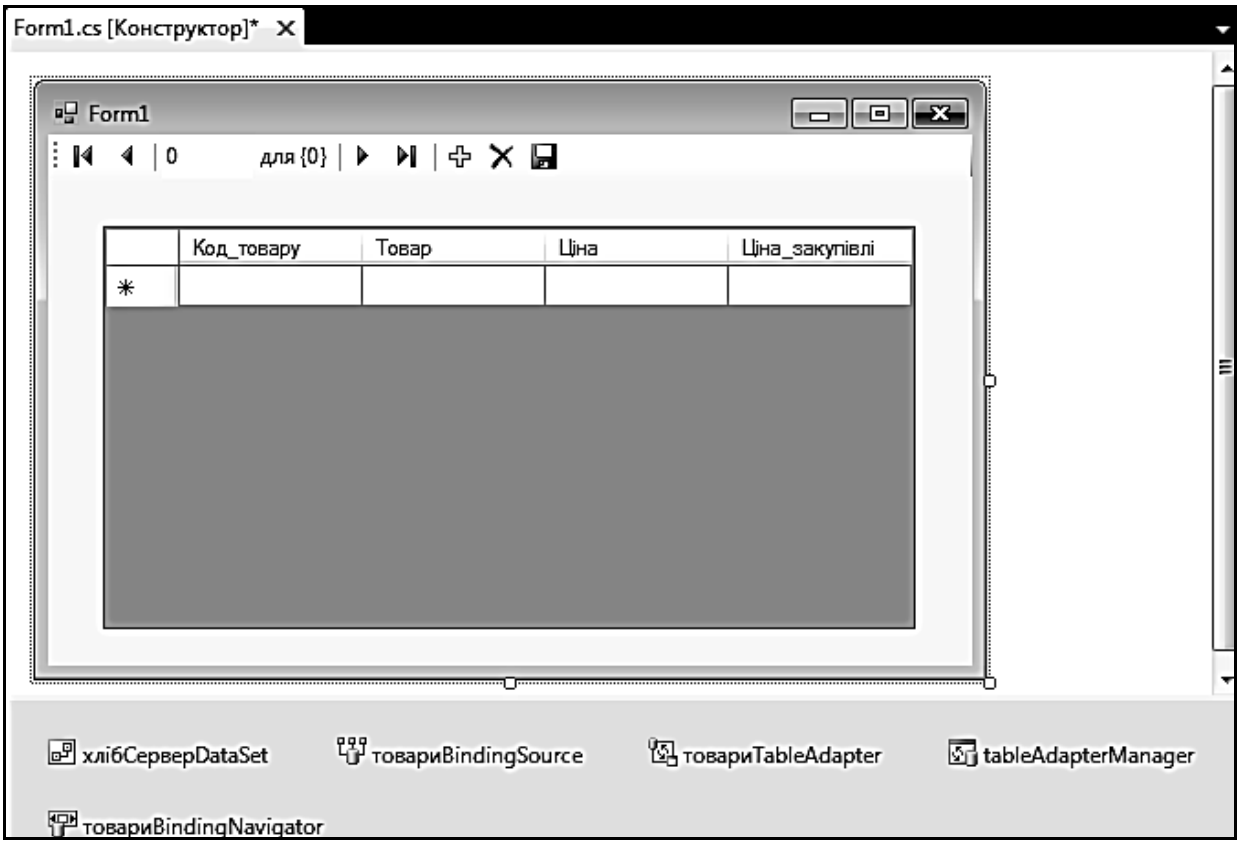


Рис. 5.13. **Результат перетягування вузла *Товари* на форму**

У нижній частині вікна конструктора форми (область компонентів) з'явилося п'ять об'єктів. Вони мають таке призначення:

***ім'яБазиДаних*DataSet** – типізований набір даних;

***ім'яТаблиці*BindingSource –** типізований з'єднувач із перетягнутою таблицею;

***ім'яТаблиці*TableAdapter** – адаптер перетягнутої таблиці;

tableAdapterManager – координатор дій адаптерів таблиць під час ієрархічного збереження;

***ім'яТаблиці*BindingNavigator** – подає елемент керування BindingNavigator.

На рис. 5.14 подано запущену на виконання форму.

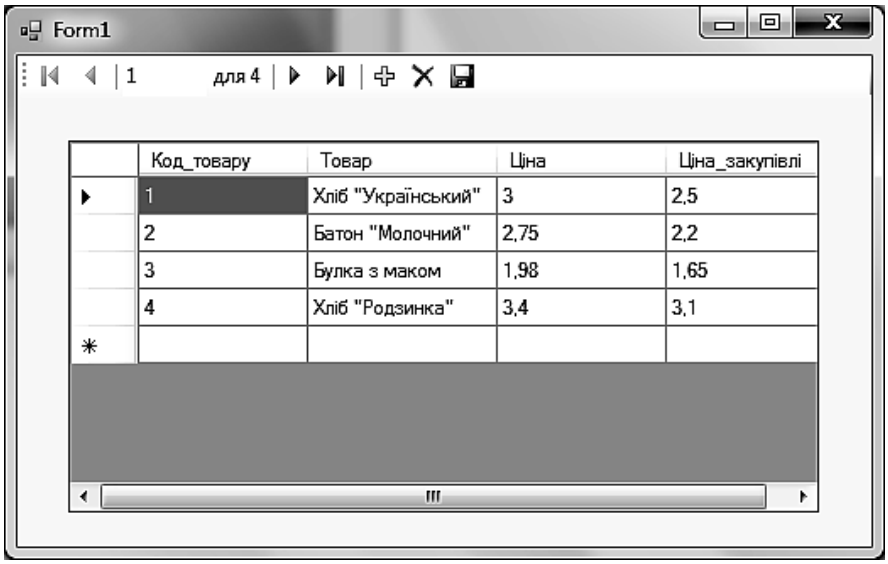


Рис. 5.14. **Форма під час виконання**

За допомогою елемента DataGridView відразу відображається кілька записів таблиці (в Access така форма називається табличною).

Можна налаштувати вузол так, щоб одночасно відображалися дані тільки одного запису, а перехід до інших записів відбувався по черзі або за номером таблиці (в Access така форма називається "в один стовпець"). Для цього у списку вузла потрібно вибрати елемент **Таблица** (рис. 5.15).

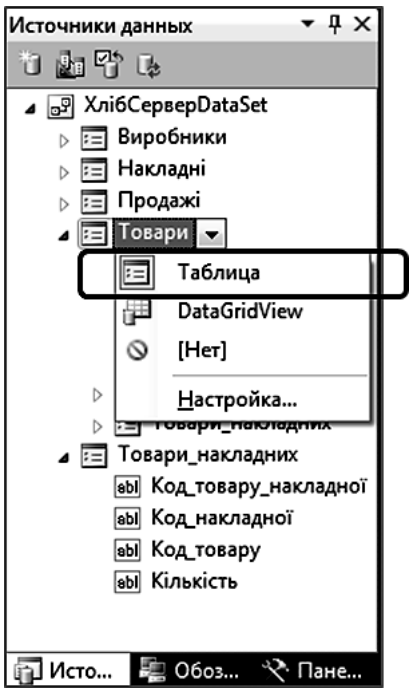


Рис. 5.15. **Встановлення режиму відображення *Таблиця***

На рис. 5.16 подано форму у вікні конструктора з елементами керування у режимі **Таблиця**, а на рис. 5.17 – цю саму форму під час виконання.

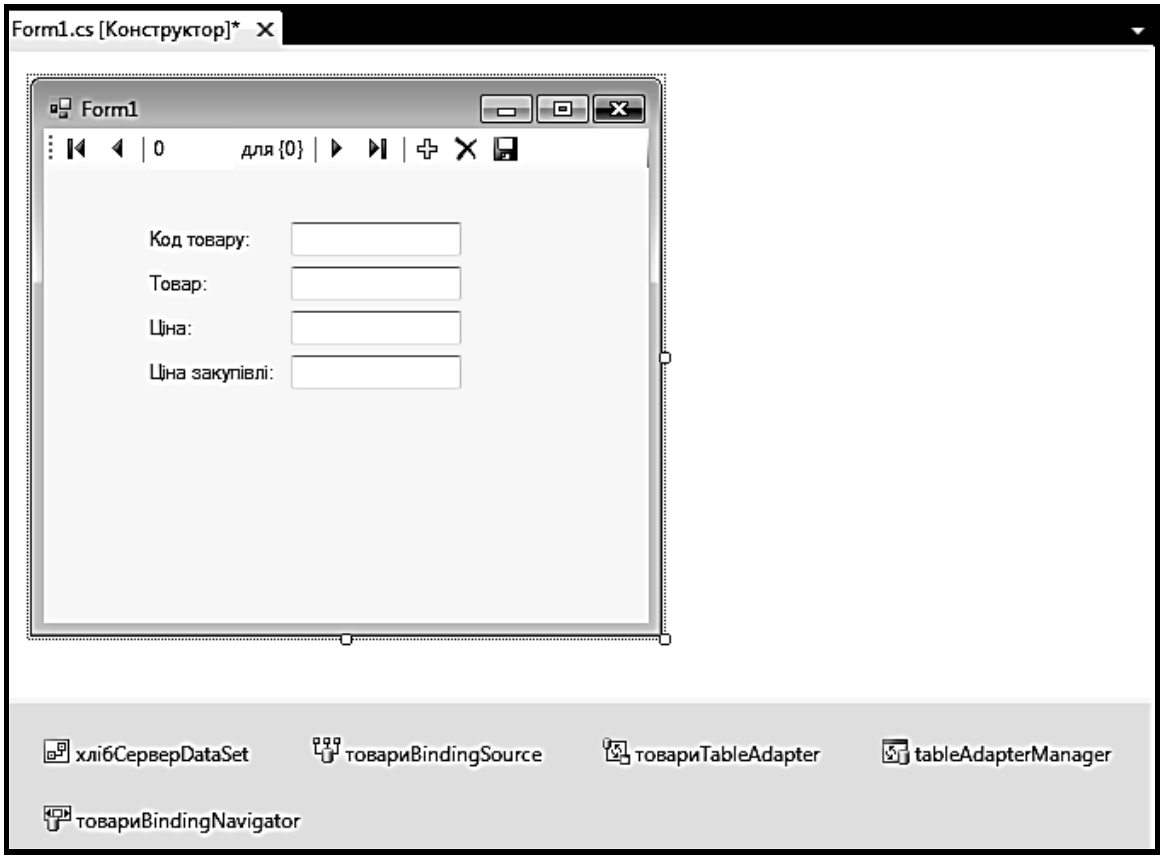


Рис. 5.16. **Форма у вікні конструктора з елементами керування у режимі *Таблица***

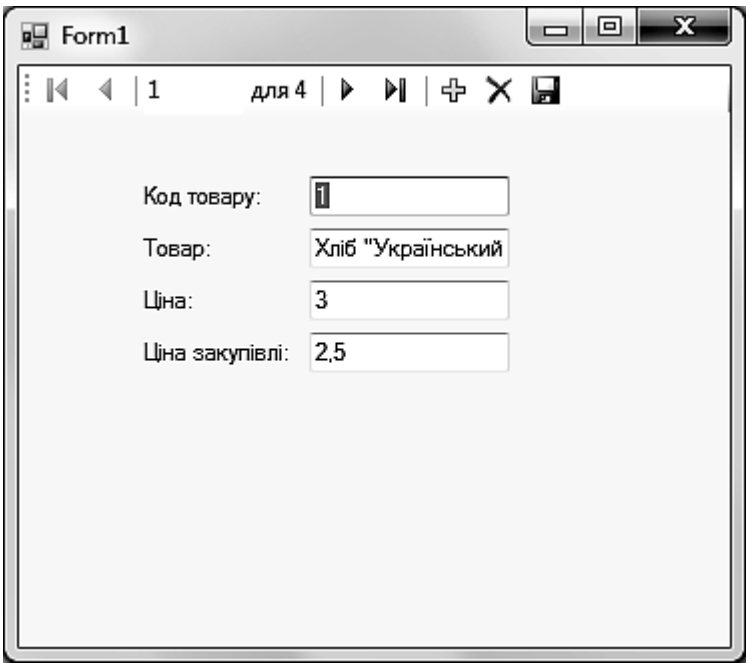


Рис. 5.17. **Виконання форми, що створена у режимі *Таблица***

Подібно до всієї форми можна підібрати елемент керування для відображення кожного поля таблиці після перетягування його з вікна **Источники данных** на форму. На рис. 5.18 зображено форму ***Продажі***, на якій поля ***Код_товару*** та ***Код_виробника*** відображаються як поля підстановки (поля зі списком ***Виробник***і ***Товар***).

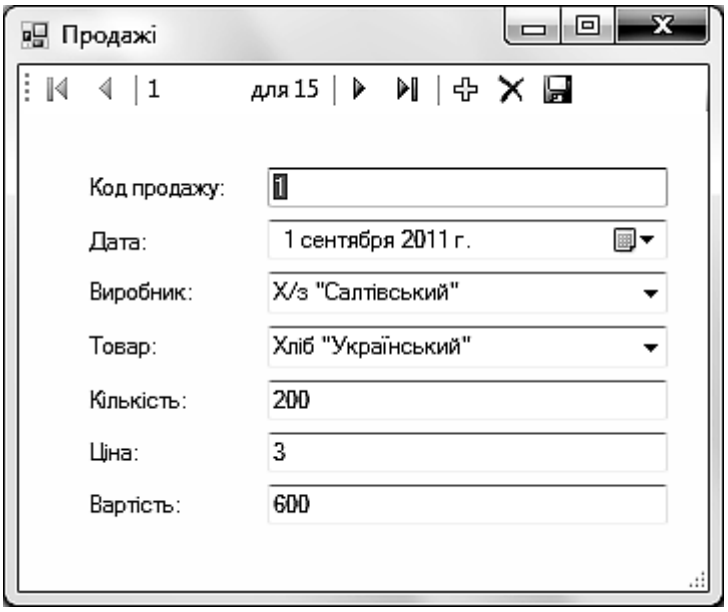


Рис. 5.18. Форма ***Продажі***

Якщо на форму перетягнути підвузол якогось вузла, то на формі з'явиться елемент керування DataGridView. У ньому відображаються записи дочірньої таблиці, що пов'язані з поточним записом батьківської таблиці, який відображається на формі. На рис. 5.19 зображено форму ***Накладна***, на якій у верхній частині відображається запис таблиці ***Накладні***, а в нижній – пов'язані з ним записи з таблиці ***Товари_накладних***. Перетягування підвузла реалізує так званий інтерфейс "master-detail", в якому подаються дані таблиць, що пов'язані співвідношенням "один-до-багатьох".

Примітка. Якщо замість підвузла таблиці ***Товари_накладних*** перетягнути на форму однойменний вузол, в елементі керування DataGridView для поточної накладної відразу будуть відображатися товари усіх накладних.

З наведених прикладів випливає, що використання типізованих наборів даних значно полегшує створення інтерфейсу користувача, надаючи розробнику засоби візуального проектування.

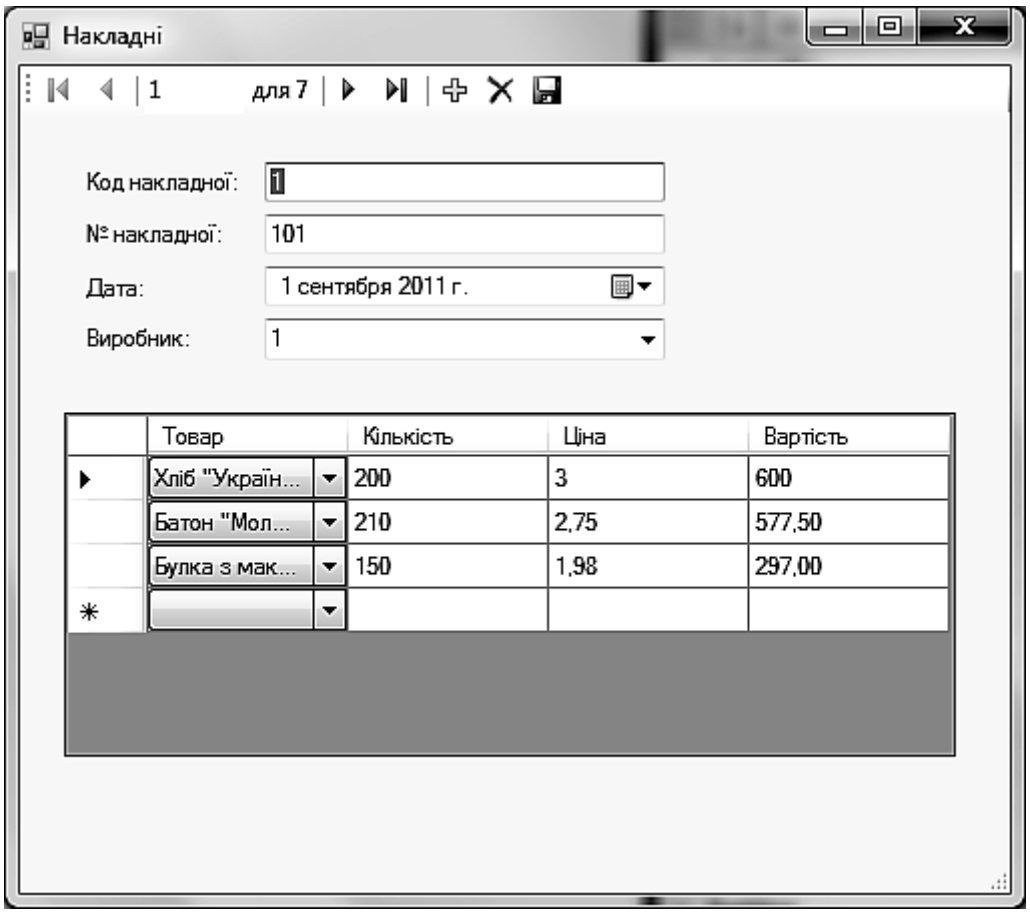


Рис. 5.19. **Форма *Накладна***

Запитання і завдання

1. Які основні переваги мають типізовані набори даних перед нетипізованими під час створення інтерфейсу користувача? Наведіть приклади їхнього використання.
2. За допомогою яких засобів створюють інтерфейс "в один стовпець"?
3. Як створити поле підстановки?
4. Опишіть алгоритм побудови форми, на якій відображаються дані двох таблиць, що пов'язані відношенням "один-до-багатьох".

Лабораторна робота № 5. Розробка застосувань на основі типізованих наборів даних

Цілі лабораторної роботи:

1. Набуття практичних навичок зі створення типізованих наборів даних.

2. Вироблення вмінь зміни властивостей об'єктів у типізованих наборах даних.

3. Набуття практичних навичок використання адаптерів таблиць.

4. Набуття практичних навичок відображення даних у застосуваннях на основі типізованих наборів даних.

5. Удосконалення навичок роботи з інтегрованим середовищем розробки Visual C# і довідковою системою Microsoft Developer Network (MSDN).

Перед виконанням лабораторної роботи студент повинен знати:

1. Основи використання бібліотеки Windows Forms.
2. Структурні елементи бази даних і їхні властивості.
3. Основи побудови SQL-запитів.
4. Організацію системи створення з'єднання в ADO.NET.
5. Принципи обробки подій в C#-програмі.

Після виконання лабораторної роботи студент повинен вміти:

1. Самостійно розробляти C#-застосування з графічним інтерфейсом користувача для роботи з базою даних.
2. Використовувати основні бібліотеки .Net Framework під час розробки програм.

Хід роботи

1. Створення кнопкової форми застосування.
2. Створення типізованого набору даних.
3. Створення форм для ведення довідкових таблиць.
4. Створення форми з деталізованим поданням даних.
5. Створення форми для ведення ієрархічних даних.
6. Додавання адаптера таблиці з агрегованою функцією.

Інструкції

Постановка загальної задачі

Вивчення засобів роботи з типізованими наборами даних проводиться шляхом створення застосування ***типізованийХліб***.

Керування роботою застосування здійснюється за допомогою кнопкової форми ***Хліб*** (рис. 5.20). Кнопки призначені для виклику відповідних функціональних форм.

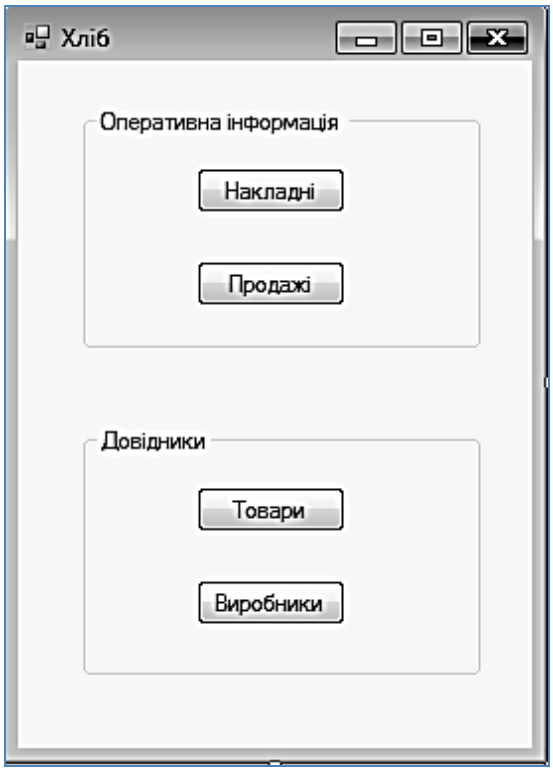
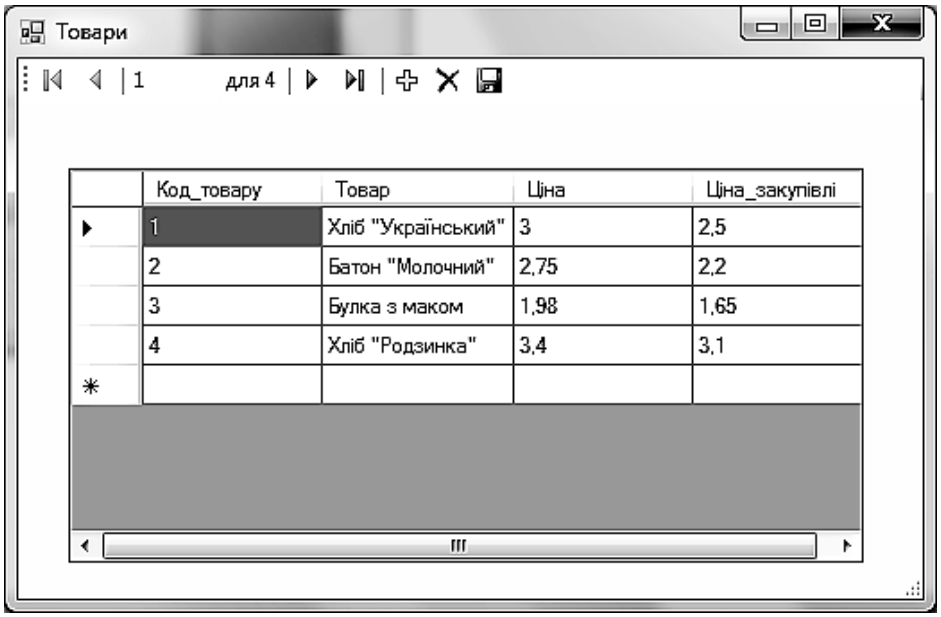


Рис. 5.20. **Кнопкова форма *Хліб***

На рис. 5.21 – 5.24 подано функціональні форми застосування. З їхньою допомогою можна переглядатий змінювати дані (обновляти, додавати й видаляти) у відповідних таблицях бази даних ***ХлібСервер***.



	Код_товару	Товар	Ціна	Ціна_закупівлі
▶	1	Хліб "Український"	3	2,5
	2	Батон "Молочний"	2,75	2,2
	3	Булка з маком	1,98	1,65
	4	Хліб "Родзинка"	3,4	3,1
*				

Рис. 5.21. **Форма для ведення таблиці *Товари***

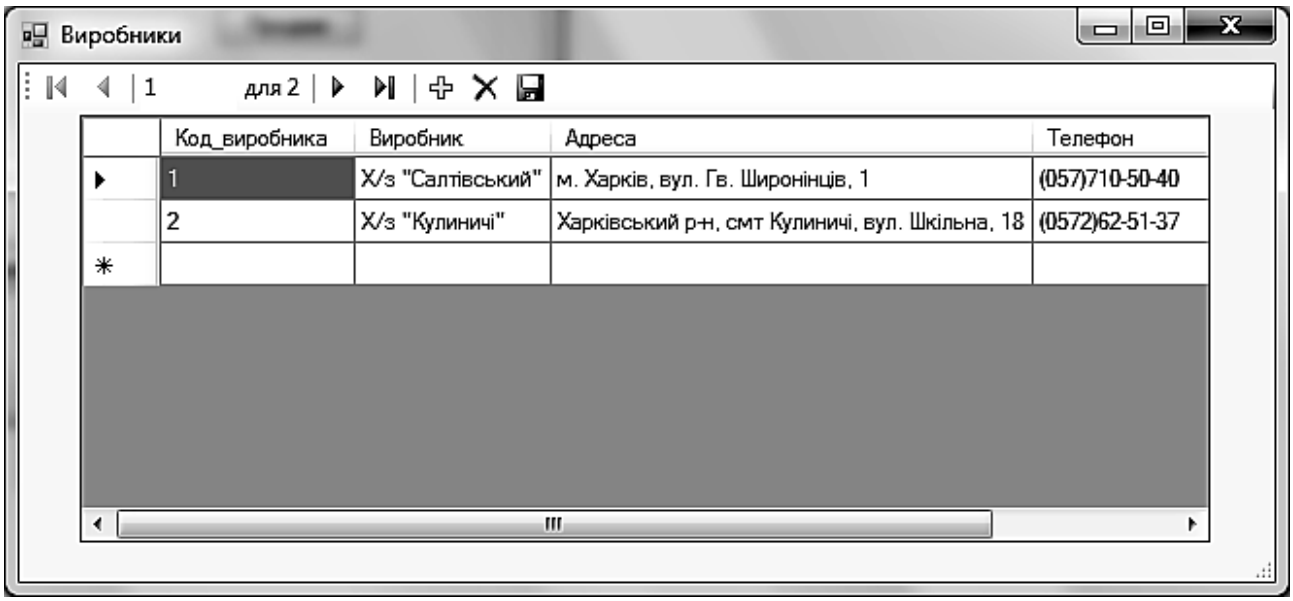


Рис. 5.22. **Форма для ведення таблиці *Виробники***

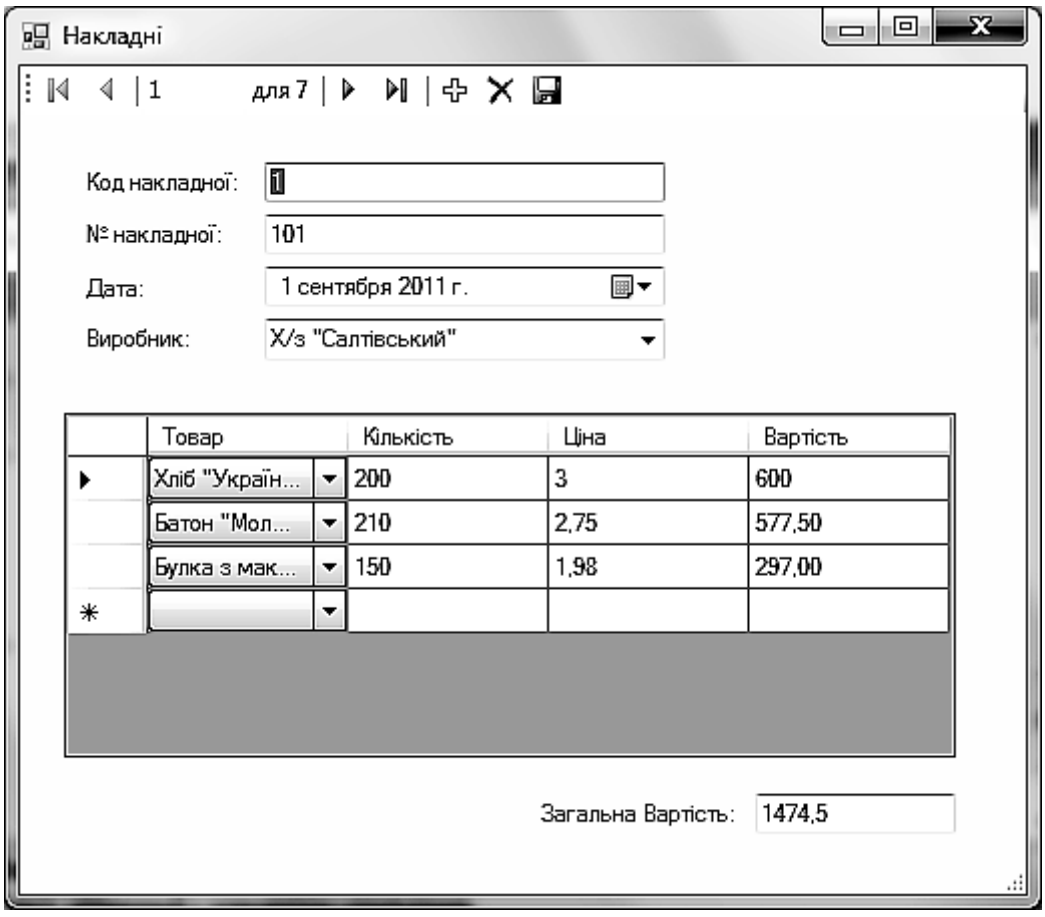


Рис. 5.23. **Форма для ведення таблиць *Накладні* й *Товари_накладних***

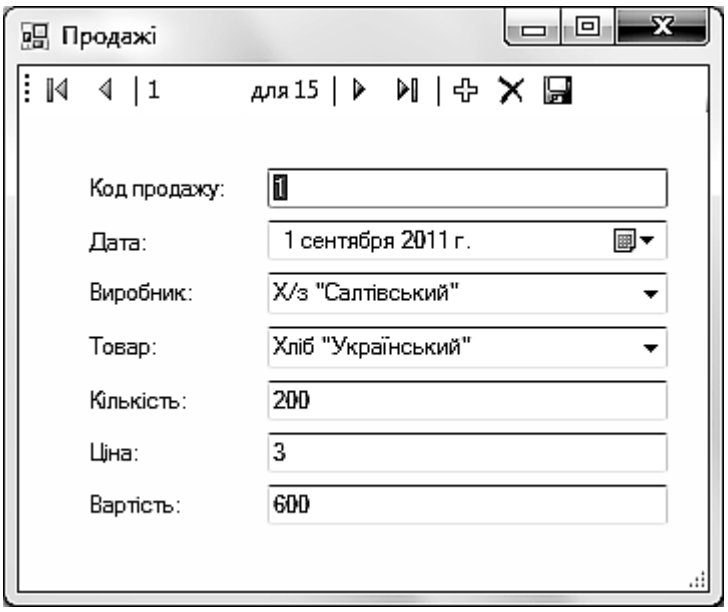


Рис. 5.24. **Форма для ведення таблиці *Продажі***

1. Створення кнопкової форми застосування

Завдання

Створити застосування і побудувати в ньому кнопкову форму (рис. 5.25).

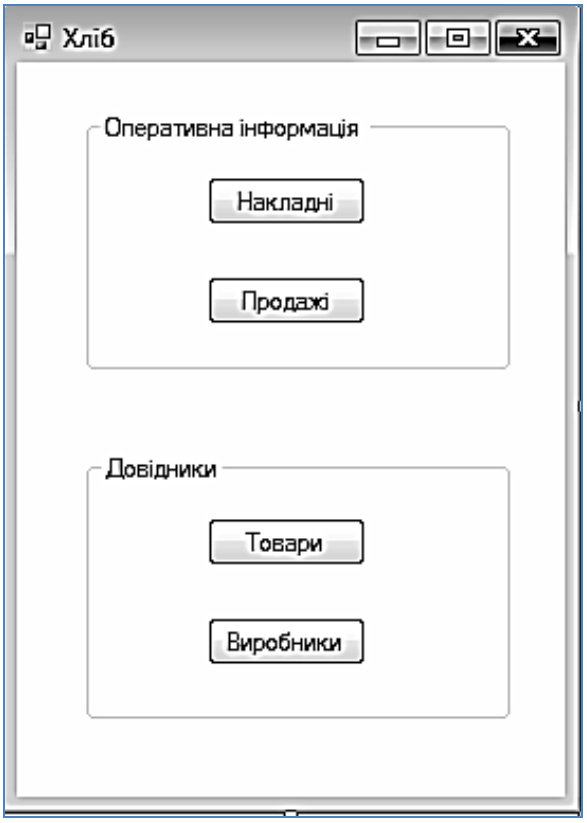


Рис. 5.25. **Кнопкова форма**

Виконання

1. Відкрийте Visual Studio.

2. Створіть проект Windows Forms мовою C# з ім'ям ***типізованийХліб*** і збережіть його.

3. У вікні **Обозреватель решений** виділіть значок ***Form1*** і для файла ***Form1.cs*** у вікні властивостей задайте ім'я ***formХліб.cs***.

4. Для властивості **Text** форми задайте значення ***Хліб***.

5. Додайте на форму елемент керування **GroupBox** і задайте значення ***Оперативна інформація*** для його властивості **Text**.

6. Додайте дві кнопки всередину елемента ***Оперативна інформація*** й установіть для них такі значення властивостей:

Кнопка	Властивість	Значення
1	Text	Накладні
	Name	buttonНакладні
2	Text	Продажі
	Name	buttonПродажі

7. Повторіть п.п. 5, 6 для групи кнопок ***Довідники***, установивши такі значення імен кнопок: ***buttonТовари*** та ***buttonВиробники***.

8. Збережіть зміни, що зроблені в проекті.

2. Створення типізованого набору даних

Завдання 1

Створити типізований набір даних ***ХлібСерверDataSet***, використовуючи майстра настроювання джерела даних.

Виконання

1. Виберіть команду **Добавить новый источник данных** у меню **Данные**.

2. Виберіть значок **База данных** як джерело даних для застосування в першому вікні майстра й клацніть кнопку **Далее**.

3. Клацніть кнопку **Создать подключение** в другому вікні майстра.

4. Клацніть кнопку **Обзор** у вікні **Добавить подключение**, знайдіть файл бази даних ***ХлібСервер.accdb***, з яким велась робота в лабораторній роботі № 3 [10], і клацніть кнопку **Открыть**.

5. Після повернення у вікно **Добавить подключение** клацніть кнопку **ОК**, а після повернення в друге вікно майстра – кнопку **Далее**.

6. Погодьтеся з тим, що файл бази даних буде скопійовано у проект, клацнувши кнопку **Да** у вікні повідомлення.

7. Клацніть кнопку **Далее** в третьому вікні майстра, погодившись із ім'ям рядка підключення.

8. Установіть прапорець у вузлі **Таблицы** в четвертому вікні майстра, вказавши, що всі таблиці бази даних будуть відображатися в наборі даних, а потім клацніть кнопку **Готово**.

У вікні **Обозреватель решений** з'явився вузол ***ХлібСерверDataSet.xsd***, що становить строго типізований набір даних (рис. 5.26).

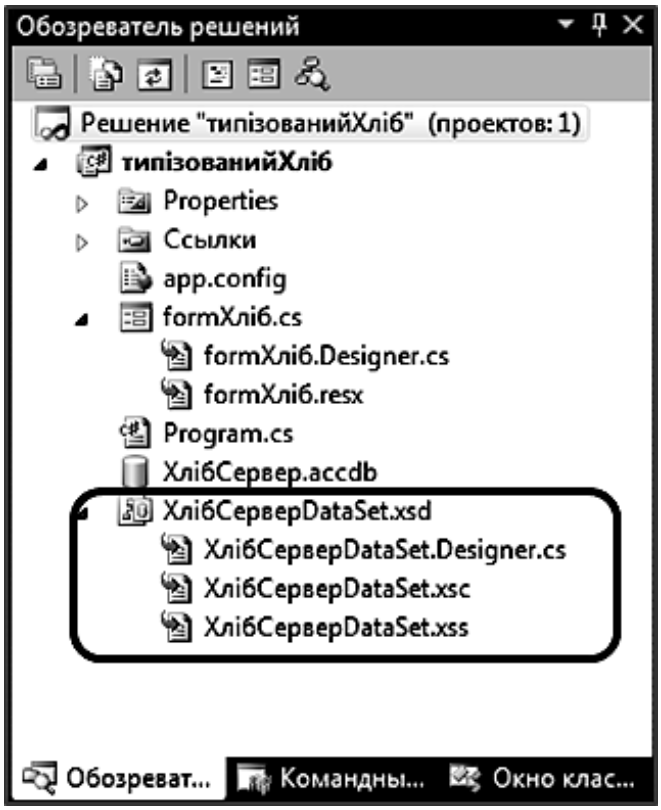


Рис. 5.26. **Вузол *ХлібСерверDataSet.xsd***

9. Збережіть зміни, що зроблені в проекті.

Завдання 2

Визначити нові об'єкти, які з'явилися під час створення типізованого набору даних ***ХлібСерверDataSet***.

Виконання

1. Двічі клацніть значок вузла типізованого набору даних ***ХлібСерверDataSet.xsd***. З'явилося вікно конструктора набору даних (рис. 5.27).

У ньому подано зображення таблиць набору, які повторюють аналогічні таблиці бази даних. У нижній частині кожної таблиці розташовані їхні адаптери. Таблиці пов'язані відношеннями.

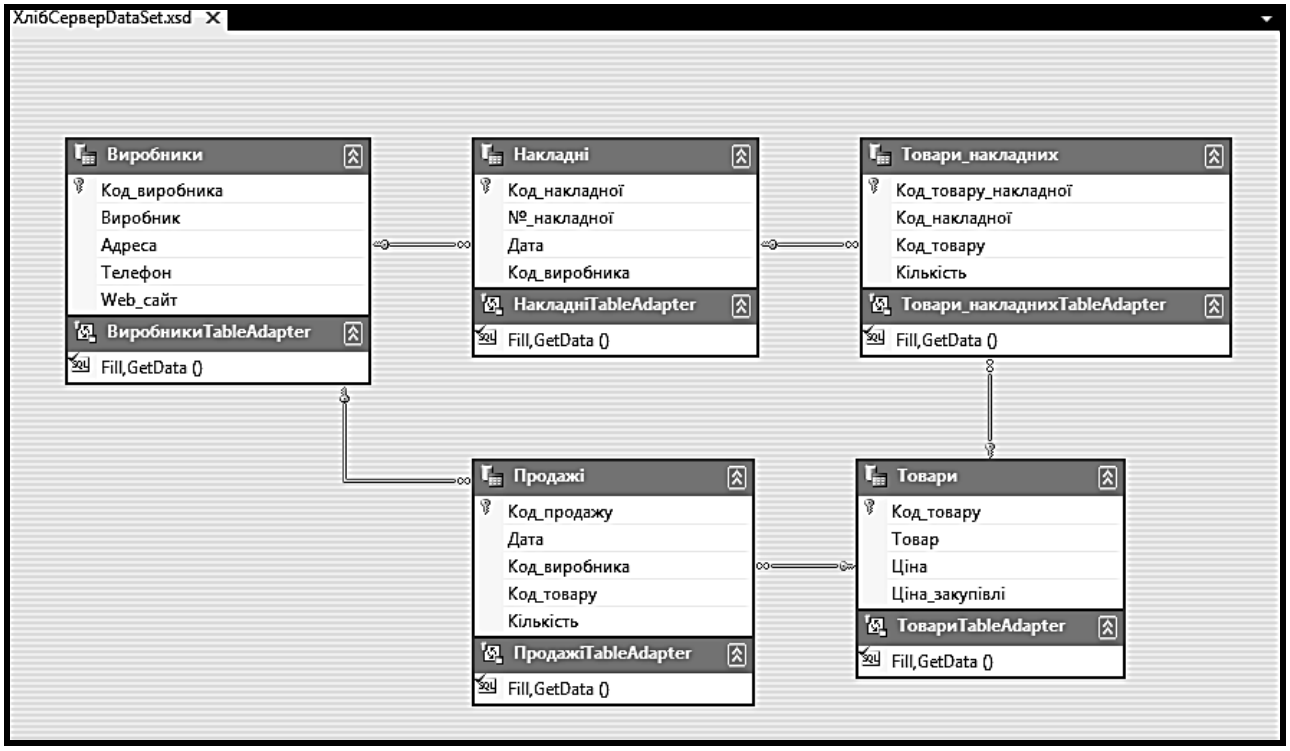


Рис. 5.27. **Набір даних у вікні конструктора**

2. Двічі клацніть перший значок підвузла ***ХлібСерверDataSet.Designer.cs***, який знаходиться у вузлі ***ХлібСерверDataSet.xsd***. З'явилося вікно коду, який згенеровано системою Visual Studio під час створення типізованого набору даних. У ньому зберігаються описи класів, їхніх методів та ін. З назвами нових класів можна ознайомитися в лівому списку, що розкривається, який розташований у верхній частині вікна (рис. 5.28). Ознайомившись із умістом вікна, закрийте його.

3. Клацніть правою кнопкою миші на значок вузла типізованого набору даних ***ХлібСерверDataSet.xsd*** і в контекстовому меню виберіть команду **Перейти к коду**. З'явилося вікно коду часткового класу (того, що може описуватися у кількох файлах, англ. *partial*) типізованого набору даних (рис. 5.29). У ньому можна додавати код, який не видаляється у разі повторного генерування набору даних. Ознайомившись із умістом вікна, закрийте його. Після закривання вікна у вузлі набору даних ***ХлібСерверDataSet.xsd*** з'явився значок файла ***ХлібСерверDataSet.cs***.

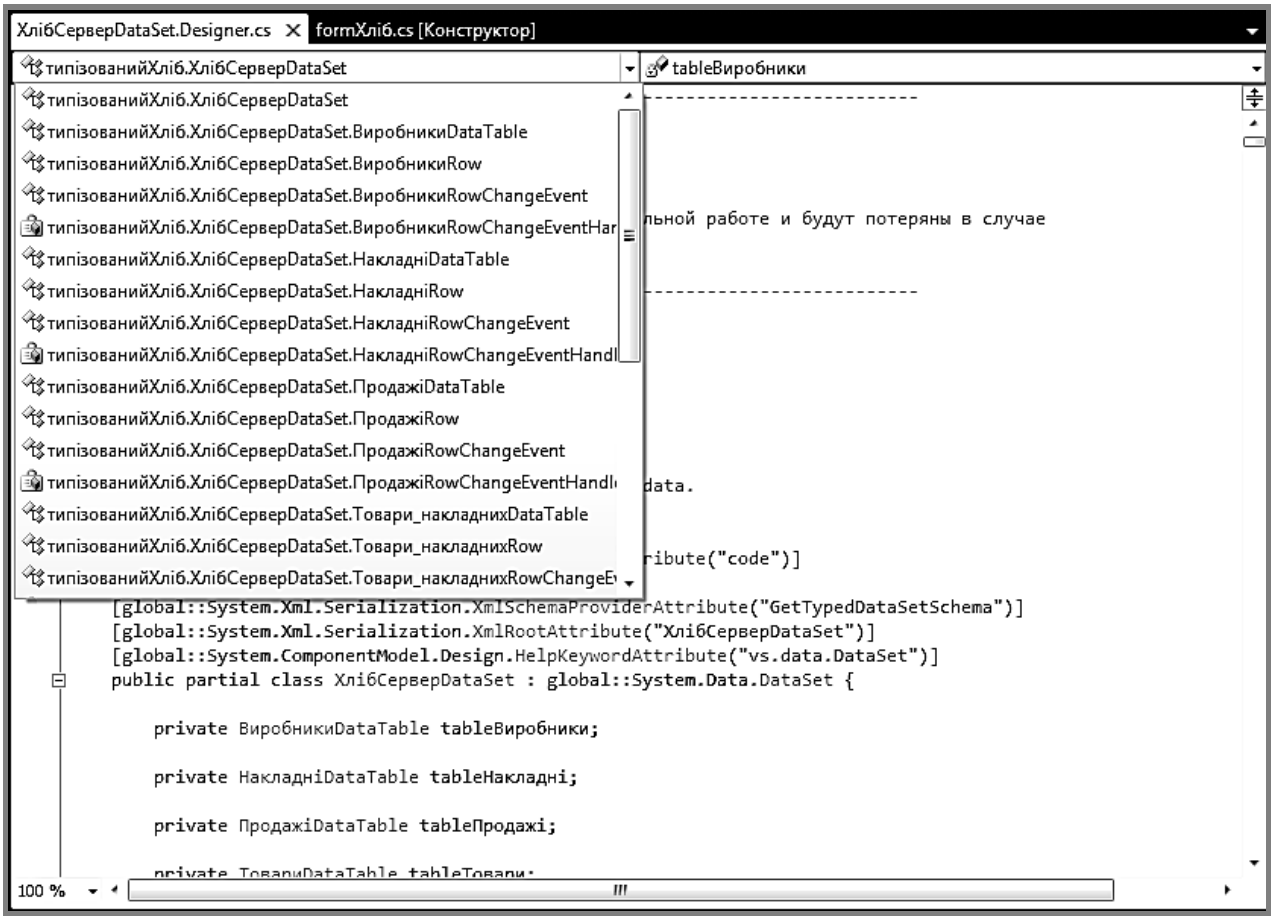


Рис. 5.28. **Вікно коду типізованого набору даних**

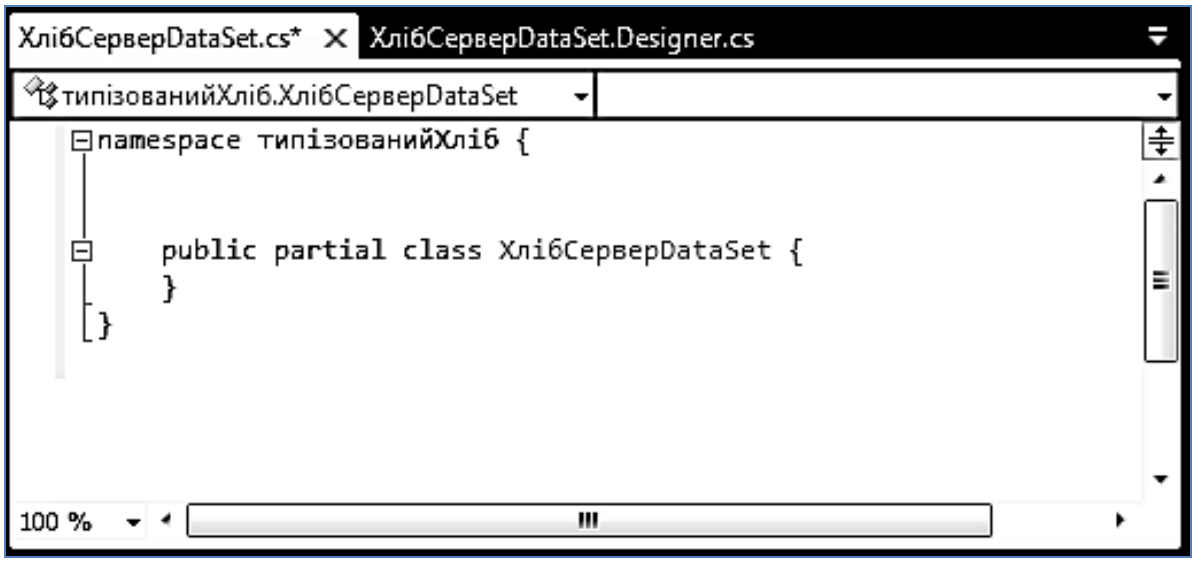


Рис. 5.29. **Вікно коду часткового класу типізованого набору даних**

4. Перейдіть у вікно **Источники данных**. У ньому відображається значок типізованого набору даних ***ХлібСерверDataSet*** у вигляді вузла. Його підвузлами є таблиці. Якщо розкрити подвузол будь-якої таблиці, відображаються її поля. Якщо таблиця має дочірню, то під полями відо-

бражаються дочірні підтаблиці (рис. 5.30). Ознайомтеся із умістом кожного вузла таблиці.

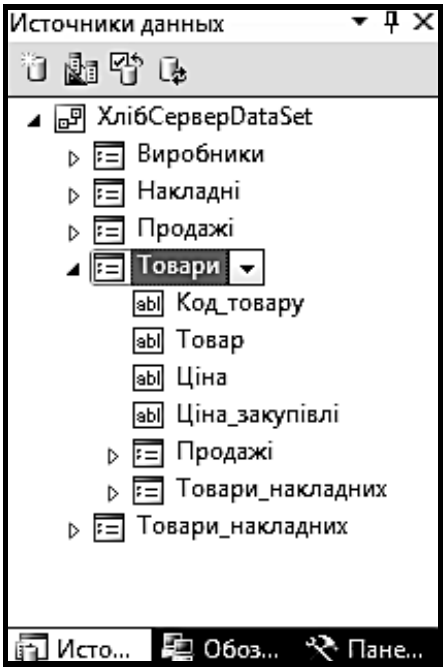


Рис. 5.30. **Поля й пов'язані дочірні таблиці у вузлі таблиці *Товари***

3. Створення форм для ведення довідкових таблиць

Завдання 1

В існуюче застосування додати форму ***Товари***, за допомогою якої можна переглядати й змінювати дані (обновляти, додавати й видаляти) в однойменній таблиці бази даних (рис. 5.31).

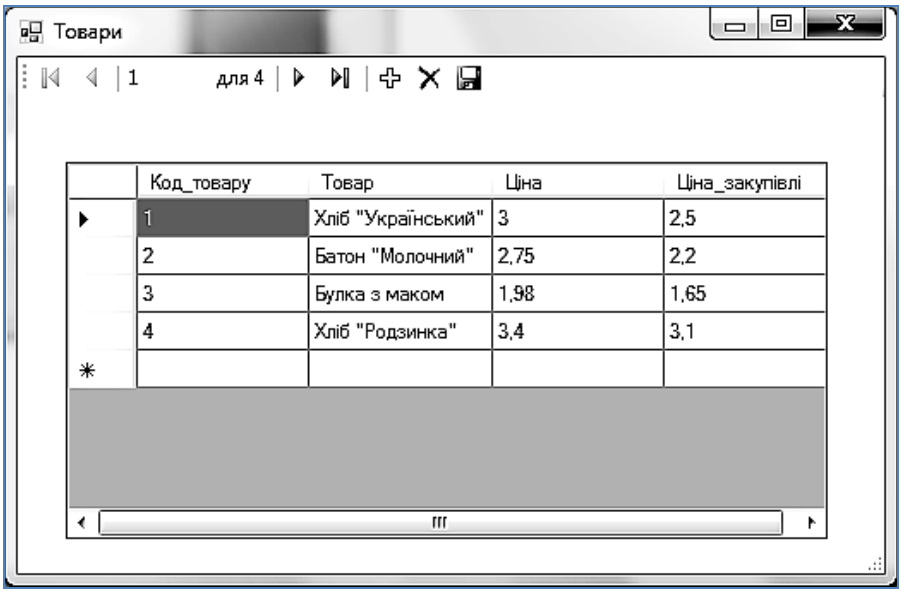


Рис. 5.31. **Форма *Товари***

Виконання

1. Додайте в застосування нову форму.

1.1. Виконайте команду **Проект – Добавить форму Windows**.

1.2. У вікні **Добавление нового элемента** введіть ім'я ***formТовари.cs*** і клацніть кнопку **Добавить**.

1.3. У вікні властивостей установіть значення ***Товар*** для властивості **Text**.

2. Перетягніть вузол таблиці ***Товари*** з вікна **Источники данных** на форму ***Товари***. На формі крім елемента керування DataGridView з даними таблиці ***Товари*** з'явилася панель навігатора, а в області компонентів – ряд об'єктів, які забезпечують роботу з даними таблиці в типізованому наборі даних (рис. 5.32). Ознайомтеся з ними й визначте призначення кожного.

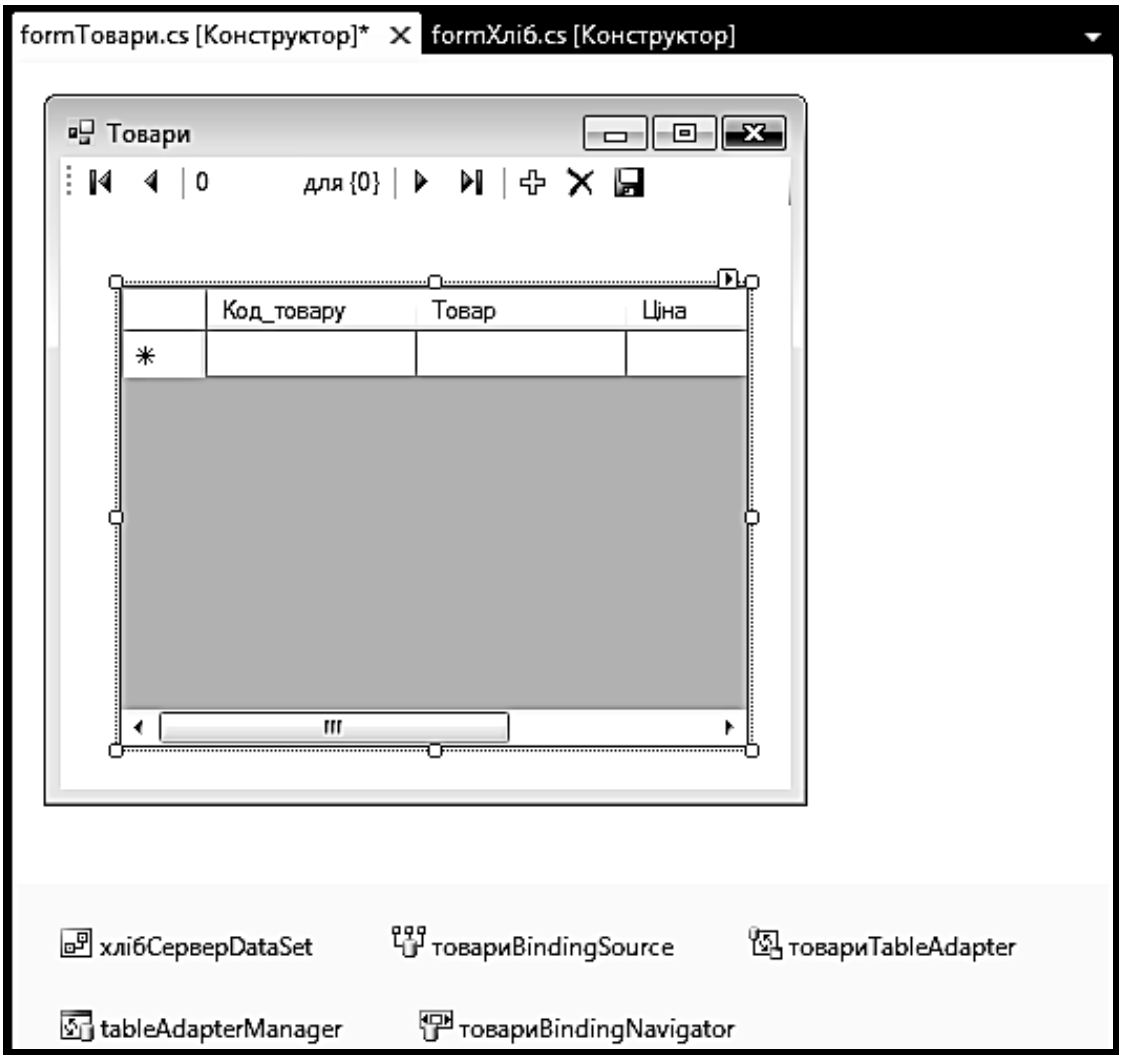


Рис. 5.32. **Вікно форми *Товари* після додавання однойменної таблиці**

3. Установіть потрібну ширину форми й елемента DataGridView, щоб дані таблиці відображалися повністю.

4. Перейдіть у вікно конструктора форми ***Хліб*** і додайте код оброблювача події "Клацання кнопки Товари".

```
private void buttonТовари_Click(object sender, EventArgs e)
        {
            formТовари вікноТовари = new formТовари();
            вікноТовари.ShowDialog();
        }
```

5. Запустіть програму на виконання й перевірте функціональність форми ***Товари*** шляхом переміщення записами, зміни, додавання й видалення записів. Спочатку виконайте операції без збереження, а потім зі збереженням у базі даних зроблених змін.

6. Закрийте форми ***Товари*** й ***Хліб***.

7. Збережіть зміни, що зроблені в проекті.

Завдання 2

У застосування ***типізованийХліб*** додати форму ***Виробники***, за допомогою якої можна переглядати й змінювати дані в однойменній таблиці бази даних (рис. 5.33).

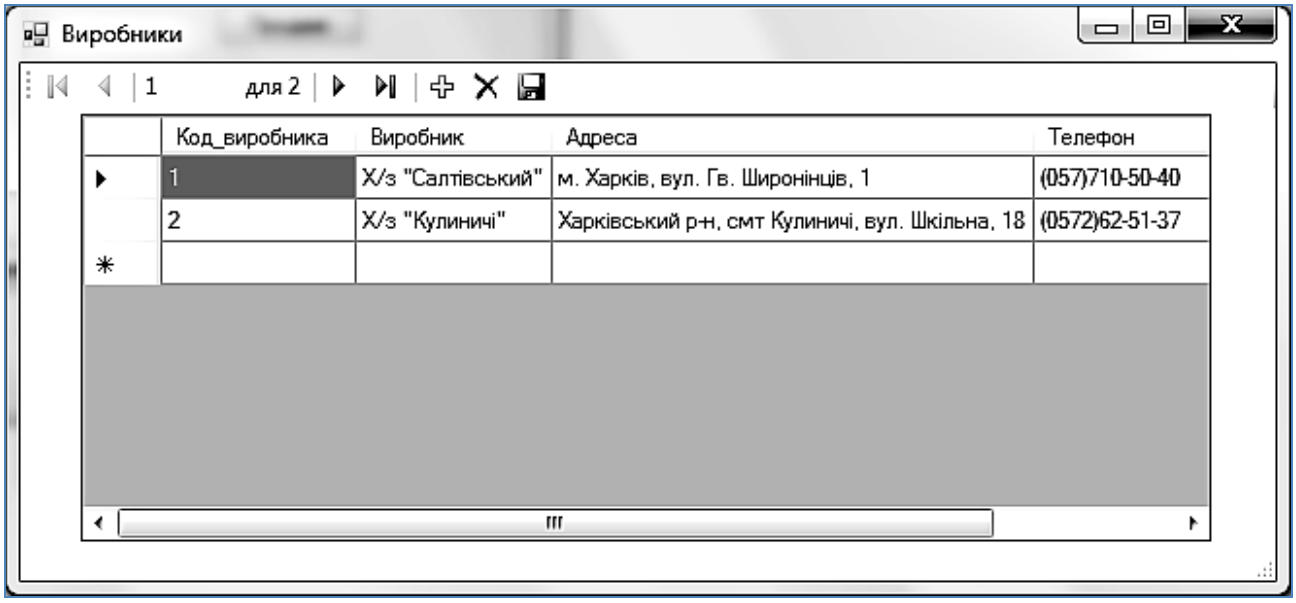


Рис. 5.33. **Форма *Виробники***

Виконання

Виконайте дії, аналогічні тим, які описані для форми ***Товари***.

4. Створення форми з деталізованим поданням даних

Завдання

Додати в застосування форму ***Продажі***, у якій будуть відображатися й змінюватися дані однойменної таблиці (рис. 5.34).

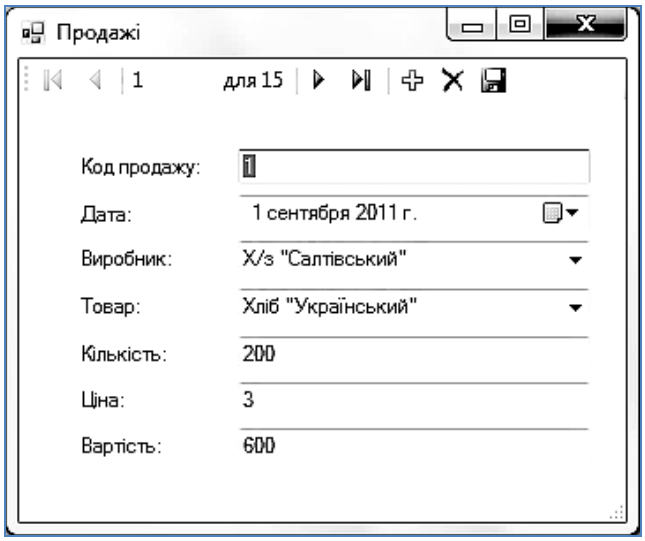


Рис. 5.34. **Форма *Продажі***

Виконання

1. Додайте в проект нову форму, давши їй ім'я ***formПродажі***.
2. Додайте стовпець ***Ціна*** в таблицю ***Продажі*** типізованого набору даних.

2.1. Перейдіть у вікно конструктора типізованого набору даних, двічі клацнувши на значок вузла набору даних ***ХлібСерверDataSet.xsd*** у вікні **Обозреватель решений**.

2.2. Клацніть лінію зв'язку між таблицями ***Товари*** й ***Продажі*** у вікні конструктора. Потім у вікні властивостей уведіть нове значення ***ТовариПродажі*** для властивості **Name**.

2.3. Клацніть правою кнопкою миші на назву таблиці ***Продажі*** й виберіть із контекстового меню команду **Добавить – Столбец** й у вікні властивостей установіть такі значення:

Властивість	Значення
Name	Ціна
DataType	System.Decimal
Expression	Parent(ТовариПродажі).Ціна

3. Додайте стовпець ***Вартість*** у таблицю ***Продажі*** типізованого набору даних, установивши у вікні властивостей такі значення:

Властивість	Значення
Name	Вартість
DataType	System.Decimal
Expression	Parent(ТовариПродажі).Ціна*Кількість

4. Закрийте вікно конструктора типізованого набору даних зі збереженням зроблених змін і перейдіть у вікно конструктора форми ***Продажі***.

5. Клацніть вузол таблиці ***Продажі*** у вікні **Источники данных** і з її списку, що розкривається, виберіть подання **Таблиця** (**Details**).

6. Розкрийте вузол таблиці ***Продажі*** й для стовпця ***Код_товару*** виберіть у списку, що розкривається, подання **ComboBox**.

7. Повторіть п. 6 для стовпця ***Код_виробника***.

8. Перетягніть вузол таблиці ***Продажі*** з вікна **Источники данных** на форму ***Продажі***. На формі з'явилися елементи керування для відображення даних таблиці й панель навігатора, а в області компонентів – ряд об'єктів, які забезпечують роботу з даними таблиці в типізованому наборі даних (рис. 5.35). Ознайомтеся з ними й визначте призначення кожного.

9. Перейдіть у вікно конструктора форми ***Хліб*** і додайте код оброблювача події "Клацання кнопки Продажі".

```
private void buttonПродажі_Click(object sender, EventArgs e)
        {
            formПродажі вікноПродажі = new formПродажі();
            вікноПродажі.ShowDialog();
        }
```

10. Запустіть програму на виконання й перевірте функціональність форми ***Продажі*** шляхом переміщення записами, зміни, додавання й видалення записів. Спочатку виконайте операції без збереження, а потім зі збереженням у базі даних зроблених змін.

11. Закрийте форми ***Продажі*** й ***Хліб***.

12. Збережіть зміни, що зроблені в проекті.

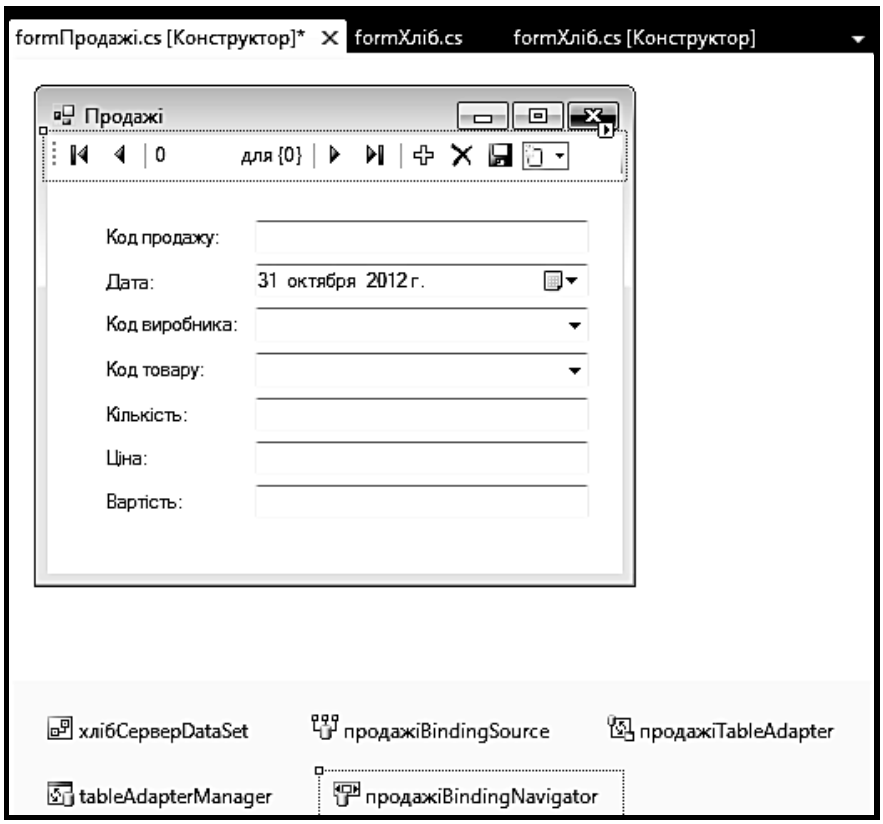


Рис. 5.35. **Вікно форми *Продажі* після додавання однойменної таблиці**

5. Створення форми для ведення ієрархічних даних

Завдання

Додати в застосування форму ***Накладні***, у якій будуть відображатися й змінюватися дані таблиць ***Накладні*** й ***Товари_накладних***, що пов'язані відношенням один-до-багатьох (рис. 5.36).

Виконання

1. Додайте в застосування нову форму, давши їй ім'я ***formНакладні***.

2. Додайте стовпці ***Ціна*** й ***Вартість*** у таблицю ***Товари_накладних*** типізованого набору даних, попередньо змінивши властивість **Name** для відношення зв'язку між таблицями ***Товари*** й ***Товари_накладних*** на ***ТовариТовари_накладних***. Дії виконуються аналогічно описаним у п.п. 2, 3 попереднього завдання.

3. Закрийте вікно конструктора типізованого набору даних зі збереженням зроблених змін і перейдіть у вікно конструктора форми ***Накладні***.

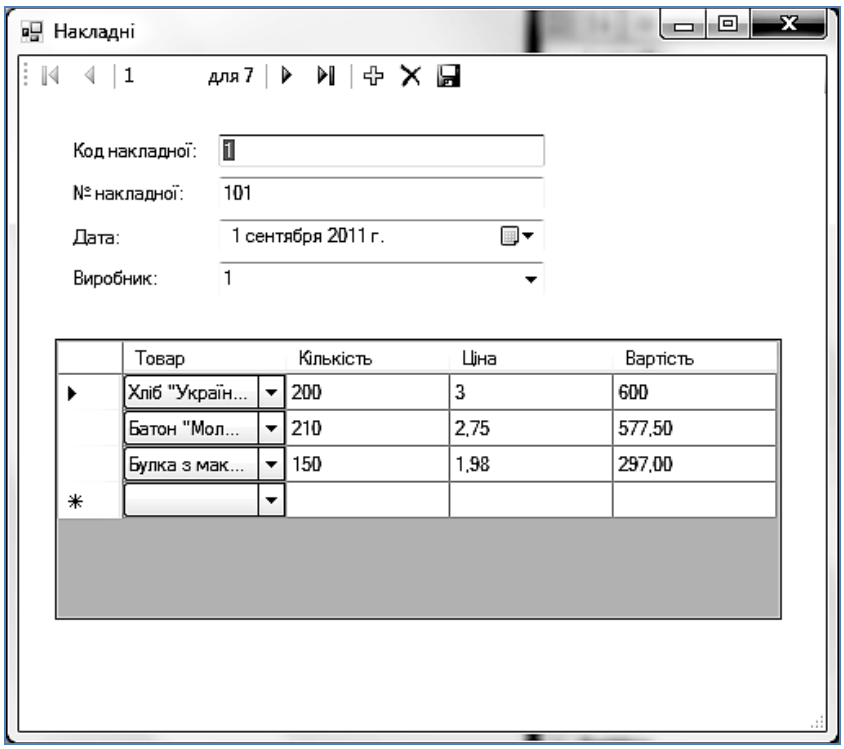


Рис. 5.36. **Форма *Накладні***

4. Клацніть вузол таблиці ***Накладні*** у вікні **Источники данных** і з її списку, що розкривається, виберіть подання **Таблиця** (**Details**).

5. Розкрийте вузол таблиці ***Накладні*** й для стовпця ***Код_виробника*** виберіть у списку, що розкривається, подання **ComboBox**.

6. Перетягніть вузол таблиці ***Накладні*** з вікна **Источники данных** на форму ***Накладні***. На формі з'явилися елементи керування для відображення даних таблиці ***Накладні*** й панель навігатора, а в області компонентів – ряд об'єктів, які забезпечують роботу з даними таблиці в типізованому наборі даних.

7. Змініть властивість **Text** для напису ***Код_виробника***, установивши нове значення ***Виробник***.

8. Щоб в елементі ComboBox для виробника відображалися назви виробників, перетягніть вузол таблиці ***Виробники*** з вікна **Источники данных** на цей ComboBox і відпустіть.

9. Розкрийте підвузол ***Товари_накладних***, який знаходиться в нижній частині вузла ***Накладні*** у вікні **Источники данных**. Потім змініть подання стовпця ***Код_товару*** на **ComboBox**.

10. Перетягніть вузол таблиці ***Товари_накладних***, який знаходиться в нижній частині вузла ***Накладні***, з вікна **Джерела даних** на форму

Накладні, помістивши його під наявними там елементами керування. На формі з'явився елемент керування DataGridView для відображення даних таблиці ***Товари_накладних***, а в області компонентів – об'єкти BindingSource і TableAdapter, які забезпечують роботу з даними таблиці ***Товари_накладних***. Але серед компонентів відсутні об'єкти для роботи з даними таблиці ***Товари***, тому стовпці ***Ціна*** й ***Вартість*** в DataGridView будуть порожніми.

11. Щоб додати об'єкти BindingSource і TableAdapter для роботи з даними таблиці ***Товари***, перетягніть будь-який стовпець (наприклад, ***Товар***) із вузла ***Товари*** на вільне місце форми. Потім вилучіть з форми елемент керування, що з'явився на ній. При цьому залишаться потрібні компоненти в області компонентів.

12. Установіть необхідну ширину форми й елемента DataGridView, щоб дані таблиці відображалися повністю.

13. Перейдіть у вікно конструктора форми ***Хліб*** і додайте код оброблювача події "Клацання кнопки Накладні".

```
private void buttonНакладні_Click(object sender, EventArgs e)
        {
            formНакладні вікноНакладні = new formНакладні();
            вікноНакладні.ShowDialog();
        }
```

14. Запустіть програму на виконання й перевірте функціональність форми ***Накладні***. Стовпець ***Код_товару*** відображається у вигляді елемента TextBox, а не ComboBox (рис. 5.37). Закінчіть виконання застосування.

15. Щоб стовпець ***Код_товару*** відображався у вигляді елемента ComboBox, виконайте таке:

15.1. Відкрийте вікно форми ***Накладні*** в режимі конструктора.

15.2. Виділіть елемент керування DataGridView.

15.3. Клацніть кнопку  у властивості **Columns** вікна властивостей. З'явиться вікно **Правка столбцов** (рис. 5.38).

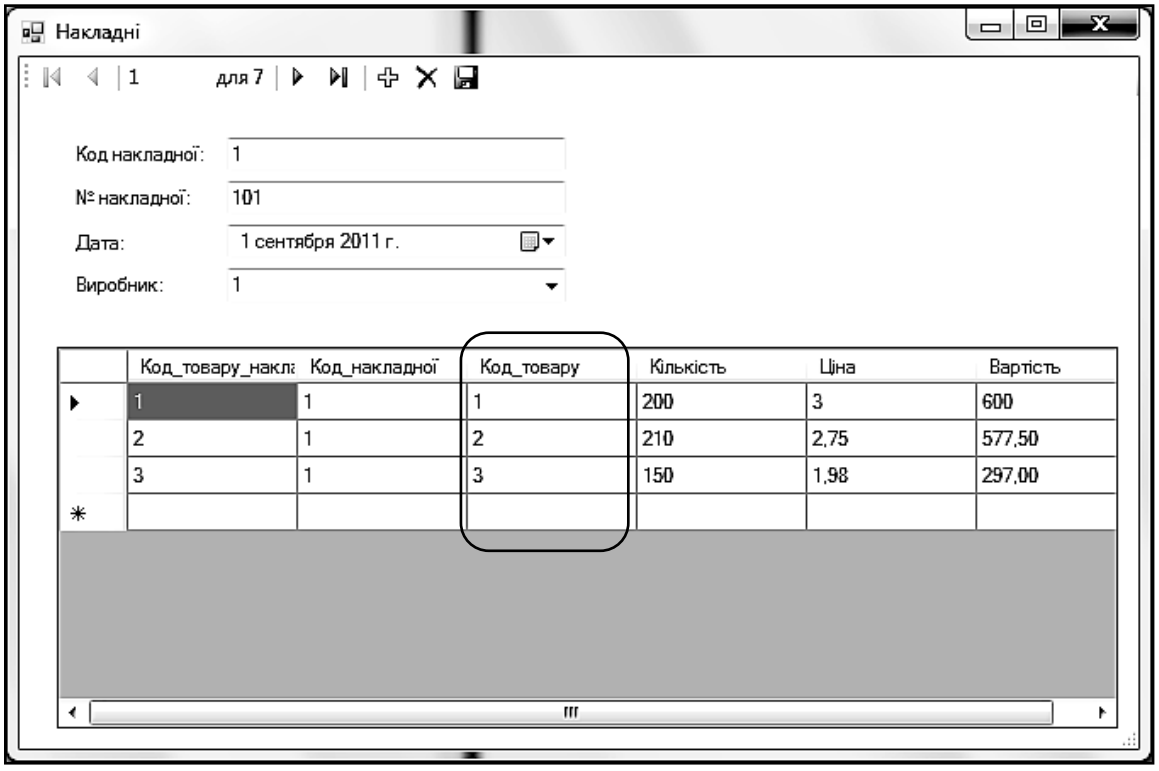


Рис. 5.37. **Стовпець *Код_товару* у вигляді елемента TextBox у вікні форми *Накладні***

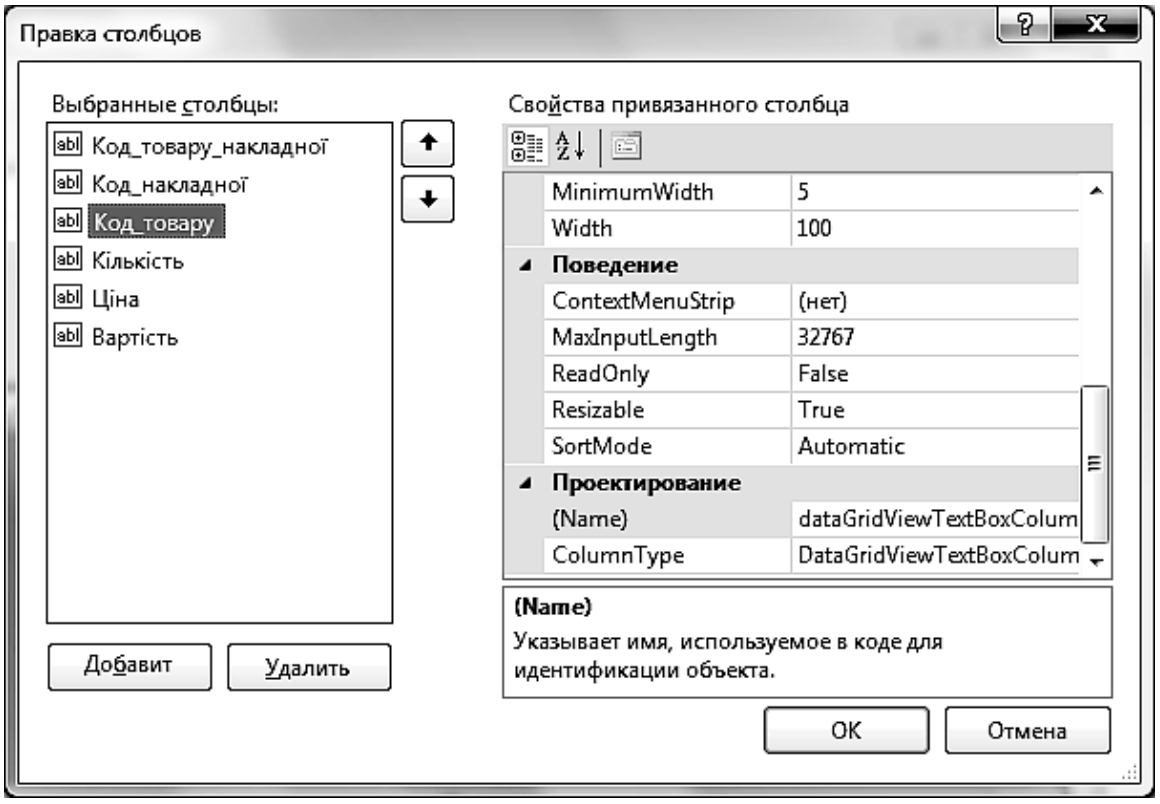


Рис. 5.38. **Вікно Правка столбцов**

15.4. Виберіть елемент ***Код_товару*** в списку **Выбранные столбцы** вікна **Правка столбцов**, а в таблиці **Свойства привязанного столбца** установіть такі значення властивостей:

Властивість	Значення
ColumnType	DataGridViewComboBoxColumn
HeaderText	Товар
DataPropertyName	Код_товару
DataSource	товариBindingSource
DisplayMember	Товар
ValueMember	Код_товару

15.5. Клацніть кнопку **ОК**.

15.6. Запустіть програму на виконання й перевірте функціональність форми ***Накладні***. Стовпець ***Код_товару*** відображається у вигляді ComboBox. Бажано не відображати стовпці ***Код_товару_накладної*** та ***Код_накладної***, які відіграють службову роль і для кінцевого користувача не цікаві.

16. Зробіть невидимими стовпці ***Код_товару_накладної*** та ***Код_накладної*** в елементі керування DataGridView. Для цього виділіть елемент керування DataGridView, у вікні властивостей клацніть кнопку  у властивості **Columns**. З'явиться вікно **Правка столбцов**. У вікні **Правка столбцов** установіть значення **False** для властивості **Visible** у стовпцях ***Код_товару_накладної*** і ***Код_накладної***.

17. Установіть потрібну ширину форми й елемента DataGridView (рис. 5.39).

18. Запустіть програму на виконання й перевірте функціональність форми ***Накладні*** шляхом переміщення записами, зміни, додавання й видалення записів. Спочатку виконайте операції без збереження, а потім зі збереженням у базі даних зроблених змін.

Усі зміни даних зберігаються коректно за виключенням додавання нової накладної. Проблема пов'язана з автоінкрементними значеннями ключа таблиці ***Накладні*** та ієрархічно пов'язаними даними. Вона розв'язується за тим самим алгоритмом, що й в лабораторній роботі № 3 [10]. У поточній роботі її винесено в задачі для самостійного виконання.

19. Закрийте форми ***Накладні*** й ***Хліб***.

20. Збережіть зміни, що зроблені в проекті.

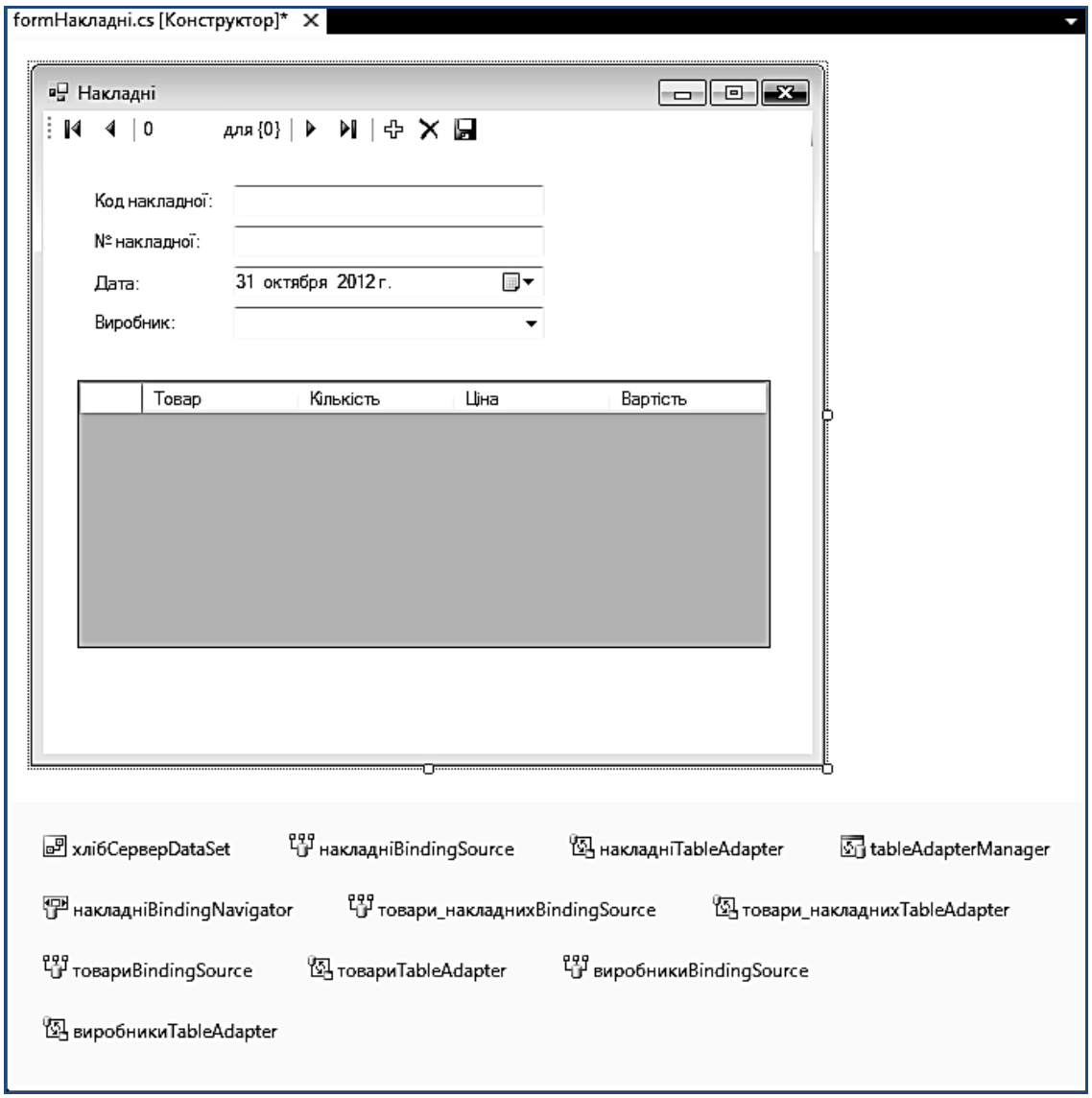


Рис. 5.39. **Вікно форми *Накладні* в режимі конструктора**

6. Додавання адаптера таблиці з агрегованою функцією

Завдання

Додати на форму ***Накладні*** загальну суму вартостей товарів за накладною (рис. 5.40).

Виконання

1. Додайте в типізований набір даних адаптер таблиці ***СумиНакладних***, який містить запит. У ньому є два стовпці – ***Код_накладної*** та ***ЗагальнаВартість***, що є сумою вартостей товарів за відповідною накладною. Для цього виконайте таке:

1.1. Перейдіть у вікно конструктора набору даних.

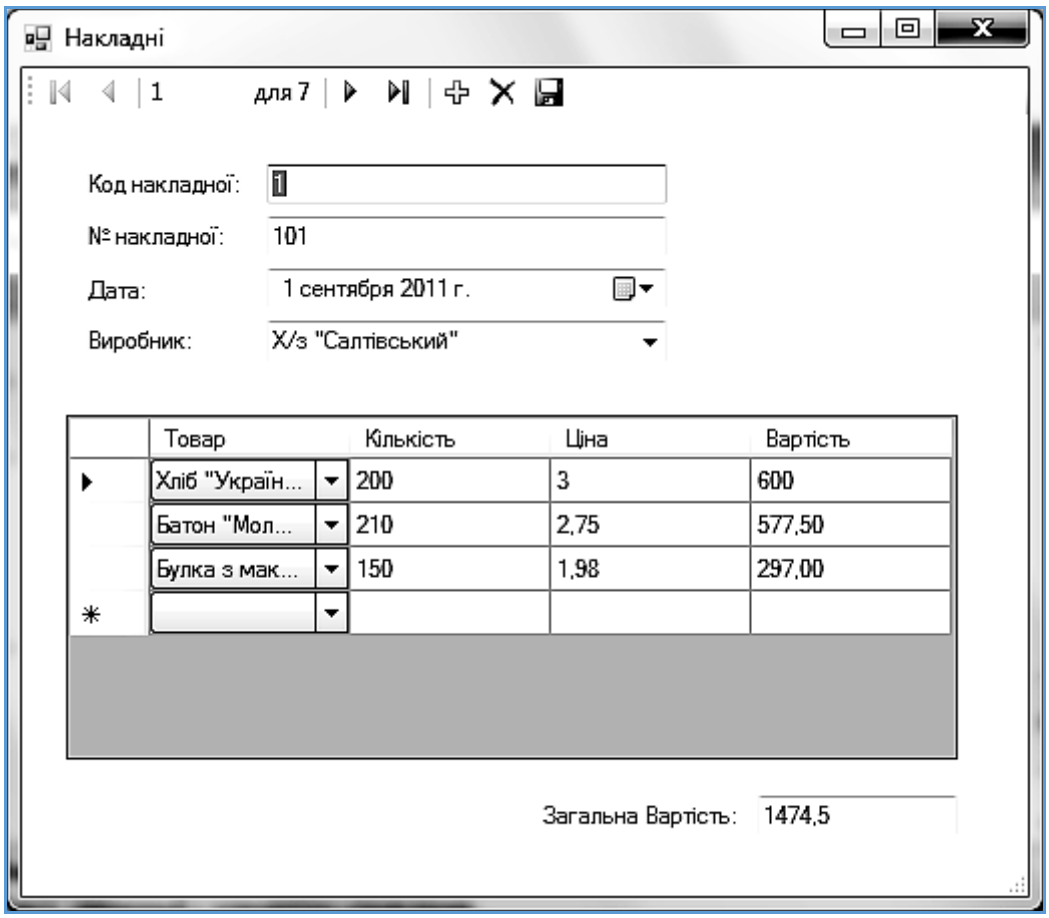


Рис. 5.40. **Форма *Накладні***

1.2. Клацніть правою клавішою миші у вільному місці вікна конструктора й виберіть команду **Добавить – Table Adapter**. З'явиться вікно майстра настроювання адаптера таблиць.

1.3. У першому вікні майстра погодьтеся з попереднім рядком підключення, клацнувши кнопку **Далее**.

1.4. У другому вікні майстра погодьтеся з тим, що адаптер буде використовувати інструкції SQL, клацнувши кнопку **Далее**.

1.5. У третьому вікні майстра введіть інструкцію SQL

```
SELECT [Товари_накладних].[Код_накладної],
    SUM(Товари.Ціна * [Товари_накладних].Кількість) AS ЗагальнаВартість
FROM (Товари INNER JOIN [Товари_накладних] ON
    Товари.[Код_товару] = [Товари_накладних].[Код_товару])
GROUP BY [Товари_накладних].[Код_накладної]
```

і клацніть кнопку **Далее**.

Примітка. Інструкцію SQL можна побудувати за допомогою візуальних засобів Visual Studio, клацнувши кнопку **Построитель запросов**

у цьому вікні, або перейшовши в застосування Access і скориставшись його візуальними засобами, а потім скопіювавши побудовану інструкцію SQL у вікно майстра настроювання адаптера таблиць.

1.6. У четвертому вікні майстра погодьтеся з іменами створюваних методів, клацнувши кнопку **Далее**.

1.7. У п'ятому вікні майстра ознайомтеся зі списком виконаних дій і клацніть кнопку **Готово**. У вікні конструктора набору даних з'явилася нова таблиця з незручним ім'ям ***Товари_накладних1***. Вона пов'язана з таблицею ***Накладні***.

1.8. Клацніть ім'я нової таблиці й у вікні властивостей уведіть більш зручне, наприклад, ***СумиНакладних***. На рис. 5.41 подано вікно конструктора набору даних з адаптером таблиці ***СумиНакладних***.

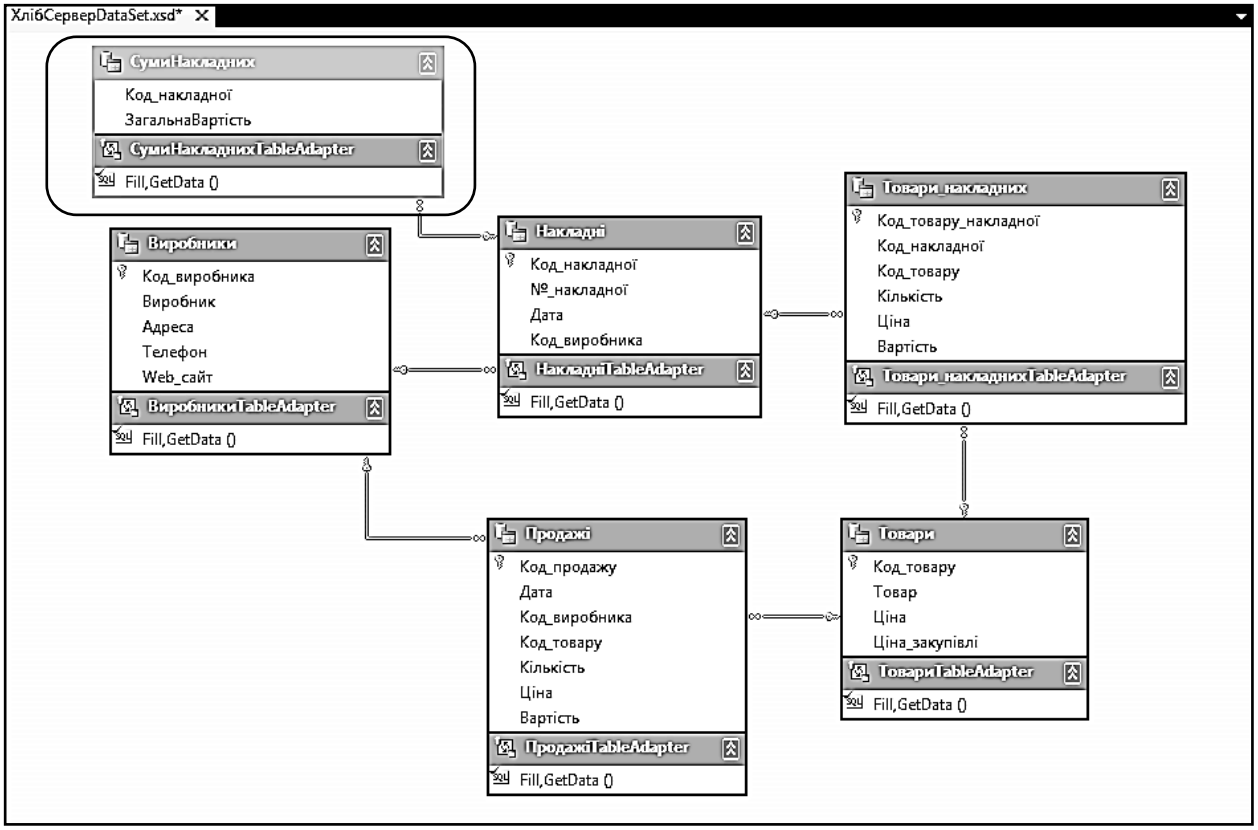


Рис. 5.41. **Вікно конструктора набору даних з адаптером таблиці *СумиНакладних***

1.9. Закрийте вікно конструктора набору даних зі збереженням зроблених змін.

2. Перейдіть у вікно конструктора форми ***Накладні*** й збільште висоту форми приблизно на один сантиметр.

3. Розкрийте вузол таблиці ***Накладні*** у вікні **Источники данных**, а в ньому – підвузол таблиці ***СумиНакладних*** і перетягніть стовпець ***ЗагальнаВартість*** на форму ***Накладні***, помістивши його під елементом DataGridView. Там з'явиться текстове поле з написом ***Загальна Вартість*** (рис. 5.42).

4. Запустіть програму на виконання й перевірте функціональність форми ***Накладні*** шляхом переміщення записами. Простежте за зміною значень у полі ***Загальна Вартість***.

5. Закрийте форми ***Накладні*** й ***Хліб***.

6. Збережіть зміни, що зроблені в проекті.

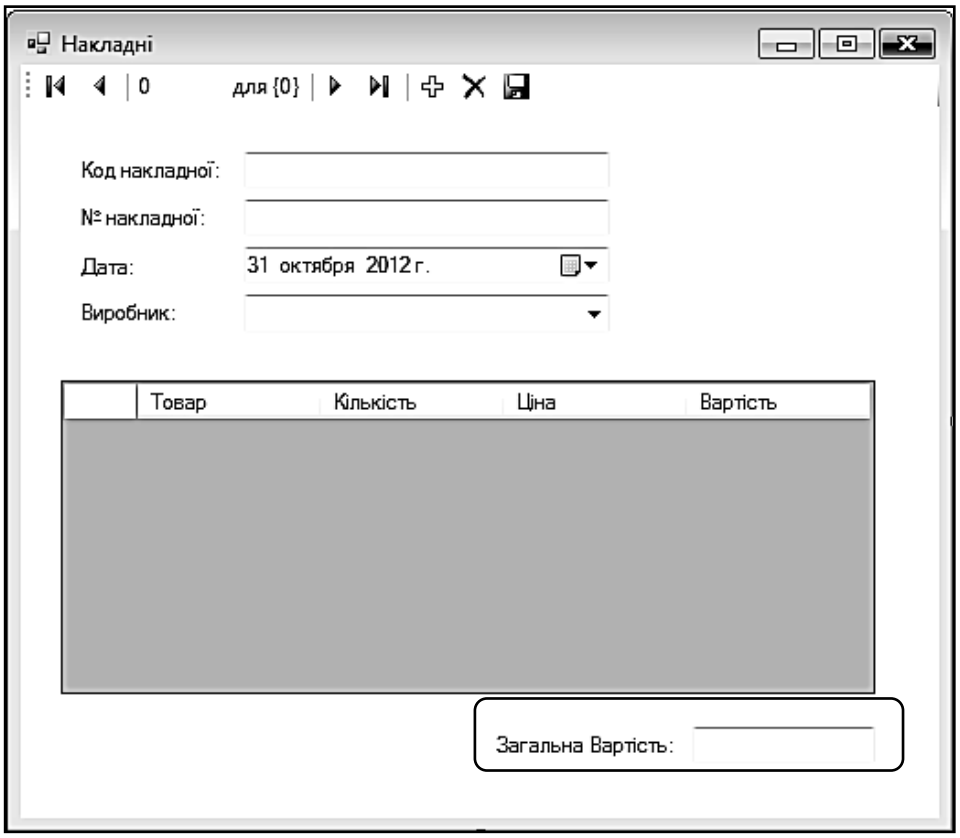


Рис. 5.42. **Вікно форми *Накладні* з полем *Загальна Вартість* у режимі конструктора**

Завдання для самостійного виконання

1. Обновити поле ***Код_продажу*** на формі ***Продажі*** після додавання нового запису.

2. Забезпечити збереження нової накладної разом із її товарами на формі ***Накладні***.

3. Відформатувати елементи керування на формах, дотримуючись таких правил:

текстові дані притискають до лівого краю елемента, а числові – до правого;

усі числа в одному стовпці відображають з однаковою кількістю цифр у дробовій частині;

грошові величини (ціна, вартість тощо) відображають з двома цифрами у дробовій частині.

4. Виконати п.п. 1 – 6 ходу роботи для індивідуальної бази даних.

Висновки

1. Типізований набір даних – це сукупність класів, що успадковують і розширюють можливості класів DataSet, DataTable і DataRow. Вони надають додаткові властивості, методи й події на базі схеми класу DataSet.

2. Типізований набір даних надає засоби створення застосувань візуальними засобами майже без використання "ручного" написання коду, а в разі його вживання значно полегшує безпомикове написання коду за рахунок використання IntelliSense.

3. Програмування з використанням типізованого набору даних є більш інтуїтивним, а код виявляється простішим в обслуговуванні. Імена таблиць, стовпців та інших об'єктів доступні через властивості, а не через використання індексів або рядків із роздільниками.

4. Типізований набір даних надає доступ до значень, що мають правильний тип. Тому помилки невідповідності типів виявляються під час компіляції коду, а не під час виконання.

5. У вікні конструктора наборів даних можна додавати стовпці в об'єкт DataTable, змінювати властивості та редагувати зв'язки між таблицями.

6. Адаптери таблиць інкапсулюють запити, що спільно використовують загальну схему з поєднаною типізованою таблицею, а також надають додаткові типізовані методи (Delete, Insert тощо).

7. Адаптер таблиці може містити кілька запитів для заповнення пов'язаної таблиці, які задовольняють різні критерії.

8. За допомогою вікна **Источники данных** можна створювати інтерфейс користувача з потрібним налаштуванням форми в цілому й окремих елементів керування на ній.

6. Технологія LINQ to DataSet

6.1. Призначення й переваги LINQ

LINQ (Language-Integrated Query) – це шаблони для запиту й зміни даних і технології, які можна розширити для підтримки джерела даних практично будь-якого типу.

У сучасному вигляді технології LINQ з'явилися у Visual Studio 2008 як спроба будувати алгоритми для відбору даних у декларативний спосіб. Цьому передувала практика роботи з даними, що зберігаються у реляційних базах даних.

Успішне застосування мови SQL у побудові запитів для відбору даних із бази породили сподівання на можливість застосувати ті самі ідеї до відбору об'єктів з множин. Отримані в результаті об'єкти повинні задовольняти певні умови, йти в певному порядку і розподілятися за групами.

Використовуючи засоби декларативної мови запитів для виконання операцій з об'єктами не обов'язково записувати кожен раз складні алгоритми. Достатньо вказати, з яких множин потрібно вибрати об'єкти і які умови має задовольняти результат. Самі ж алгоритми інкапсульовані від розробника, який записує запит засобами мови програмування C# чи Visual Basic. Ідея перенесення засобів роботи з реляційними базами даних мовою SQL у мову програмування реалізувалася у вигляді технологій LINQ. Кажуть, що LINQ поєднує світ об'єктів зі світом даних.

Використання технологій LINQ під час розробки застосувань з базами даних дає такі переваги:

1. LINQ робить запити дуже зручною конструкцією мов C# і Visual Basic на відміну від традиційних запитів до даних, які виражаються у вигляді простих рядків без перевірки типів під час компіляції та підтримки засобів IntelliSense.

2. Під час налагодження коду LINQ забезпечує покрокове виконання, встановлення точок зупинки, перегляд результатів у вікнах налагоджувальника.

Запитання і завдання

1. Яке призначення мають технології LINQ?

2. Які переваги мають технології LINQ? Наведіть приклади їхнього використання під час розробки застосувань з базами даних.

3. За допомогою яких засобів ви відбирали потрібні дані з бази у роз'єднаному і з'єднаному середовищах? В чому полягали їхні недоліки?

6.2. Види технологій LINQ

LINQ є родиною технологій, що вбудовані у мови програмування C# та Visual Basic. Завдяки перевагам, які надають технології LINQ, вони з'явилися і в інших мовах програмування (F#, Java Script, PHP тощо). Структуру технологій LINQ подано на рис. 6.1. Далі розглянуто її елементи.

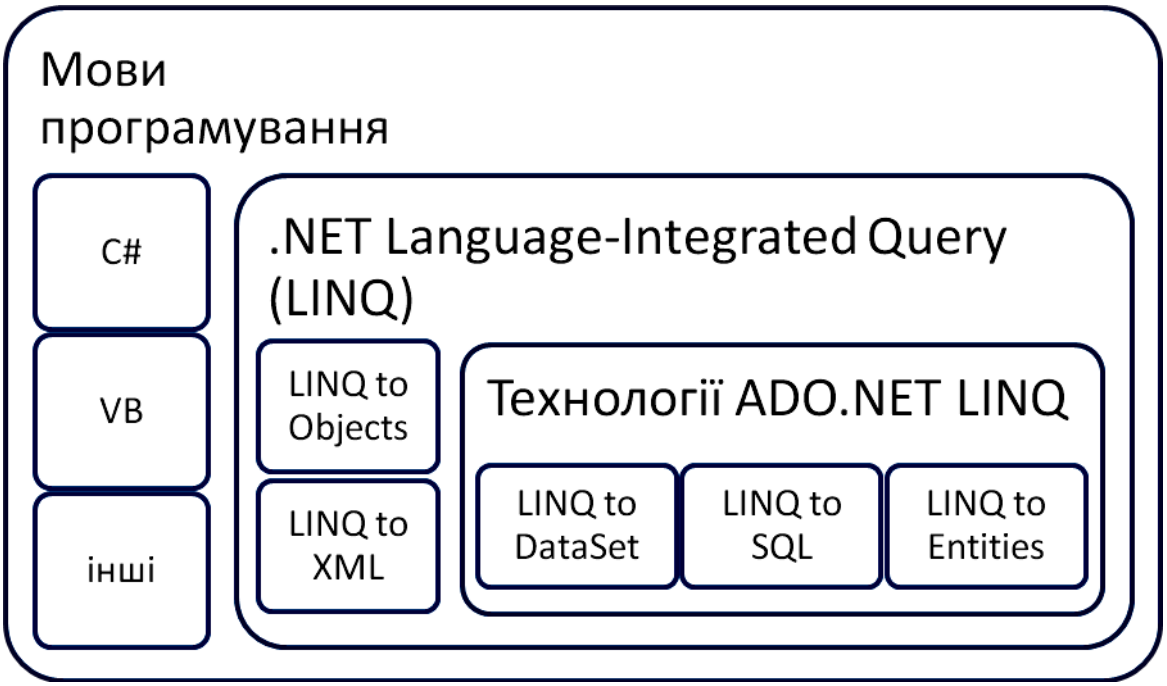


Рис. 6.1. **Структура технологій LINQ**

На даний час добре розроблені такі технології LINQ:

LINQ to Objects – дозволяє виконувати запити до масивів та колекцій об'єктів (чисел, рядків тощо);

LINQ to XML – підтримує запити до XML-сховищ даних;

технології ADO.NET LINQ – забезпечують виконання запитів із базами даних реляційного типу.

Останній вид технологій є найбільш цікавим для побудови застосувань, які працюють з базами даних. До їхнього складу входять такі технології:

LINQ to DataSet – забезпечує розширені й оптимізовані запити до DataSet (типізованого і нетипізованого);

LINQ to SQL – реалізує запити до баз даних, що побудовані на основі СКБД SQL Server;

LINQ to Entities – дозволяє виконувати запити до сутнісної моделі даних EDM.

Першу з технологій ADO.NET LINQ розглянуто у поточному розділі, а останню – у наступному. Технологію LINQ to SQL в даному навчальному посібнику не буде розглянуто через її вузьку спрямованість і ту обставину, що вона повсюди замінюється технологією LINQ to Entities, яка найбільш інтенсивно розвивається останнім часом.

Запитання і завдання

1. В яких мовах програмування реалізовані технології LINQ?
2. Які види технологій LINQ вам відомі? Опишіть їхнє призначення.
3. Порівняйте призначення технологій LINQ to SQL та LINQ to Entities?
4. Відшукайте в Інтернеті відомості про те, для яких СКБД реалізовано технологію LINQ to Entities.

6.3. Технологія LINQ to DataSet і запити

Клас DataSet є ключовим елементом моделі автономного програмування, що побудована на основі ADO.NET. Він дозволяє явно кешувати дані з різних джерел.

До створення LINQ to DataSet був розрив між засобами, за допомогою яких дані вибирають із бази, і тими, за допомогою яких дані вибирають з DataSet. У першому випадку використовували мову SQL, а в другому – алгоритмічну, хоча структури бази даних і DataSet дуже подібні. В обох випадках – це таблиці, між якими є зв'язки.

Технологія LINQ to DataSet дала можливість ліквідувати цей розрив. Вона спростила й прискорила написання запитів до даних в об'єкті DataSet, використовуючи конструкції мови, які подібні до тих, що вживаються у мові SQL. При цьому запити формуються безпосередньо мовою програмування без застосування окремої мови запитів.

Під час створення застосувань технологія LINQ to DataSet використовується для реалізації задач звітності, аналізу й бізнес-аналітики. В операціях ведення бази даних (додавання, зміна і видалення даних із таблиць) технологія LINQ to DataSet не надає переваг перед традиційними засобами, тому в них ця технологія за звичай не використовується.

Перед виконанням запиту до об'єкта DataSet за допомогою технології LINQ to DataSet необхідно помістити дані в об'єкт DataSet.

Запити LINQ можуть виконуватися до однієї чи кількох таблиць DataSet. В останньому разі використовують стандартні оператори запиту Join та GroupJoin.

У запитах LINQ до DataSet обробляються перерахування об'єктів типу DataRow, що містяться як в типізованих, так і нетипізованих об'єктах DataSet.

У запиті вказують, яку інформацію потрібно вибрати із джерела або джерел даних. У разі необхідності у запиті також зазначають спосіб сортування, групування й формування даних перед поверненням.

Запит зберігається у змінній запиту й ініціюється виразом запиту.

Запитання і завдання

1. Яке призначення має технологія LINQ to DataSet?
2. Які об'єкти використовують в технології LINQ to DataSet?
3. Під час реалізації яких задач доцільно використовувати технологію LINQ to DataSet? Наведіть приклади.
4. Яку інформацію вказують у запитах LINQ to DataSet?

6.4. Вираз запиту

Спочатку слід розглянути такий приклад запиту до DataSet:

Приклад 6.1. Запит до DataSet.

```
EnumerableRowCollection<DataRow> query =
       from товар in tableТовари.AsEnumerable()
       where товар.Field<int>("Код_групи") == 1
       orderby товар.Field<decimal>("Ціна") descending
       select товар;
```

У ньому сказано, що вибирають дані з локальної таблиці tableТовари. В результаті залишають тільки дані про товари з групи, код якої дорівнює 1, і вони йдуть у порядку зменшення ціни. Запит зберігають у змінній запиту query, що є перерахованою колекцією рядків таблиці (EnumerableRowCollection<DataRow>).

У прикладі вираз запиту містить такі речення: from, where, orderby і select. Порядок, в якому записані речення, є протилежним до порядку, якого дотримуються у мові SQL, де спочатку вказують речення Select,

а потім From. Зазначені в прикладі речення мають такий самий сенс, як і вмові SQL:

from – вказує джерело даних,

where – містить умову фільтрування,

orderby – застосовує сортування,

select – визначає тип елементів, що повертаються.

Треба дати визначення основних понять, що пов'язані із запитами LINQ.

Запит – це набір інструкцій, що описують, які дані необхідно вибрати із зазначеного джерела (або джерел) даних, а також визначають форму й організацію даних, що вибирають.

Вираз запиту – це запит, який записано за допомогою синтаксису запиту. На відміну від синтаксису запиту у LINQ також використовують синтаксис методів, який розглядається у подальшому.

Вираз запиту складається з **набору речень**, що написані у декларативному синтаксисі, подібному до SQL.

Кожне **речення**, у свою чергу, містить один або кілька виразів C#, які є виразами запиту або можуть містити вираз запиту.

Змінна запиту – це будь-яка змінна, що зберігає запит замість результатів запиту. Вона завжди має перерахований тип.

Тип змінної запиту визначається типом елементів у реченні select.

Можна також використовувати ключове слово **var** для задавання анонімного типу даних. У цьому разі компілятор самостійно визначає тип змінної запиту під час компіляції.

Приклад 6.2. Використання анонімного типу у запиті прикладу 6.1.

```
var query =
       from товар in tableТовари.AsEnumerable()
       where товар.Field<int>("Код_групи") == 1
       orderby товар.Field<decimal>("Ціна") descending
       select товар;
```

Під час побудови запиту LINQ потрібно дотримуватися таких правил:

1. Вираз запиту повинен починатися реченням **from** і закінчуватися реченням **select** або **group**.

2. Між першим реченням **from** і останнім реченням **select** або **group** може міститися одне або кілька необов'язкових речень **where**, **orderby**, **join**, **let** або навіть додаткових речень **from**.

3. Можна використовувати ключове слово **into**, щоб результат речення **join** або **group** слугував джерелом додаткових речень запиту в тому самому виразі запиту.

Запитання і завдання

1. У чому полягає різниця між поняттями "запит" і "вираз запиту"?
2. Яку інформацію містить змінна запиту?
3. У яких випадках доцільно використовувати анонімний тип?
4. Яких правил слід дотримуватися під час побудови запиту LINQ?

6.5. Речення запиту

Далі розглянуто призначення і синтаксис кожного виду речення у запиті LINQ.

from. Починає вираз запиту і визначає:

джерело даних;

локальну змінну діапазону, що становить кожний елемент вихідної послідовності.

Виконання подібне до ітерації в інструкції foreach.

Приклад 6.3. Речення from.

```
from товар in tableТовари.AsEnumerable()
```

select. Закінчує вираз запиту. Результатом його виконання є або послідовність із тим самим типом об'єктів, що й в об'єктів, які містяться в джерелі даних (наприклад, select товар), або перетворена послідовність вихідних даних (проекція).

Приклад 6.4. Речення select (проекція).

```
select new
{
   Дата = продаж.Field<Datetime>("Дата"),
   Товар = товар.Field<string>("Товар"),
   Ціна = товар.Field<decimal>("Ціна"),
   Кількість = продаж.Field<Int16>("Кількість"),
   Вартість = товар.Field<decimal>("Ціна") *
```

```
                продаж.Field<Int16>("Кількість")
};
```

group. Закінчує вираз запиту. Використовують для одержання послідовності груп, що утворюються на основі зазначеного ключа. Ключем можуть бути дані будь-якого типу.

Приклад 6.5. Речення group.

```
var продажі =
    from продаж in tableПродажі.AsEnumerable()
    group продаж by продаж.Field<int>("Код_товару");
```

into. Ключове слово into використовують у реченні select або group для створення тимчасового ідентифікатора, у якому зберігається запит. Застосовують, коли потрібно виконати в запиті додаткові операції після групування або вибору.

Приклад 6.6. Ключове слово into.

```
var продажі =
from продаж in tableПродажі.AsEnumerable()
group продаж by продаж.Field<int>("Код_товару")into g
select new
{
Код_товару =g.Key,
Кількість = g.Sum(p =>p.Field<Int16>("Кількість"))
};
```

where. Використовують для фільтрування елементів із джерела даних за однією або кількома умовами.

Приклад 6.7. Речення where.

```
where
(товар.Field<decimal>("Ціна")>=decimal.Parse(textBoxВід.Text))
&&
(товар.Field<decimal>("Ціна")<=decimal.Parse(textBoxДо.Text))
```

orderby. Вживають для сортування результатів у порядку зростання (ascending) або спадання (descending). Ключове слово ascending є необов'язковим.

Приклад 6.8. Речення orderby.

```
orderby товар.Field<decimal>("Ціна") descending
```

join. Застосовують для встановлення зв'язку елементів одного джерела даних з елементами іншого джерела на основі порівняння на рівність певних ключів у кожному елементі. Після об'єднання двох послідовностей необхідно використати речення select або group, щоб указати елемент для збереження у вихідній послідовності.

Приклад 6.9. Речення join.

```
var query =
from товар in tableТовари.AsEnumerable()
    join продаж in продажіon товар.Field<int>("Код_товару") equals
 продаж.Код_товару
selectnew
    {
        Товар = товар.Field<string>("Товар"),
        Кількість = продаж.Кількість
    };
```

let. Використовують для збереження результатів виконання якоїсь складової частини виразу, щоб уживати його в наступних реченнях.

Приклад 6.10. Речення let.

```
from товар in tableТовари.AsEnumerable()
let price = товар.Field<decimal>("Ціна")
where
    (price >= decimal.Parse(textBoxВід.Text))
&&
    (price <= decimal.Parse(textBoxДо.Text))
```

Примітка. Порівняйте з прикладом 6.7.

Вкладений запит. Речення запиту саме може містити вираз запиту, який називають вкладеним запитом. Кожний вкладений запит починається власним реченням from, яке може вказувати на джерело даних, що відмінне від джерела першого речення from.

Приклад 6.11. Вкладений запит.

```
var query =
 from продаж in tableПродажі.AsEnumerable()
   group продаж by продаж.Field<DateTime>("Дата") into Дні
```

```
 select new
    {
        Дата = Дні.Key,
Відпущено = (from день in Дні
        select (int)день.Field<Int16>("Кількість")).Sum()
    };
```

Запитання і завдання

1. Які речення запиту вам відомі? Перерахуйте їх.

2. Чи можна записати вираз запиту, у якому немає речень from і select? Обґрунтуйте відповідь.

3. Запишіть речення, в якому задають дворівневе сортування даних локальної таблиці Товари спочатку за стовпцем Ціна_закупівлі (за спадання), а потім – стовпцем Товар (за алфавітом).

4. У чому полягає різниця між можливостями речення join в LINQ і аналогічним у SQL?

5. У яких випадках використовують вкладені запити? Наведіть приклади.

6.6. Синтаксис методів

Вирази запитів записують за допомогою декларативного синтаксису запитів. Однак у самому середовищі CLR .NET відсутнє поняття синтаксису запиту. Тому під час компіляції вирази запитів перетворюються у виклики методів для джерела даних (послідовності).

Ці методи називаються стандартними операторами запитів і мають імена **Where**, **Select**, **GroupBy**, **Join**, **Max**, **Average** тощо.

Їх можна викликати безпосередньо, використовуючи синтаксис методів замість синтаксису запитів. Методи викликають за допомогою оператора крапки.

Приклад 6.12. Запит, що записаний за допомогою виразу запитів.

```
EnumerableRowCollection<DataRow> query =
    from товар in tableТовари.AsEnumerable()
    where товар.Field<int>("Код_групи") == 1
    orderby товар.Field<decimal>("Ціна") descending
    select товар;
```

Той самий запит подано за допомогою синтаксису методів.

```
EnumerableRowCollection<DataRow> query =
      tableТовари.AsEnumerable().
      Where(товар => товар.Field<int>("Код_групи") == 1).
      OrderByDescending(товар =>товар.Field<decimal>("Ціна"));
```

Примітка. У стандартних операторах Where та OrderByDescending застосовано лямбда-вирази.

Для таких операцій запитів, як Count, Sum, Average, Min або Max, немає відповідних речень у виразах запитів, тому їх необхідно записувати за допомогою виклику методів.

Приклад 6.13. Обчислення сумарної кількості проданих товарів.

```
int Сума = продажі.Sum(p =>p.Кількість);
```

Синтаксис виразів запитів і синтаксис методів можна поєднувати один з одним різними способами (див. приклади 6.6 та 6.11).

Порівнюючи синтаксис виразів запитів із синтаксисом методів, можна прийти до таких висновків:

1. Під час компіляції вираз запитів перетвориться у виклики методів стандартних операторів запиту.
2. Будь-який запит, який можна записати за допомогою синтаксису запитів, також можна подати за допомогою синтаксису методів. Однак синтаксис запитів у більшості випадків зрозуміліший й лаконічніший.
3. Між цими двома видами подання запитів немає відмінностей ні в семантиці, ні в продуктивності.
4. У ході створення LINQ-запитів рекомендується використовувати синтаксис запитів скрізь, де це можливо, і синтаксис методів, якщо це необхідно.

Запитання і завдання

1. У чому полягає синтаксис методів подання запиту? У яких випадках його доцільно використовувати? Наведіть приклади.
2. Що становлять стандартні оператори запитів? Наведіть приклади таких операторів.
3. Якому виду синтаксису подання запиту слід надавати перевагу? Обґрунтуйте відповідь.
4. Запишіть запити, що подані в прикладах 6.6 та 6.9, використавши синтаксис методів.

6.7. Відкладені й негайні операції

Під час налагодження програми, коли виконують покрокове виконання, можна переглядати результати обчислень. Тоді можна зауважити, що результати виконання одних запитів LINQ стають відомими відразу у точці, де оголошено запит, а в інших запитах – згодом. Це явище пов'язане з негайним і відкладеним виконанням.

Негайне виконання означає читання джерела даних і виконання операції в тій точці коду, де оголошено запит.

Відкладене виконання означає, що операція не виконана в тій точці коду, де оголошено запит. Він виконується тільки після перерахування змінної запиту, наприклад, у таких випадках:

в операторі foreach,

перетворенні до DataView,

прив'язці до BindingSource.

Якщо змінна запиту перераховується багаторазово, щоразу можуть повертатися різні результати.

Стандартні оператори запитів відрізняються за часом виконання залежно від того, повертають вони одноелементне значення чи послідовність значень.

Якщо використовують метод, що повертає одноелементне значення (наприклад, Average і Sum), запит виконуються негайно.

Методи, що повертають послідовність елементів, відкладають виконання запиту й повертають об'єкт, що перераховується. Такі запити виконуються під час безпосереднього звертання до елементів послідовності, що утворюються в результаті.

Запитання і завдання

1. У чому полягає різниця між негайним і відкладеним виконанням запитів LINQ? Чим вона зумовлена?
2. За якими ознаками можна стверджувати, що запит LINQ виконується негайно?
3. За допомогою яких засобів можна негайно виконати запит LINQ, якщо його результатом є послідовність елементів.

6.8. Типізовані DataSet

Якщо схема об'єкта DataSet відома під час розробки застосування, під час його створення рекомендується використовувати типізований об'єкт DataSet.

Типізований об'єкт DataSet представляє клас, що є похідним від DataSet. Тому він успадковує всі методи, події й властивості класу DataSet.

Крім того, типізований об'єкт DataSet надає методи, події й властивості зі строгою типізацією. Це означає, що доступ до таблиць і стовпців можна одержати за іменами, не використовуючи методи на основі колекцій.

Технологія LINQ to DataSet підтримує запити і до типізованих об'єктів DataSet. Під час запису запиту LINQ можна використовувати всі переваги іменування, які надають типізовані об'єкт DataSet. Це спрощує запити й покращує їхнє читання. В цьому разі немає необхідності використовувати універсальний метод Field або метод SetField для доступу до даних стовпців. Імена властивостей доступні під час компіляції, тому що відомості про тип містяться в об'єкті DataSet.

Запит LINQ to DataSet надає доступ до значень стовпців правильного типу, тому помилки невідповідності типів виявляються вже під час компіляції, а не під час виконання.

Приклад 6.14. Запит LINQ до типізованого DataSet.

```
var query =
   from товар in хлібСерверDataSet.Товари.AsEnumerable()
   where товар.Код_групи == 1
   orderby товар.Ціна descending
   select товар;
```

Запитання і завдання

1. У чому полягає різниця між запитами LINQ до типізованих і нетипізованих об'єктів DataSet?
2. Які переваги мають запити LINQ до типізованих DataSet?
3. Запишіть запити, що подані в прикладах 6.6 та 6.9, використавши типізований DataSet з ім'ям хлібСерверDataSet.

Лабораторна робота № 6. Розробка програм на основі технології LINQ to DataSet

Цілі лабораторної роботи:

1. Набуття практичних навичок формування LINQ-запитів до DataSet.
2. Вироблення вмінь та навичок групування даних і обчислення агрегованих величин у технології LINQ to DataSet.
3. Опанування операцій об'єднання даних кількох таблиць та створення підзапитів.
4. Набуття практичних навичок побудови діаграм із використанням технології LINQ to DataSet.
5. Удосконалювання навичок роботи з інтегрованим середовищем розробки Visual C# і довідковою системою Microsoft Developer Network (MSDN).

Перед виконанням лабораторної роботи студент повинен знати:

1. Основи використання бібліотеки Windows Forms.
2. Основи побудови запитів мовою SQL.
3. Структуру DataSet і принципи роботи з його таблицями.
4. Принципи прив'язування даних до елементів інтерфейсу.

Після виконання лабораторної роботи студент повинен вміти:

1. Самостійно розробляти прості застосування для роботи з даними на основі технології LINQ to DataSet.
2. Використовувати основні бібліотеки .Net Framework при розробці програм.
3. Виконувати прості завдання аналізу даних.
4. Подавати дані, що містяться в DataSet, у вигляді діаграм.

Хід роботи

1. Підготовка застосування до використання технології LINQ to DataSet.
2. Сортування й фільтрація даних.
3. Групування даних. Обчислення агрегованих величин.

4. Об'єднання даних таблиць. Вкладені запити.
5. Аналіз даних із діаграмною візуалізацією.
6. Робота з індивідуальною базою даних.

Інструкції

Постановка загальної задачі

Вивчення засобів технології LINQ to DataSet проводиться шляхом розширення застосування ***типізованийХліб***, що створено у попередній лабораторній роботі. До нього додають функції аналізу даних на основі операцій сортування, фільтрування, групування і об'єднання. Результати подають у табличному вигляді та у вигляді діаграми.

Керування роботою застосування здійснюється за допомогою групи кнопок ***Аналіз***, що додаються на форму ***Хліб*** (рис. 6.2). Кнопки призначені для виклику відповідних функціональних форм.

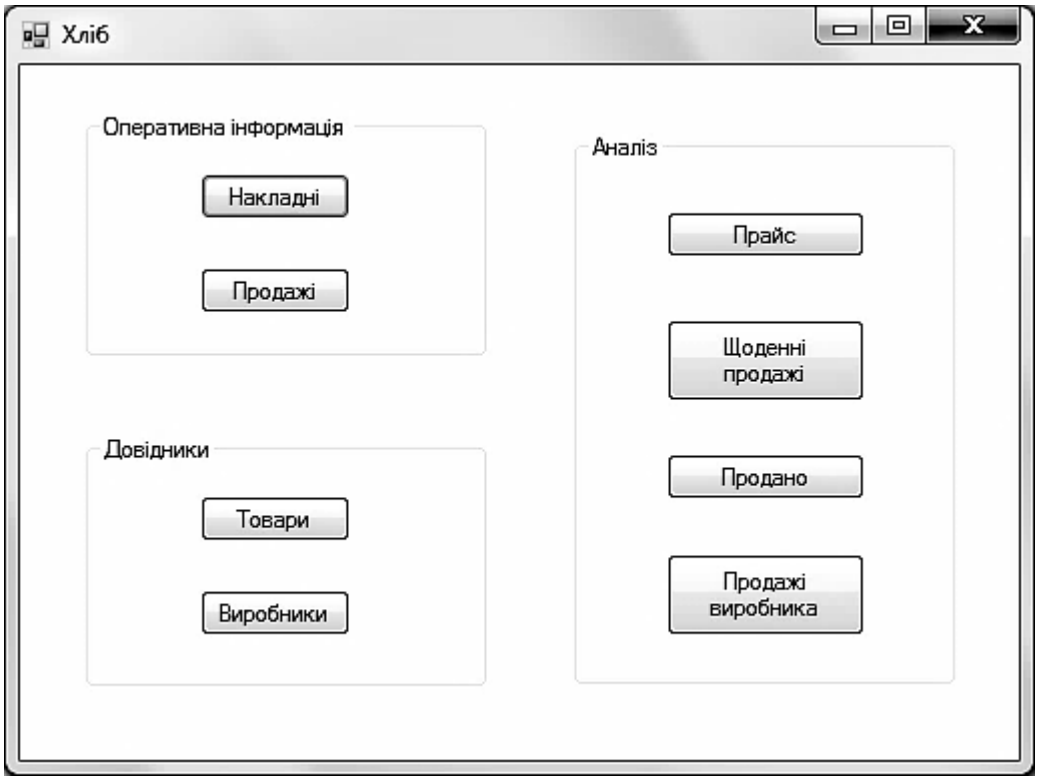


Рис. 6.2. **Кнопкова форма *Хліб* з групою кнопок *Аналіз***

На рис. 6.3 – 6.6 подано нові функціональні форми застосування. З їхньою допомогою можна переглядати дані, що містяться у таблицях бази даних ***ХлібСервер***. Далі розглянемо призначення цих форм.

Форма ***Прайс*** слугує для перегляду цін товарів. Причому можна відфільтрувати дані про товари, задавши діапазон цін (рис. 6.3).

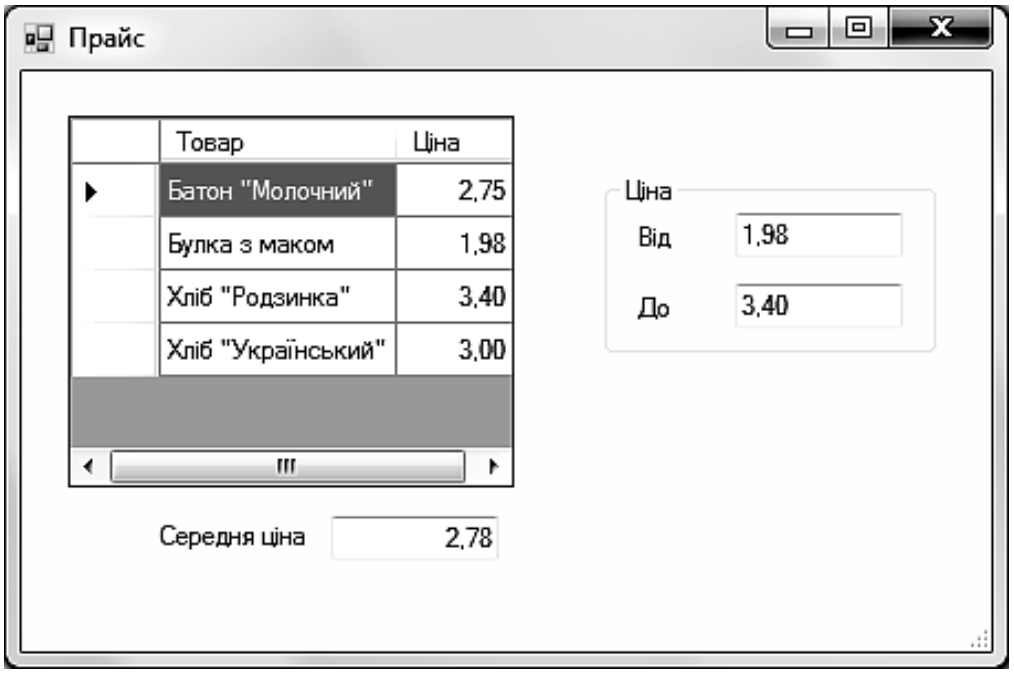


Рис. 6.3. **Форма *Прайс***

На формі ***Щоденні продажі*** подають кількість одиниць усіх товарів, які продано за відповідний день (рис. 6.4).

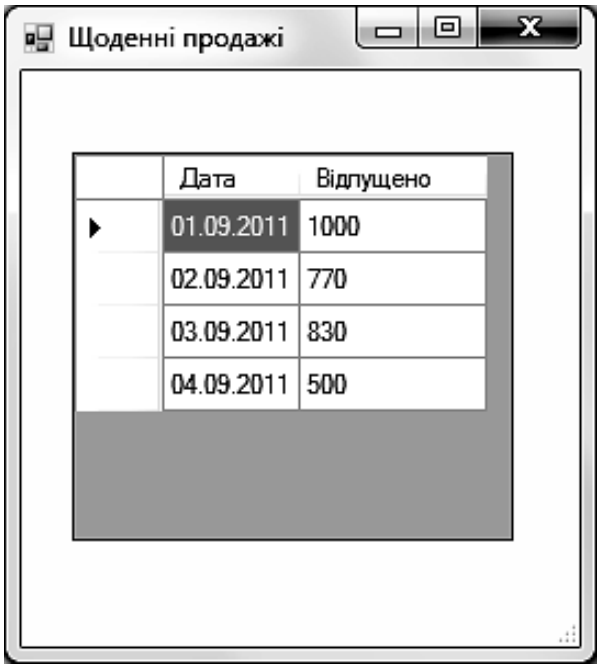


Рис. 6.4. **Форма *Щоденні продажі***

Форма ***Продано*** використовується для аналізу того, як продавалися товари різних видів (рис. 6.5). Ліворуч подано список товарів, у яких величина ***Кількість*** більша нуля, а праворуч – усіх товарів. В останньому видно, які товари зовсім не продавалися (***Кількість = 0***).

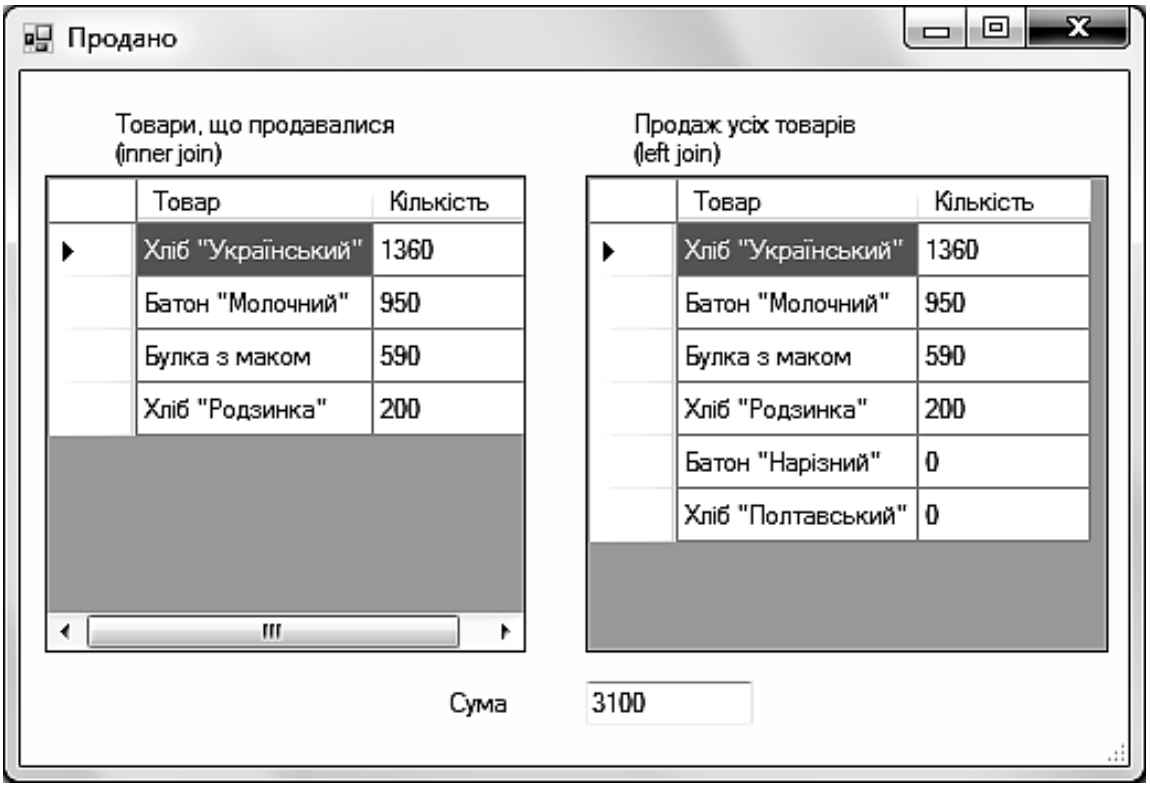


Рис. 6.5. **Форма *Продано***

Форма ***Продажі виробника*** застосовується для аналізу продажів щодо обраного виробника. Тут окрім табличного подання даних використовується графічне. Із кругової діаграми можна дізнатися, яку частину займає кожний товар у загальній сумі продажів обраного виробника (рис. 6.6).

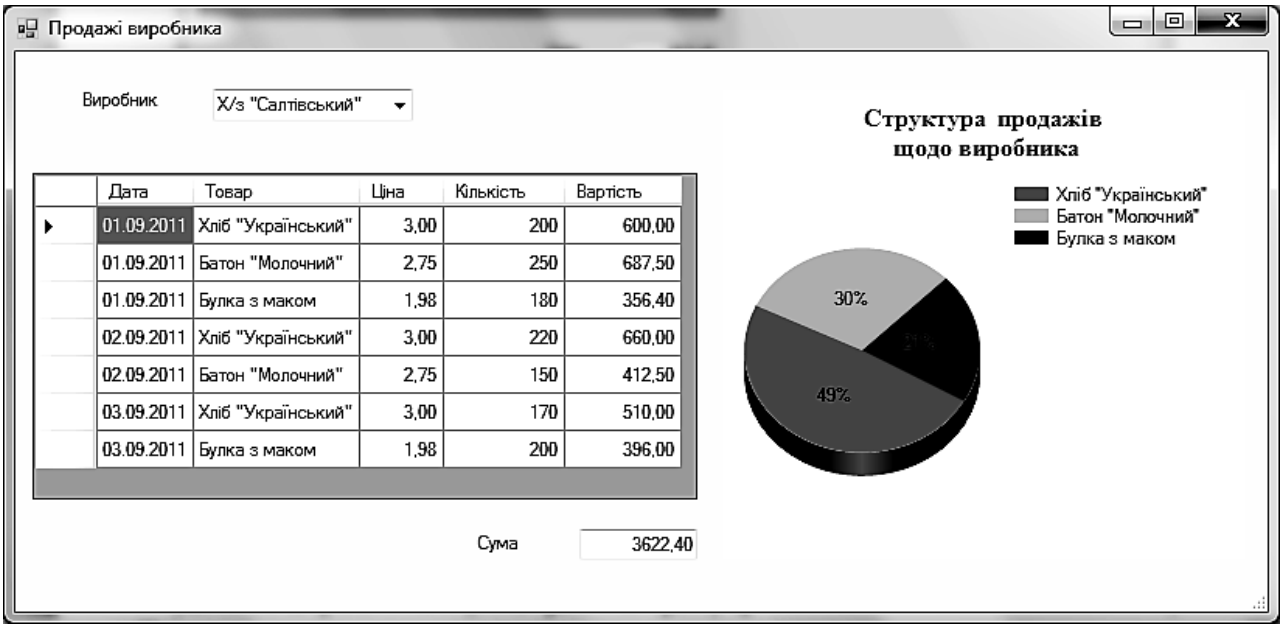


Рис. 6.6. **Форма *Продажі виробника***

1. Підготовка застосування до використання технології LINQ to DataSet

Завдання

Скопіювати застосування для подальшої роботи і додати на кнопкову форму в ньому групу кнопок ***Аналіз***.

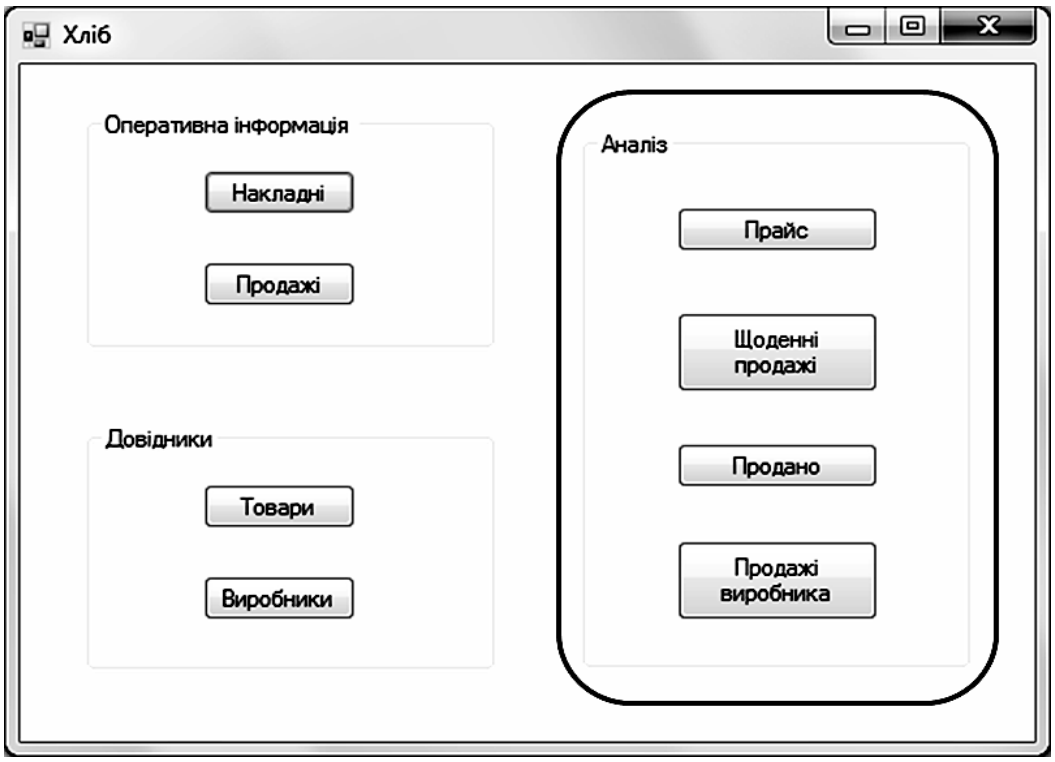


Рис. 6.7. **Група кнопок *Аналіз* на кнопковій формі *Хліб***

Виконання

1. Створіть нову папку з ім'ям ***linqХліб*** і скопіюйте в неї проект ***типізованийХліб***, що створено у лабораторній роботі № 5.

2. Відкрийте проект ***типізованийХліб***, що знаходиться у папці ***linqХліб***. Запустіть його на виконання і перевірте функціональність кнопок, що містяться на формі ***Хліб***. Потім зніміть застосування з виконання.

3. Перейдіть у вікно конструктора форми ***Хліб***.

4. Додайте на форму елемент керування **GroupBox** і задайте значення ***Аналіз*** для його властивості **Text**.

5. Додайте чотири кнопки усередину елемента ***Аналіз*** й установіть для них такі значення властивостей:

Кнопка	Властивість	Значення
1	Text	Прайс
	Name	buttonПрайс
2	Text	Щоденні продажі
	Name	buttonЩоденніПродажі
3	Text	Продано
	Name	buttonПродано
4	Text	Продажі виробника
	Name	buttonПродажіВиробника

6. Збережіть зміни, що зроблені в проекті.

2. Сортування й фільтрація даних (форма Прайс)

Постановка задачі

Вивчення засобів сортування та фільтрації даних як елемента аналізу проведено на прикладі форми ***Прайс***. Вона слугує для перегляду цін на товари (рис. 6.8).

В елементі DataGridView відображаються два стовпці даних із таблиці ***Товари***. Причому дані відсортовані за полем ***Товар***. Нижче в елементі TextBox відображається результат обчислення середньої ціни. Праворуч у групі ***Ціна*** подано мінімальну й максимальну ціну. Вони використовуються як значення за замовчуванням, щоб під час завантаження форми відображалися дані про всі товари. Тут можна задати нові межі діапазону цін, відповідно до яких будуть відбиратися товари в елементі DataGridView (фільтрація).

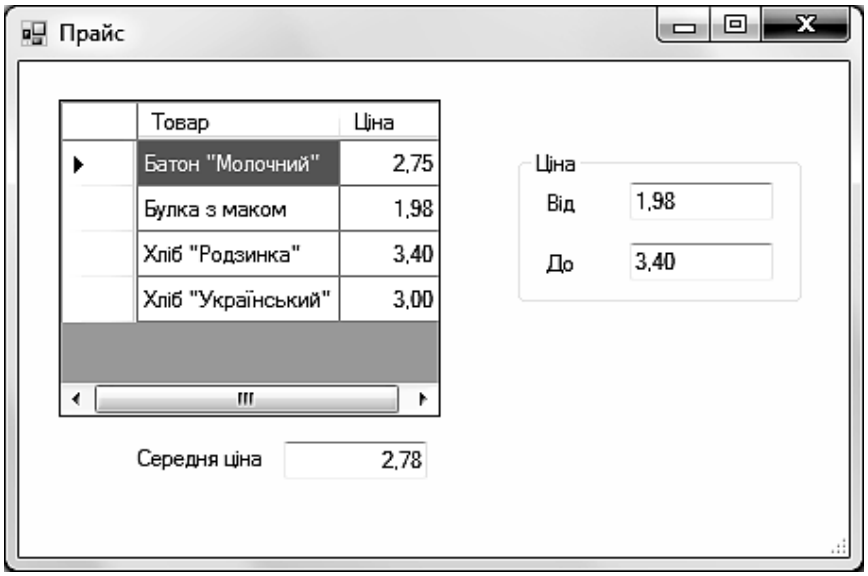


Рис. 6.8. **Форма *Прайс***

Завдання 1

Додати в застосування форму ***Прайс***, у якій будуть відображатися назви й ціни товарів (рис. 6.9).

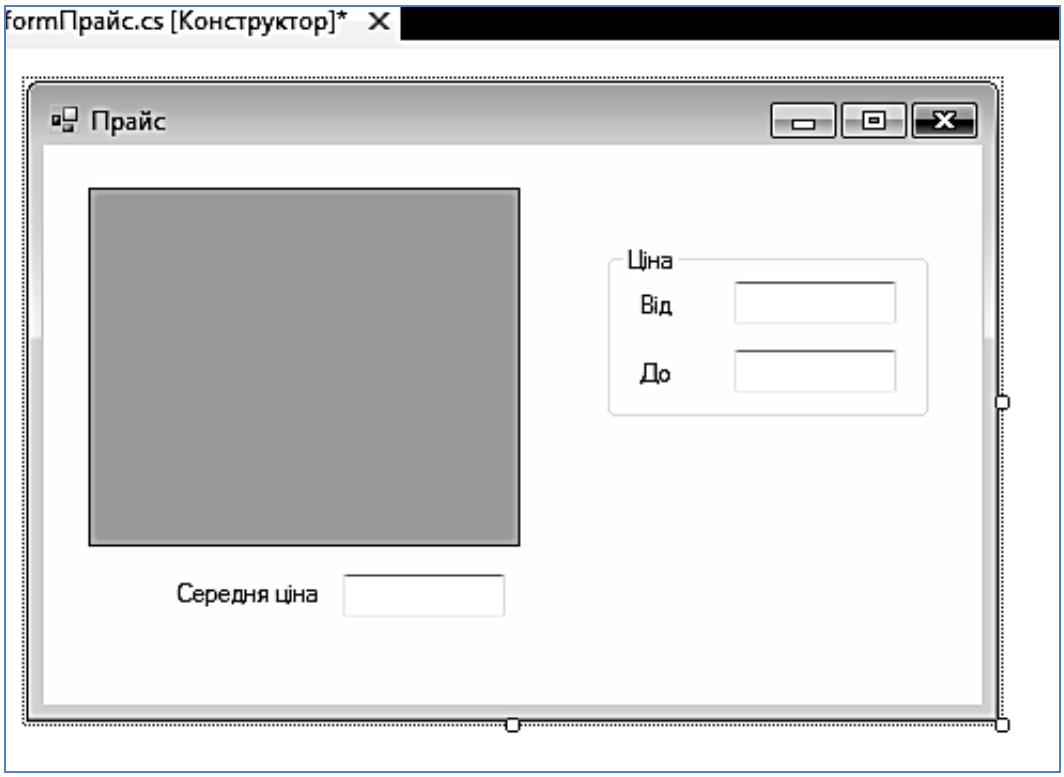


Рис. 6.9. **Форма *Прайс*у вікні конструктора**

Виконання

1. Додайте в застосування нову форму з ім'ям файла ***form-Прайс.cs*** і значенням ***Прайс*** властивості **Text**.

2. Для відображення даних про товари додайте на форму елемент **DataGridView** зі значенням ***dataGridViewПрайс*** властивості **Name**.

3. Для подання середньої ціни товарів під елементом DataGridView розмістіть елементи **Label** і **TextBox**. Для властивості **Text** елемента **Label** задайте значення ***Середня ціна***, а для властивості **Name** елемента **TextBox** – значення ***textBoxСередня_ціна***.

4. Для встановлення меж діапазону цін праворуч від елемента DataGridView додайте елемент **GroupBox** зі значенням ***Ціна*** для властивості **Text**.

5. Усередині елемента GroupBox розмістіть два елементи **Label** зі значеннями ***Від*** і ***До*** для властивості **Text**, а праворуч – по одному елементу **TextBox** зі значеннями ***textBoxВід*** та ***textBoxДо*** для властивості **Name** відповідно.

6. Двічі клацніть у вільному місці форми ***Прайс*** й уведіть оператори тіла оброблювача події ***formПрайс_Load***. Він має такий вигляд:

```
private void formПрайс_Load(object sender, EventArgs e)
{
    // Створюємо типізований набір даних
    ХлібСерверDataSet хлібСерверDataSet = new ХлібСерверDataSet();

    // Заповнюємо таблицю Товари
    ХлібСерверDataSetTableAdapters.ТовариTableAdapter
       товариTableAdapter =
       new ХлібСерверDataSetTableAdapters.ТовариTableAdapter();
    товариTableAdapter.Fill(хлібСерверDataSet.Товари);

    // Задаємо значення за замовчанням
    if (textBoxВід.Text == "")
    {
        decimal Від =
        (from товар in хлібСерверDataSet.Товари.AsEnumerable()
        select товар.Ціна).Min();
        textBoxВід.Text = Від.ToString("0.00");
    }

    if (textBoxДо.Text == "")
    {
        decimal До =
        (from товар in хлібСерверDataSet.Товари.AsEnumerable()
        select товар.Ціна).Max();
        textBoxДо.Text = До.ToString("0.00");
    }

    // Готуємо дані для DataGridView і серед. ціни
    var query =
        from товар in хлібСерверDataSet.Товари.AsEnumerable()
        let price = товар.Ціна
        where (price >= decimal.Parse(textBoxВід.Text)) &&
            (price <= decimal.Parse(textBoxДо.Text))
        orderby товар.Товар
        select new
        {
            Товар = товар.Товар,
            Ціна = товар.Ціна
        };
    // Відображаємо дані
    BindingSource bindingТовари = new BindingSource();
    bindingТовари.DataSource = query;
    dataGridViewПрайс.DataSource = bindingТовари;
    // Форматування виведених даних
```

```
    dataGridViewПрайс.AutoResizeColumns();
    dataGridViewПрайс.Columns["Ціна"].DefaultCellStyle.Format="0.00";
    dataGridViewПрайс.Columns["Ціна"].DefaultCellStyle.Alignment =
        DataGridViewContentAlignment.MiddleRight;

    // Середня ціна
    decimal Середня_ціна = query.Average(t => t.Ціна);
    textBoxСередня_ціна.Text = Середня_ціна.ToString("0.00");
    textBoxСередня_ціна.TextAlign = HorizontalAlignment.Right;
}
```

7. Перейдіть у вікно конструктора форми ***Хліб***, двічі клацніть кнопку ***Прайс*** і додайте код оброблювача події "Клацання на кнопці Прайс".

```
private void buttonПрайс_Click(object sender, EventArgs e)
{
    formПрайс вікноПрайс = new formПрайс();
    вікноПрайс.ShowDialog();
}
```

8. Перевірте функціональність форми ***Прайс***.
9. Збережіть зміни, що зроблені в проекті.

Завдання 2

Додати функцію фільтрації даних у разі зміни значень в елементах ***Від*** і ***До***.

Ідея розв'язку

Оскільки обробка події "Зміна значення" призведе до багаторазового виклику оброблювача після додавання кожного символу, більш раціонально обробляти подію "Втрата фокуса". Вона виникає тільки один раз після закінчення зміни значення в елементі TextBox. У якості оброблювача можна використати наявний оброблювач події "Завантаження форми" (функція formПрайс_Load).

Виконання

1. Перейдіть у вікно конструктора форми ***Прайс*** і клацніть поле ***Від*** (елемент ***textBoxВід***).
2. У вікні властивостей відобразіть перелік подій, клацнувши кнопку **События**  .

3. Установіть значення ***formПрайс_Load*** для властивості **Leave**, вибравши його зі списку, що розкривається.

4. Повторіть п.п. 1 – 3 для поля ***До*** (елемент ***textBoxДо***).

5. Запустіть програму на виконання й перевірте функціональність полів ***Від*** і ***До***.

6. Збережіть зміни, що зроблені в проекті.

7. У висновках з лабораторної роботи вкажіть, чи був сенс використовувати технологію LINQ to Dataset в оброблювачі події ***formПрайс_Load***, чи можна це було зробити з використанням класичної технології.

3. Групування даних. Обчислення агрегованих величин (форма Щоденні продажі)

Постановка задачі

Вивчення засобів відбору даних у групи за значеннями певного поля і виконання обчислень у кожній групі буде проведено на прикладі форми ***Щоденні продажі***. Вона слугує для перегляду кількості одиниць усіх товарів, які продано за відповідний день (рис. 6.10).

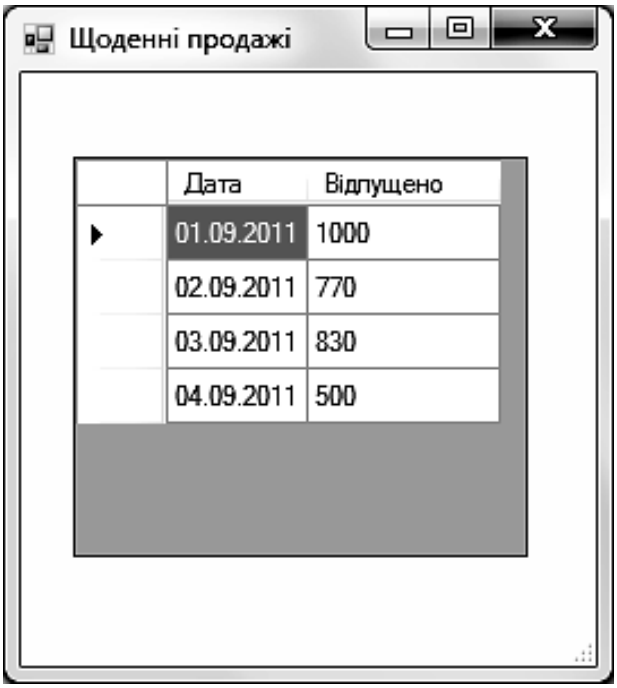


Рис. 6.10. **Форма *Щоденні продажі***

В елементі DataGridView відображаються два стовпці даних із таблиці ***Продажі***. У першому стовпці відображаються дати, в які здійснювався продаж товарів, а в другому – кількість одиниць товарів, які відпуще-

но у відповідний день. Тобто результати продажів зібрано у групи за значеннями поля ***Дата*** і в кожній групі виконано обчислення суми значень поля ***Кількість***.

Завдання

Додати в застосування форму ***Щоденні продажі***, у якій будуть відображатися дати й кількість одиниць товарів, які продано у відповідний день.

Виконання

1. Додайте в застосування нову форму з ім'ям файла ***formЩоденніПродажі.cs*** і значенням ***Щоденні продажі*** для властивості **Text**.

2. Для відображення даних про товари додайте на форму елемент **DataGridView** зі значенням ***dataGridViewЩоденніПродажі*** властивості **Name**.

3. Двічі клацніть у вільному місці форми ***Щоденні продажі*** й уведіть оператори тіла оброблювача події ***formЩоденніПродажі_Load***. Оброблювач має такий вигляд:

```
private void formЩоденніПродажі_Load(object sender, EventArgs e)
{
    // Створюємо типізований набір даних
    ХлібСерверDataSet хлібСерверDataSet = new ХлібСерверDataSet();

    // Заповнюємо таблицю Продажі
    ХлібСерверDataSetTableAdapters.ПродажіTableAdapter
        продажіTableAdapter =
        new ХлібСерверDataSetTableAdapters.ПродажіTableAdapter();
    продажіTableAdapter.Fill(хлібСерверDataSet.Продажі);

    // Готуємо дані для DataGridView
    var query =
        from продаж in хлібСерверDataSet.Продажі.AsEnumerable()
        group продаж by продаж.Дата into Дні
        select new
        {
            Дата = Дні.Key,
            Відпущено = (from день in Дні
            select (int)день.Кількість).Sum()
        };
    // Відображаємо дані
    BindingSource bindingПродажі = new BindingSource();
```

```
    bindingПродажі.DataSource = query;
    dataGridViewЩоденніПродажі.DataSource = bindingПродажі;
    dataGridViewЩоденніПродажі.AutoResizeColumns();
}
```

4. Перейдіть у вікно конструктора форми ***Хліб***, двічі клацніть кнопку ***Щоденні продажі*** і додайте код оброблювача події "Клацання на кнопці Щоденні продажі".

```
private void buttonЩоденніПродажі_Click(object sender, EventArgs e)
{
    formЩоденніПродажі вікноЩоденніПродажі= new formЩоденніПродажі();
    вікноЩоденніПродажі.ShowDialog();
}
```

5. Перевірте функціональність форми ***Щоденні продажі***.
6. Збережіть зміни, що зроблені в проекті.

4. Об'єднання даних таблиць. Вкладені запити (форма Продано)

Постановка задачі

Вивчення запитів, у яких виконуються об'єднання записів двох таблиць різними способами, буде проведено на прикладі форми ***Продано***. Вона слугує для аналізу результатів продажів за кожним видом товарів (рис. 6.11).

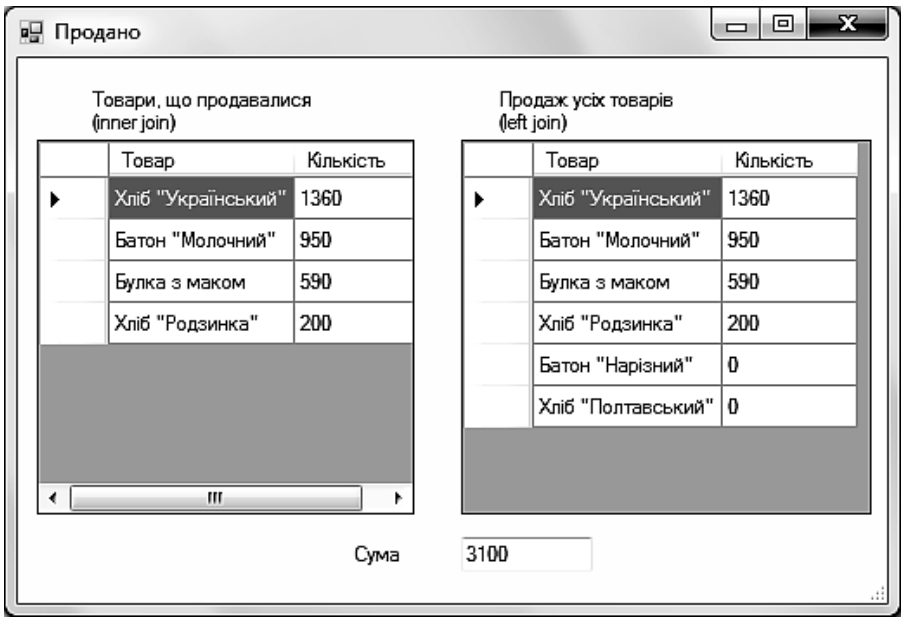


Рис. 6.11. **Форма *Продано***

Ідеї розв'язку

Спочатку слід створити перший запит (підсумковий), у якому обчислюється кількість проданих товарів кожного виду. Він будується на основі даних таблиці ***Продажі***. Цей запит складається із двох полів – ***Код_товару*** й ***Кількість***.

Щоб на формі відображати не коди товарів, а їхні назви, треба побудувати другий запит. У ньому об'єднаємо дані таблиці ***Товари*** й результати першого запиту за полем ***Код_товару***. З таблиці ***Товари*** взяти поле ***Товар***, а з першого запиту – ***Кількість***.

Об'єднання буде зроблено двома способами. У першому способі в результат другого запиту потрапляють тільки ті записи, у яких збігаються значення полів ***Код_товару***, тобто стануть відомі результати продажів тих товарів, про які є дані в таблиці ***Продажі*** (inner join). Якщо ж за якимись товарами не було продажів, вони не потраплять у результуючий список.

У другому способі об'єднання в результат другого запиту потрапляють усі записи з таблиці ***Товари***, навіть ті, які не продавалися (left join).

Для обчислення загальної суми кількості проданих товарів достатньо результатів першого запиту.

Завдання

Додати в застосування форму ***Продано***. У ній будуть відображатися дані про кількості продажів тільки тих товарів, які продавалися, а також усіх товарів, які можуть постачатися в кіоск.

Виконання

1. Додайте в застосування нову форму з ім'ям файла ***formПродано.cs***і значенням ***Продано*** властивості **Text**.

2. Для відображення даних про кількості проданих товарів додайте на форму два елементи **DataGridView**. Першому дайте ім'я ***dataGridViewПродано***, а другому – ***dataGridViewПродано2***.

3. Над кожним елементом DataGridView помістіть напис (елемент Label). У властивість Text першого напису введіть значення

Товари, що продавалися
(inner join)
а другого –

Продаж усіх товарів
(left join).

4. Для подання загальної суми кількості проданих товарів під елементами DataGridView розмістіть елементи **Label** і **TextBox**. Для властивості **Text** елемента **Label** задайте значення ***Сума***, а для властивості ***Name*** елемента **TextBox** – ***textBoxСума***.

5. Двічі клацніть у вільному місці форми ***Продано*** й уведіть оператори тіла оброблювача події ***formПродано_Load***. Оброблювач має такий вигляд:

```
private void formПродано_Load(object sender, EventArgs e)
{
    // Створюємо типізований набір даних
    ХлібСерверDataSet хлібСерверDataSet = new ХлібСерверDataSet();

    // Заповнюємо таблицю Товари
    ХлібСерверDataSetTableAdapters.ТовариTableAdapter товариTableAdapter
    =newХлібСерверDataSetTableAdapters.ТовариTableAdapter();
    товариTableAdapter.Fill(хлібСерверDataSet.Товари);

    // Заповнюємо таблицю Продажі
    ХлібСерверDataSetTableAdapters.ПродажіTableAdapterпродажіTableAdapter
    = newХлібСерверDataSetTableAdapters.ПродажіTableAdapter();
    продажіTableAdapter.Fill(хлібСерверDataSet.Продажі);

    // Обчислюємо сумарну кількість продажі в покожному виду товарів
    // (черезКод_товару)
    var продажі = fromпродажінхлібСерверDataSet.Продажі.AsEnumerable()
       groupпродажбупродаж.Код_товаруintog
       select new
       {
           Код_товару = g.Key,
           Кількість = g.Sum(p => p.Кількість)
       };

    // Додаємо назви товарів для кожної кількості (inner join)
    var query = from товар in хлібСерверDataSet.Товари.AsEnumerable()
    join продаж in продажі
        on товар.Код_товару equals продаж.Код_товару
        select new
        {
            Товар = товар.Товар,
            Кількість = продаж.Кількість
        };

    // Відображаємо дані
```

```
    BindingSource bindingТовари = new BindingSource();
    bindingТовари.DataSource = query;
    dataGridViewПродано.DataSource = bindingТовари;
    dataGridViewПродано.AutoResizeColumns();

    // Додаємо назви товарів для кожної кількості (left join)
    var query2 = from товар in хлібСерверDataSet.Товари.AsEnumerable()
         join продаж in продажі
         on товар.Код_товару equals продаж.Код_товару into pg
        from t in pg.DefaultIfEmpty(new { Код_товару = 0, Кількість= 0 })
         select new
         {
             Товар = товар.Товар,
             Кількість = t.Кількість
         };
// Відображаємо дані
    BindingSource bindingТовари2 = new BindingSource();
    bindingТовари2.DataSource = query2;
    dataGridViewПродано2.DataSource = bindingТовари2;
    dataGridViewПродано2.AutoResizeColumns();

    //Сума
    int Сума = продажі.Sum(p => p.Кількість);
    textBoxСума.Text = Сума.ToString();
}
```

6. Перейдіть у вікно конструктора форми ***Хліб***, двічі клацніть кнопку ***Продано*** і додайте код оброблювача події "Клацання на кнопці Продано".

```
private void buttonПродано_Click(object sender, EventArgs e)
{
    formПродано вікноПродано = new formПродано();
    вікноПродано.ShowDialog();
}
```

7. Перевірте функціональність форми ***Продано***. Для цього:

7.1. Запустіть на виконання застосування.

7.2. Викличте форму ***Товари***, введіть дані про два нових товари і збережіть їх у базі даних. Потім закрийте форму ***Товари***.

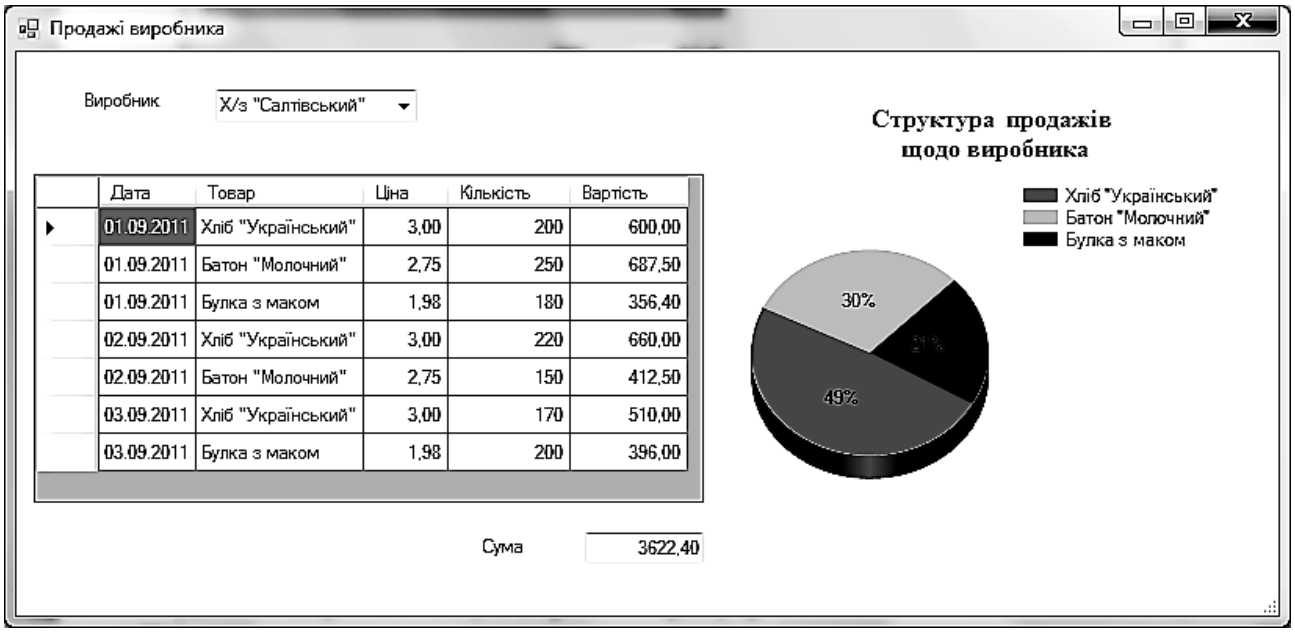
7.2. Викличте форму ***Продано*** і переконайтеся в тому, що у правому списку з'явилися назви доданих товарів з нульовими значеннями у стовпці ***Кількість***.

8. Збережіть зміни, що зроблені в проекті.

5. Аналіз даних із діаграмною візуалізацією (форма Продажі виробника)

Постановка задачі

Вивчення запитів для інтерактивного аналізу даних, у яких відображаються дані про обраного виробника, буде проведено на прикладі форми ***Продажі виробника***. У ній відображаються дані в табличній формі й у вигляді об'ємної кругової діаграми (рис. 6.12).



Дата	Товар	Ціна	Кількість	Вартість
01.09.2011	Хліб "Український"	3,00	200	600,00
01.09.2011	Батон "Молочний"	2,75	250	687,50
01.09.2011	Булка з маком	1,98	180	356,40
02.09.2011	Хліб "Український"	3,00	220	660,00
02.09.2011	Батон "Молочний"	2,75	150	412,50
03.09.2011	Хліб "Український"	3,00	170	510,00
03.09.2011	Булка з маком	1,98	200	396,00

Рис. 6.12. **Форма *Продажі виробника***

Ідеї розв'язку

Спочатку слід створити поле зі списком ***Виробник***. Обране в ньому значення використовується як умова відбору даних у запиті. Його дані будуть відображатися в елементах DataGridView і Chart. На основі цього запиту також обчислюється загальна сума вартостей проданих товарів обраного виробника.

Для забезпечення інтерактивності обробляється подія **SelectedIndexChanged** для поля зі списком ***Виробник***.

Побудову форми можна виконувати в такому порядку. Спочатку створити форму з текстовими даними, а потім додати діаграму.

Після введення коду обох оброблювачів подій слід протестувати форму, спостерігаючи за синхронним відображенням даних у елементах DataGridView та Chart.

Завдання 1

Додати в застосування форму ***Продажі виробника***. У ній будуть відображатися текстові дані про продажі товарів обраного виробника (рис. 6.13).

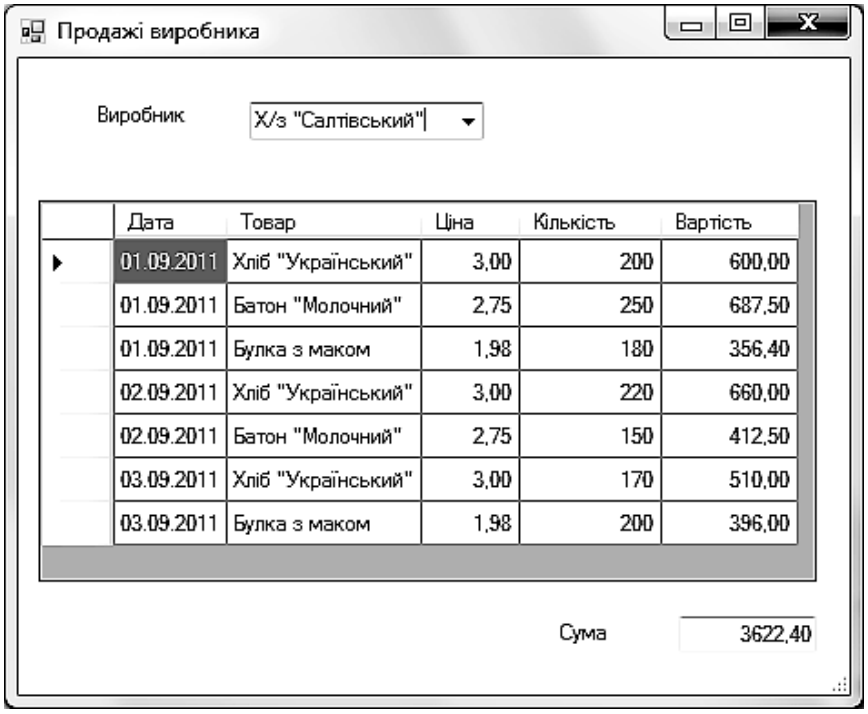


Рис. 6.13. **Форма *Продажі виробника* з текстовими даними**

Виконання

1. Додайте в застосування нову форму з ім'ям файла ***formПродажіВиробника.cs*** і значенням ***Продажі виробника*** властивості **Text**.

2. Для відображення списку виробників додайте на форму елемент ComboBox, а ліворуч від нього – елемент Label. Для властивості **Text** елемента **Label** задайте значення ***Виробник***, а для властивості **Name** елемента **ComboBox** – значення ***comboBoxВиробник***.

3. Для відображення результатів продажів про товари обраного виробника додайте на форму елемент **DataGridView** з ім'ям ***dataGridViewПродажіВиробника***.

4. Для подання загальної суми вартостей проданих товарів обраного виробника під елементом DataGridView розмістіть елементи **Label** і **TextBox**. Для властивості **Text** елемента **Label** задайте значення ***Сума***, а для властивості **Name** елемента **TextBox** – значення ***textBoxСума***.

5. Двічі клацніть у вільному місці форми ***Продажі виробника*** й уведіть оператори тіла оброблювача події ***formПродажіВиробника_Load***. Він має такий вигляд:

```
// Спільні об'єкти
ХлібСерверDataSet хлібСерверDataSet = new ХлібСерверDataSet();
BindingSource bindingВиробники = new BindingSource();
BindingSource bindingПродажіВиробника = new BindingSource();
BindingSource bindingСтруктура = new BindingSource();

private void formПродажіВиробника_Load(object sender, EventArgs e)
{
    //*******************
    // comboBoxВиробник *
    //*******************

    // Заповнюємо таблицю Виробники
    ХлібСерверDataSetTableAdapters.ВиробникиTableAdapter
        виробникиTableAdapter = new
        ХлібСерверDataSetTableAdapters.ВиробникиTableAdapter();
    виробникиTableAdapter.Fill(хлібСерверDataSet.Виробники);

    // Відображаємо дані
    bindingВиробники.DataSource = хлібСерверDataSet.Виробники;
    comboBoxВиробник.DataSource = bindingВиробники;
    comboBoxВиробник.DisplayMember = "Виробник";
    comboBoxВиробник.ValueMember = "Код_виробника";

    //*******************************
    // dataGridViewПродажіВиробника *
    //*******************************

    // Заповнюємо таблицю Товари
    ХлібСерверDataSetTableAdapters.ТовариTableAdapter
        товариTableAdapter = new
        ХлібСерверDataSetTableAdapters.ТовариTableAdapter();
    товариTableAdapter.Fill(хлібСерверDataSet.Товари);

    // Заповнюємо таблицю Продажі
    ХлібСерверDataSetTableAdapters.ПродажіTableAdapter
        продажіTableAdapter = new
        ХлібСерверDataSetTableAdapters.ПродажіTableAdapter();
    продажіTableAdapter.Fill(хлібСерверDataSet.Продажі);

    // БудуємозапитзаданимитаблицьПродажі
    // і Товари для вибраного виробника
    var queryПродажіВиробника =
        from продаж in хлібСерверDataSet.Продажі.AsEnumerable()
```

```
        join товар in хлібСерверDataSet.Товари.AsEnumerable()
        on продаж.Код_товару equals товар.Код_товару
       where продаж.Код_виробника == (int)comboBoxВиробник.SelectedValue
        selectnew
        {
            Дата = продаж.Дата,
            Товар = товар.Товар,
            Ціна = товар.Ціна,
            Кількість = продаж.Кількість,
            Вартість = товар.Ціна * продаж.Кількість
        };
    // Відображаємо дані
    bindingПродажіВиробника.DataSource = queryПродажіВиробника;
    dataGridViewПродажіВиробника.DataSource = bindingПродажіВиробника;
    dataGridViewПродажіВиробника.AutoResizeColumns();
    // Форматуємо виведені дані (грошовий стиль)
    dataGridViewПродажіВиробника.Columns["Ціна"].DefaultCellStyle.Format
        = "0.00";
    dataGridViewПродажіВиробника.Columns["Ціна"].DefaultCellStyle.
    Alignment= DataGridViewContentAlignment.MiddleRight;
    dataGridViewПродажіВиробника.Columns["Вартість"].DefaultCellStyle.Format
        = "0.00";
    dataGridViewПродажіВиробника.Columns["Вартість"].DefaultCellStyle.
    Alignment = DataGridViewContentAlignment.MiddleRight;
    // ФорматуємовиведеніданістовпцяКількість (цілечисло)
    dataGridViewПродажіВиробника.Columns["Кількість"].
    DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleRight;
    //Сума
    decimalСума = queryПродажіВиробника.Sum(p =>p.Вартість);
    textBoxСума.Text = Сума.ToString("0.00");
    textBoxСума.TextAlign = HorizontalAlignment.Right;
}
```

6. Перейдіть у вікно конструктора форми ***Хліб***, двічі клацніть кнопку ***Продажі виробника*** і додайте код оброблювача події "Клацання кнопки Продажі виробника".

```
private void buttonПродажіВиробника_Click(object sender, EventArgs e)
{
    formПродажіВиробника вікноПродажіВиробника = new formПродажіВиробника();
    вікноПродажіВиробника.ShowDialog();
}
```

7. Перевірте функціональність форми ***Продажі виробника*** і збережіть зміни, що зроблені в проекті.

Завдання 2

Додати інтерактивність формі, щоб при виборі в полі зі списком іншого виробника, на формі відображалися відповідні дані.

Ідея розв'язку

Щоб при зміні виробника відображалися нові дані, слід обробити подію **SelectedIndexChanged** для поля зі списком ***Виробник***.

В оброблювачі перечитати запит з урахуванням нового значення в умові відбору й обновити прив'язки даних. Така обробка буде проводитися після повного завантаження форми, коли виконується умова

if (this.CanFocus).

Виконання

1. Перейдіть у вікно конструктора форми ***Продажі виробника*** й двічі клацніть поле зі списком ***Виробник***.

2. Уведіть оператори тіла оброблювача події ***comboBoxВиробник_SelectedIndexChanged***. Він має такий вигляд:

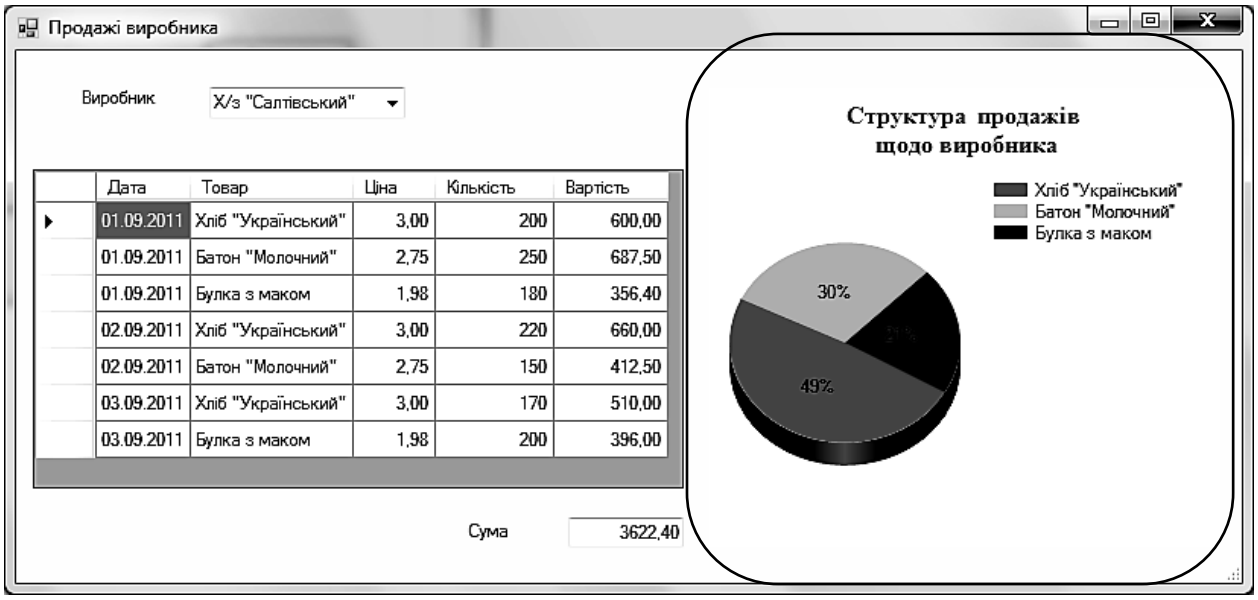
```
private void comboBoxВиробник_SelectedIndexChanged(object sender,
    EventArgs e)
{
    // Після завантаження
    if (this.CanFocus)
    {
        var queryПродажіВиробника =
        from продаж in хлібСерверDataSet.Продажі.AsEnumerable()
        join товар in хлібСерверDataSet.Товари.AsEnumerable()
        on продаж.Код_товару equals товар.Код_товару
        where продаж.Код_виробника =
           (int)comboBoxВиробник.SelectedValue
        select new
        {
            Дата = продаж.Дата,
            Товар = товар.Товар,
            Ціна = товар.Ціна,
            Кількість = продаж.Кількість,
            Вартість = товар.Ціна * продаж.Кількість
        };
        // Відображаємо дані
```

```
        bindingПродажіВиробника.DataSource = queryПродажіВиробника;
    }
}
```

3. Перевірте функціональність поля зі списком ***Виробник*** на формі ***Продажі виробника*** і збережіть зміни, що зроблені в проекті.

Завдання 3

Додати на форму ***Продажі виробника*** об'ємну кругову діаграму (рис. 6.14). У ній відображається структура продажів товарів обраного виробника (яку частину займає кожний вид товару у загальній сумі продажів його товарів).



	Дата	Товар	Ціна	Кількість	Вартість
▶	01.09.2011	Хліб "Український"	3,00	200	600,00
	01.09.2011	Батон "Молочний"	2,75	250	687,50
	01.09.2011	Булка з маком	1,98	180	356,40
	02.09.2011	Хліб "Український"	3,00	220	660,00
	02.09.2011	Батон "Молочний"	2,75	150	412,50
	03.09.2011	Хліб "Український"	3,00	170	510,00
	03.09.2011	Булка з маком	1,98	200	396,00

Рис. 6.14. **Форма *Продажі виробника* з круговою діаграмою**

Виконання

1. Перейдіть у вікно конструктора форми ***Продажі виробника*** й збільште її ширину приблизно на 12 см.

2. Додайте на форму елемент **Chart** і дайте йому ім'я ***chartСтруктура***.

3. Перейдіть у вікно коду форми і додайте таке посилання на простір імен:

```
//Для діаграми
using System.Windows.Forms.DataVisualization.Charting;
```

4. Перебуваючи у вікні коду форми, додайте в оброблювач події ***formПродажіВиробника_Load*** такий код, помістивши його перед фігурною дужкою оброблювача, яка закривається:

```
//************
// Діаграма *
//************

// Запит
var queryСтруктура =
from товар inqueryПродажіВиробника
group товар by товар.Товар into t
    select new
    {
        Товар = t.Key,
        Вартість = t.Sum(p => p.Вартість)
    };
    // Дані для відображення
    bindingСтруктура.DataSource = queryСтруктура;
    chartСтруктура.DataSource = bindingСтруктура;
    chartСтруктура.Series["Series1"].XValueMember = "Товар";
    chartСтруктура.Series["Series1"].YValueMembers = "Вартість";

// Тип диаграми
chartСтруктура.Series["Series1"].ChartType = SeriesChartType.Pie;

// Підписи на діаграмі
chartСтруктура.Series["Series1"].Label = "#PERCENT{P0}";
//"#VALX\n#PERCENT";

// Підписи як виноски
// this.chartСтруктура.Series["Series1"]["PieLabelStyle"] = "Outside";

// Об'ємний варіант (3D)
this.chartСтруктура.ChartAreas[0].Area3DStyle.Enable3D = true;

// Заголовок
chartСтруктура.Titles.Add("Заголовок");
chartСтруктура.Titles[0].Text= "Структура продажів\n щодо виробника";
chartСтруктура.Titles[0].Font
= new Font("Times New Roman", 12, FontStyle.Bold);
    chartСтруктура.Titles[0].ForeColor = Color.Red;

// Легенда
chartСтруктура.Series["Series1"].IsVisibleInLegend = true;
chartСтруктура.Series["Series1"].LegendText = "#VALX";//"#AXISLABEL";
```

Примітка. Деякі конструкції коду закоментовані. Вони можуть знадобитися під час побудови інших діаграм.

5. Перевірте можливість відображати діаграму після завантаження форми.

6. У вікні коду форми додайте такий код в оброблювач події comboBoxВиробник_SelectedIndexChanged, помістивши його перед фігурною дужкою оператора if, яка закривається:

```
//***********
// Діаграма *
//***********

// Запит
var queryСтруктура =
from товар inqueryПродажіВиробника
    group товар by товар.Товар into t
    select new
    {
        Товар = t.Key,
        Вартість = t.Sum(p => p.Вартість)
    };
// Дані для відображення
bindingСтруктура.DataSource = queryСтруктура;
if (queryСтруктура.Count() > 0)
{
    chartСтруктура.DataSource = bindingСтруктура;
    chartСтруктура.Series["Series1"].XValueMember = "Товар";
    chartСтруктура.Series["Series1"].YValueMembers = "Вартість";
    chartСтруктура.Series["Series1"].ChartType = SeriesChartType.Pie;
}
```

7. Перевірте функціональність поля зі списком ***Виробник*** щодо можливості змінювати діаграму. Прослідкуйте за тим, щоб на діаграмі відображалися дані, які відповідають тим, що подаються в елементі керування DataGridView.

8. Збережіть зміни, що зроблені в проекті.

9. У висновках з лабораторної роботи вкажіть, чи був сенс використовувати технологію LINQ to DataSet в оброблювачах подій форми ***Продажі виробника***, чи можна це було зробити з використанням класичної технології.

Завдання для самостійного виконання

1. За аналогією з формою ***Продано*** побудувати форму ***Поставки виробників***. У ній відобразити дані про вартість усіх товарів, які надійшли від кожного виробника (inner join та left join).

2. Додати стовпчикову діаграму на форму ***Продажі виробників***. На ній подати вартості проданих товарів обраного виробника за датами (на осі X). Бажано додати лінії тренда за кожним товаром.

3. Попередньо додати в базу даних таблицю ***Групи*** з полями ***Код_групи*** й ***Група*** й пов'язати її за зовнішнім ключем із таблицею ***Товари***. Після цього виконати такі завдання:

3.1. Додати поле зі списком ***Група*** на форму ***Прайс***. З його допомогою будуть відбиратися товари заданої групи.

3.2. Додати поле зі списком ***Група*** на форму ***Продажі***, щоб після вибору групи в списку ***Товар*** відображалися товари тільки обраної групи.

4. Додати на форму ***Накладні*** список ***Журнал накладних***, який дозволяє швидко знайти потрібну накладну. У списку передбачити такі стовпці: ***Код_накладної***, ***Дата*** відпуску товарів за накладною, ***Виробник***, від якого отримано товари та ***Сума*** вартостей усіх товарів за накладною. Значенням вибраного елемента списку є ***Код_накладної***.

5. У проекті з індивідуальною базою даних застосувати прийоми, які були розглянуті в базовій частині лабораторної роботи, а також у завданнях для самостійного виконання.

Висновки

1. LINQ (Language-Integrated Query) – це шаблони для запиту й зміни даних і технології, які можна розширити для підтримки джерела даних практично будь-якого типу.

2. Технології LINQ дозволяють будувати алгоритми для відбору даних у декларативний спосіб.

3. Використання технологій LINQ під час розробки застосувань з базами даних робить запити дуже зручною конструкцією мов C# і Visual Basic на відміну від традиційних запитів до даних, які виражаються у вигляді простих рядків без перевірки типів під час компіляції або підтримки IntelliSense, а також під час налагодження коду LINQ забезпечує покрокове виконання, встановлення точок зупинки та перегляд результатів у вікнах налагоджувальника.

4. LINQ є родиною технологій, що вбудовані у мови програмування C# та Visual Basic. Вона складається з таких технологій: LINQ to Objects, LINQ to XML, LINQ to DataSet, LINQ to SQL та LINQ to Entities. Три останні входять до групи технологій ADO.NET LINQ, які забезпечують виконання запитів із базами даних реляційного типу.

5. Технологія LINQ to DataSet спростила й прискорила написання запитів до даних в об'єкті DataSet, використовуючи конструкції мови, які подібні до тих, що вживаються у мові SQL.

6. Запити LINQ to DataSet виконуються до однієї чи кількох таблиць DataSet. В останньому разі використовують стандартні оператори запиту Join та GroupJoin.

7. Основними поняттями, що пов'язані із запитами LINQ, є такі: запит, вираз запиту та змінна запиту.

8. Тип змінної запиту визначається типом елементів у реченні select. Можна також використовувати ключове слово var для задавання анонімного типу даних. У цьому разі компілятор самостійно визначає тип змінної запиту під час компіляції.

9. Під час компіляції вирази запитів перетворюються у виклики методів для джерела даних (послідовності). Ці методи називаються стандартними операторами запитів і мають імена Where, Select, GroupBy, Join, Max, Average тощо. Їх можна викликати безпосередньо, використовуючи синтаксис методів замість синтаксису запитів. Методи викликають за допомогою оператора крапки.

10. При створенні LINQ-запитів рекомендується використовувати синтаксис запитів скрізь, де це можливо, а синтаксис методів – якщо це необхідно.

11. Негайне виконання означає читання джерела даних і виконання операції в тій точці коду, де оголошено запит, а відкладене виконання – операція не виконана в тій точці коду, де оголошено запит (виконується тільки після перерахування змінної запиту).

12. Технологія LINQ to DataSet підтримує запити як до нетипізованих, так і типізованих об'єктів DataSet. Під час запису запиту LINQ до типізованого DataSet можна використовувати всі переваги іменування. Це спрощує запити й покращує їхнє читання, дозволяє виявляти помилки невідповідності типів вже під час компіляції, а не під час виконання.

7. Платформа Entity Framework

7.1. Призначення платформи Entity Framework

Провідною концепцією сховищ даних є реляційні бази даних, розроблення застосувань – об'єктно орієнтоване програмування. Метою платформи Entity Framework є об'єднання цих концепцій в одну. Реалізація такого підходу дозволяє в коді оперувати з даними таблиць бази як з об'єктами програми без використання мови SQL.

Приклад 7.1. Встановлення нової ціни для товару "Булка з маком" в базі даних із використанням платформи Entity Framework.

```
ХлібEntities db = new ХлібEntities();
var товар=db.Products.Where(p => p.Товар == @"Булка з маком").First();
товар.Ціна = 3;
db.SaveChanges();
```

У прикладі 7.1 спочатку створюють екземпляр класу ХлібEntities, потім знаходять дані про товар "Булка з маком" і встановлюють нову ціну. В останньому рядку зберігають у базі даних зроблені зміни. Отже, всі дії з даними бази виконані на рівні коду програми.

З початками такого підходу відбулося ознайомлення під час вивчення технології LINQ to DataSet у попередньому розділі. Там оперування з даними, що знаходяться в типізованому DataSet, виконувалося на об'єктному рівні, але відрізок між DataSet і базою даних реалізовувався реляційними засобами (SQL-запитами) (рис. 7.1). Платформа Entity Framework спрямована на інкапсуляцію цього відрізка. При її використанні у розробника складається враження, що він оперує даними бази безпосередньо з коду програми (без використання проміжної ланки DataSet) (рис. 7.2).

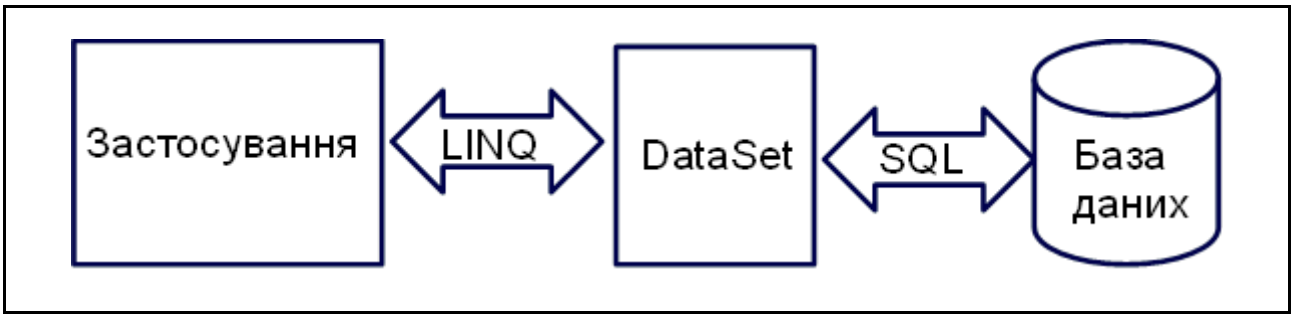


Рис. 7.1. **Доступ до даних у типізованому DataSet**

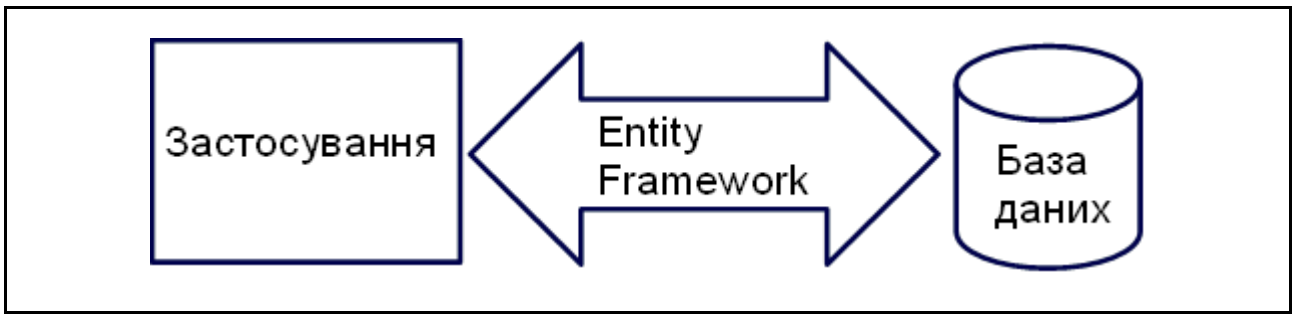


Рис. 7.2. **Доступ до даних в Entity Framework**

Платформа Entity Framework – це набір технологій ADO.NET, що забезпечують розробку застосувань, пов'язаних з обробкою даних на основі концепції об'єктно-реляційного відображення.

Запитання і завдання

1. Яке призначення має платформа Entity Framework?
2. У чому полягає різниця між технологією типізованого набору даних і платформою Entity Framework?
3. У яких випадках краще користуватися технологією типізованого набору даних, а в яких – платформою Entity Framework? Наведіть приклади.

7.2. Моделі й основні поняття в Entity Framework

Щоб реалізувати відображення між об'єктною моделлю і базою даних створюють сутнісну модель даних EDM (Entity Data Model). Вона складається із трьох компонентів – концептуальної моделі даних, моделі збереження (фізична модель) та опису відповідності між елементами кожної моделі (рис. 7.3).

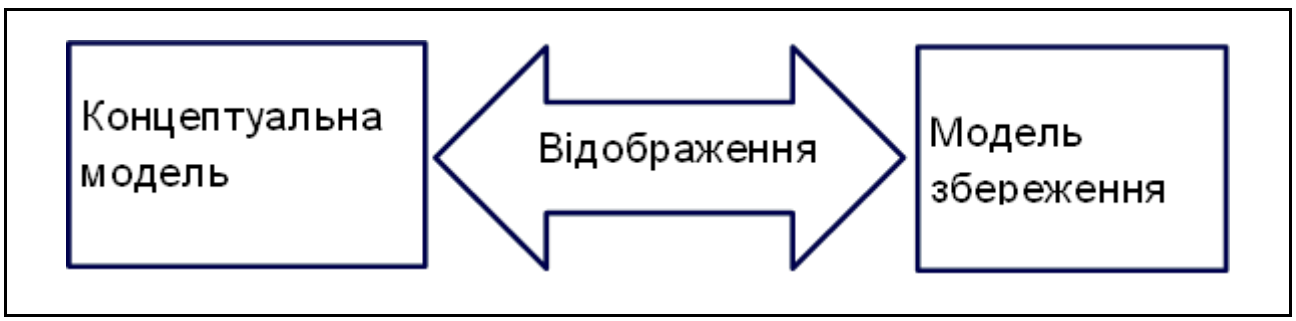


Рис. 7.3. **Сутнісна модель даних**

У концептуальній моделі визначають сутності і зв'язки між ними як класи, що використовують у програмі незалежно від того, як дані збері-

гаються у базі. Кожній сутності відповідає рядок таблиці в базі даних, а зв'язку (асоціації) – відношення між таблицями.

Концептуальну модель записують засобами мови CSDL (Conceptual Schema Definition Language), модель збереження даних – засобами мови SSDL (Store Schema Definition Language), а відображення однієї схеми на іншу – засобами мови MSL (Mapping Schema Language). Усі три мови є діалектами мови XML. У проекті Visual Studio всі вони зберігаються в одному XML-файлі, що має розширення EDMX. Цей файл генерується автоматично. Його структуру подано на рис. 7.4.

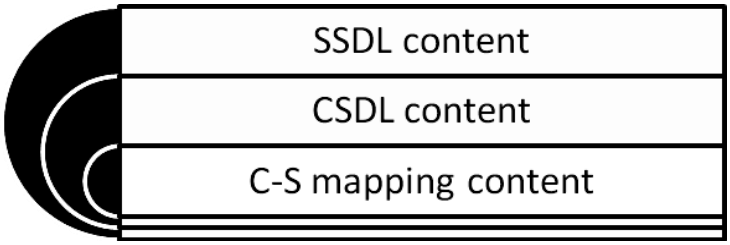


Рис. 7.4. **Структура EDMX-файла**

У табл. 7.1 подано відповідність термінів, що використовують в концептульній і фізичній моделях даних.

Таблиця 7.1

Терміни концептуальної і фізичної моделей

Концептульна модель	Фізична модель
Контекст об'єкта	База даних
Набір сутностей	Таблиця
Сутність	Рядок таблиці
Властивість	Стовпець таблиці
Асоціація	Відношення між таблицями

Далі розглянуто їх детальніше.

Контекст об'єкта – контейнер сутностей у концептуальній моделі. Він містить з'єднання з базою даних, відстежує зміни даних і забезпечує їхнє збереження у базі даних. Контекст об'єкта подається екземпляром класу ObjectContext.

Набір сутностей – логічний контейнер для сутностей одного типу (таблиці бази даних). Майстер побудови сутнісної моделі, яку буде не-

забаром розглянуто, має опцію підтримки імен таблиць у множині при створенні сутнісних класів. Наприклад, для таблиці ***Products*** формується сутність ***Product*** і набір сутностей ***Products***. Ця опція діє тільки для англомовних імен. Наприклад, для таблиці ***Продукти*** формується сутність ***Продукти*** і набір сутностей ***ПродуктиSet***.

Сутність – певний об'єкт бізнес-даних (наприклад, деякий товар чи виробник). Кожна сутність повинна мати унікальний **ключ** усередині набору сутностей. Сутність будується на основі шаблону, який називається типом сутності (аналог класу для екземплярів цього класу). У концептуальній моделі типи сутностей конструюються із **властивостей** і описують структуру основних концептуальних елементів верхнього рівня, таких, як товари чи виробники в бізнес-застосуванні.

Властивість – аналог властивості класу, що визначається ім'ям і типом даних (примітивним чи структурованим).

Асоціація – зв'язок між двома типами сутностей (такими, як товар і продажі). Кожна асоціація має дві кінцеві точки асоціації, які визначають типи сутностей, що беруть участь в асоціації. Кожна кінцева точка асоціації також визначає її кратність, яка може мати значення "один" (1), "нуль або один" (0..1) або "багато" (*). Сутності однієї кінцевої точки асоціації доступні за допомогою властивостей навігації (рис. 7.5).

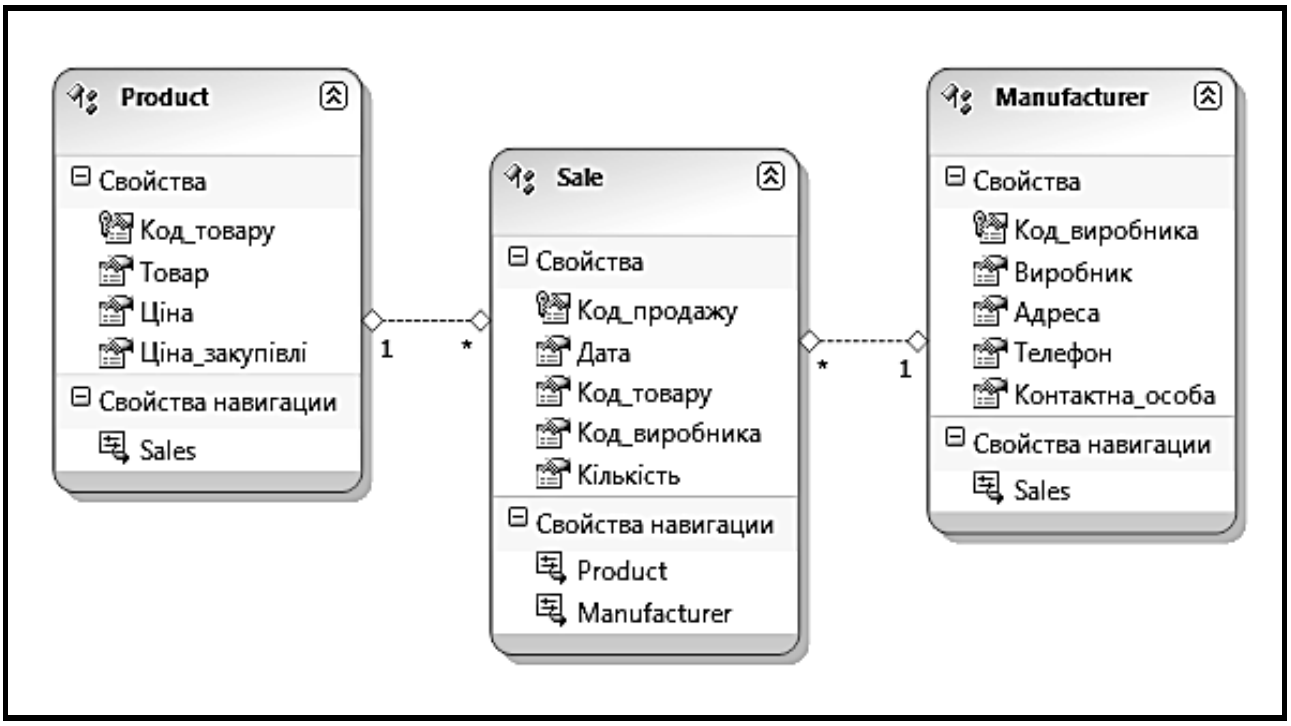


Рис. 7.5. **Сутнісна модель продажів**

Запитання і завдання

1. З яких компонентів складається модель даних EDM? Опишіть призначення кожного з них.

2. Які мови використовують для створення моделі даних EDM? Перерахуйте їх.

3. За рахунок чого можна подати компоненти моделі даних EDM, що написані різними мовами, в одному файлі?

4. Опишіть співвідношення термінів концептуальної і фізичної моделей.

7.3. Сценарії створення моделі EDM

Роботу з базою даних на платформі Entity Framework починають на основі одного з таких сценаріїв:

Model First;

DB First;

Code First.

Далі коротко розглядається кожний сценарій.

Model First – спочатку створюють модель EDM, а на її основі генерують SQL-скрипт для сторення бази даних (таблиць і з'язків між ними). Сценарій використовують, коли ще немає ні бази даних, ні моделі.

DB First – створюють модель EDM на основі існуючої бази даних. За трудозатратами цей сценарій є найлегшим.

Code First – спочатку записують код об'єктної (концептуальної) моделі, потім створюють модель EDM на основі бази даних без генерування коду і додають клас, що є нащадком від класу ObjectContext. Це найбільш трудозатратний сценарій, але він дає найефективніший код.

Далі подано алгоритми побудови моделі EDM за двома першими сценаріями. Третій сценарій викладено в розділі 8 навчального посібника. Більш детально можна ознайомитися з цими сценаріями у книгах [11 – 14].

Якщо база даних вже існує, для роботи з її даними в проект додають модель EDM. Для цього виконують таке (сценарій DB First):

1. Вибирають команду **Добавить новый элемент** в меню **Проект**. З'являється вікно **Добавление нового елемента**.

2. Вибирають шаблон **Данные** в списку **Установленные шаблоны** і елемент **Модель EDM ADO.NET** в центральному списку, а в ни-

жній частині вікна вводять ім'я моделі, потім клацають кнопку **Добавить** (рис. 7.6). З'являється перше вікно майстра моделі EDM.

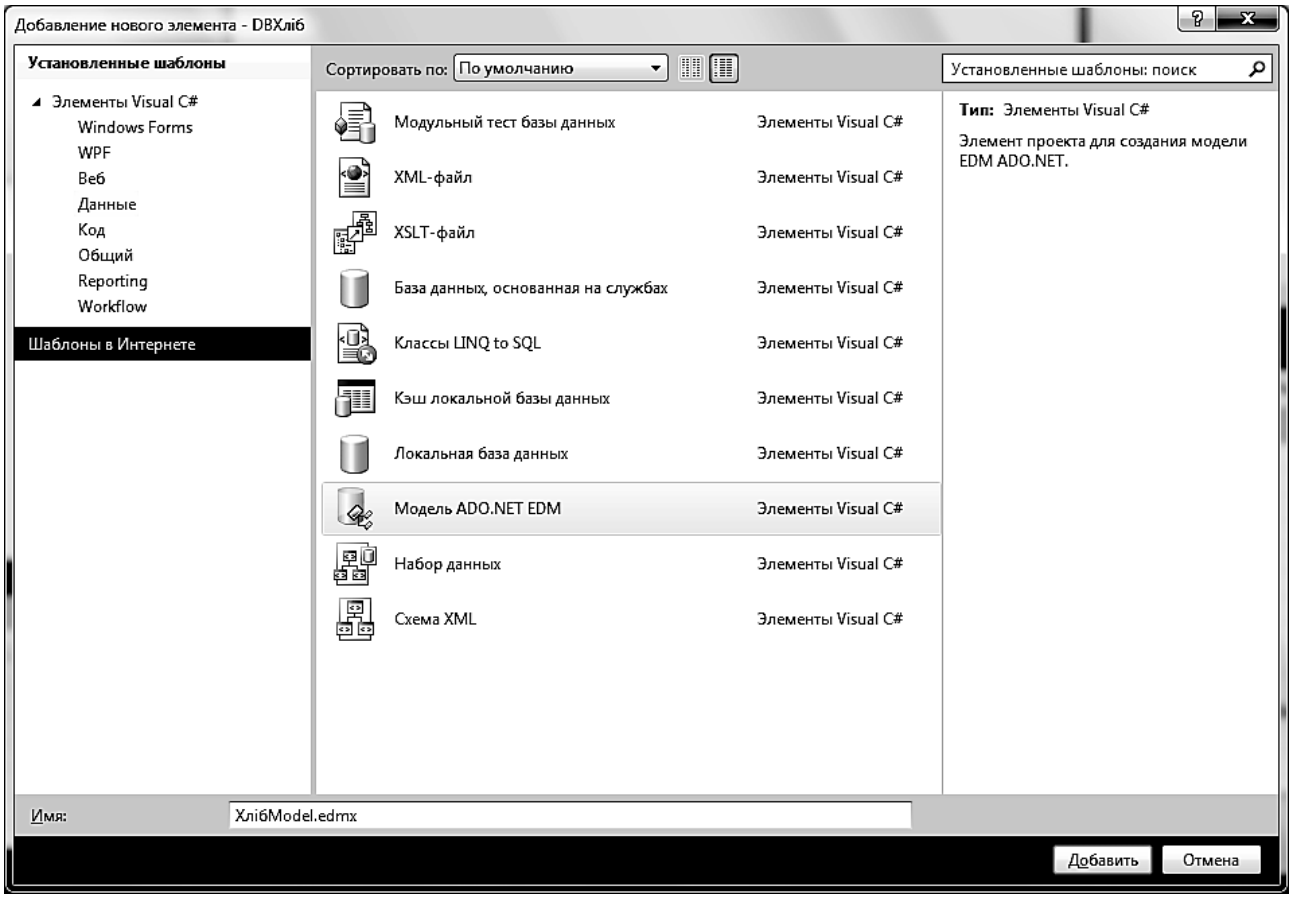


Рис. 7.6. **Вікно *Добавление нового элемента***

3. Вибирають значок **Создать из базы данных** у вікні **Выбор содержимого модели** і клацають кнопку **Далее** (рис. 7.7).

4. Клацають кнопку **Создать соединение** у вікні **Выбор подключения к данным**.

5. Вказують файл бази даних у вікні **Свойства подключения** і клацають кнопку **ОК**.

6. Повернувшись у вікно **Выбор подключения к данным**, клацають кнопку **Далее** і вирішують, чи копіювати файл бази даних у проект. З'явилося вікно **Выбор объектов базы данных**.

7. Вибирають об'єкти бази даних, що мають увійти до складу моделі EDM, і вмикають прапорець **Формировать имена объектов во множественном или единственном числе** (рис. 7.8). Потім клацають кнопку **Готово**.

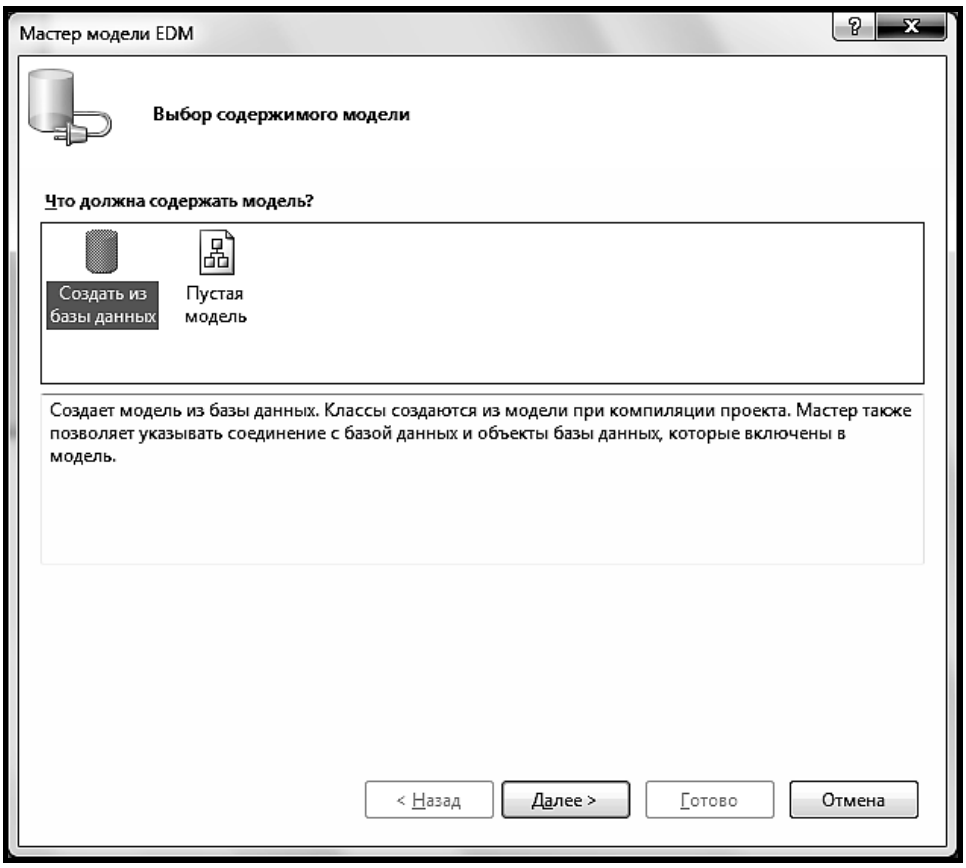


Рис. 7.7. **Вікно *Выбор содержимого модели***

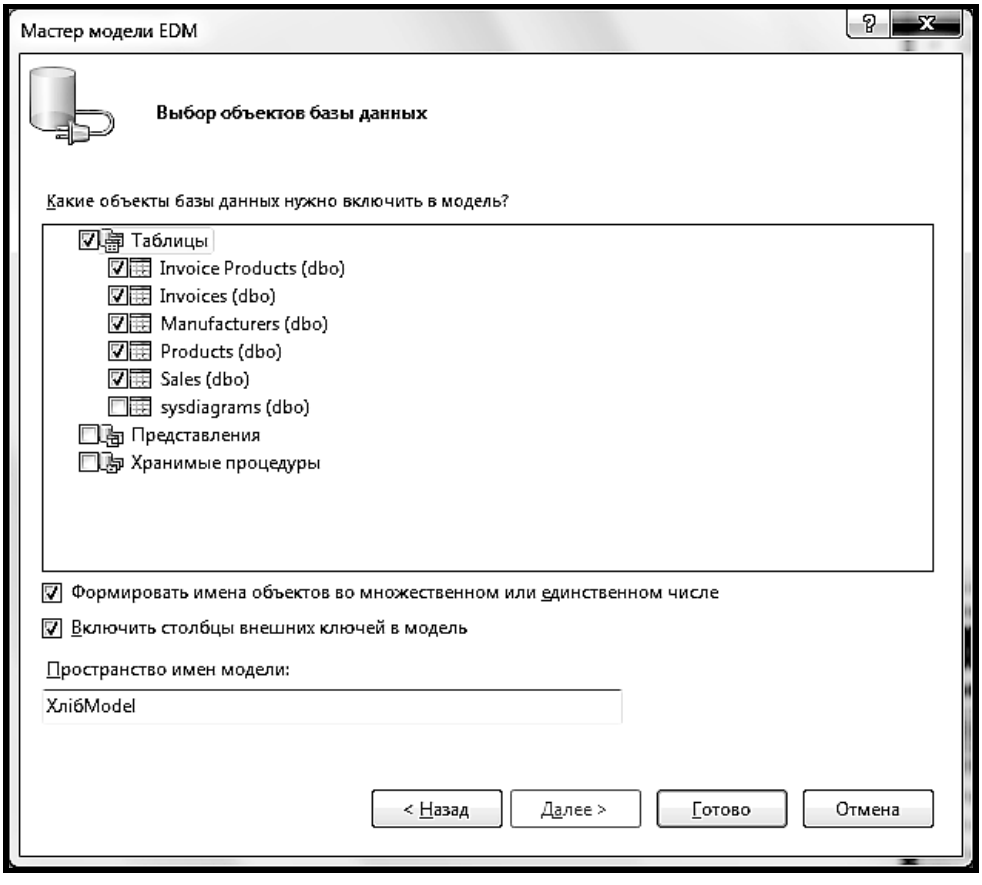


Рис. 7.8. **Вікно *Выбор объектов базы данных***

У результаті роботи майстра відкрилося вікно конструктора моделі EDM (рис. 7.9). У ньому можна переглядати елементи моделі, змінювати їх, додавати нові об'єкти й асоціації, а також вилучати зайві.

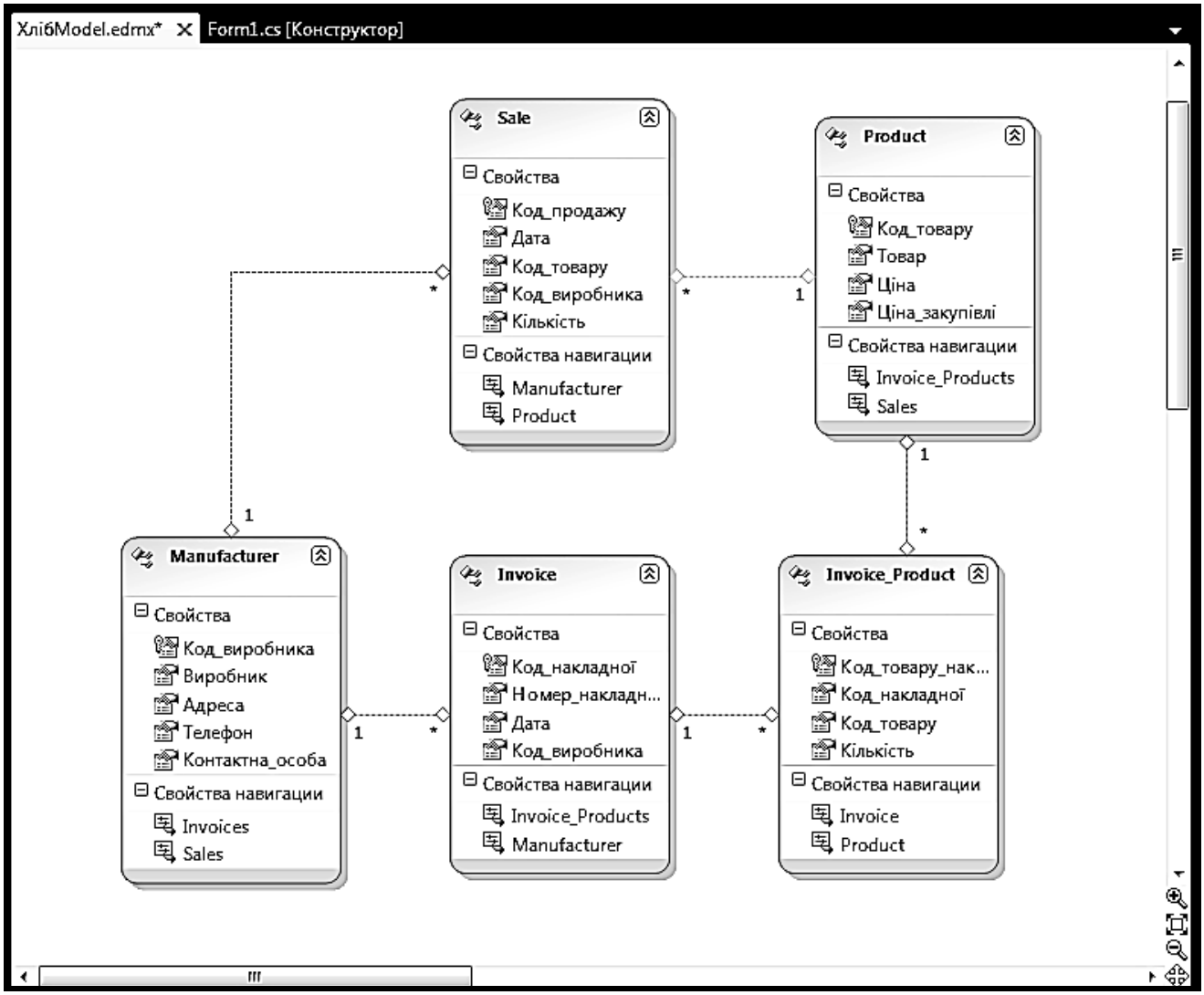


Рис. 7.9. **Вікно конструктора моделі EDM**

Якщо бази даних ще немає, спочатку створюють модель EDM, а на її основі – базу даних. Сценарій Model First здійснюється у такій послідовності:

1. Щоб створити порожню модель EDM виконують таке:

1.1. Вибирають команду **Добавить новый элемент** в меню **Проект**. З'являється вікно **Добавление нового элемента**.

1.2. Вибирають шаблон **Данные** в списку **Установленные шаблоны** і елемент **Модель EDM ADO.NET** в центральному списку, а в нижній частині вікна вводять ім'я моделі, потім клацають кнопку **Добавить**. З'являється перше вікно майстра моделі EDM.

1.3. Вибирають значок **Пустая модель** у вікні **Выбор содержимого модели** і клацають кнопку **Готово** (рис. 7.10).

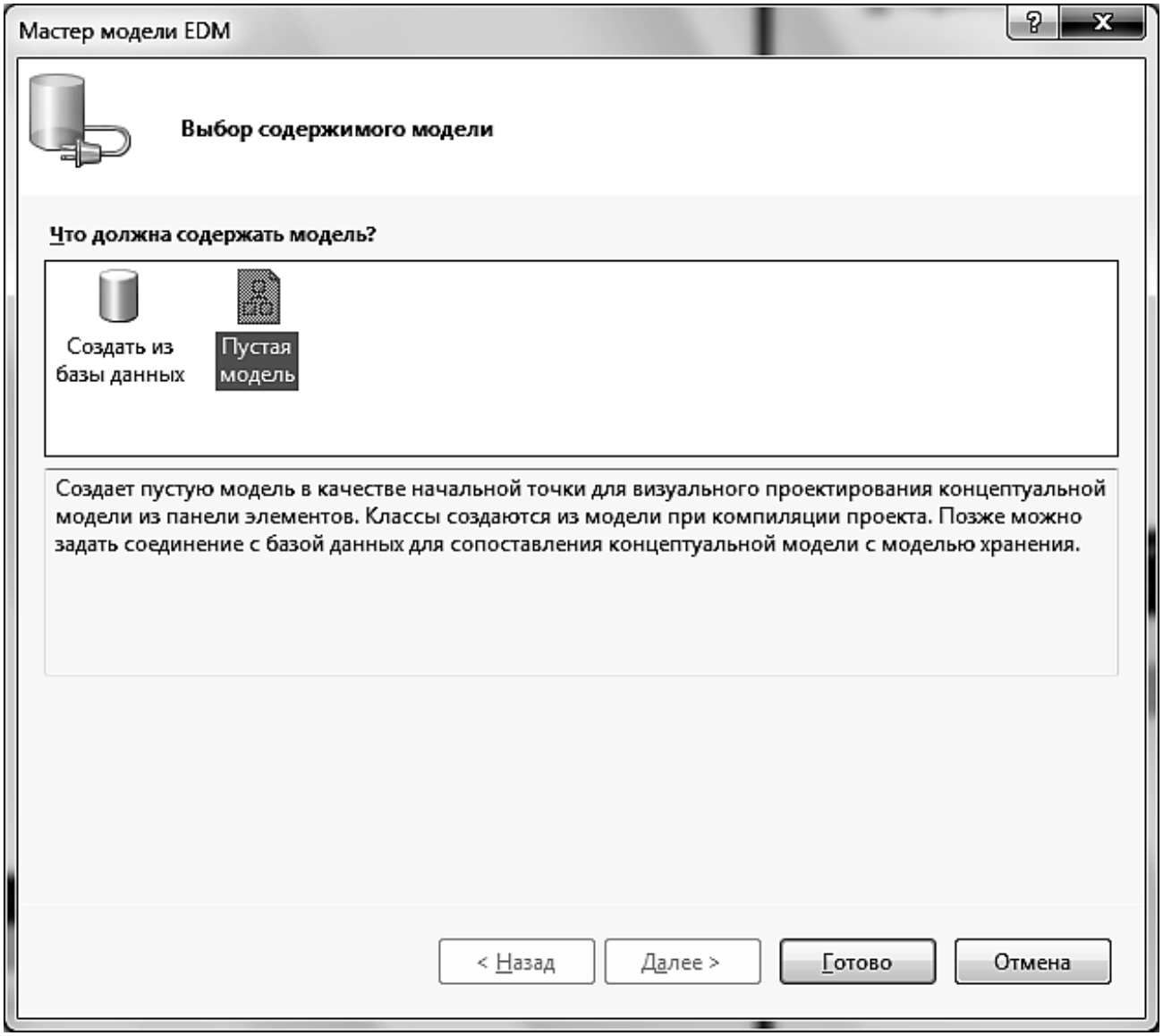


Рис. 7.10. **Вибір порожньої моделі у вікні *Выбор содержимого модели***

З'явилося порожнє вікно конструктора моделей EDM. У ньому далі будують саму модель.

2. Щоб створити сутності виконують таке:

2.1. Встановлюють значення ***True*** для властивості **Переводить новые объекты во множественное число** у вікні властивостей.

2.2. З панелі елементів перетягують значок **Сущность**у вікно конструктора моделей EDM.

2.3. Вводять ім'я сутності як значення властивості **Имя** у вікні властивостей.

Примітка. Бажано давати імена сутностей англійською мовою. Тоді значення властивості **Имя набора сущностей** формується автоматично у множині. Множинну форму отримає також таблиця у базі даних, яка створиться на основі даного об'єкта моделі.

2.4. Змінюють ім'я ключової властивості сутності із значення ***Идентификатор*** на підходяще, наприклад, ***Код_товару***.

Примітка. Імена властивостей сутності можна подавати будь-якою мовою, дотримуючись вимог, які ставляться до ідентифікаторів у програмі.

2.5. Додають властивості сутності, вибравши команду **Добавить** – **Скалярное свойство** у контестовому меню сутності і встановивши потрібні значення у вікні властивостей.

2.6. Повторюють п.п. 2.2 – 2.5 для створення решти сутностей (рис. 7.11).

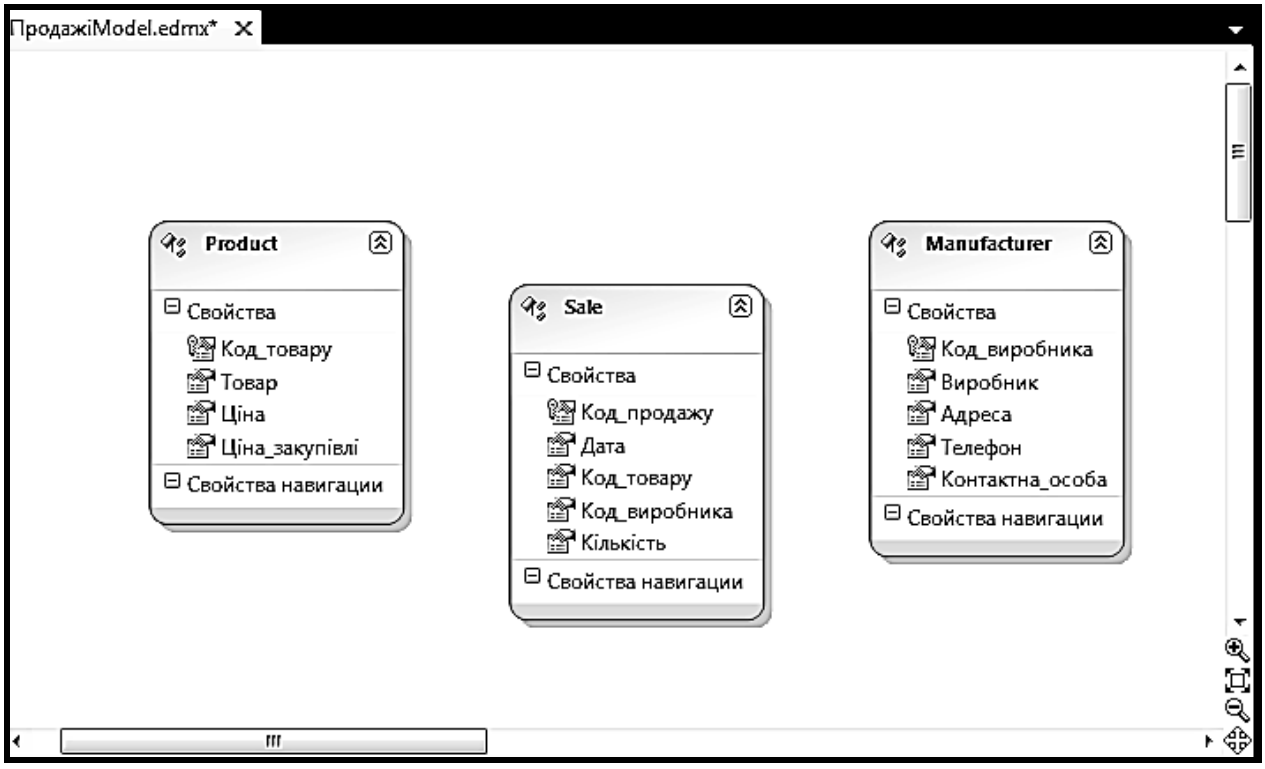


Рис. 7.11. **Сутності моделі EDM**

3. Щоб встановити асоціації між сутностями виконують таке:

3.1. Клацають правою клавишею миші у вільному місці вікна конструктора моделей EDM і вибирають команду **Добавить – Ассоциация** в контекстовому меню. З'являється вікно **Добавить ассоциацию**.

3.2. Встновлюють імена сутностей, що беруть участь в асоціації, та інші її властивості, у разі потреби вимкнувши прапорець **Добавить свойства внешнего ключа к сущности** (рис. 7.12).

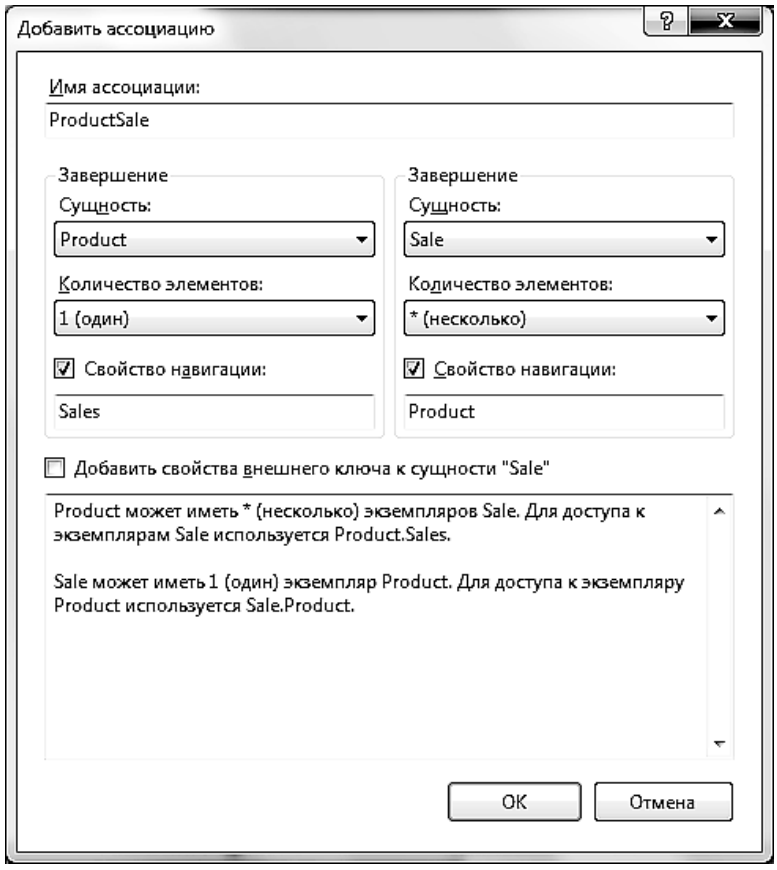


Рис. 7.12. **Вікно *Добавить ассоциацию***

3.3. Двічі клацнувши на лінії асоціації вказують поля, за якими здійснюється асоціація (рис. 7.13).

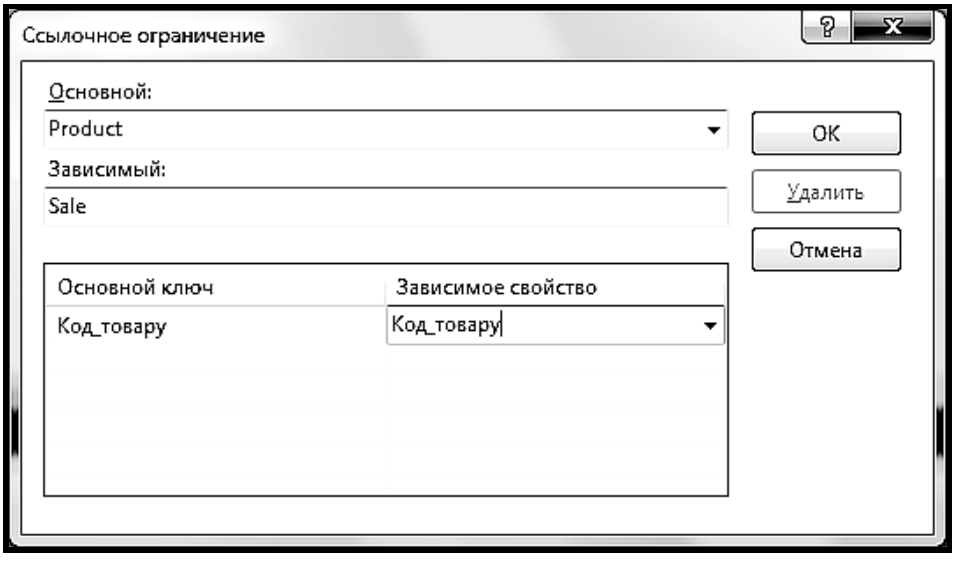


Рис. 7.13. **Поля, за якими здійснюється асоціація**

3.4. Повторюють п.п. 3.1 – 3.3 для створення решти асоціацій (рис. 7.14).

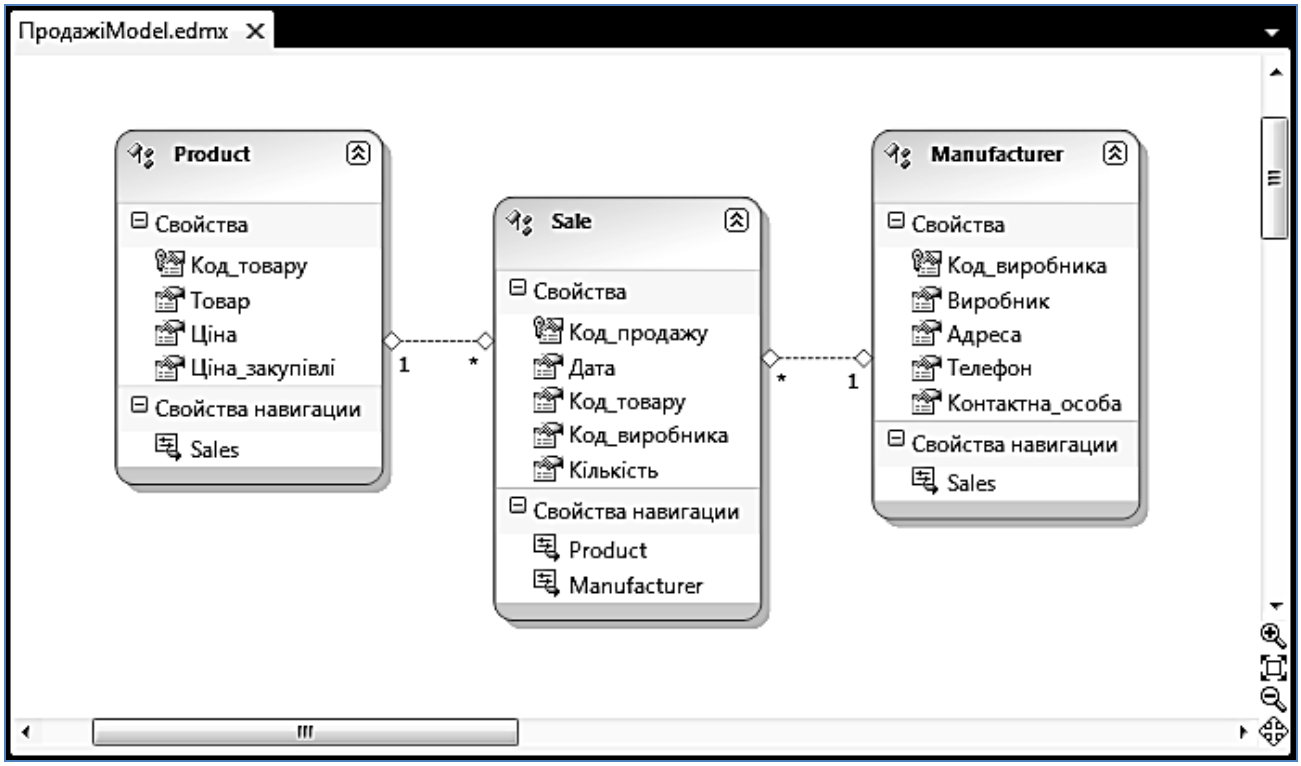


Рис. 7.14. **Модель EDM з асоціаціями**

4. Щоб згенерувати SQL-скрипт для створення бази даних на основі моделі EDM виконують таке:

4.1. Клацають правою кнопкою миші у вільному місці вікна конструктора моделей EDM і вибирають команду **Сформировать базу данных на основе модели** в контекстовому меню. З'являється вікно **Database Generation Workflow Manager**.

4.2. Погоджуються зі стратегією **TablePerTypeStrategy**, клацнувши кноку **Next**. З'являється вікно майстра формування бази даних.

4.3. У першому вікні майстра формування бази даних вибирають рядок з'єднання чи створюють новий, клацнувши кнопку **Создать соединение**. В останньому разі створюють новий рядок виконуючи дії, подібні до тих, що й під час створення підключення до бази даних (тут не вибирають вже існуючу базу даних, а вводять нове ім'я).

4.4. Після того, як майстер формування бази даних згенерує SQL-скрипт для створення бази даних, клацають кнопку **Готово**. З'являється вікно редактора Transact-SQL (рис. 7.15).

Рис. 7.15. **Вікно редактора Transact-SQL**

5. Щоб створити базу даних, клацають кнопку **Выполнить SQL** на панелі редактора Transact-SQL. Після з'єднання з базою даних створюються усі її таблиці і зв'язки між ними.

Отриману базу даних можна переглянути у вікні **Обозреватель серверов**, а її схему – у вікні конструктора баз даних (рис. 7.16).

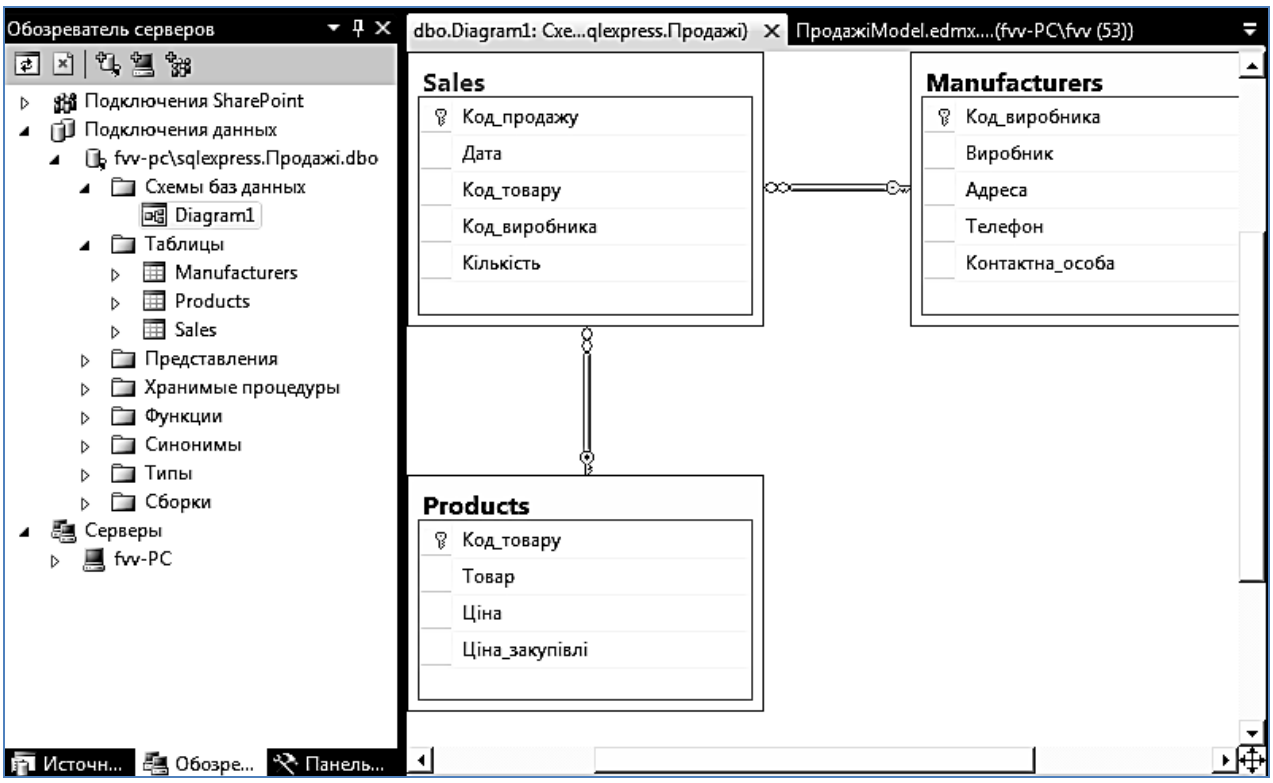


Рис. 7.16. **Вікна *Обозреватель серверов* і *Конструктор баз даних***

Запитання і завдання

1. Які сценарії створення моделей EDM вам відомі. В яких випадках використовують кожен із них?
2. З яких основних етапів складається сценарій DB First?
3. У чому полягає різниця між створенням моделі EDM за сценарієм Model First й аналогічної моделі даних засобами ERWin?
4. Обґрунтуйте, чому бажано давати імена сутностям англійською мовою?

7.4. Операції CRUD

Далі розглянуто основні операції ведення бази даних засобами Entity Framework.

Усі операції з базою даних здійснюють через екземпляр класу ObjectContext, в якому містяться сутності концептуальної моделі. Після створення такого екземпляру виконують потрібні операції з даними.

Для збереження змін у базі даних викликають метод SaveChanges класу ObjectContext.

Читання даних. Щоб прочитати дані з таблиці бази даних у найпростішому разі достатньо виконати два кроки:

1. Створюють екземпляр класу ObjectContext.
2. Прив'язують набір сутностей до елемента DataGridView.

Приклад 7.2. Відображення даних таблиці ***Products***.

```
// Створюють екземпляр класу ObjectContext
ХлібEntities db = new ХлібEntities();

// Прив'язують набір сутностей до елемента DataGridView
gvТовары.DataSource = db.Products;
```

Додавання даних. Щоб додати запис до таблиці бази даних виконують чотири кроки:

1. Створюють екземпляр класу ObjectContext.
2. Створюють і заповнюють новий об'єкт.
3. Додають новий об'єкт до набору.
4. Зберігають зміни.

Приклад 7.3. Додавання нового запису до таблиці ***Products***.

```
// Створюють екземпляр класу ObjectContext
ХлібEntities db = new ХлібEntities();

// Створюють і заповнюють дані про новий товар
Product product = new Product()
{
    Товар = "Калач",
    Ціна = 2,
    Ціна_закупівлі = 1.5M
};

// Додають новий товар до набору товарів
db.Products.AddObject(product);

// Зберігають у базі даних зміни
db.SaveChanges();
```

Зміна даних. Щоб змінити запис у таблиці бази даних виконують чотири кроки:

1. Створюють екземпляр класу ObjectContext.
2. Знаходять потрібну сутність.
3. Змінюють знайдені дані.
4. Зберігають зміни.

Приклад 7.4. Підвищення ціни закупівлі товару ***Хліб "Український"*** на 10 %.

```
// Створюють екземпляр класу ObjectContext
ХлібEntities db = new ХлібEntities();

// Знаходять дані про товар
Product product = (from товар in db.Products
        where товар.Товар == @"Хліб ""Український"""
        select товар).Single<Product>();
// Підвищують ціну
product.Ціна_закупівлі = product.Ціна_закупівлі * 1.1M;

// Зберігають у базі даних зміни
db.SaveChanges();
```

Видалення даних. Щоб видалити запис з таблиці бази даних виконують чотири кроки:

1. Створюють екземпляр класу ObjectContext.
2. Знаходять потрібну сутність.
3. Видаляють знайдені дані.
4. Зберігають зміни.

Приклад 7.5. Видалення даних про товар ***Батон "Молочний"***.

```
// Створюють екземпляр класу ObjectContext
ХлібEntities db = new ХлібEntities();

// Знаходять дані про товар
Product product = (from товар in db.Products
        where товар.Товар == @"Батон ""Молочний"""
        select товар).Single<Product>();

// Видаляють дані про знайдений товар
db.Products.DeleteObject(product);

// Зберігають у базі даних зміни
db.SaveChanges();
```

Запитання і завдання

1. У чому полягає різниця між виконанням операцій CRUD з використанням технології типізованих наборів даних і платформи Entity Framework?

2. Запишіть фрагмент програми, який реалізує додавання даних до таблиці ***Manufacturers***.

3. Запишіть фрагмент програми, який реалізує видалення даних про всі товари.

7.5. LINQ to Entities

Технологія LINQ to Entities забезпечує підтримку LINQ у запитах до сутностей. Вона дозволяє писати запити до концептуальної моделі Entity Framework мовою Visual C# або Visual Basic.

Подібно до інших технологій LINQ у LINQ to Entities можна використовувати синтаксис виразів запитів або синтаксис запитів на основі методів.

У запиті вказують, які дані треба одержати із сутності. В ньому також можна вказати, як повинні сортуватися, групуватися й оформлятися дані, що повертаються.

Проекція. Операцію застосовують, якщо в запиті потрібно отримати не всі властивості сутностей, а також результати обчислень з їхніми властивостями. Властивості, що входять до складу результату вказують у реченні **select**.

Приклад 7.6. Дані про всі товари без їхніх кодів, але з обчисленням торгової надбавки (*Ціна – Ціна_закупівлі*).

```
var query = from товар in db.Products
select new
    {
        Товар = товар.Товар,
        Ціна = товар.Ціна,
        Ціна_закупівлі = товар.Ціна_закупівлі,
        Торгова_надбавка = товар.Ціна - товар.Ціна_закупівлі
    };
```

Фільтрація. Операцію застосовують, якщо в запиті потрібно отримати не всі сутності набору, а лише ті, що задовольняють певні умови. Ці умови вказують у реченні **where**.

Приклад 7.7. Дані про товари, ціна закупівлі яких знаходиться в межах від 2 до 3 грн.

```
var query = from товар in db.Products
  where (товар.Ціна_закупівлі >= 2) && (товар.Ціна_закупівлі <= 3)
      select товар;
```

Сортування. Операцію застосовують, якщо в запиті потрібно отримати сутності у певному порядку. Операцію сортування вказують у реченні **orderby**, а порядок – за допомогою слів **ascending** (за зростанням) та **descending** (за спаданням).

LINQ підтримує багаторівневе сортування. Воно полягає в тому, що спочатку виконують сортування за першим критерієм. Якщо є кілька сутностей, у яких однакове значення цього критерію, утворюються групи сутностей з тим самим його значенням. Після цього у кожній групі виконують сортування за другим критерієм і т. д. Критерії перераховують у реченні **orderby** через кому.

Приклад 7.8. Розташування даних про всі товари за зменшенням ціни закупівлі, а в групах з однаковою ціною закупівлі – в алфавітному порядку назв товарів.

```
var query = from товар in db.Products
        orderby товар.Ціна_закупівлі descending, товар.Товар ascending
        select товар;
```

Агрегатні оператори. Оскільки для таких операцій запитів, як **Count**, **Sum**, **Average**, **Min** або **Max**, немає відповідних речень у виразах запитів, їх створюють за допомогою виклику методів.

Приклад 7.9. Середня ціна товарів та назва найдешевшого товару.

```
Decimal Середня_ціна = db.Products.Average(t => t.Ціна);

string Товар_мін = (from товар in db.Products
                    where товар.Ціна==db.Products.Min(t=>t.Ціна)
                    select товар.Товар).First();
```

Приклад 7.10. Сумарна кількість проданих товарів кожного виду.

```
var продажі = from продаж in db.Sales
              group продаж by продаж.Код_товару into g
              select new
              {
                  Код_товару = g.Key,
                  Кількість = g.Sum(p => p.Кількість)
              };
```

Секціонування. Операцію застосовують, якщо в запиті потрібно отримати певну кількість із множини сутностей. Використовують такі методи:

Take – вибирає задану кількість сутностей з початку послідовності, а решту пропускає;

TakeWhile – вибирає сутності, що задовольняють вказані умови, з початку послідовності, а решту пропускає;

Skip – пропускає задану кількість сутностей з початку послідовності і вибирає решту;

SkipWhile – пропускає сутності з початку послідовності, що задовольняють вказані умови, а решту вибирає (у тому числі ті, що задовольняють умовам, але знаходяться після першого відібраного елемента).

Примітка. Перед вживанням методів Skip та SkipWhile обов'язково виконують сортування.

Секціонування (англ. paging) застосовують для організації виведення даних із довгих списків на web-сторінку. В цьому разі немає потреби очікувати, коли завантажиться увесь список – для початку виведення достатньо даних, які заповнюють видиму частину списку.

Приклад 7.11. Назви перших двох товарів, а потім решти.

```
var query = from товар in db.Products
orderby товар.Код_товару
            select new  {
Товар = товар.Товар
                        };
            var перші2 = query.Take(2);
            var решта = query.Skip(2);
```

Навігація за зв'язками. Операцію застосовують, якщо в запиті потрібно отримати властивість з іншої сутності, яка пов'язана асоціацією з поточною сутністю. Властивості навігації дозволяють користувачеві переходити від однієї сутності до іншої або від сутності до пов'язаних сутностей у наборі асоціацій.

Приклад 7.12. Вартість проданих товарів кожного виду, які отримані від виробника з кодом ***2***.

```
var query = from продаж in db.Sales
            where продаж.Код_виробника == 2
            select new
```

```
            {
                Дата = продаж.Дата,
                Товар = продаж.Product.Товар,
                Ціна = продаж.Product.Ціна,
                Кількість = продаж.Кількість,
                Вартість = продаж.Product.Ціна * продаж.Кількість
            };
```

Примітка. Властивості ***Товар*** і ***Ціна*** вибираються через властивість навігації ***Product***.

Приклад 7.13. Список накладних, у якому відображаються такі атрибути кожної накладної: код накладної в базі даних, дата відпуску товарів за накладною, виробник, від якого отримано товари та сума вартостей усіх товарів за накладною.

```
var накладні = from накладна in db.Invoices
            select new
               {
                   Код_накладної = накладна.Код_накладної,
                   Дата = накладна.Дата,
                   Виробник = накладна.Manufacturer.Виробник,
                   Сума =
накладна.Invoice_Products.Sum(t=>t.Product.Ціна_закупівлі
*t.Кількість)
               };
```

Примітки. 1. Властивість ***Виробник*** вибирається через властивість навігації ***Manufacturer***, властивість ***Ціна_закупівлі*** – спочатку через властивість навігації ***Invoice_Products***, а потім через властивість навігації ***Product***, властивість ***Кількість*** – через властивість навігації ***Invoice_Products***.

2. У запиті використано властивість ***Ціна_закупівлі*** оскільки розглядаються прибуткові накладні.

З'єднання. Операцію застосовують у запитах до джерел даних, що не мають доступних для переходу зв'язків один з одним за властивостями навігації.

Операція з'єднання здійснює взаємозв'язок об'єктів одного джерела даних з об'єктами, що використовують спільну властивість в іншому джерелі даних.

Приклад 7.14. Сумарна кількість проданих товарів кожного виду із зазначенням назви товару.

```
// Обчислюємо суммарну кількість продаж по кажному виду товарів
// (через Код_товару)
var продажі = from продаж in db.Sales
              group продаж by продаж.Код_товару into g
              select new
              {
                  Код_товару = g.Key,
                  Кількість = g.Sum(p => p.Кількість)
              };
// Додаємо назви товарів для кожної кількості (inner join)
var query = from товар in db.Products
            join продаж in продажі
            on товар.Код_товару equals продаж.Код_товару
            select new
            {
                Товар = товар.Товар,
                Кількість = продаж.Кількість
            };
```

Було викладено далеко не всі можливості технології LINQ to Entities. Більш детально можна ознайомитися з ними в роботі [15].

Запитання і завдання

1. Порівняйте технології LINQtoDataSet та LINQ to Entities.
2. У чому полягає різниця між операціями навігації за зв'язками та з'єднаннями?
3. Запишіть фрагмент програми, який реалізує визначення сумарної вартості проданих товарів кожного продавця із зазначенням назви продавця.

7.6. Відображення даних

Відібрані в запиті дані потрібно відобразити в елементах керування на формі. Це можна зробити такими способами:

1. Вказати запит як джерело даних елемента керування.
2. Вказати запит як джерело даних з'єднувача.
3. Використати візуальні засоби на основі панелі **Источники данных**.

Останні два способи передбачають використання об'єкта класу BindingSource. Вони вважаються більш правильними, виходячи з переваг з'єднувача, які розглянуто в розділі 3.7. Перший спосіб є коротшим, але, зважаючи на його недоліки, використовується у невеликих проектах.

Спочатку слід навести приклади застосування перших двох способів, а потім детально розглянути третій спосіб відображення даних.

Приклад 7.15. Запит LINQ як джерело даних елемента керування.

```
// Створюємо об'єкт класу ObjectContext
ХлібEntities db = new ХлібEntities();

// Відбираємо дані
var query = from товар in db.Products
            select new
            {
                Товар = товар.Товар,
                Ціна = товар.Ціна
            };

// Відображаємо дані
gvТовари.DataSource = query;
```

Приклад 7.16. Запит LINQ як джерело даних з'єднувача.

```
// Створюємо об'єкт класу ObjectContext
ХлібEntities db = new ХлібEntities();

// Відбираємо дані
var query = from товар in db.Products
            select new
            {
                Товар = товар.Товар,
                Ціна = товар.Ціна
            };

// Створюємо з'єднувача
BindingSource bindingТовари = new BindingSource();

// Вказуємо джерело даних для з'єднувача
bindingТовари.DataSource = query;

// Відображаємо дані
gvТовари.DataSource = bindingТовари;
```

Використання візуальних засобів на основі панелі **Источники данных** дуже подібне до того, що було розглянуто в типізованих наборах даних, але в Entity Framework ця технологія менш розвинута і вимагає додаткових операцій.

Для відображення даних із використанням візуальних засобів на основі панелі **Источники данных** виконують таке.

1. Додають нове джерело даних на панель **Источники данных**. Для цього:

1.1. Запускають майстра настроювання джерела даних вибором команди **Добавить новый источник данных**, що знаходиться в меню **Данные**.

1.2. Вибирають значок **Объект** у вікні **Выбор типа источника данных** і клацають кнопку **Далее** (рис. 7.17).

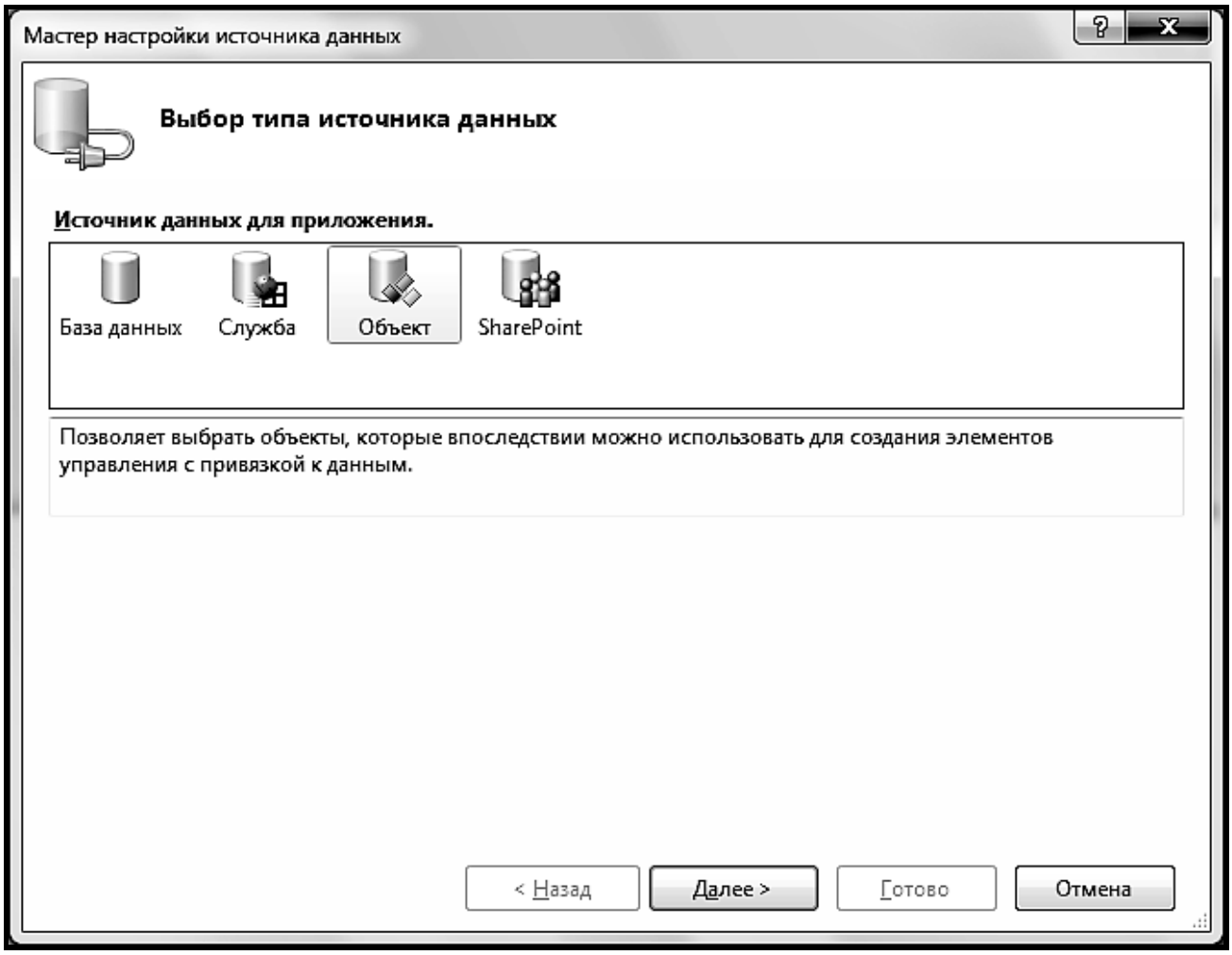


Рис. 7.17. **Вікно *Выбор типа источника данных***

1.3. Вибирають сутності, дані яких знадобляться для відображення, у вікні **Выбор объектов данных** і клацають кнопку **Готово** (рис. 7.18).

У результаті роботи майстра у вікні **Источники данных** з'явилося зображення вибраних сутностей. Кожна сутність тут подана вузлом у ви-

гляді значка. Елементами вузла є імена властивостей. Якщо сутність має асоціацію з іншою сутністю, то остання відображається у вигляді під-двузла. На рис. 7.19 у розкритому вузлі сутності ***Product*** відображаються її властивості та дві дочірні сутності – ***Invoice_Products*** і ***Sales***.

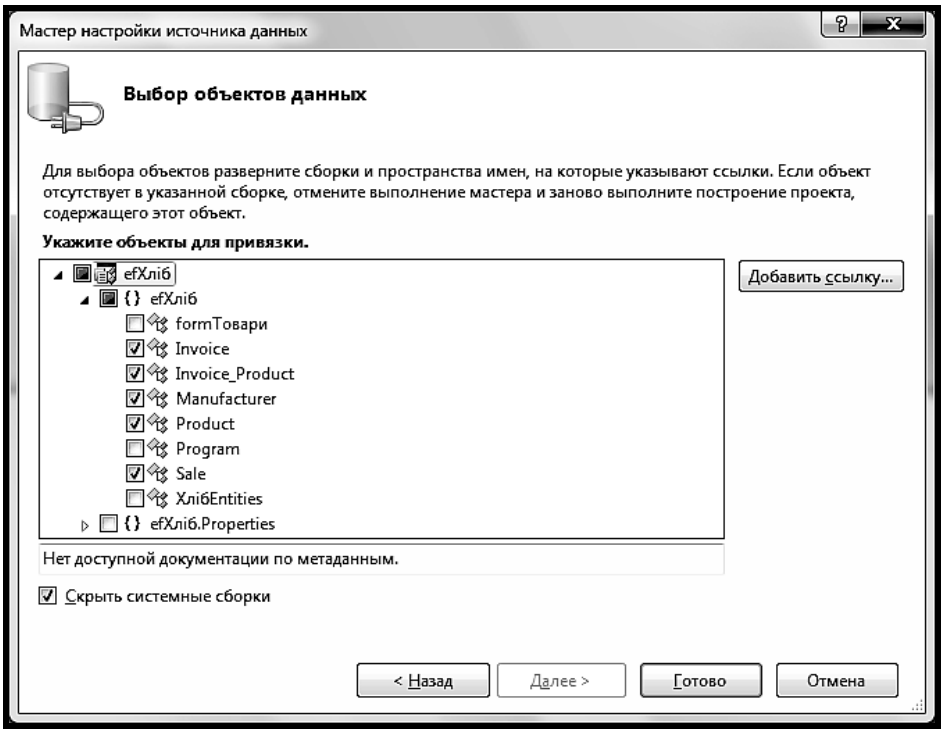


Рис. 7.18. **Вікно *Выбор объектов данных***

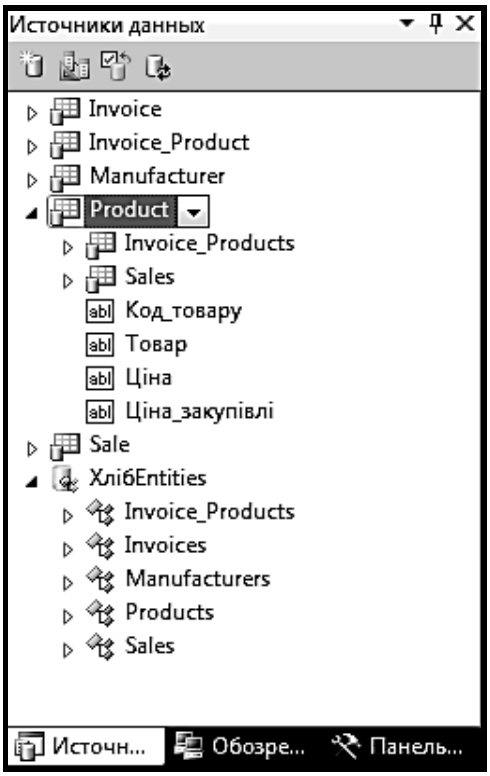


Рис. 7.19. **Вікно *Источники данных***

2. Перетягують цілі сутності або їхні окремі властивості на форму, як це описано в розділі 5.6, щоб отримати відображення даних (рис. 7.20).

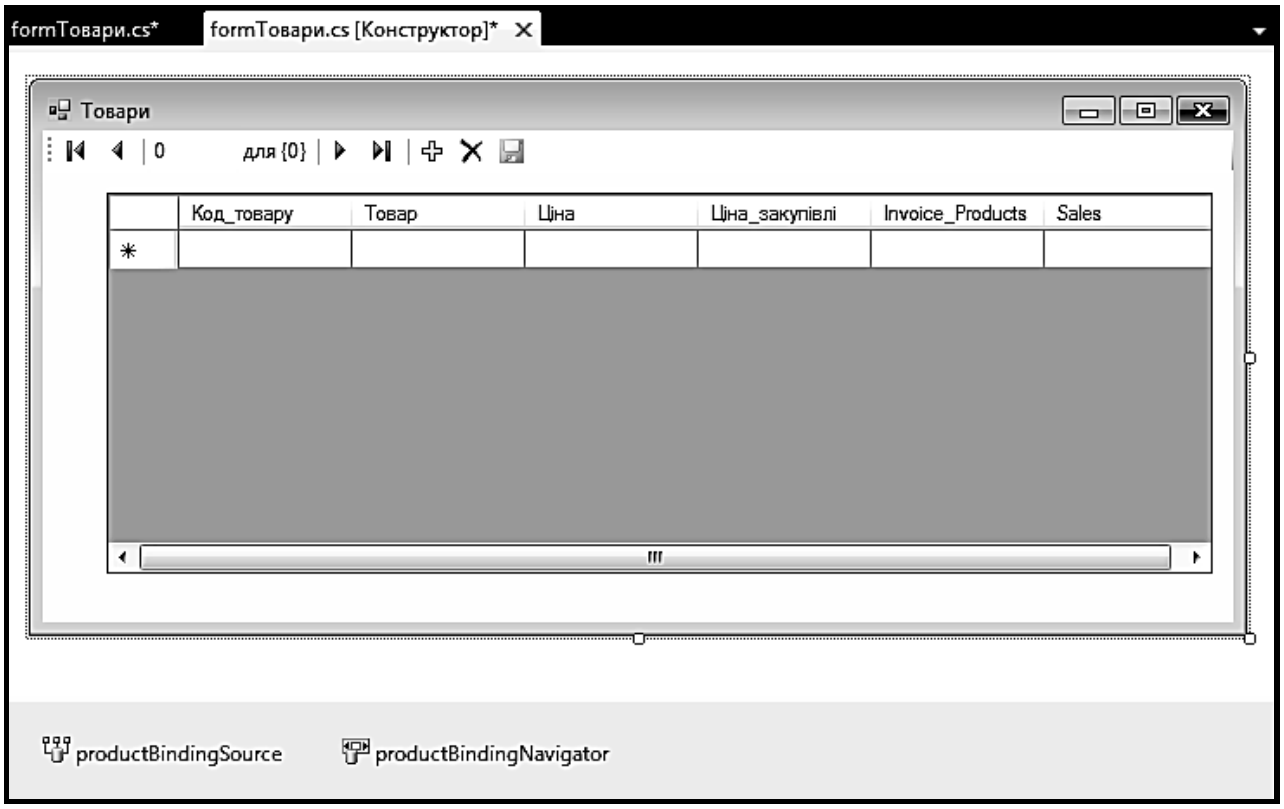


Рис. 7.20. **Зовнішній вигляд форми після перетягання сутності**

3. Двічі клацають на вільному місці форми й у вікні коду вводять код оброблювача події завантаження форми. В ньому вказують методи для заповнення з'єднувача даними.

Приклад 7.17. Оброблювач події завантаження форми при використанні візуальних засобів на основі панелі **Источники данных**.

```
private void formТовари_Load(object sender, EventArgs e)
{
    // Створюємо об'єкт класу ObjectContext
    ХлібEntities db = new ХлібEntities();

    // Вказуємо джерело даних для з'єднувача
    productDataGridView.DataSource = db.Products;
}
```

Запитання і завдання

1. Які засоби відображення сутностей вам відомі? Порівняйте їх.
2. Запишіть фрагмент програми, який реалізує відображення даних запиту LINQ про результати продажів як джерела даних елементів керування.
3. Запишіть фрагмент програми, який реалізує відображення даних запиту LINQ про результати продажів як джерела даних з'єднувача. Порівняйте фрагменти програм, що отримані в п.п. 2 і 3.
4. Опишіть дії, які потрібно виконати для розв'язання задачі п. 3 з використанням візуальних засобів на основі панелі **Источники данных**.

Лабораторна робота № 7. Розробка застосувань на основі платформи Entity Framework

Цілі лабораторної роботи:

1. Набуття практичних навичок створення концептуальних моделей.
2. Освоєння сценаріїв Model First і DB First.
3. Оволодіння технологією LINQ to Entities.
4. Набуття практичних навичок відображення даних у застосуваннях на основі платформи Entity Framework.
5. Удосконалення навичок роботи з інтегрованим середовищем розробки Visual C# і довідковою системою Microsoft Developer Network (MSDN).

Перед виконанням лабораторної роботи студент повинен знати:

1. Основи проектування реляційних баз даних.
2. Структурні елементи бази даних і їхні властивості.
3. Основи побудови SQL-запитів.
4. Принципи прив'язування даних до елементів інтерфейсу.
5. Основи роботи з ієрархічними даними.
6. Принципи обробки подій в C#-програмі.

Після виконання лабораторної роботи студент повинен вміти:

1. Створювати концептуальні моделі засобами Microsoft Visual Studio.
2. Самостійно розробляти C#-застосування на основі платформи Entity Framework.

3. Використовувати основні бібліотеки .Net Framework під час розробки програм.

Хід роботи

1. Створення кнопкової форми застосування.
2. Сценарій Model First.
3. Сценарій DB First.
4. Операції CRUD у довідкових таблицях.
5. Робота з допоміжними сутностями.
6. Керування ієрархічними сутностями.
7. Аналіз даних у Entity Framework.

Інструкції

Постановка загальної задачі

Вивчення засобів роботи з даними на платформі Entity Framework проводиться шляхом створення застосування ***efХліб***. Дані зберігаються в базі даних ***Хліб*** під керівництвом СКБД SQL Server. База даних створюється в процесі виконання лабораторної роботи.

Керування застосуванням здійснюється за допомогою кнопкової форми ***Хліб*** (рис. 7.21). Кнопки призначені для виклику відповідних функціональних форм.

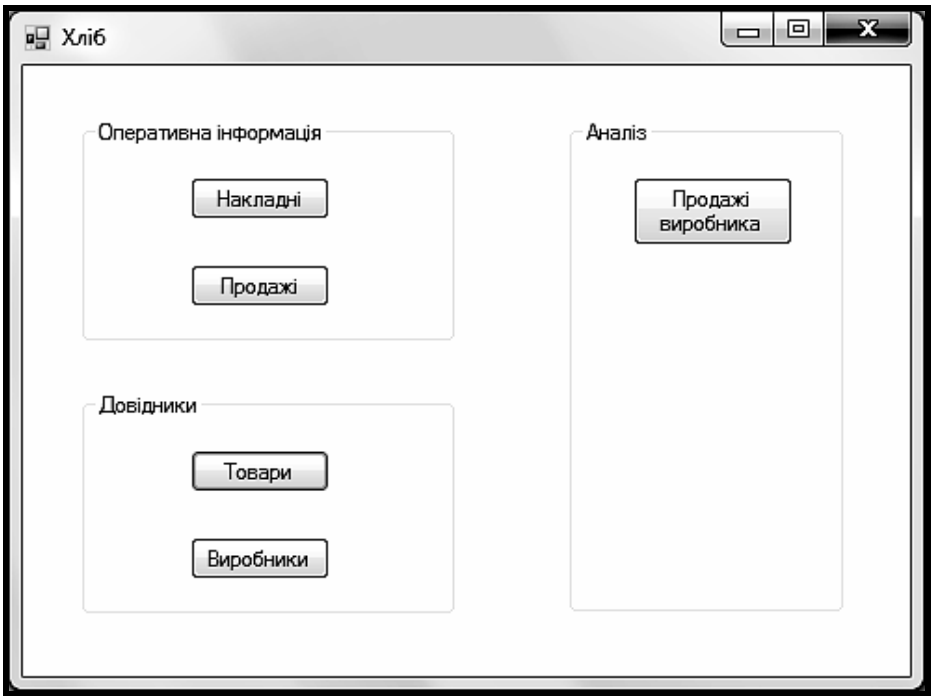


Рис. 7.21. **Кнопкова форма *Хліб***

На рис. 7.22 – 7.26 подано функціональні форми застосування. З їхньою допомогою можна переглядати й змінювати дані (обновляти, додавати й видаляти) у відповідних таблицях бази даних ***Хліб***, а також виконувати аналіз даних.

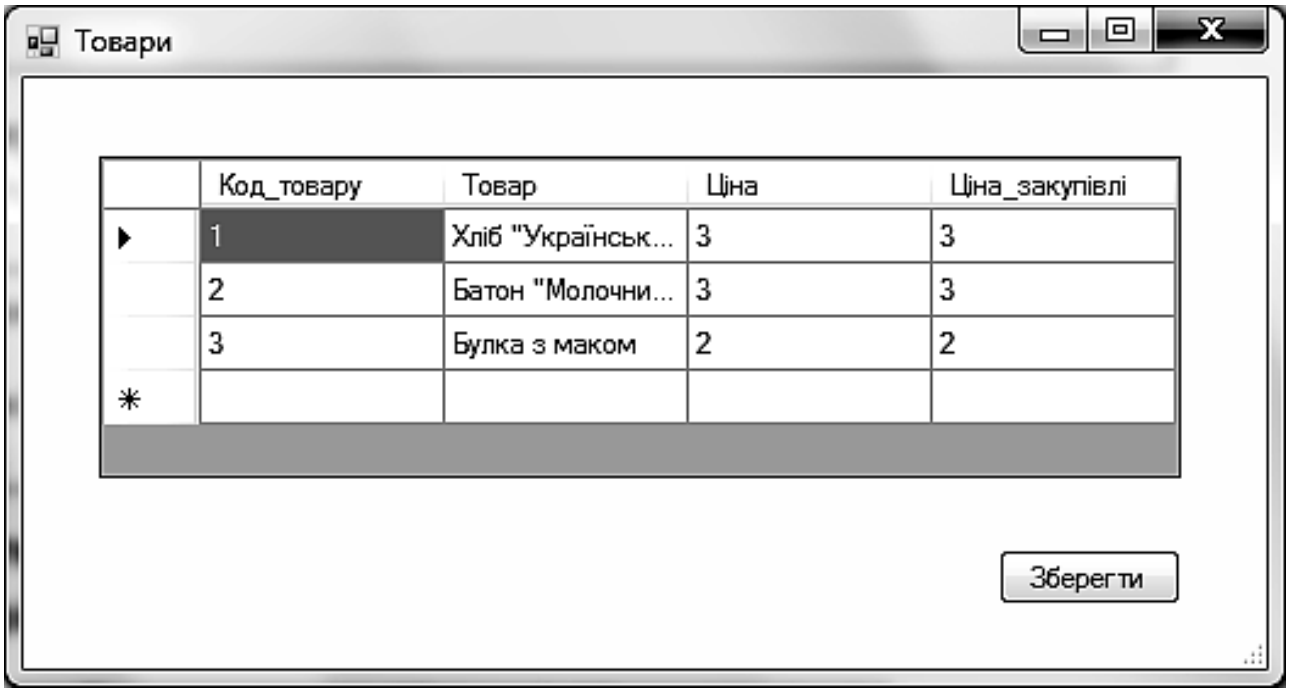


Рис. 7.22. **Форма для ведення сутності *Product***

Рис. 7.23. **Форма для ведення сутності *Manufacturer***

Рис. 7.24. **Форма для ведення сутності *Sale***

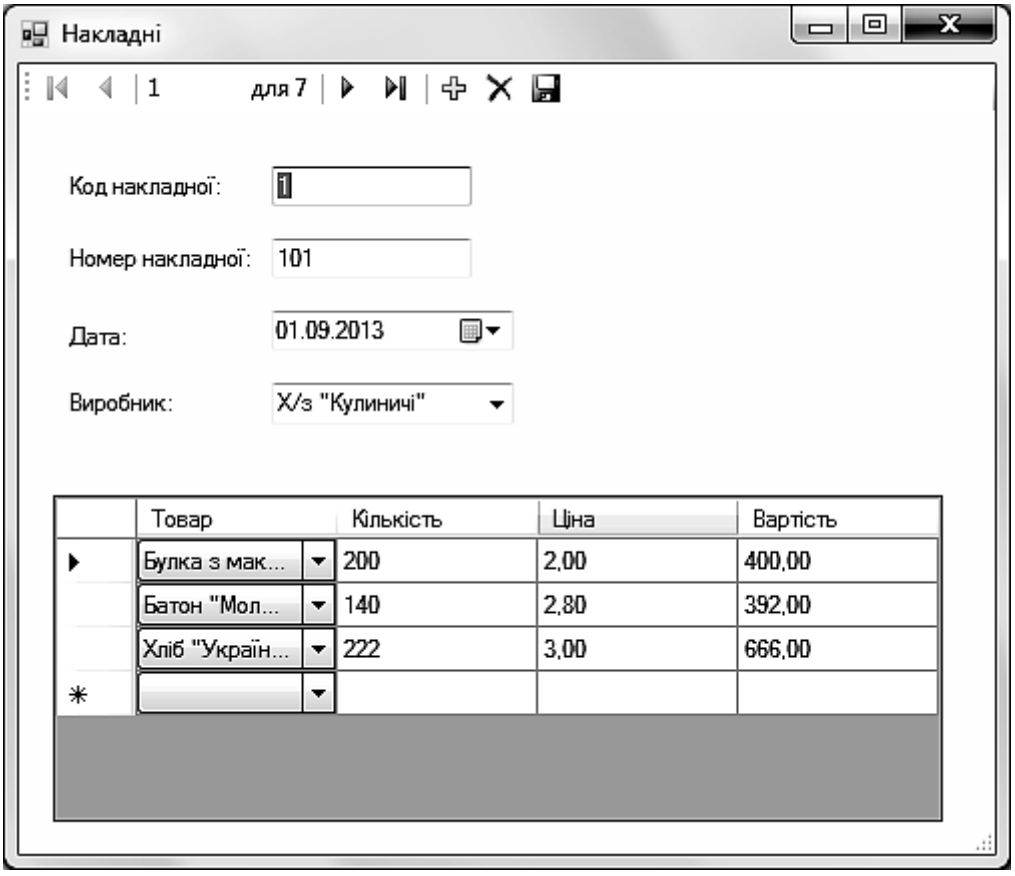


Рис. 7.25. **Форма для ведення сутностей *Invoice* й *Invoice Product***

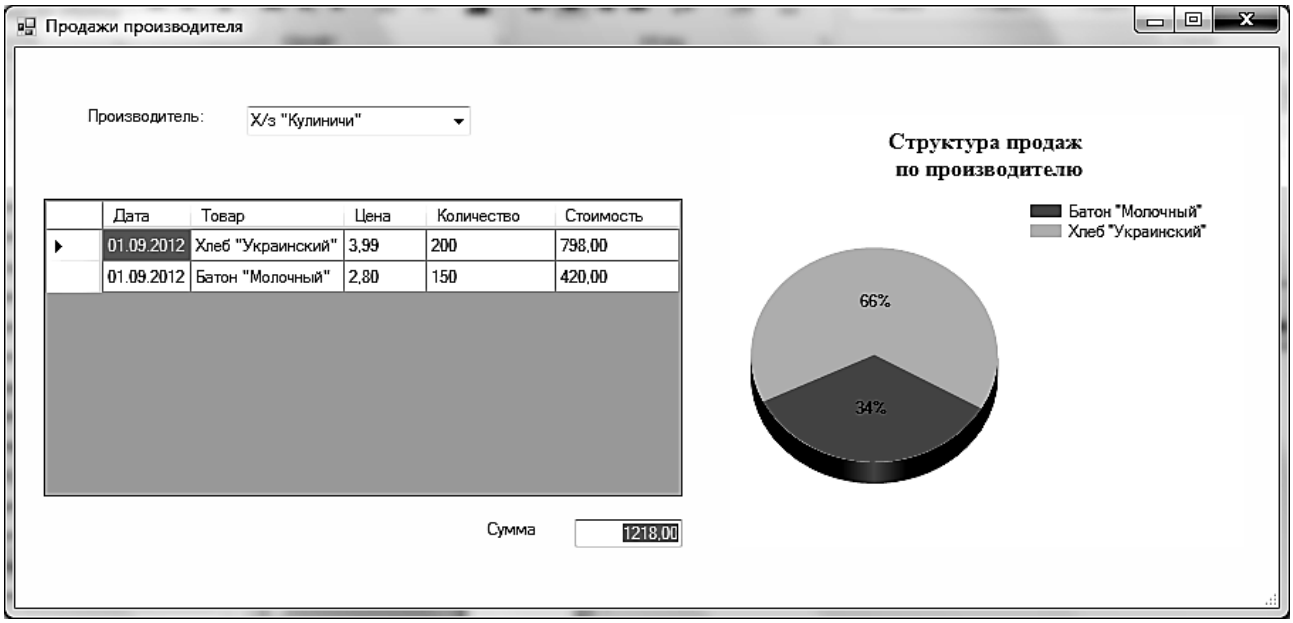


Рис. 7.26. **Форма для виконання аналізу з продажу товарів вибраного виробника**

1. Створення кнопкової форми застосування

Завдання

Створити застосування і побудувати в ньому кнопкову форму (рис. 7.27).

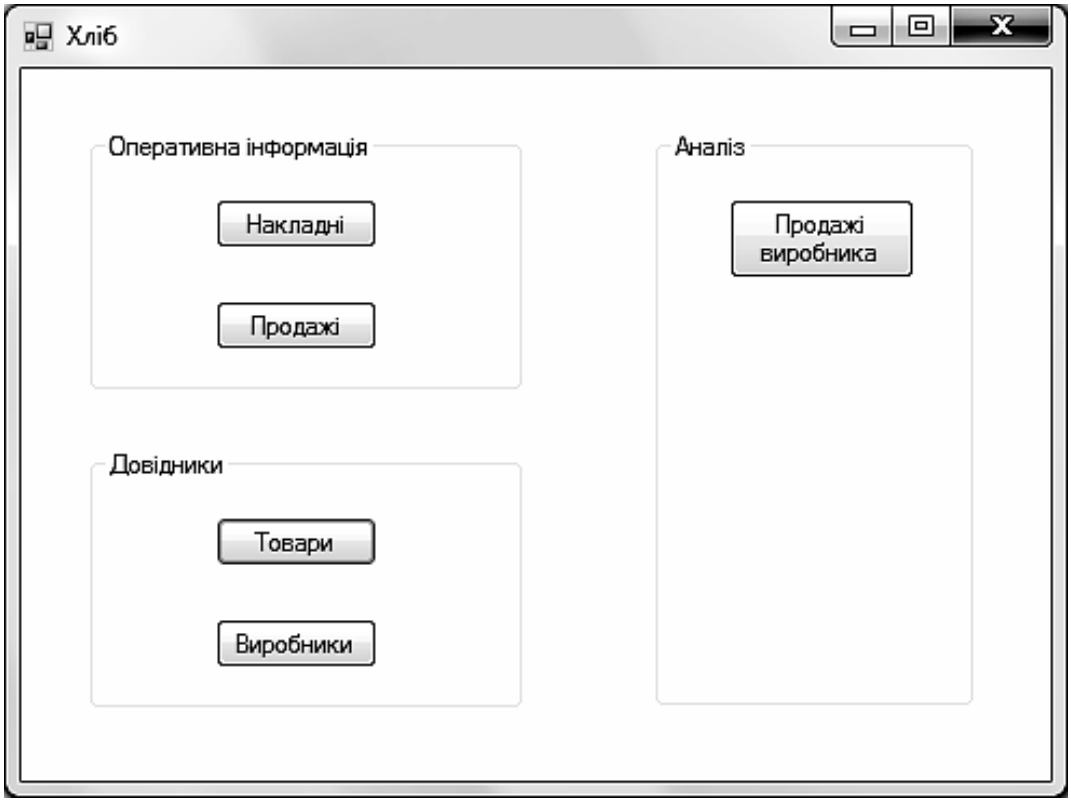


Рис. 7.27. **Кнопкова форма**

Виконання

1. Відкрийте Visual Studio.
2. Створіть проект Windows Forms мовою C# з ім'ям ***efХліб*** і збережіть його.
3. У вікні **Обозреватель решений** виділіть значок ***Form1*** і для файлу ***Form1.cs*** у вікні властивостей задайте ім'я ***formХліб.cs***.
4. Для властивості **Text** форми задайте значення ***Хліб***.
5. Додайте на форму елемент керування **GroupBox** і задайте значення ***Оперативна інформація*** для його властивості **Text**.
6. Додайте дві кнопки всередину елемента ***Оперативна інформація*** й установіть для них такі значення властивостей:

Кнопка	Властивість	Значення
1	Text	Накладні
	Name	buttonНакладні
2	Text	Продажі
	Name	buttonПродажі

7. Повторіть п.п. 5, 6 для групи кнопок ***Довідники***, встановивши такі значення імен кнопок: ***buttonТовари*** та ***buttonВиробники***.
8. Повторіть ще раз п.п.5, 6 для групи ***Аналіз***, встановивши значення імені кнопки ***buttonПродажіВиробника***. У рамці групи залиште вільне місце для кнопок викликання форм з іншими видами аналізу, які додадуться під час роботи над завданнями для самостійного виконання.

2. Сценарій Model First

Вивчення засобів формування бази даних на основі сценарію Model First виконується в 4 етапи:

1. Створення порожньої моделі EDM.
2. Побудува концептуальної моделі обліку продажів товарів.
3. Створення порожньої бази даних у вигляді файла бази даних SQL Server.
4. Формування бази даних на основі моделі EDM.

У результаті виконання цих етапів отримаємо концептуальну модель і відповідну базу даних. Застосовуючи засоби платформи Entity Framework, можна використовувати першу для операцій з другою. Цей сценарій використовують, коли ще базу даних не створено.

Примітка. Якщо на основі концептуальної моделі створюють базу даних усередині SQL Server, а не у вигляді локального mdf-файла, етап 3 можна пропустити.

Завдання 1

Створити порожню модель EDM.

Виконання

1. Виберіть команду **Добавить новый элемент** в меню **Проект**. З'являється вікно **Добавление нового елемента**.

2. Виберіть шаблон **Данные** в списку **Установленные шаблоны** і елемент **Модель EDM ADO.NET** в центральному списку, а в нижній частині вікна введіть ім'я моделі ***ХлібModel.edmx***, потім клацніть кнопку **Добавить**. З'являється перше вікно майстра моделі EDM (рис. 7.28).

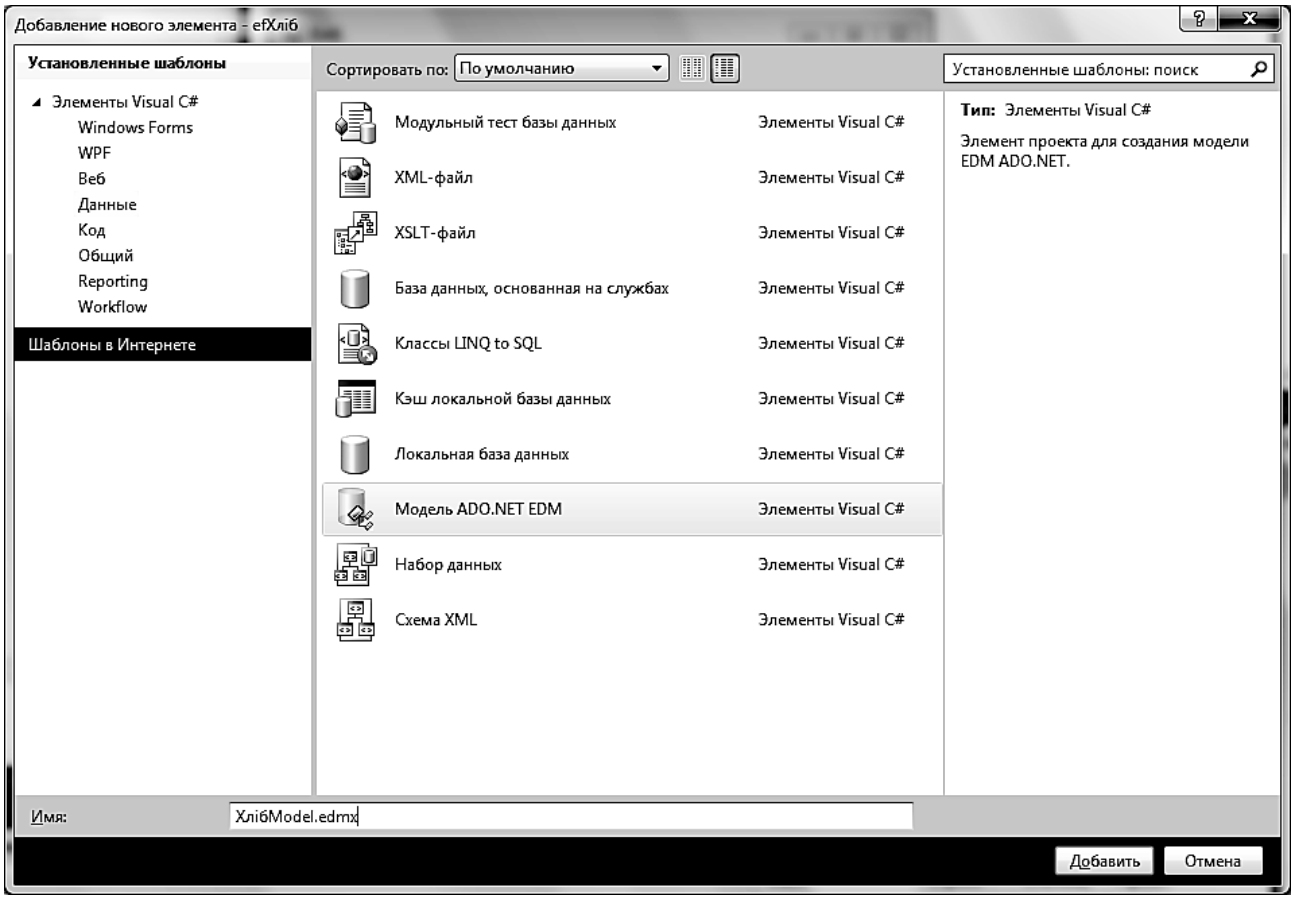


Рис. 7.28. **Вибір елемента моделі EDM у вікні *Добавление нового элемента***

3. Виберіть значок **Пустая модель** у вікні **Выбор содержимого модели** і клацніть кнопку **Готово** (рис. 7.29).

З'явилося порожнє вікно конструктора моделей EDM. У ньому далі побудуємо саму модель.

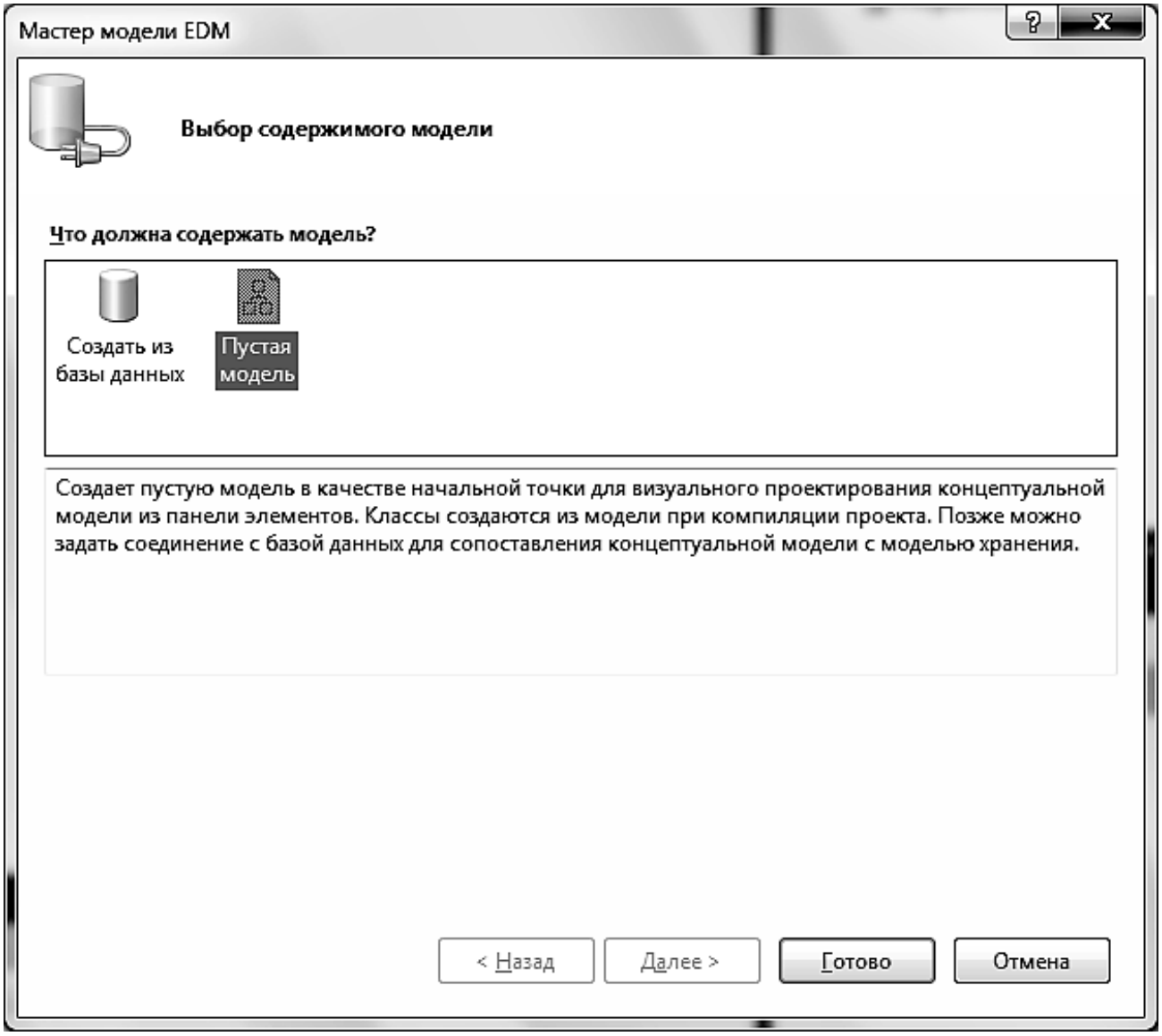


Рис. 7.29. **Вибір порожньої моделі у вікні *Выбор содержимого модели***

Завдання 2

Побудувати концептуальну модель, в якій подати дані з обліку продажів товарів.

Виконання

1. Перебуваючи у вікні конструктора моделей EDM, встановіть значення ***True*** для властивості **Переводить новые объекты во множественное число** у вікні властивостей.

2. Перетягніть значок **Сущность** з панелі елементів у вікно конструктора моделей EDM.

3. Уведіть ім'я сутності ***Product*** як значення властивості **Имя** у вікні властивостей.

4. Змініть ім'я ключової властивості сутності із значення ***Идентификатор*** на ***Код_товару***.

5. Додайте властивості сутності ***Product***, вибравши команду **Добавить – Скалярное свойство** у контекстовому меню сутності і встановивши такі значення у вікні властивостей:

Властивість сутності	Властивість у вікні Свойства (Properties)	Значення
Товар	Имя	Товар
	Тип	String
	Максимальная длина	25
Ціна	Имя	Ціна
	Тип	Decimal
	Точность	5
	Масштаб	2
Ціна_закупівлі	Имя	Ціна_закупівлі
	Тип	Decimal
	Точность	5
	Масштаб	2

Зовнішній вигляд сутності ***Product*** подано на рис. 7.30.

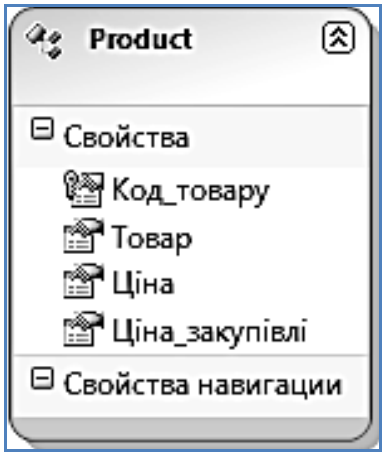


Рис. 7.30. **Сутність *Product***

6. Повторіть п.п. 1 – 5 для створення сутностей ***Manufacturer*** та ***Sale***, встановивши такі значення їхніх властивостей:

Сутність	Властивість сутності	Властивість у вікні Свойства (Properties)	Значення
Manufacturer	Код_виробника	Имя	Код_виробника
		Ключ сущности	True
	Виробник	Имя	Виробник
		Тип	String
		Максимальная длина	20
	Адреса	Имя	Адреса
		Тип	String
		Максимальная длина	30
	Телефон	Имя	Телефон
		Тип	String
		Максимальная длина	15
	Контактна_особа	Имя	Контактна_особа
		Тип	String
		Максимальная длина	20
Sale	Код_продажу	Имя	Код_продажу
		Ключ сущности	True
	Дата	Имя	Дата
		Тип	DateTime
	Код_товару	Имя	Код_товару
		Тип	Int32
	Код_виробника	Имя	Код_виробника
		Тип	Int32
	Кількість	Имя	Кількість
		Тип	Int16

Створені сутності подано на рис. 7.31.

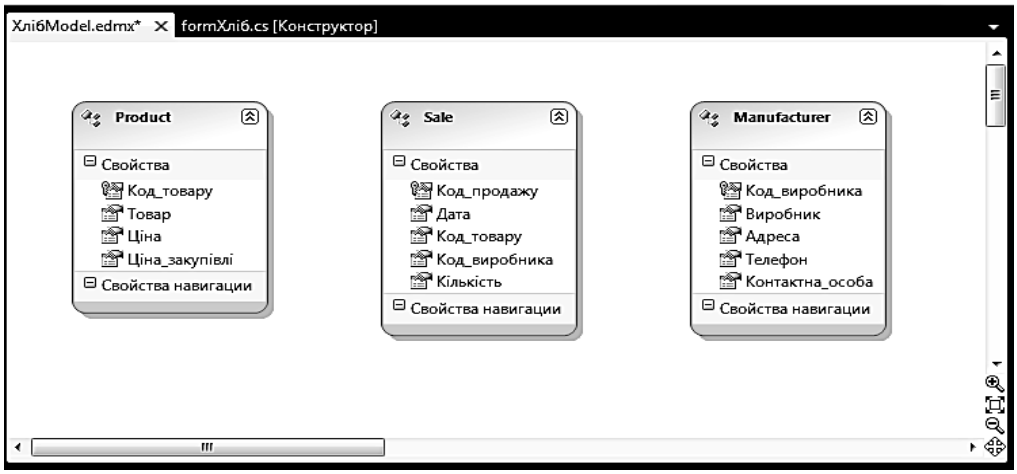


Рис. 7.31. **Сутності моделі EDM**

7. Щоб встановити асоціацію між сутностями ***Product*** і ***Sale*** виконайте таке:

7.1. Клацніть правою кнопкою миші у вільному місці вікна конструктора моделей EDM і виберіть команду **Добавить – Ассоциация** в контекстовому меню. З'являється вікно **Добавить ассоциацию**.

7.2. Встановіть імена сутностей ***Product*** і ***Sale*** у списках **Сущность** і вимкніть прапорець **Добавить свойства внешнего ключа к сущности** (рис. 7.32).

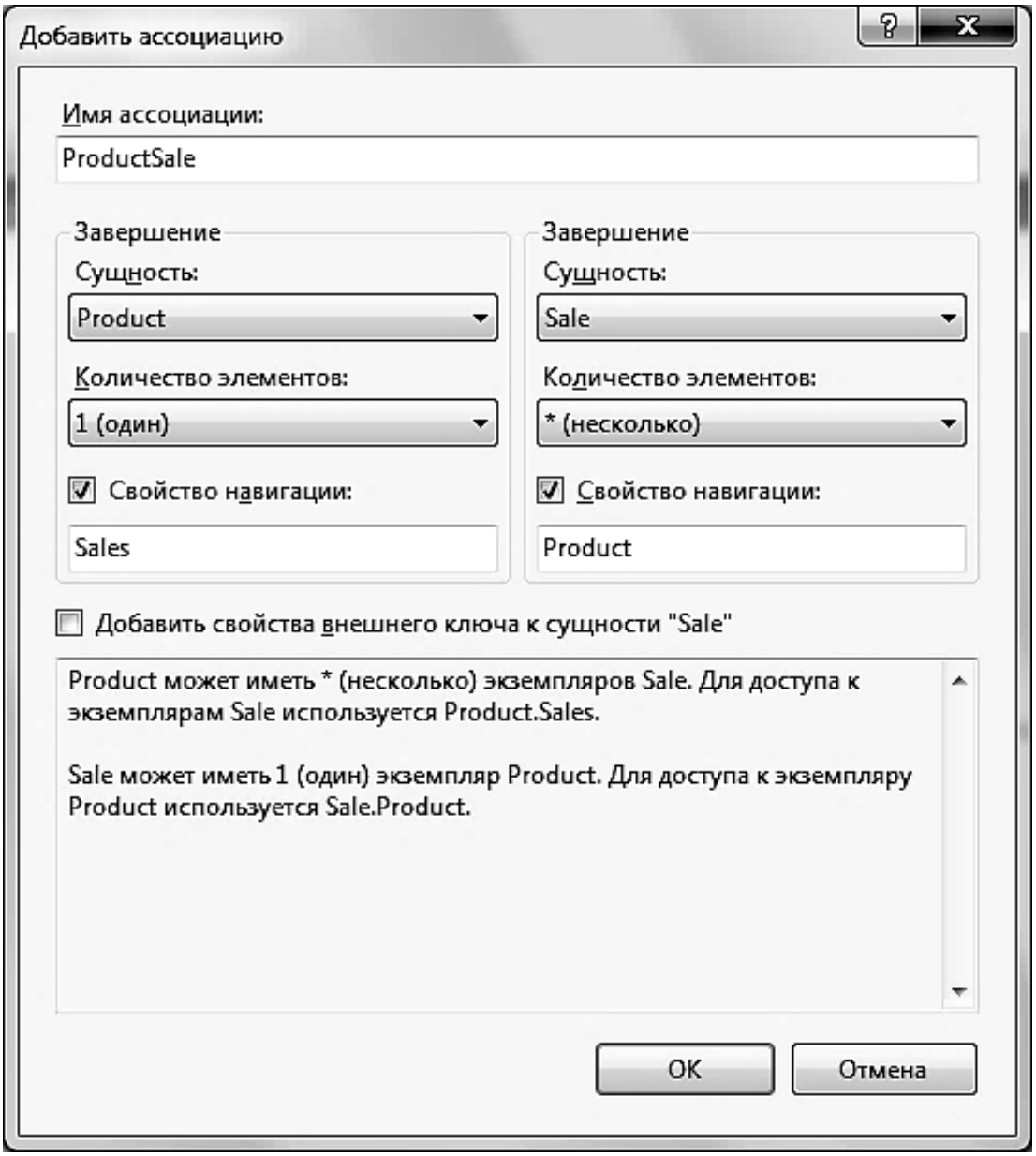


Рис. 7.32. **Вікно *Добавить ассоциацию***

Між сутностями ***Product*** і ***Sale*** з'явилася лінія асоціації, яка відображає відношення один-до-багатьох (рис. 7.33).

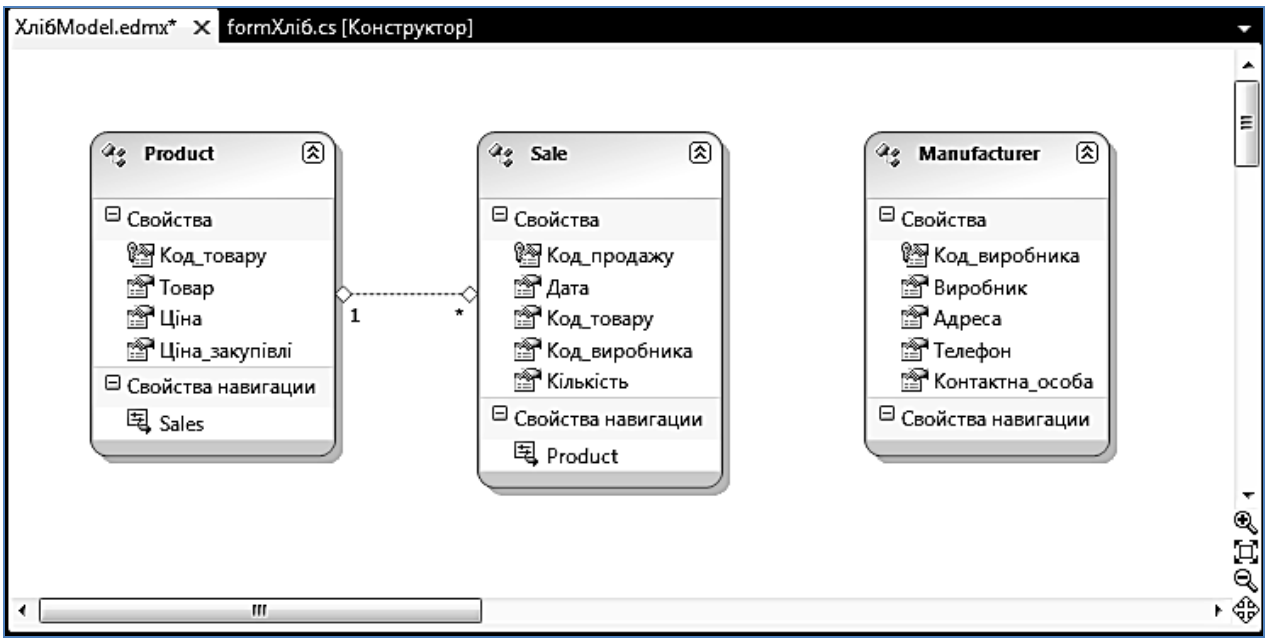


Рис. 7.33. **Лінія асоціації між сутностями *Product* і *Sale***

1.1. Двічі клацніть на лінії асоціації і вкажіть властивості сутностей, за якими здійснюється асоціація (рис. 7.34).

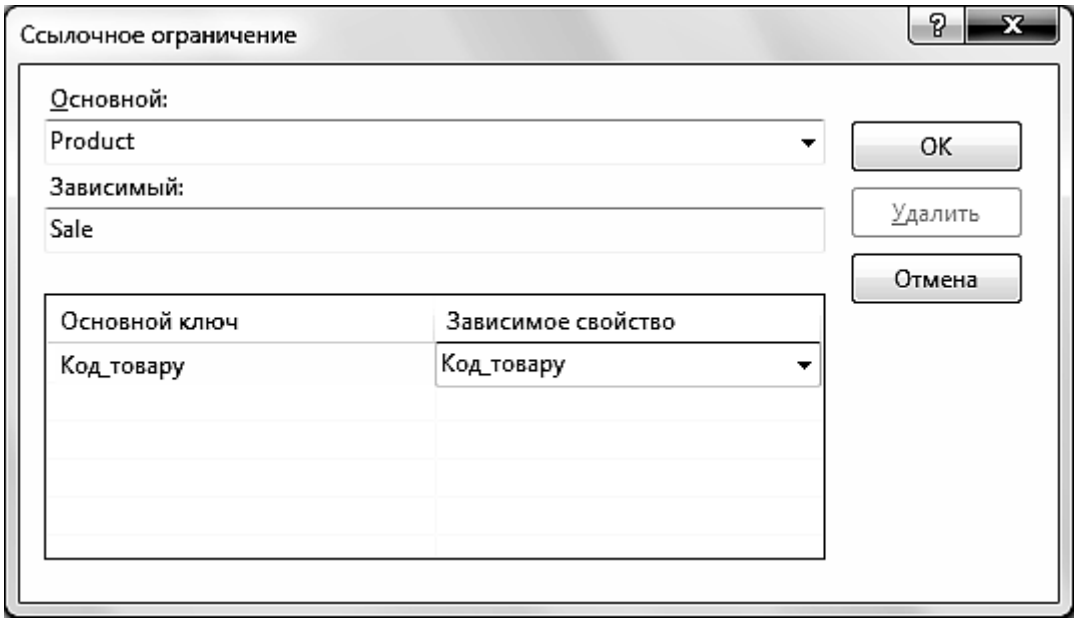


Рис. 7.34. **Властивості сутностей, за якими здійснюється асоціація**

1.2. Задайте властивість каскадного видалення, вказавши значення ***Cascade*** для властивості **Событие OnDelete элемента End1** у вікні властивостей.

2. Повторіть п. 7 для створення асоціації між сутностями ***Manufacturer*** і ***Sale*** (рис. 7.35).

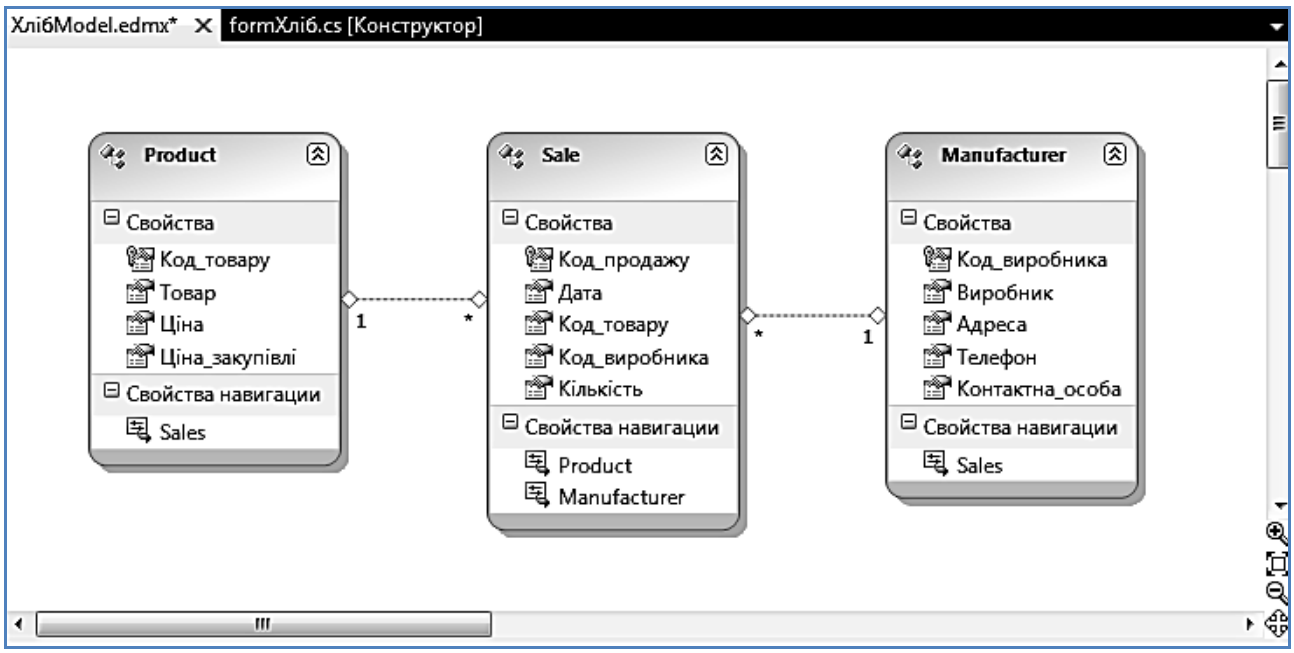


Рис. 7.35. **Модель EDM з асоціаціями**

3. Збережіть зміни, що зроблені в проекті.

Завдання 3

Створити порожню базу даних як файл бази даних SQL Server.

Виконання

1. Виберіть команду **Добавить новый элемент** у меню **Проект**. З'являється вікно **Добавление нового элемента**.

2. Виберіть шаблон **Данные** в списку **Установленные шаблоны** і елемент **База данных, основанная на службах** у центральному списку, а в нижній частині вікна введіть ім'я бази даних ***Хліб.mdf***, потім клацніть кнопку **Добавить** (рис. 7.36). З'являється перше вікно майстра налаштування джерела даних.

3. Клацніть кнопку **Отмена** у вікні **Выбор модели базы данных** (рис. 7.37).

У вікні **Обозреватель решений** з'явився значок бази даних ***Хліб.mdf***.

Примітка. Якщо у вікні **Добавление нового элемента** (рис. 7.36) вибрати елемент **Локальная база данных**, створиться sdf-файл бази даних SQL Server Compact Edition. Усі операції, які описані нижче, будуть правильними і для нього.

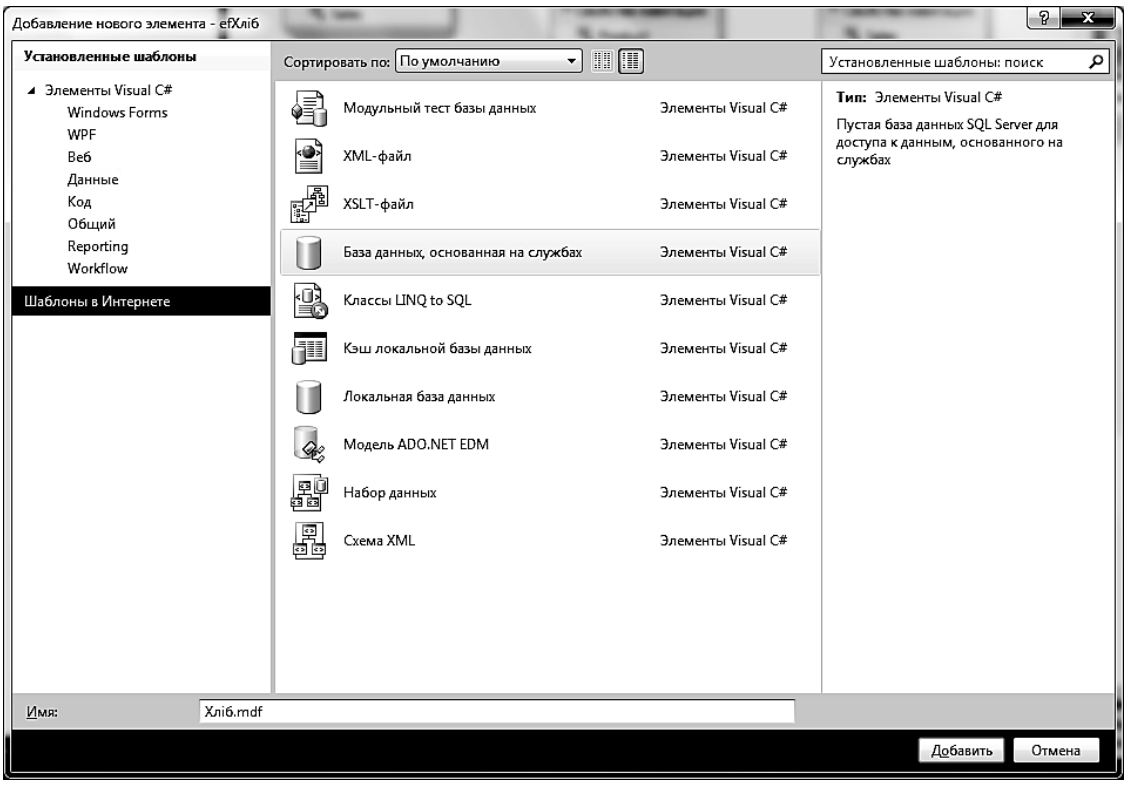


Рис. 7.36. **Вибір елемента бази даних у вікні *Добавление нового элемента***

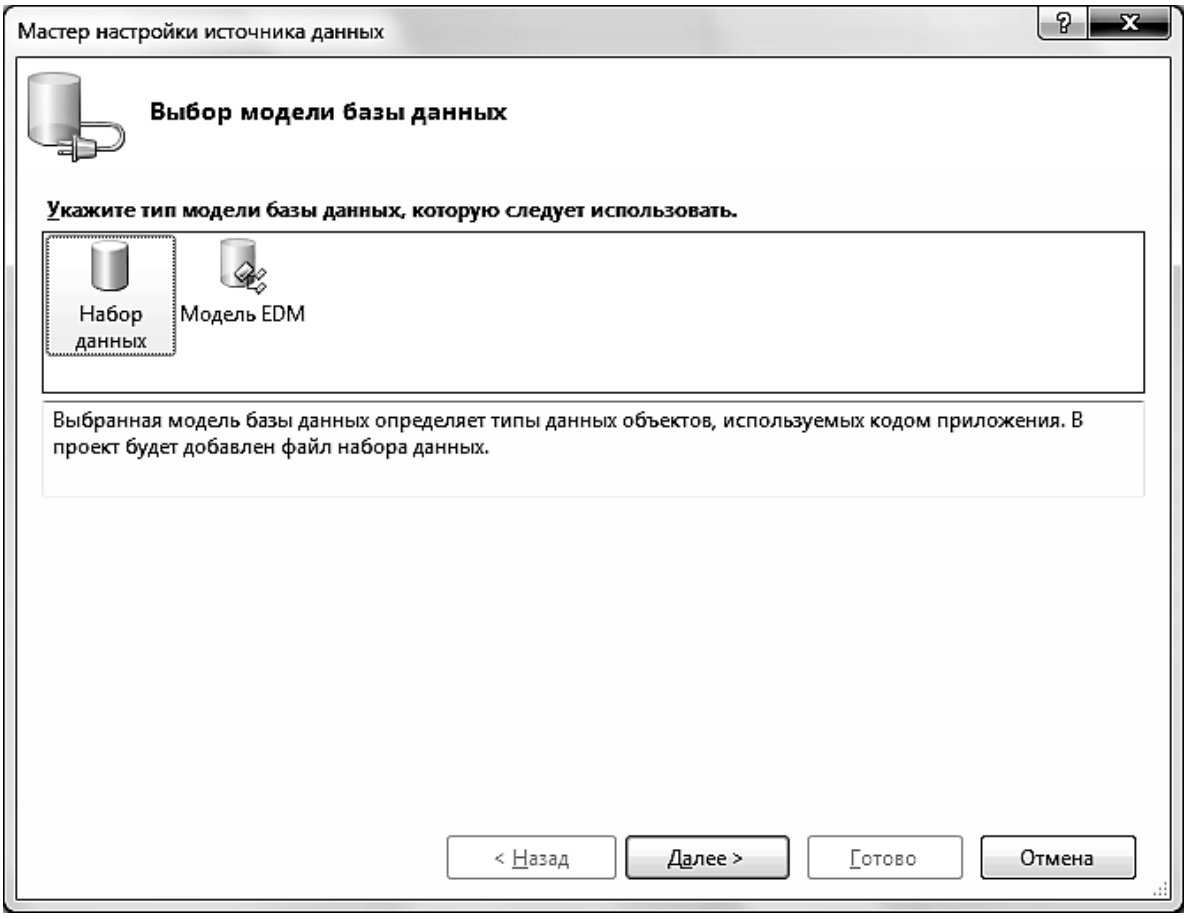


Рис. 7.37. **Останнє вікно під час створення порожньої бази даних**

Завдання 4

Сформувати базу даних на основі моделі EDM.

Виконання

1. Перейдіть у вікно конструктора моделей EDM, двічі клацнувши на значку моделі ***ХлібModel.edmx*** у вікні **Обозреватель решений**.

2. Клацніть правою кнопкою миші у вільному місці вікна конструктора моделей EDM і виберіть команду **Сформировать базу данных на основе модели** в контекстовому меню. З'являється вікно **Database Generation Workflow Manager**.

3. Погодьтеся зі стратегією **TablePerTypeStrategy**, клацнувши кнопку **Next**. З'являється вікно майстра формування бази даних.

4. У першому вікні майстра виберіть рядок з'єднання з базою даних ***Хліб.mdf*** і клацніть кнопку **Далее** (рис. 7.38).

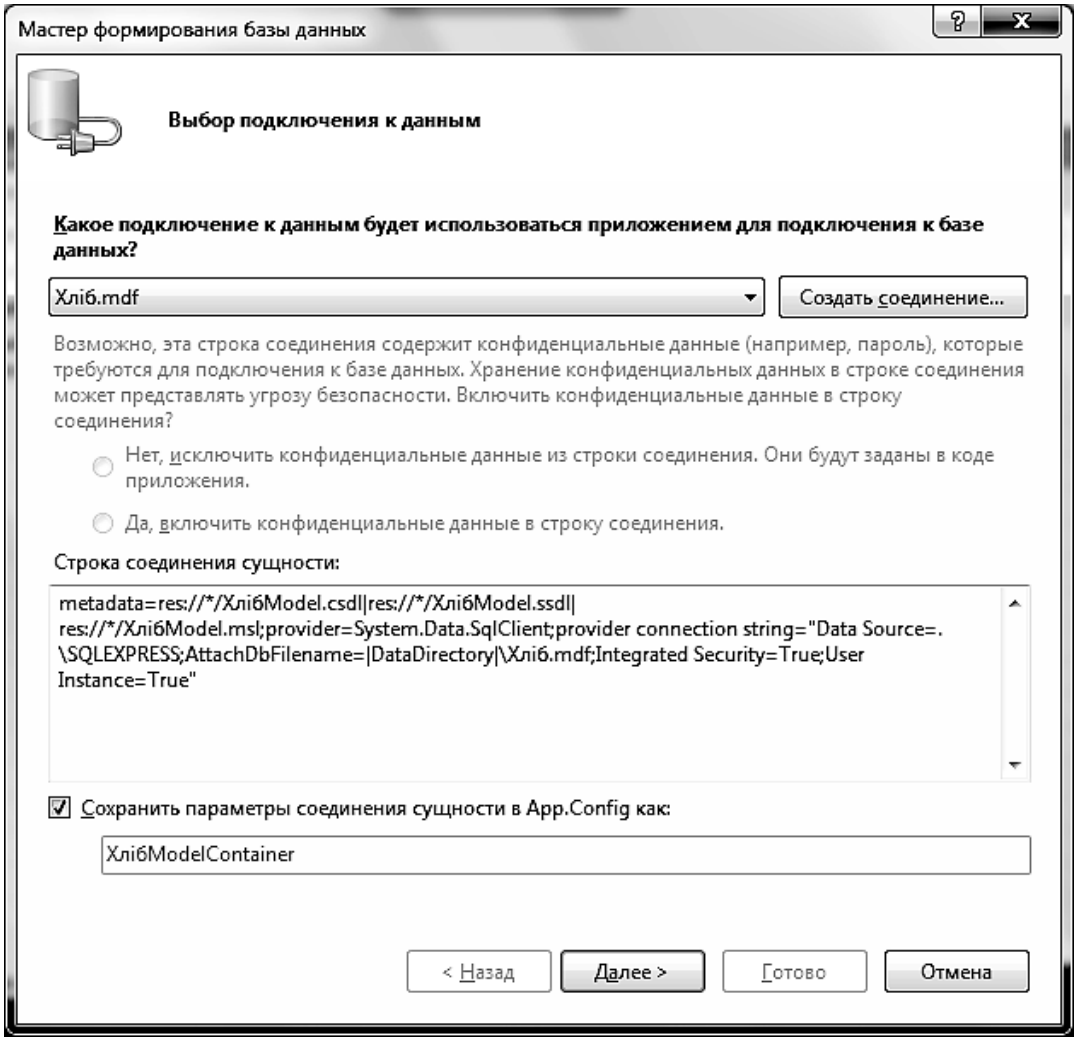


Рис. 7.38. **Вибір з'єднання з базою даних**

5. Після того, як майстер формування бази даних згенерує SQL-скрипт для створення бази даних, клацніть кнопку **Готово** у другому вікні майстра (рис. 7.39).

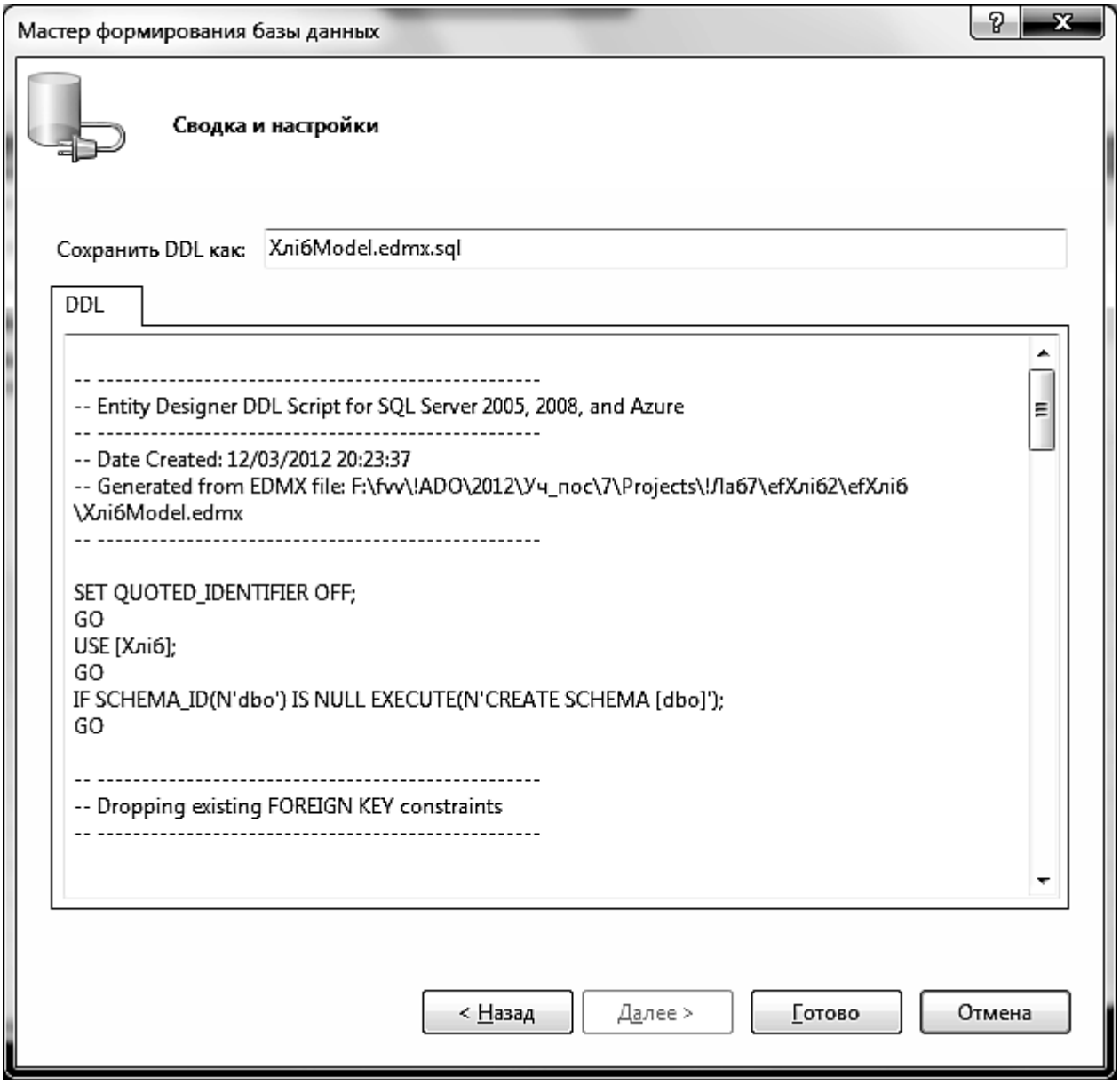


Рис. 7.39. **Вікно майстра з SQL-скриптом**

З'являється вікно редактора Transact-SQL (рис. 7.40).

6. У скрипті закоментуйте речення

```
USE [Хліб];
```

поставивши перед ним подвійний знак "мінус". Воно набуде такого вигляду:

```
--USE [Хліб];
```

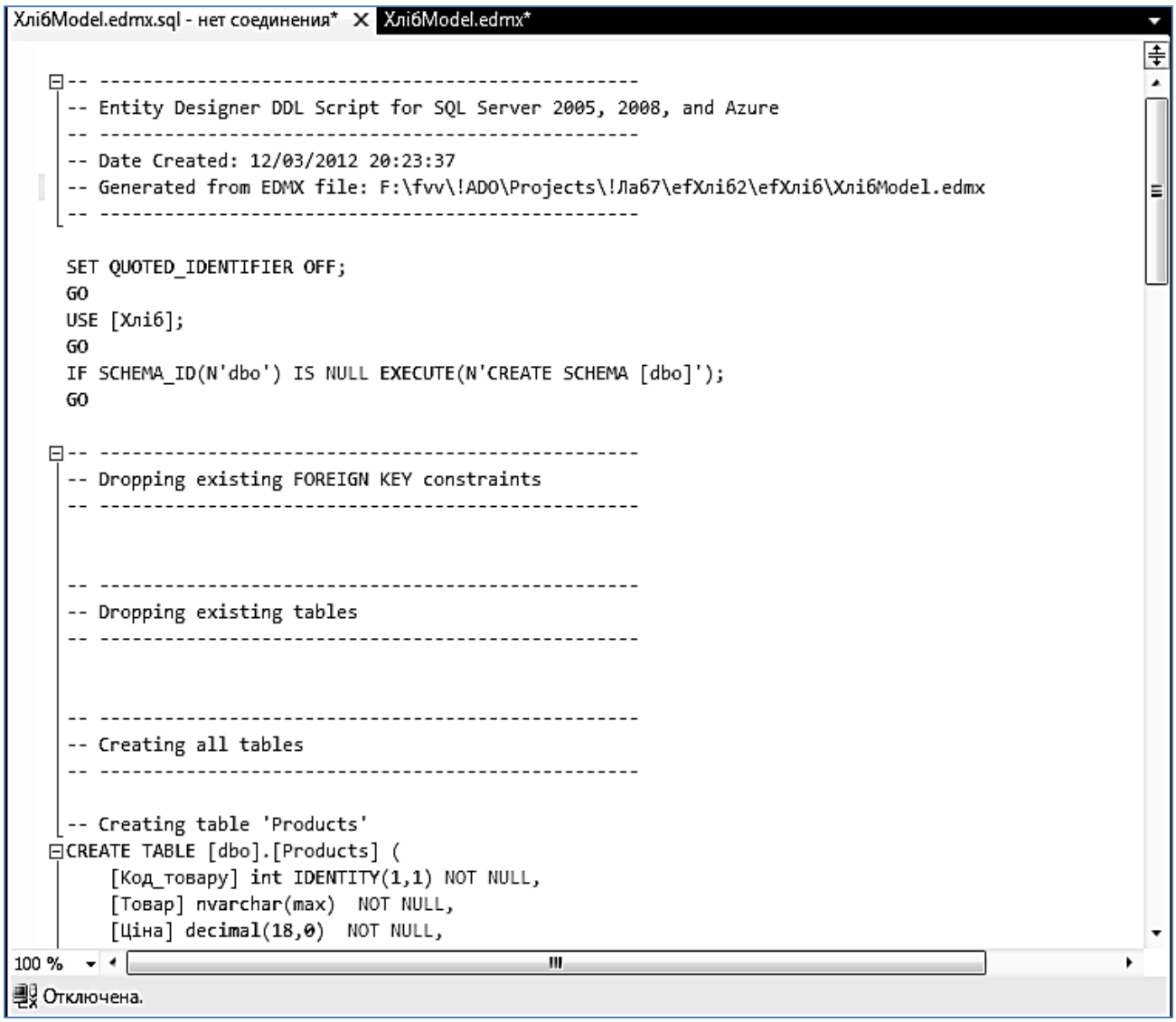


Рис. 7.40. **Вікно редактора Transact-SQL**

7. Виділіть значок бази даних ***Хліб.mdf*** у вікні **Обозреватель серверов**і скопіюйте значення властивості **Строка подключения** з вікна властивостей у буфер. Воно має такий формат:

```
Data Source=.\SQLEXPRESS; AttachDbFilename = Шлях до проекта
\efХлі6\Хлі6.mdf; Integrated Security=True;User Instance=True
```

8. Перейдіть у вікно редактора Transact-SQL з файлом ***ХлібModel.edmx.sql*** і клацніть кнопку **Выполнить SQL** на панелі редактора Transact-SQL.

9. Клацніть кнопку **Параметры** у вікні **Подключиться к компоненту Database Engine** (рис. 7.41).

Рис. 7.41. **Вікно *Подключиться к компоненту Database Engine***

10. Вставте з буфера рядок з'єднання у вкладці **Дополнительные параметры соединения** і клацніть кнопку **Соединить** (рис. 7.42).

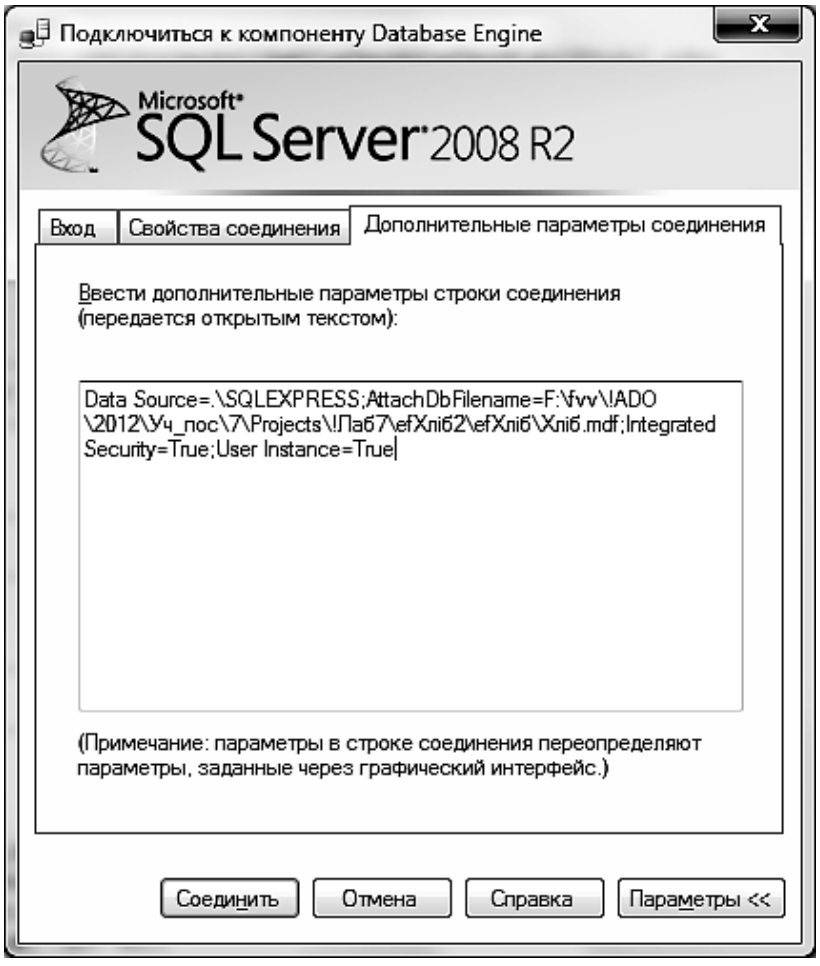


Рис. 7.42. Вкладка **Дополнительные параметры соединения**

Після з'єднання з базою даних створюються усі її таблиці і зв'язки між ними. Отриману базу даних можна переглянути у вікні **Обозреватель серверов**, а її схему – у вікні конструктора баз даних (рис. 7.43).

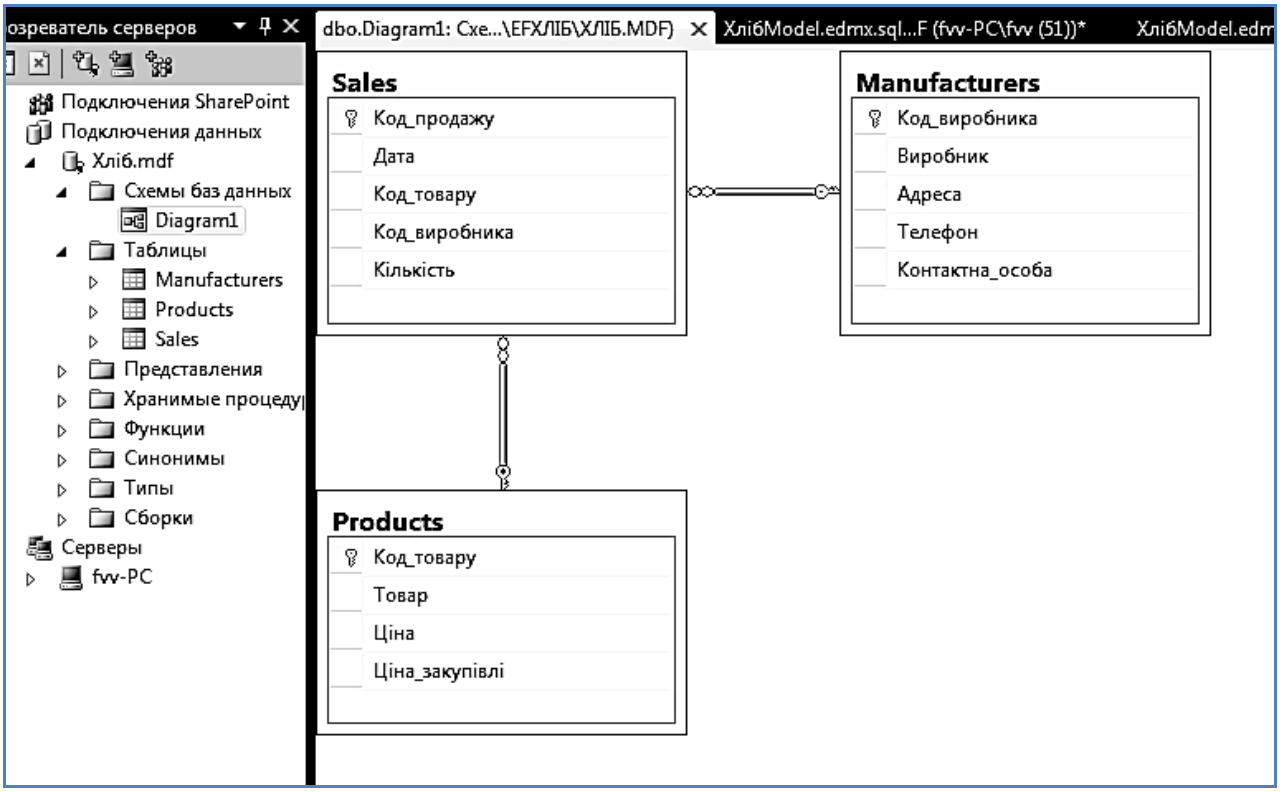


Рис. 7.43. **Вікна *Обозреватель серверов* і *Конструктор баз даних***

11. Збережіть зміни, що зроблені в проекті.

3. Сценарій DB First

Вивчення сценарію DB First здійснюється шляхом оновлення концептуальної моделі на основі зміненої бази даних. Воно виконується в 2 етапи:

1. Зміна бази даних.
2. Оновлення концептуальної моделі.

У результаті виконання цих етапів отримано базу даних, в якій додано облік надходжень товарів за накладними і відповідну концептуальну модель. Застосовуючи засоби платформи Entity Framework, можна використовувати другу для операцій з першою.

Завдання 1

Змінити базу даних, додавши до неї таблиці ***Invoices*** та ***Invoice Products*** (Накладні та Товари накладної).

Виконання

1. Перейдіть у вікно **Обозреватель серверов** і відкрийте вузол бази даних ***Хліб.mdf***, а потім – підвузол **Таблицы**. В останньому знаходяться три таблиці: Manufacturers, Sales та Products (рис. 7.44).

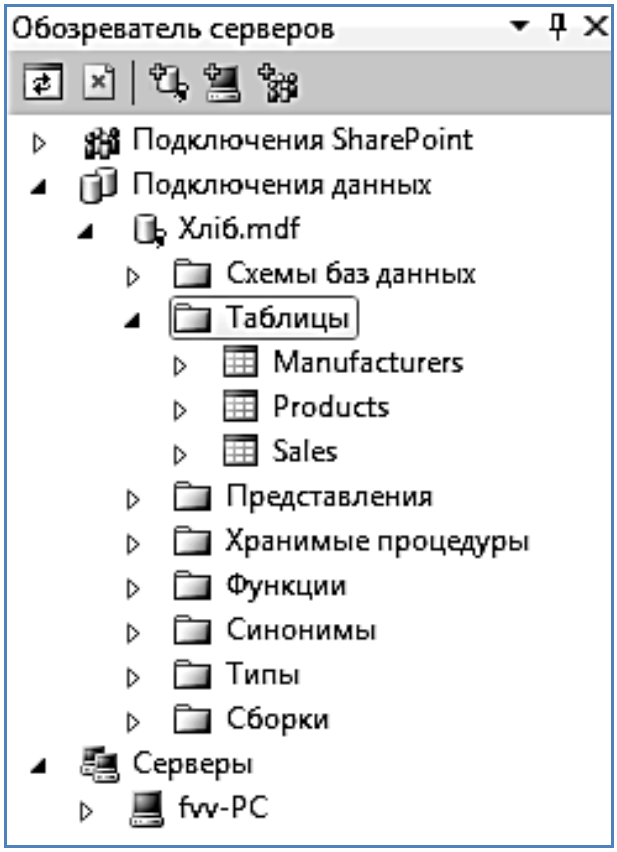


Рис. 7.44. **Підвузол *Таблицы* у вікні *Обозреватель серверов***

2. Клацніть правою кнопкою миші на підвузлі **Таблицы** і з контекстового меню виберіть команду **Добавить новую таблицу**.

3. Задайте схему таблиці ***Invoices*** у вікні конструктора таблиць, встановивши такі значення властивостей її полів:

Поле	Властивість	Значення
Код_накладної	Тип данных	int
	Спецификация идентифицирующего столбца	Да
	Первинний ключ	
Номер_накладної	Тип данных	varchar
	Длина	6
Дата	Тип данных	date
Код_виробника	Тип данных	int

Зовнішній вигляд схеми таблиці ***Invoices*** подано на рис. 7.45.

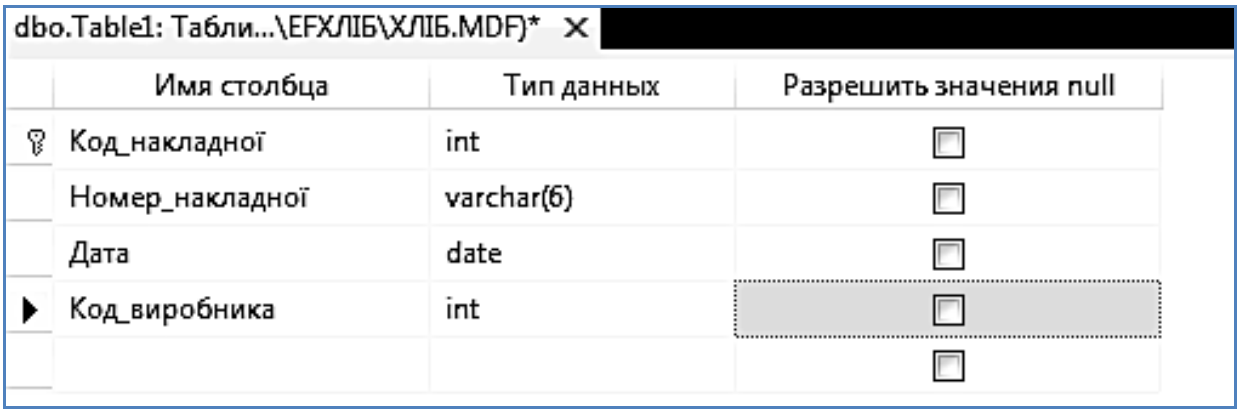


Рис. 7.45. **Схема таблиці *Invoices***

4. Клацніть кнопку **Сохранить** і введіть ім'я таблиці ***Invoices***.

5. Повторіть п.п. 2 – 4 для створення таблиці ***Invoice Products***, встановивши такі значення властивостей її полів:

Поле	Властивість	Значення
Код_товару_накладної	Тип данных	int
	Спецификация идентифицирующего столбца	Да
	Первинний ключ	
Код_накладної	Тип данных	int
Код_товару	Тип данных	int
Кількість	Тип данных	smallint

Зовнішній вигляд схеми таблиці ***Invoice Products*** подано на рис. 7.46.

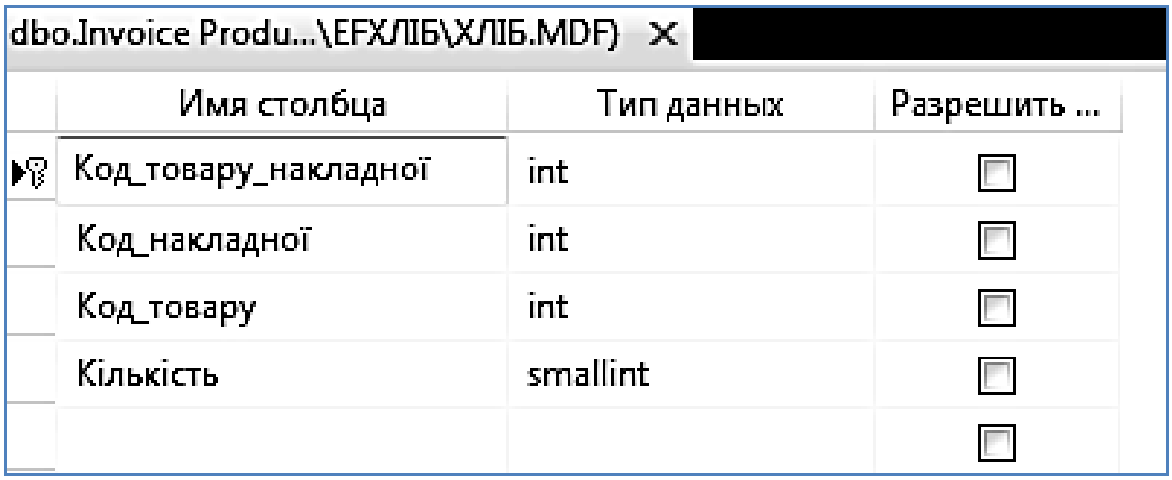


Рис. 7.46. **Схема таблиці *Invoice Products***

6. Встановіть зв'язки з новими таблицями. Для цього:

6.1. Відкрийте вікно діаграми, що знаходиться у підвузлі **Схемы баз данных** вікна **Обозреватель серверов**.

6.2. Клацніть правою кнопкою миші на вільному місці конструктора баз даних і з контекстового меню виберіть команду **Добавить таблицу**.

6.3. Додайте таблиці ***Invoices*** та ***Invoice Products*** за допомогою вікна **Добавление таблицы**.

6.4. Встановіть зв'язки між такими парами таблиць:

Manufacturers і ***Invoices***;

Invoices і ***Invoice Products***;

Products і ***Invoice Products***.

При цьому перетягуйте поле первинного ключа з батьківської таблиці на відповідне поле зовнішнього ключа дочірньої таблиці.

На рис. 7.47 подано зовнішній вигляд оновленої схеми бази даних.

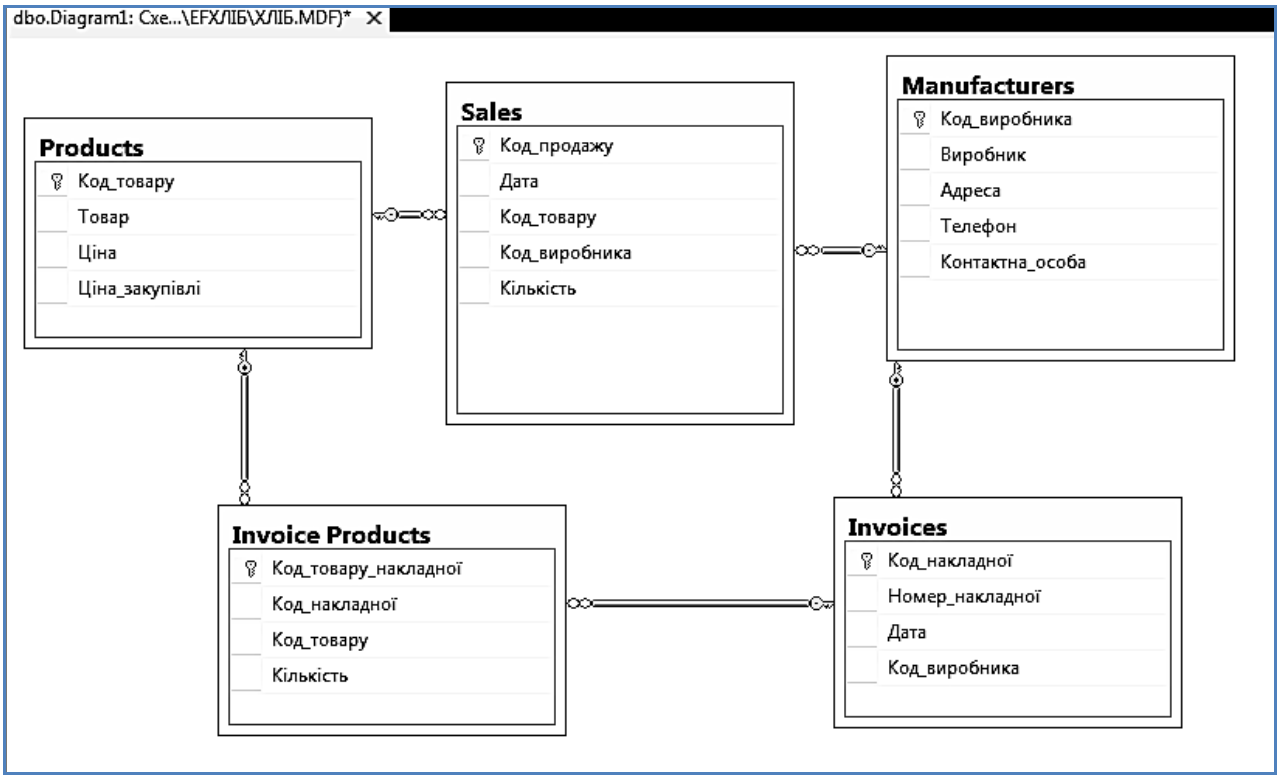


Рис. 7.47. **Схема бази даних *Хліб***

7. Збережіть зміни, що зроблені в проекті.

Завдання 2

Оновити концептуальну модель відповідно до тих змін, що зроблені у базі даних.

Виконання

1. Відкрийте вікно конструктора моделей EDM, двічі клацнувши на значку моделі ***ХлібModel.edmx*** у вікні **Обозреватель решений**.

2. Клацніть правою клавішею миші у вільному місці вікна конструктора моделей EDM і виберіть команду **Обновить модель из базы данных** у контекстовому меню. З'являється вікно майстра оновлення.

3. Виберіть таблиці ***Invoices*** і ***Invoice Products*** у вкладці **Добавить** і клацніть кнопку **Готово** (рис. 7.48).

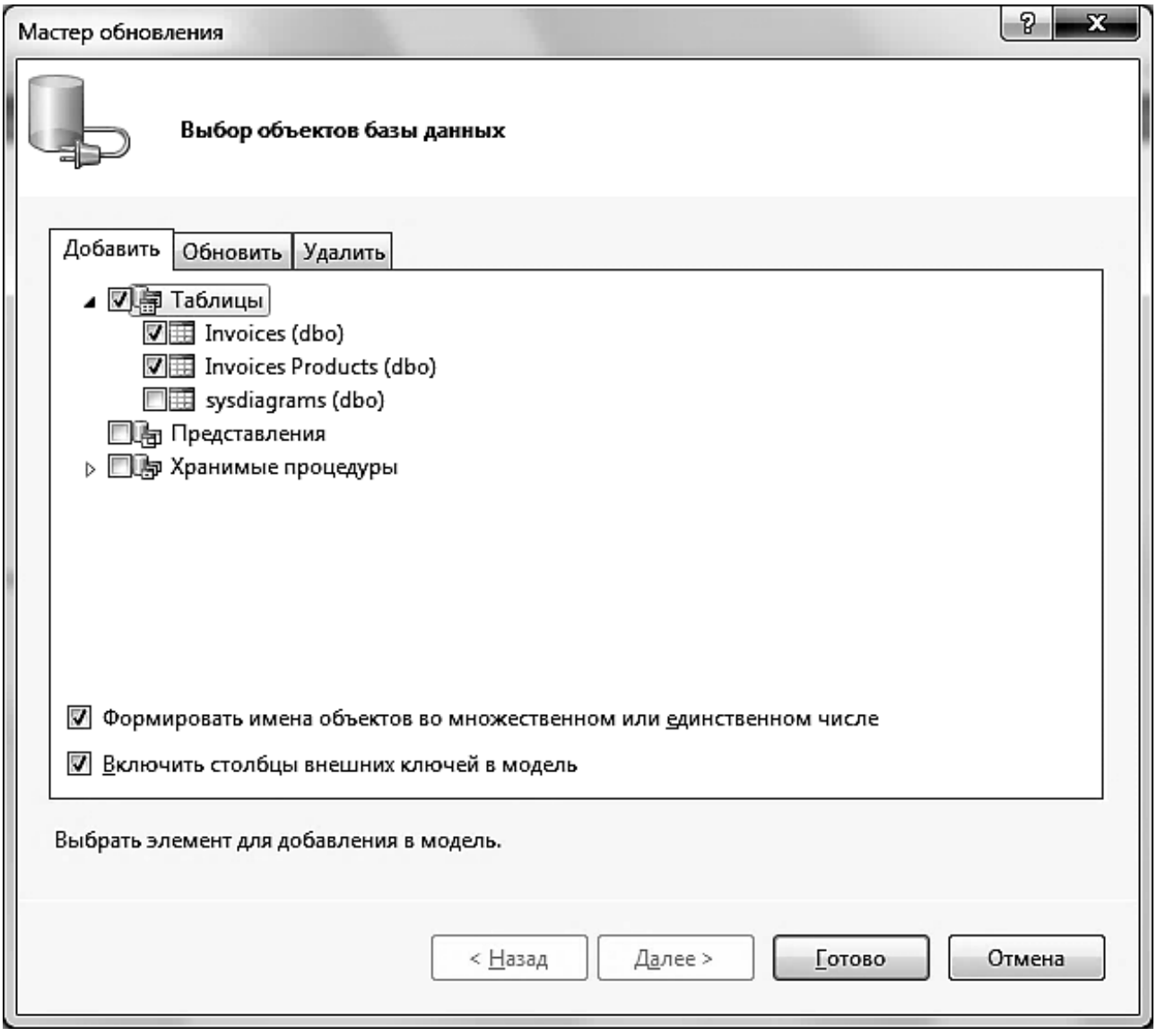


Рис. 7.48. **Вибір таблиць для додавання до моделі EDM**

У вікні конструктора моделей EDM до моделі ***ХлібModel.edmx*** додалися сутності ***Invoice*** та ***Invoice_Product*** з відповідними асоціаціями (рис. 7.49).

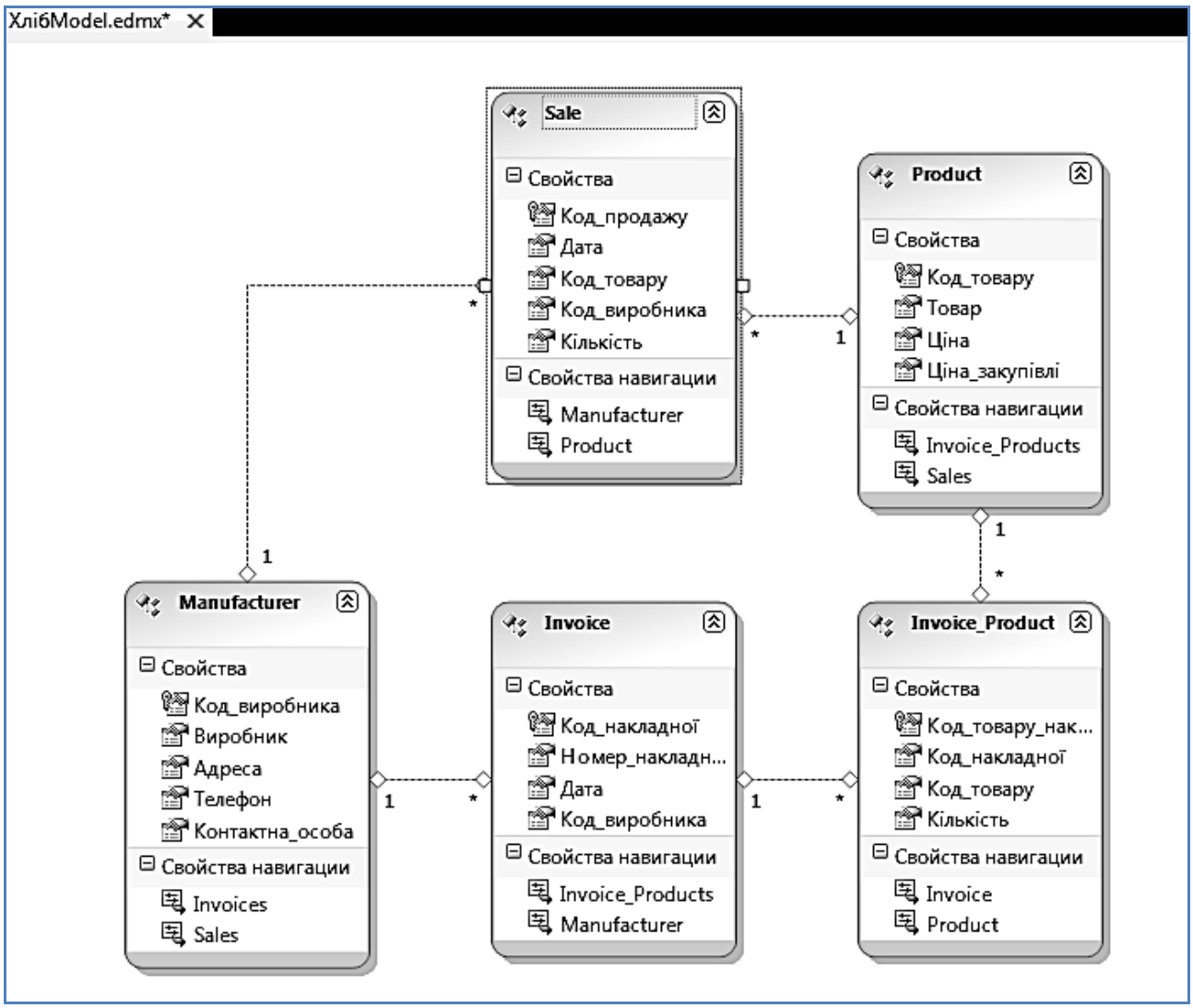


Рис. 7.49. **Модель EDM із доданими сутностями**

4. Перейдіть у вікно властивостей концептуальної моделі і змініть значення властивості **Имя контейнера сущностей** з ***ХлібModelContainer*** на ***ХлібEntities***. Аналогічні зміни виконайте у файлі **App.Config** для властивості **name**.

5. Збережіть зміни, що зроблені в проекті.

4. Операції CRUD в довідкових таблицях

Після створення концептуальної моделі її набори сутностей можна відображати в елементах керування на формі і виконувати з ними операції додавання, оновлення і зміни зі збереженням у базі даних. Для найпростіших сутностей ***Product*** і ***Manufacturer*** застосувати такі способи відображення даних:

1. Набір сутностей як джерело даних елемента керування.
2. Набір сутностей як джерело даних з'єднувача.

Перший спосіб застосувати для набору ***Products***, а другий – для набору ***Manufacturer***.

Завдання 1

Додати в застосування форму ***Товари***, у якій будуть відображатися й змінюватися дані набору сутностей ***Products*** (рис. 7.50).

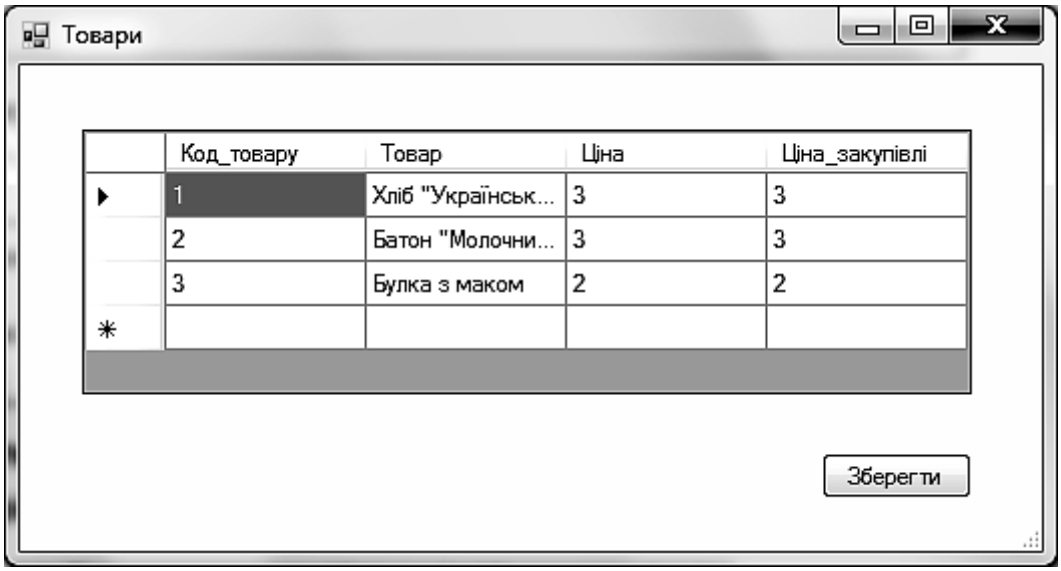


Рис. 7.50. **Форма *Товари***

Виконання

1. Додайте в застосування нову форму з ім'ям файла ***formТовари.cs*** і значенням ***Товари*** для властивості **Text**.

2. Додайте на форму елемент **DataGridView** з ім'ям ***gvТовари***, для відображення набору сутностей ***Products*** (рис. 7.51).

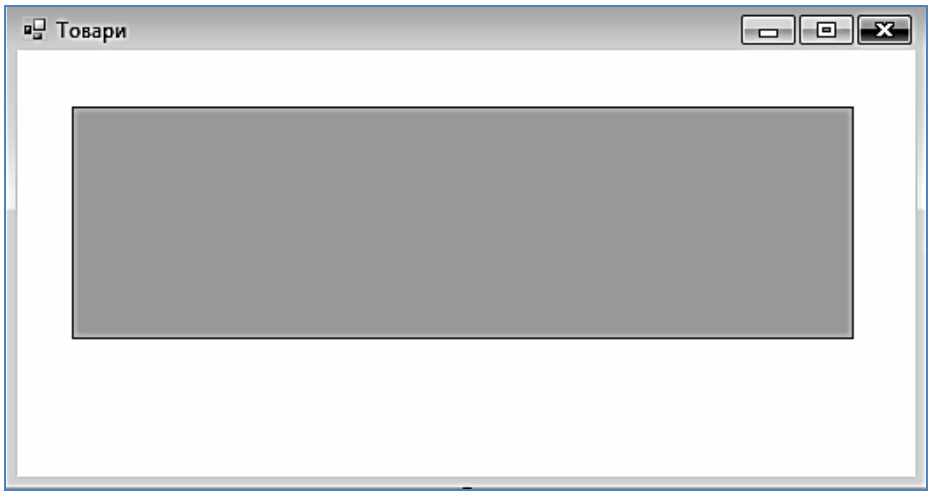


Рис. 7.51. **Елемент DataGridView на формі *Товари***

3. Двічі клацніть у вільному місці форми ***Товари*** й у вікні коду форми введіть оператори тіла оброблювача події ***formТовари_Load***. Він має такий вигляд:

```
ХлібEntities db;

private void formТовари_Load(object sender, EventArgs e)
{
    // Створюємо екземпляр класу ObjectContext
    db = new ХлібEntities();

    // Прив'язуємо набір сутностей до елемента DataGridView
    gvТовари.DataSource = db.Products;
}
```

4. Перейдіть у вікно конструктора форми ***formХліб***, двічі клацніть кнопку ***Товари*** й у вікні коду введіть код оброблювача події "Клацання кнопки Товари".

```
private void buttonТовари_Click(object sender, EventArgs e)
{
    formТовари вікноТовари = new formТовари();
    вікноТовари.ShowDialog();
}
```

5. Перевірте функціональність форми. Після її завантаження повинні відображатися всі властивості сутності ***Product***, і тому числі властивості навігації ***Sales*** та ***Invoice_Products***. Останні потрібно вилучити.

6. Перейдіть у вікно коду форми ***formТовари*** й додайте оператори в кінець тіла оброблювача події ***formТовари_Load*** для вилучення стовпців ***Sales*** та ***Invoice_Products*** з елемента DataGridView. Після цього оброблювач набуде такого вигляду:

```
private void formТовари_Load(object sender, EventArgs e)
{
    // Створюємо екземпляр класу ObjectContext
    db = new ХлібEntities();
    // Прив'язуємо набір сутностей до елемента DataGridView
    gvТовари.DataSource = db.Products;
    // Вилучаємо властивості навігації з інтерфейсу
    gvТовари.Columns.Remove("Invoice_Products");
    gvТовари.Columns.Remove("Sales");
}
```

7. Для збереження змін у базі даних додайте на форму ***formТовари*** кнопку ***Зберегти*** і код її оброблювача.

```
private void buttonЗберегти_Click(object sender, EventArgs e)
{
    db.SaveChanges();
}
```

8. Перевірте функціональність кнопки ***Зберегти***, додавши дані про товари і зберігши їх у базі даних. Потрібно ввести такі дані:

Товар	Ціна	Ціна_закупівлі
Хліб "Український"	3,00	2,50
Батон "Молочний"	2,80	2,50
Булка з маком	2,00	1,80

9. Збережіть зміни, що зроблені в проекті.

Завдання 2

Додати в застосування форму ***Виробники***, у якій будуть відображатися й змінюватися дані набору сутностей ***Manufacturers*** (рис. 7.52).

Рис. 7.52. **Форма *Виробники***

Виконання

1. Додайте в застосування нову форму з ім'ям файла ***formВиробники.cs*** і значенням ***Виробники*** для властивості **Text**.

2. Додайте на форму елемент **DataGridView** з ім'ям ***gvВиробники***, для відображення набору сутностей ***Manufacturers***.

3. Двічі клацніть у вільному місці форми ***Виробники*** й у вікні коду форми введіть оператори тіла оброблювача події ***formВиробники_Load***. Він має такий вигляд:

```
ХлібEntities db;

private void formВиробники_Load(object sender, EventArgs e)
{
    // Створюємо екземпляр класу ObjectContext
    db = new ХлібEntities();

    // Створюємо з'єднувача
    BindingSource bindingВиробники = new BindingSource();

    // Вказуємо джерело даних для з'єднувача
    bindingВиробники.DataSource = db.Manufacturers;

    // Відображаємо дані
    gvВиробники.DataSource = bindingВиробники;

    //Вилучаємо властивості навігації з інтерфейсу
    gvВиробники.Columns.Remove("Invoices");
    gvВиробники.Columns.Remove("Sales");

}
```

4. Перейдіть у вікно конструктора форми ***formХліб***, двічі клацніть кнопку ***Виробники*** й у вікні коду введіть код оброблювача події "Клацання кнопки Товари".

```
private void buttonВиробники_Click(object sender, EventArgs e)
{
    formВиробники вікноВиробники = new formВиробники();
    вікноВиробники.ShowDialog();
}
```

5. Для збереження змін у базі даних додайте на форму ***formВиробники*** кнопку ***Зберегти*** і код її оброблювача.

```
private void buttonЗберегти_Click(object sender, EventArgs e)
{
    db.SaveChanges();
}
```

6. Перевірте функціональність кнопки ***Зберегти***, додавши дані про виробників і зберігши їх у базі даних. Потрібно ввести такі дані:

Виробник	Адреса	Телефон	Контактна_особа
Х/з "Салтівський"	вул. Гв. Широнінців, 1	(057)710-50-40	Іванов І. І.
Х/з "Кулиничі"	смт Кулиничі, вул. Шкільна, 18	(0572)62-51-37	Петренко П. П.

7. Збережіть зміни, що зроблені в проекті.

5. Робота з допоміжними сутностями

Для побудови форми слід використовувати візуальні засоби на основі панелі **Источники данных**. Побудова форми виконується в 2 етапи:

1. Додавання джерела даних на основі об'єкта.
2. Створення форми візуальними засобами.

У результаті виконання цих етапів буде отримано форму ***Продажі***, в якій можна переглядати, додавати, видаляти і змінювати дані сутності ***Sale***. Для зручності роботи з даними слід використати як допоміжні дані сутностей ***Product*** та ***Manufacturer*** (рис. 7.53).

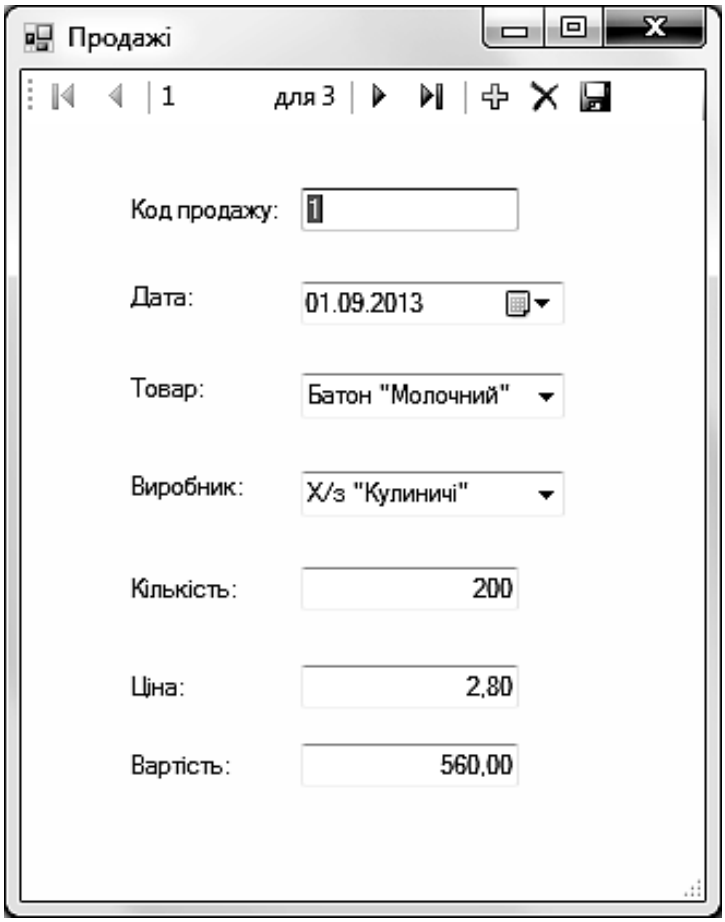


Рис. 7.53. **Форма *Продажі***

Завдання 1

Додати в застосування джерела даних на основі об'єкта ***ХлібEntities***.

Виконання

1. Викличте майстра настроювання джерела даних, вибравши команди **Добавить новый источник данных**, що знаходиться в меню **Данные**.

2. Виберіть значок **Объект** у вікні **Выбор типа источника данных** і клацніть кнопку **Далее** (рис. 7.54).

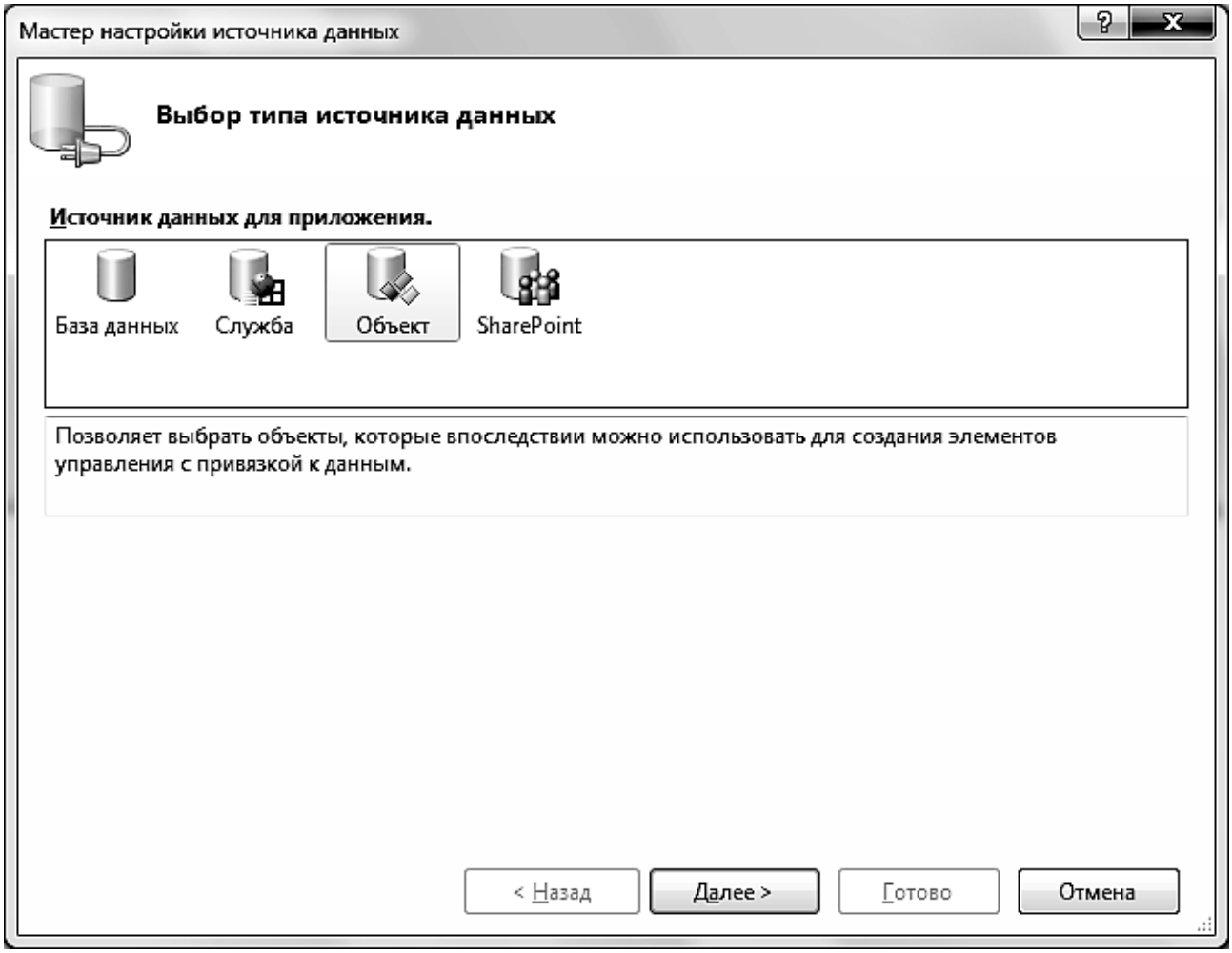


Рис. 7.54. **Вікно *Выбор типа источника данных***

3. Виберіть сутності ***Invoice***, ***Invoice Product***, ***Manufacturer***, ***Product*** і ***Sale***, дані яких знадобляться для відображення, у вікні **Выбор объектов данных** і клацніть кнопку **Готово** (рис. 7.55).

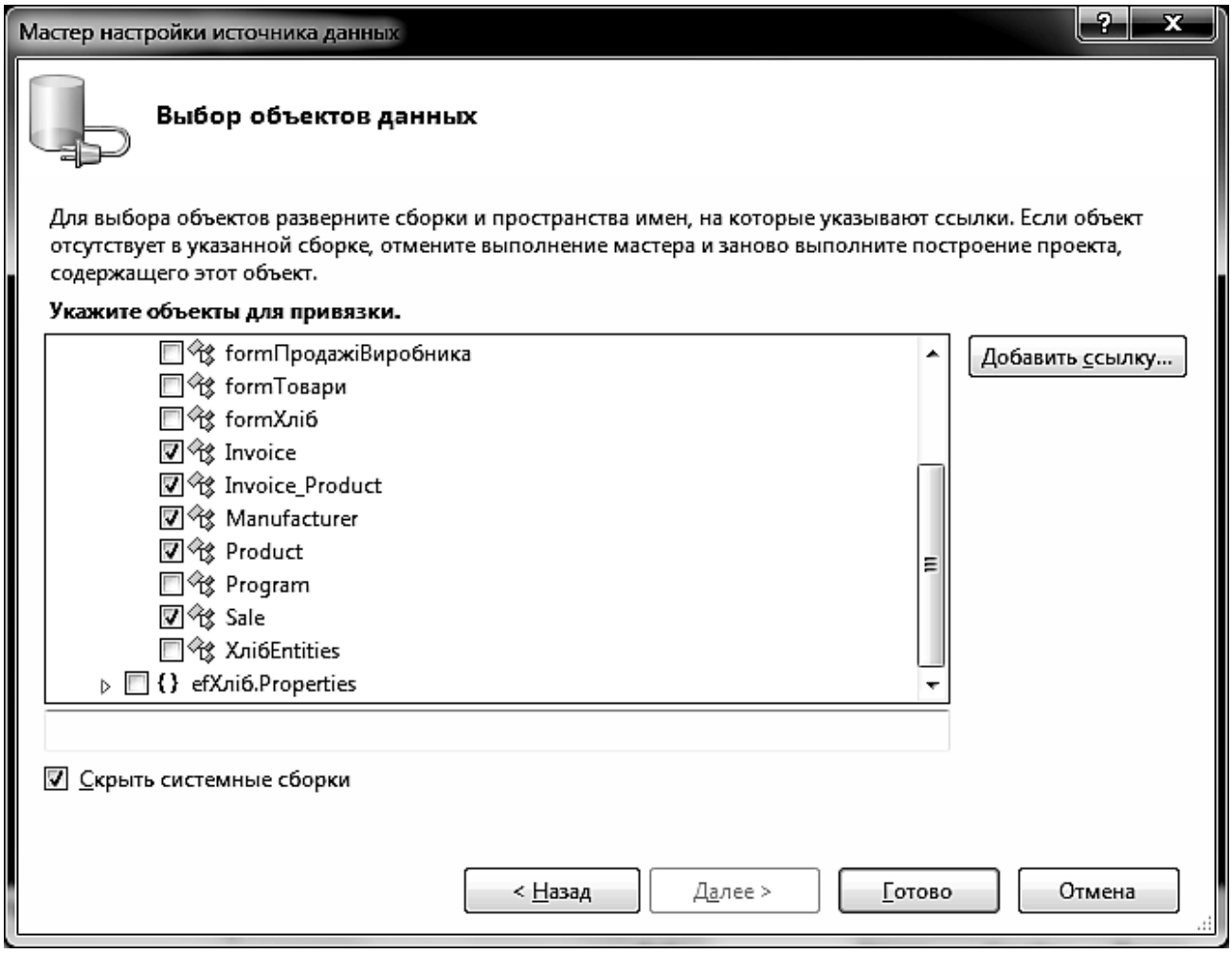


Рис. 7.55. **Вікно *Выбор объектов данных***

У результаті роботи майстра у вікні **Источники данных** з'явилося зображення вибраних сутностей. Кожна сутність тут подана вузлом у вигляді значка. Елементами вузла є імена властивостей. Якщо сутність має асоціацію з іншою сутністю, то остання відображається у вигляді підвузла. На рис. 7.56 у вузлі сутності ***Sale*** відображаються її властивості та дві батьківські сутності – ***Manufacturer*** і ***Product***. Вони будуть використовуватися під час побудови форми.

Примітка. На рис. 7.56 показано вікно **Источники данных**, коли поряд відкрите вікно коду. Якщо поряд з вікном **Источники данных** відкрите вікно конструктора форми, об'єкти вікна **Источники данных** набувають вигляду елементів інтерфейсу (форми та елементів керування). Саме в цьому режимі їх використовують для побудови вікон шляхом перетягування вузлів чи окремих елементів на форму. У разі потреби тут можна змінити вид елементів керування, скориставшись списком, що розкривається. Він розташований у правому кінці кожного елемента.

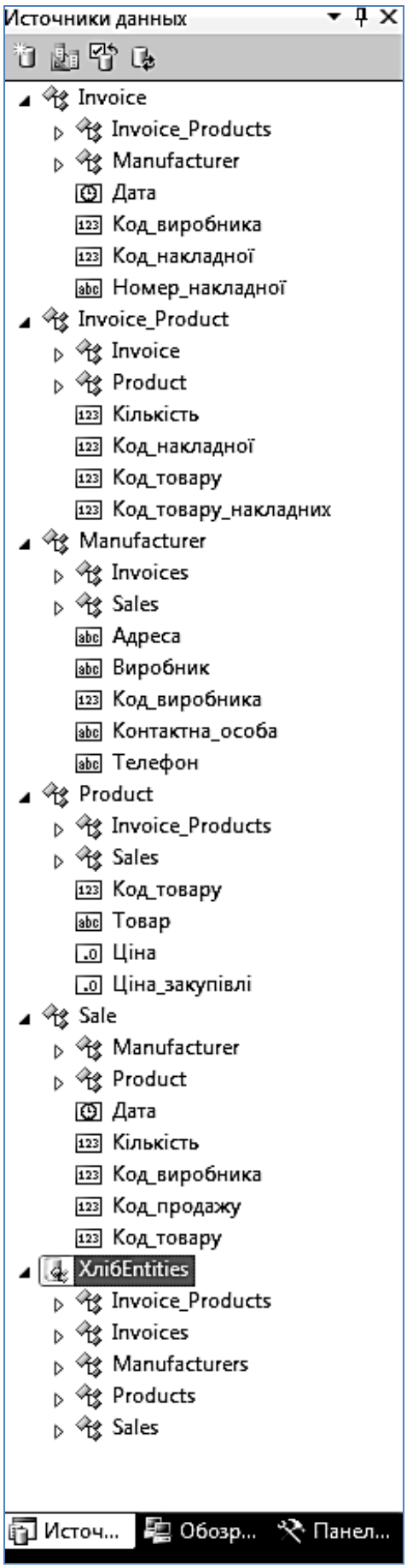


Рис. 7.56. **Вікно *Источники данных***

Завдання 2

Створити форму ***Продажі*** з використанням візуальних засобів (рис. 7.57).

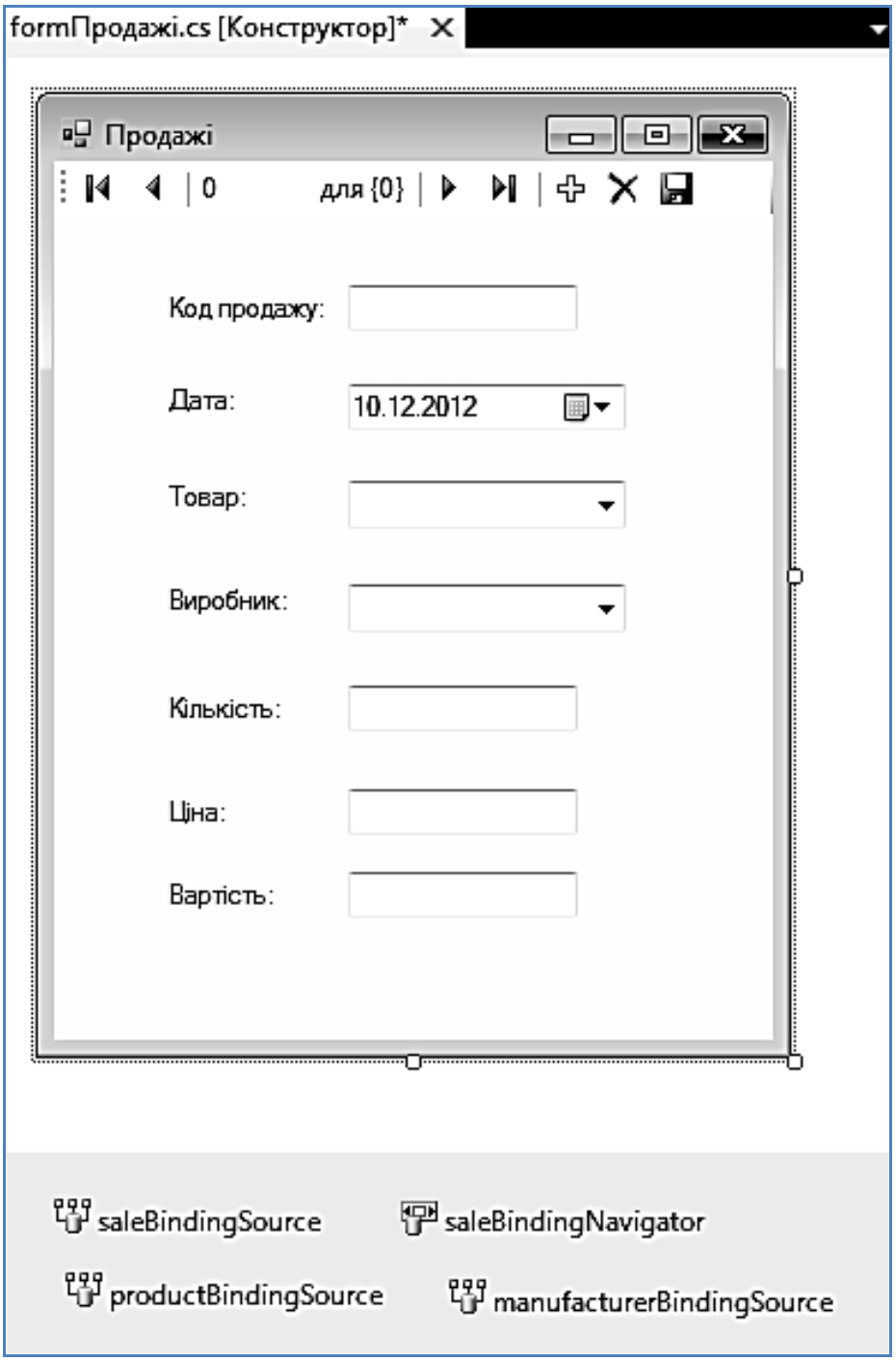


Рис. 7.57. **Форма *Продажі* в конструкторі**

Виконання

1. Додайте в застосування нову форму, з ім'ям файла ***formПродажі.cs*** і значенням ***Продажі*** для властивості **Text**.

2. Клацніть вузол сутності ***Sale*** у вікні **Источники данных** і з її списку, що розкривається, виберіть подання **Таблиця** (**Details**).

3. Виберіть подання **ComboBox** у вузлі ***Sale*** для властивості ***Код_товару*** у списку, що розкривається.

4. Повторіть п. 3 для властивості ***Код_виробника***.

5. Перетягніть з вузла ***Sale*** на форму ***Продажі*** по черзі такі властивості:

Код_продажу,

Дата,

Код_товару,

Код_виробника,

Кількість.

На формі з'явилися елементи керування для відображення даних й панель навігатора, а в області компонентів – ряд об'єктів, які забезпечують роботу з даними сутності (рис. 7.58). Ознайомтеся з ними й визначте призначення кожного.

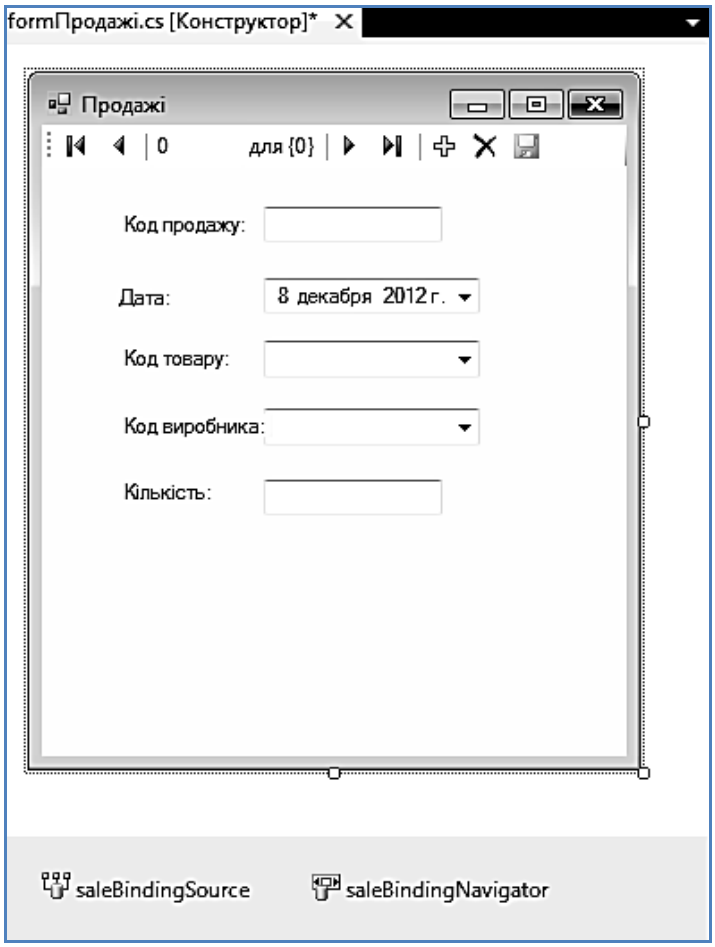


Рис. 7.58. **Вікно форми *Продажі* після додавання сутності *Sale***

6. Клацніть елемент DateTimePicker і для його властивості **Format** встановіть значення ***Short***.

7. Змініть властивість **Text** для написів ***Код_товару*** й ***Код_виробника***, задавши нові значення ***Товар*** і ***Виробник*** відповідно.

8. Щоб в елементі ComboBox для товару відображалися назви товарів, перетягніть вузол сутності ***Product*** з вікна ***Источники данных*** на цей ComboBox і відпустіть. Потім для елемента ComboBox***Товар*** встановіть такі значення властивостей:

Властивість	Значення
(DataBidings)-SelectedValue	saleBindingSource - Код_товару
DataSource	productBindingSource
DisplayMember	Товар
ValueMember	Код_товару

Примітка. Значення цих чотирьох властивостей краще встановлювати у вікні задач ComboBox (рис. 7.59).

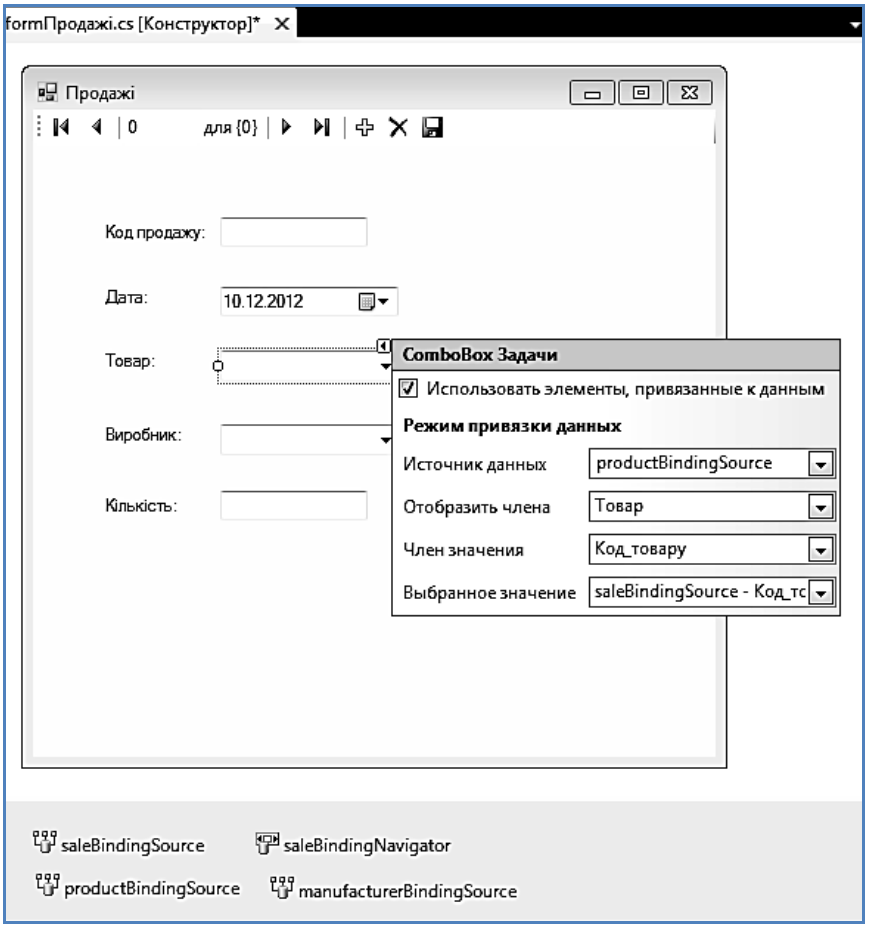


Рис. 7.59. **Встановлення значень властивостей прив'язки для елемента ComboBox**

9. Повторіть п. 8 для ComboBox ***Виробник***.

Примітка. В область компонентів форми автоматично додалися компоненти BindingSource для сутностей ***Product*** й ***Manufacturer***.

10. Двічі клацніть у вільному місці форми ***Продажі*** й у вікні коду форми введіть оператори тіла оброблювача події ***formПродажі_Load***. Він має такий вигляд:

```
ХлібEntities db;

private void formПродажі_Load(object sender, EventArgs e)
{
    // Створюємо екземпляр класу ObjectContext
    db = new ХлібEntities();

    // Завантажуємо дані для productBindingSource
    // та manufacturerBindingSource
    productBindingSource.DataSource = db.Products;
    manufacturerBindingSource.DataSource = db.Manufacturers;

    // Завантажуємо дані для saleBindingSource
    saleBindingSource.DataSource = db.Sales;
}
```

11. Перейдіть у вікно конструктора форми ***Продажі***, клацніть правою кнопкою миші на клавішу ***Зберегти***, що розташована елементі ***saleBindingNavigator*** і виберіть значення **Enable** з контекстового меню. Потім додайте код оброблювача події "Клацання кнопки Зберегти".

```
private void saleBindingNavigatorSaveItem_Click(object sender,
EventArgs e)
{
    saleBindingSource.EndEdit();
    db.SaveChanges();
}
```

12. Перейдіть у вікно конструктора форми ***Хліб*** і додайте код оброблювача події "Клацання кнопки Продажі":

```
private void buttonПродажі_Click(object sender, EventArgs e)
{
    formПродажі вікноПродажі = new formПродажі();
    вікноПродажі.ShowDialog();
}
```

13. Запустіть програму на виконання й перевірте функціональність форми ***Продажі***, додавши дані про продаж одного товару і зберігши їх (рис. 7.60).

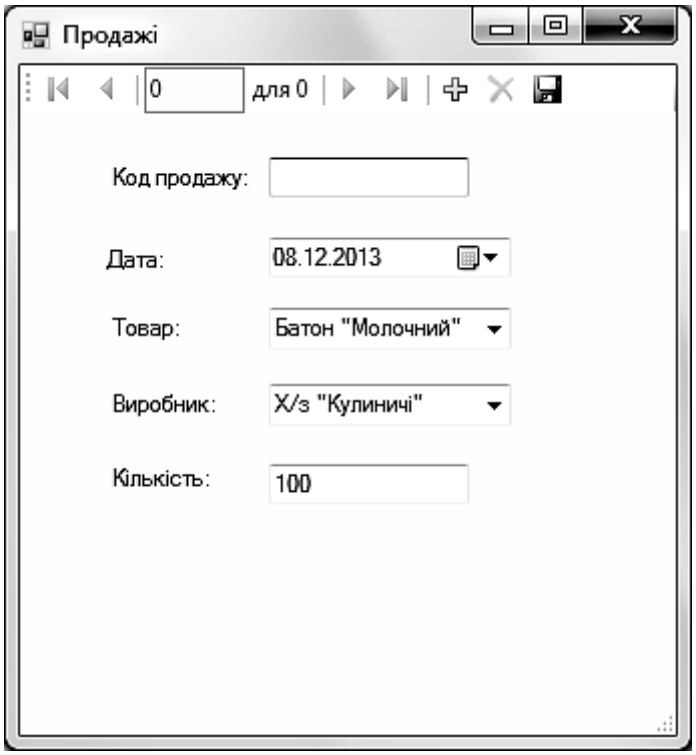


Рис. 7.60. **Додавання даних про продаж одного товару**

14. Закрийте форми ***Продажі*** й ***Хліб*** і збережіть зміни, що зроблені в проекті.

15. Додайте на форму текстове поле ***textBoxЦіна*** з відповідним підписом. Потім перейдіть у вікно коду форми і введіть такий оператор у кінці тіла оброблювача події ***formПродажі_Load***:

```
   //Прив'язуємо Ціну з productBindingSource до цінаTextBox
   цінаTextBox.DataBindings.Add("Text", productBindingSource, "Ціна");
```

16. Додайте на форму текстове поле ***textBoxВартість*** з відповідним підписом. Потім додайте до проекту клас ***Sale.cs*** і введіть його код.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace efХліб
{
```

```
    public partial class Sale
    {
        public decimal Вартість
        {
            get
            {
                if (Product != null)
                    return Product.Ціна * Кількість;
                else
                    return 0;
            }
        }
    }
}
```

17. Перейдіть у вікно коду форми ***formПродажі*** й додайте оператор у кінець тіла оброблювача події ***formПродажі_Load*** для відображення властивості ***Вартість***:

```
//Відображаємо властивість Вартість
вартістьTextBox.DataBindings.Add("Text",saleBindingSource,"Вартість");
```

Після цього оброблювач набуде такого вигляду:

```
private void formПродажі_Load(object sender, EventArgs e)
{
    // Створюємо екземпляр класу ObjectContext
    db = new Хліб6Entities();

    // Завантажуємо дані для productBindingSource
    // та manufacturerBindingSource
    productBindingSource.DataSource = db.Products;
    manufacturerBindingSource.DataSource = db.Manufacturers;

    // Завантажуємо дані для saleBindingSource
    saleBindingSource.DataSource = db.Sales;

    //Прив'язуємо Ціну з productBindingSource до цінаTextBox
   цінаTextBox.DataBindings.Add("Text", productBindingSource, "Ціна");
    //Відображаємо властивість Вартість
    вартістьTextBox.DataBindings.Add("Text", saleBindingSource, "Вартість");
}
```

18. Запустіть програму на виконання й перевірте функціональність форми ***Продажі***, додавши дані про продаж кількох товарів і зберігши їх.

6. Керування ієрархічними сутностями

Завдання

Додати в застосування форму ***Накладні***, у якій будуть відображатися й змінюватися дані сутностей ***Invoice***та ***Invoice_Product***, що пов'язані відношенням один-до-багатьох (рис. 7.61).

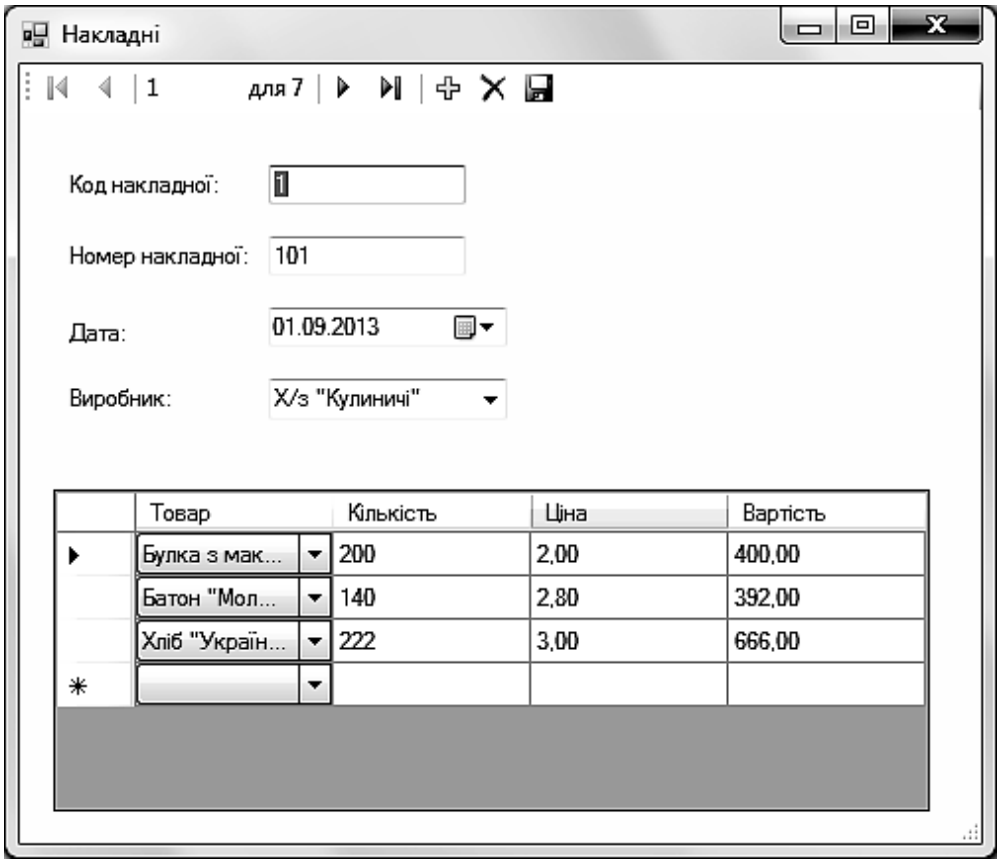


Рис. 7.61. **Форма *Накладні***

Виконання

1. Додайте в застосування нову форму, давши їй ім'я ***formНакладні***.
2. Клацніть вузол сутності ***Invoice*** у вікні **Источники данных** і з її списку, що розкривається, виберіть подання **Таблиця** (**Details**).
3. Виберіть подання **ComboBox** у вузлі ***Invoice*** для властивості ***Код_виробника*** у списку, що розкривається.
4. Перетягніть з вузла ***Invoice*** на форму ***Накладні*** по черзі такі властивості:

Код_накладної,

Номер_накладної,

Дата,

Код_виробника.

На формі з'явилися елементи керування для відображення даних й панель навігатора, а в області компонентів – об'єкти, які забезпечують роботу з даними сутності (рис. 7.62).

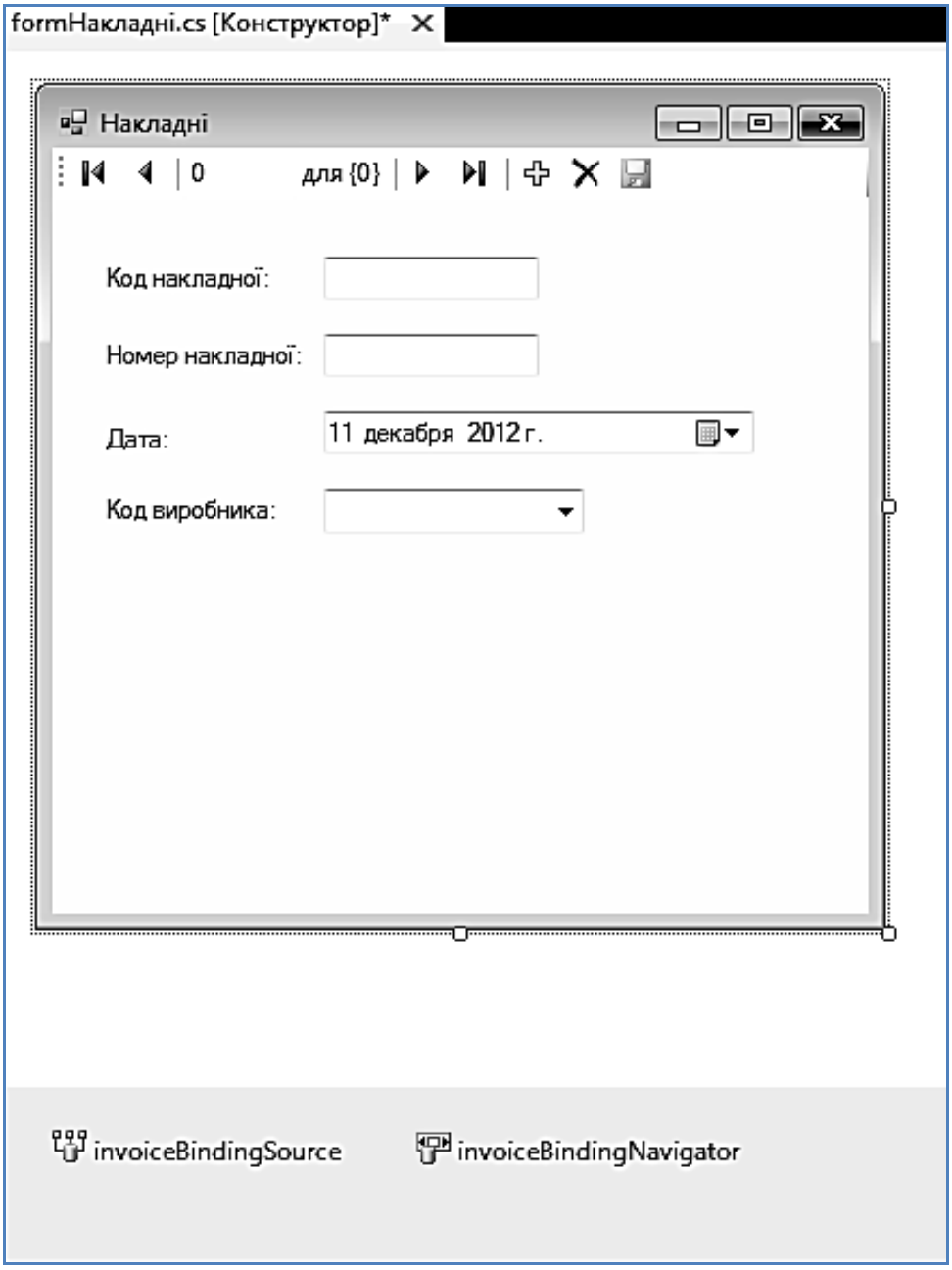


Рис. 7.62. **Вікно форми *Накладні* після додавання сутності *Invoice***

5. Налаштуйте елемент DateTimePicker, встановивши для його властивості **Format** значення ***Short***, а також змініть значення властивості **Text** для напису ***Код_виробника***, установивши нове значення ***Виробник***.

6. Щоб в елементі ComboBox відображалися назви виробників, перетягніть вузол сутності ***Manufacturer*** з вікна ***Источники данных*** на цей ComboBox і відпустіть. Потім для елемента ComboBox ***Виробник*** встановіть такі значення властивостей:

Властивість	Значення
(DataBidings)-SelectedValue	invoiceBindingSource - Код_виробника
DataSource	manufacturerBindingSource
DisplayMember	Виробник
ValueMember	Код_виробника

7. Двічі клацніть у вільному місці форми ***Накладні*** й у вікні коду форми введіть оператори тіла оброблювача події ***formНакладні_Load***. Він має такий вигляд:

```
Хліб Entities db;

private void formНакладні_Load(object sender, EventArgs e)
{
    // Створюємо екземпляр класу ObjectContext
    db = new Хліб Entities();

    // Завантажуємо дані для manufacturerBindingSource
    manufacturerBindingSource.DataSource = db.Manufacturers;

    // Завантажуємо дані для invoiceBindingSource
    invoiceBindingSource.DataSource =
      db.Invoices.Execute(System.Data.Objects.MergeOption.AppendOnly);
      //Execute - для успішного збереження доданих накладних
}
```

8. Перейдіть у вікно конструктора форми ***Накладні***, клацніть правою кнопкою миші на кнопці ***Зберегти***, що розташована елементі ***invoiceBindingNavigator*** і виберіть значення **Enable** з контекстового меню. Потім додайте код оброблювача події "Клацання кнопки Зберегти".

```
private void invoiceBindingNavigatorSaveItem_Click(object sender,
EventArgs e)
{
    invoiceBindingSource.EndEdit();
    db.SaveChanges();
}
```

9. Перейдіть у вікно конструктора форми ***Хліб*** і додайте код оброблювача події "Клацання кнопки Накладні":

```
private void buttonНакладні_Click(object sender, EventArgs e)
{
    formНакладні вікноНакладні = new formНакладні();
    вікноНакладні.ShowDialog();
}
```

10. Запустіть програму на виконання й перевірте функціональність форми ***Накладні***, додавши дані про накладну і зберігши їх (рис. 7.63).

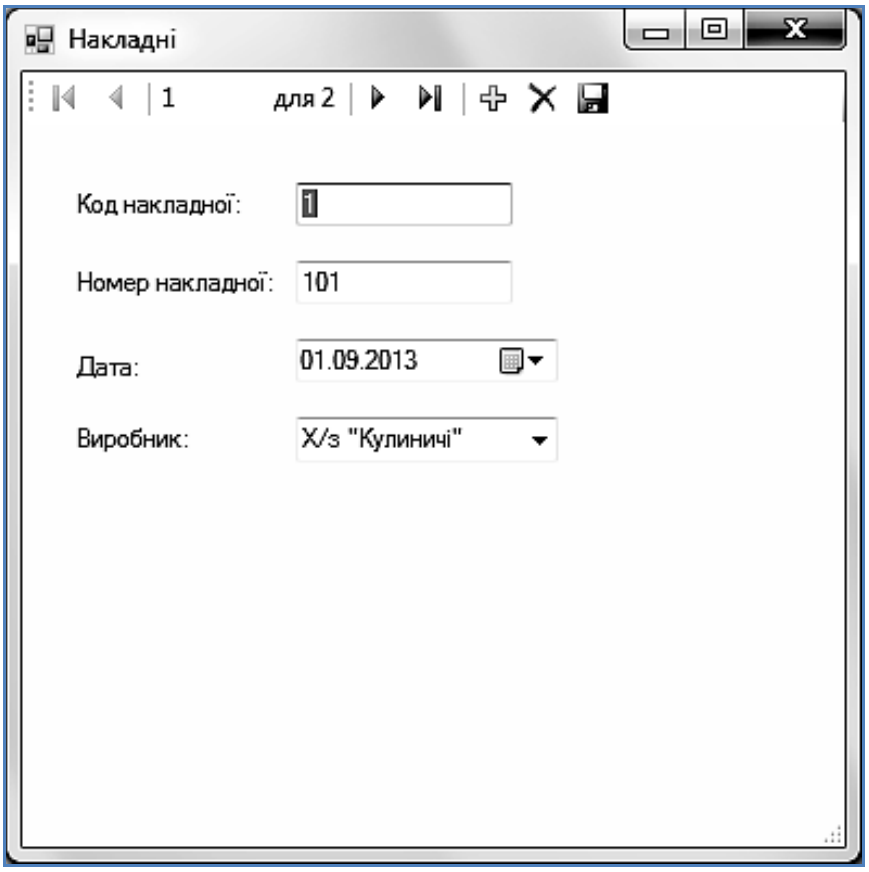


Рис. 7.63. **Додавання даних про накладну**

11. Закрийте форми ***Накладні*** та ***Хліб*** і збережіть зміни, що зроблені в проекті.

12. Перетягніть підвузол ***Invoice_Products***, що знаходиться у вузлі ***Invoice*** вікна **Источники данных**, у нижню частину форми ***Накладні***. Перевірте функціональність форми ***Накладні*** (рис. 7.64).

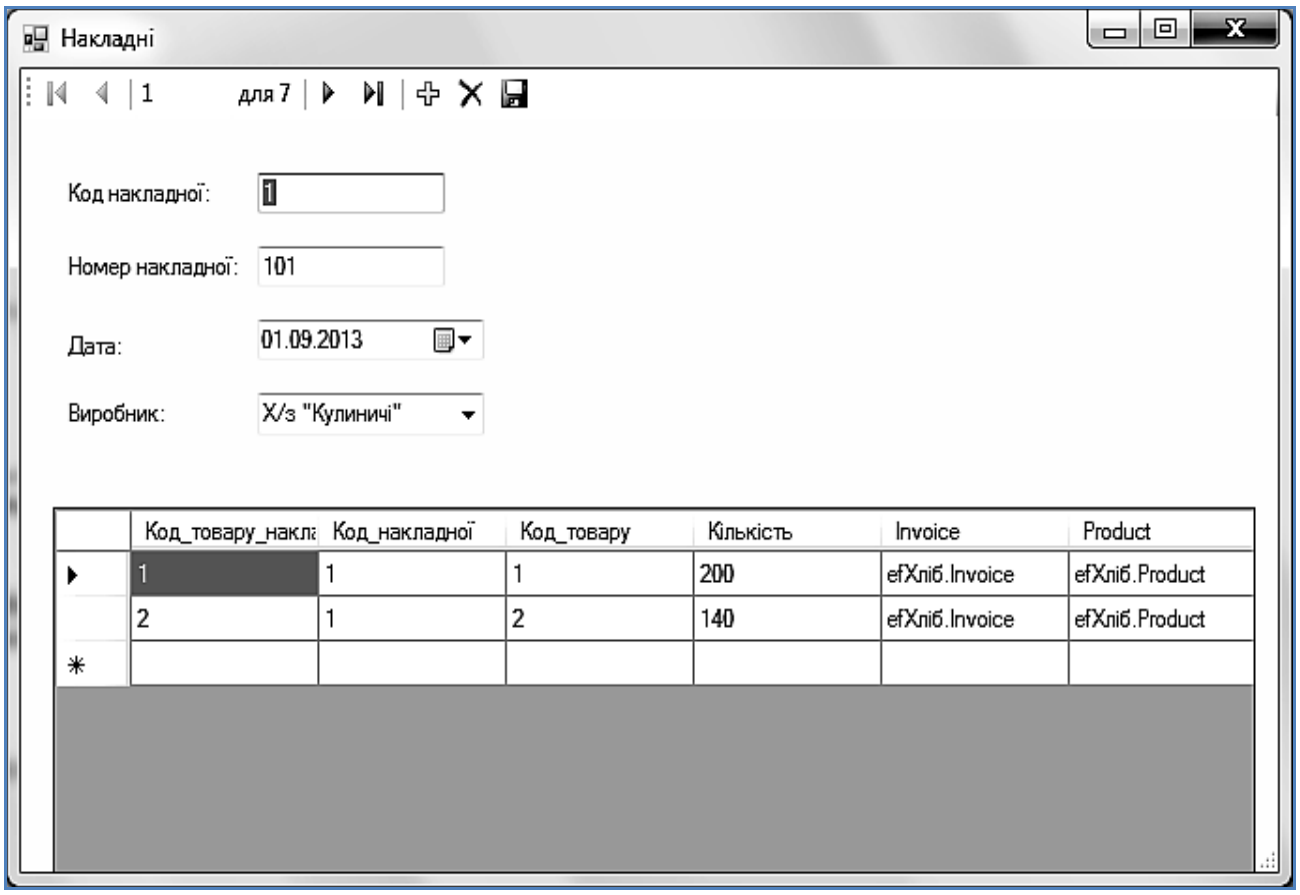


Рис. 7.64. **Форма *Накладні* з доданими товарами накладних**

13. Налаштуйте відображенння елемента ***invoice_ProductsData GridView***, зробивши невидимими стовпці ***Код_товару_накладної*** і ***Код_накладної*** та видаливши стовпці ***Invoice*** і ***Product***. Для цього можна використати властивість **Columns**.

14. Налаштуйте властивість ***Код_виробника*** у вигляді елемента ComboBox. Для цього:

14.1. Перетягніть на форму яке-небудь поле (наприклад, ***Товар***) з вузла ***Products*** вікна **Источники данных** у вільне місце форми й відразу вилучіть його. У нижній частині вікна конструктора залишиться елемент ***productBindingSource***.

14.2. Перейдіть у вікно **Правка столбцов** за допомогою властивості **Columns** для елемента ***invoice_ProductsDataGridView***.

14.3. Змініть тип стовпця ***Код_товару*** на **DataGridViewComboBoxColumn**, дайте йому ім'я ***код_товаруDataGridViewComboBoxColumn*** і заголовок ***Товар***. Тут також установіть значення властивостей у групі **Данные**:

Властивість	Значення
DataPropertyName	Код_товару
DataSource	productBindingSource
DisplayMember	Товар
ValueMember	Код_товару

Вікно **Правка столбцов** зі встановленими значеннями властивостей у групі **Данные** подано на рис. 7.65.

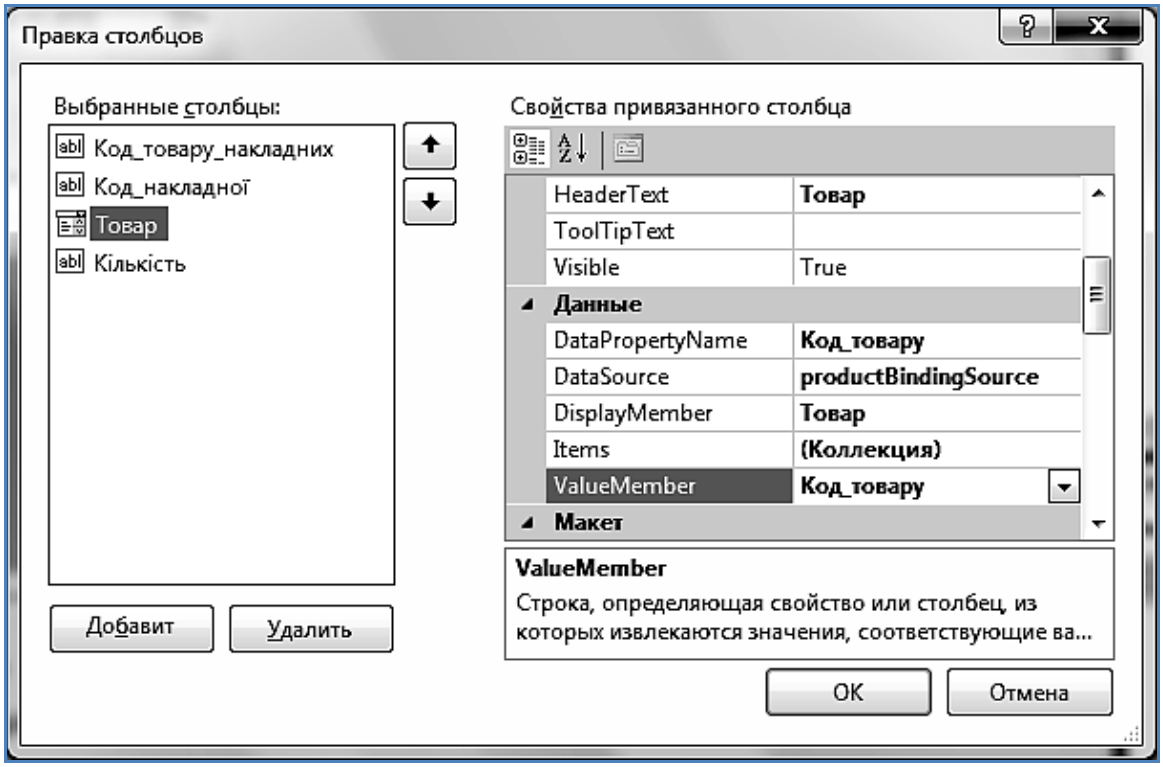


Рис. 7.65. **Встановлення значень властивостей у групі *Данные***

14.4. Перейдіть у вікно коду форми ***Накладні*** й додайте оператор для завантаження даних для ***productBindingSource*** в тілі оброблювача події ***formНакладні_Load***:

```
   // Завантажуємо дані для productBindingSource
   productBindingSource.DataSource = db.Products;
```

Після цього оброблювач набуде такого вигляду:

```
private void formПродажі_Load(object sender, EventArgs e)
{
    // Створюємо екземпляр класу ObjectContext
    db = new ХлібEntities();
```

```
    // Завантажуємо дані для manufacturerBindingSource
    manufacturerBindingSource.DataSource = db.Manufacturers;
    // Завантажуємо дані для productBindingSource
    productBindingSource.DataSource = db.Products;
    // Завантажуємо дані для invoiceBindingSource
    invoiceBindingSource.DataSource
    = db.Invoices.Execute(System.Data.Objects.MergeOption.AppendOnly);
   // Execute - для успішного збереження доданих накладних
}
```

14.5. Додайте оператор завершення редагування ***invoice_ProductsBindingSource*** в тілі оброблювача події ***invoiceBindingNavigatorSaveItem_Click***:

```
invoice_ProductsBindingSource.EndEdit();
```

Після цього оброблювач набуде такого вигляду:

```
private void invoiceBindingNavigatorSaveItem_Click(object sender, EventArgs e)
{
    invoice_ProductsBindingSource.EndEdit();
    invoiceBindingSource.EndEdit();
    db.SaveChanges();
}
```

14.6. Запустіть програму на виконання й перевірте функціональність форми ***Накладні***, додавши дані про надходження кількох товарів і зберігши їх (рис. 7.66).

15. Додайте стовпці ***Ціна*** і ***Вартість*** в елемент ***invoice_Products-DataGridView***. Для цього:

15.1. Додайте до проекту клас ***Invoice_Product.cs*** і введіть код:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace efХліб
{
    public partial class Invoice_Product
    {
        public decimal Ціна
        {
            get
```

```
            {
                if (this.Product != null)
                    return this.Product.Ціна;
                else
                    return 0;
            }
        }
        public decimal Вартість
        {
            get
            {
                if (this.Product != null)
                    return Ціна * Кількість;
                else
                    return 0;
            }
        }
    }
}
```

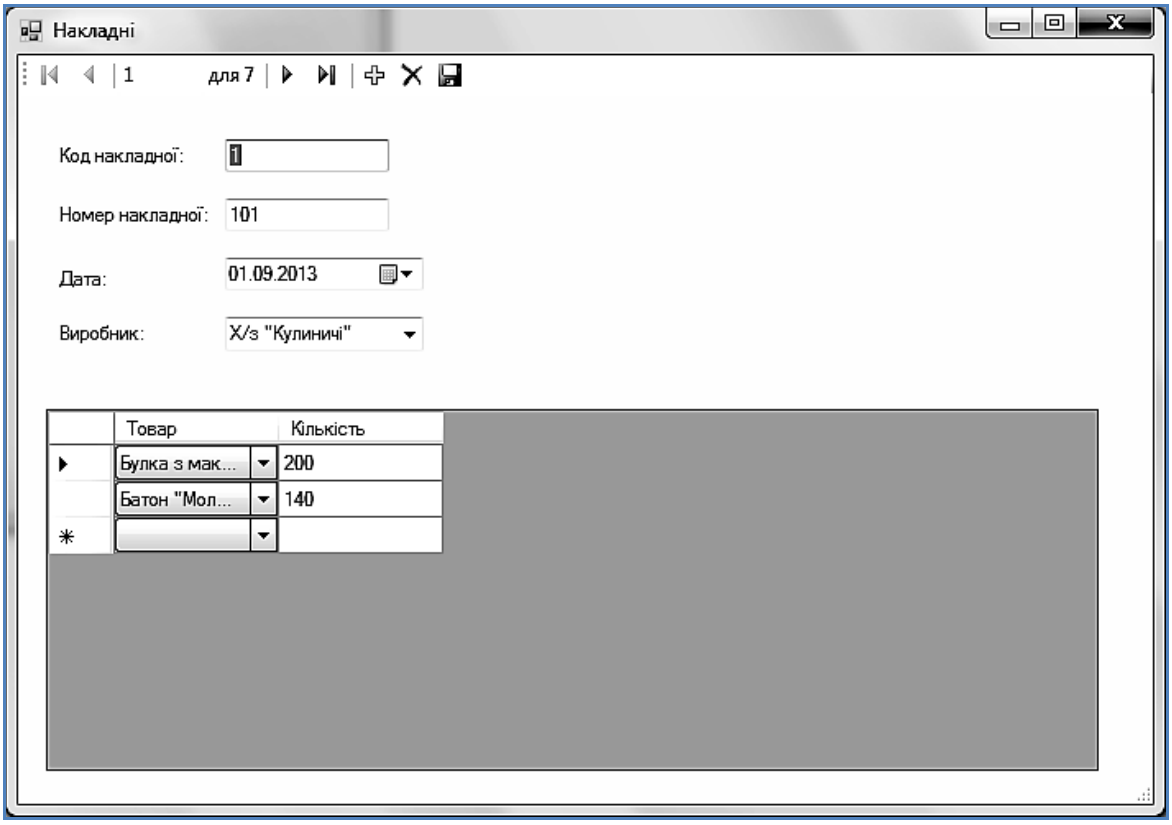


Рис. 7.66. **Форма *Накладні* з відображенням властивості *Код_товару* у вигляді ComboBox**

15.2. Перейдіть у вікно конструктора форми ***Накладні***, виділіть в ньому елемент ***invoice_ProductsDataGridView*** і за допомогою властивості **Columns** викличте вікно **Правка столбцов**.

15.3. Додайте два текстових стовпці ***Ціна*** і ***Вартість*** (рис. 7.67).

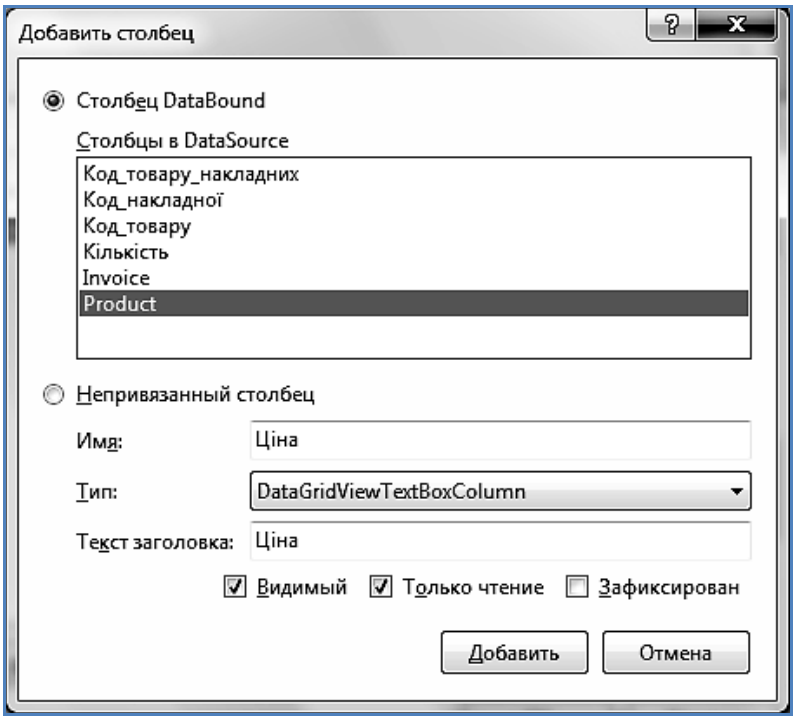


Рис. 7.67. **Додавання текстового стовпця *Ціна***

15.4. Перемістіть текстові стовпці ***Ціна*** і ***Вартість*** у кінець списку у вікні **Правка столбцов**. Тут також встановіть значення ***Ціна*** і ***Вартість*** для властивості **DataPropertyName** для кожного з доданих стовпців відповідно (рис. 7.68).

16. Щоб під час спроби додати дані ще про один товар в існуючу накладну не видавалося повідомлення про необхідність задати значення за замовчуванням для стовпця ***Товар***, задайте це значення так:

16.1. Виділіть елемент ***invoice_ProductsDataGridView*** й у вікні властивостей двічі клацніть подію **DefaultValuesNeeded**.

16.2. Додайте тіло коду оброблювача ***invoice_ProductsDataGridView_DefaultValuesNeeded***. Він має такий вигляд:

```
private void invoices_ProductsDataGridView_DefaultValuesNeeded(object sender,
          DataGridViewRowEventArgs e)
{
    //Значення за замовчанням для стовпця Товар
    e.Row.Cells["код_товаруDataGridViewComboBoxColumn"].Value =
```

```
            db.Products.Min(t => t.Код_товару);
}
```

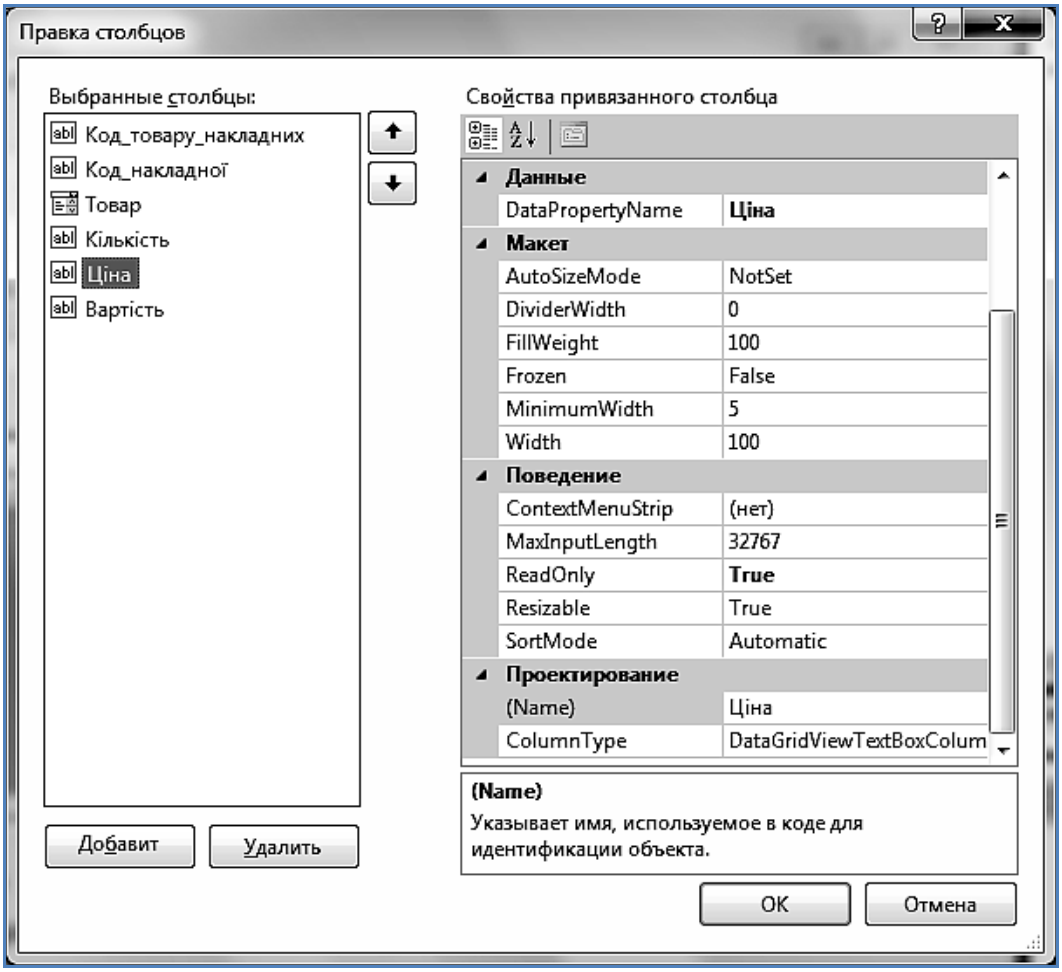


Рис. 7.68. **Встановлення значення *Ціна* для властивості DataPropertyName однойменного стовпця**

17. Запустіть програму на виконання й перевірте функціональність форми ***Накладні***, додавши дані про надходження кількох товарів і зберігши їх.
18. Закрийте форми ***Накладні*** й ***Хліб***.
19. Збережіть зміни, що зроблені в проекті.

7. Аналіз даних в Entity Framework

Завдання

Додати в застосування форму ***Продажі виробника***. У ній відображаються дані про результати продажів товарів вибраного виробника в табличній формі та у вигляді об'ємної кругової діаграми (рис. 7.69).

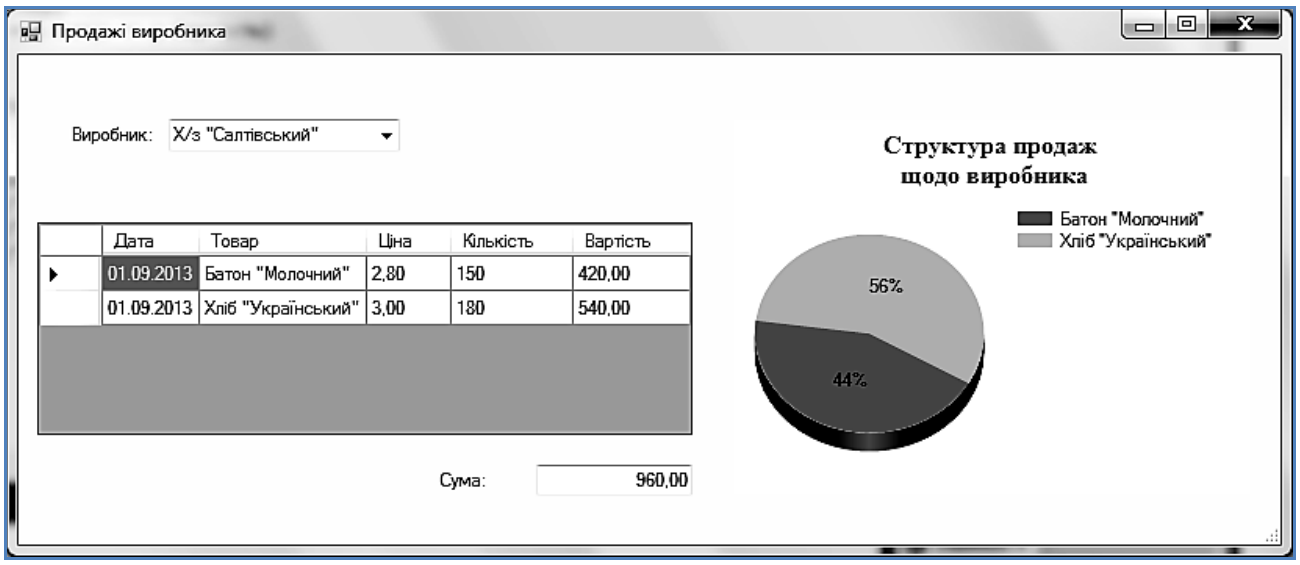


Рис. 7.69. **Форма *Продажі виробника***

Виконання

1. Додайте в застосування нову форму, давши їй ім'я ***formПродажіВиробника.***

2. Додайте на форму поле зі списком ***Виробник***. Для цього:

2.1. Виберіть у вікні **Источники данных** подання **ComboBox** для властивості ***Код_виробника*** у вузлі ***Manufacturer***.

2.2. Перетягніть властивість ***Код_виробника***з вузла ***Manufacturer*** у вікні ***Источники данных*** на форму ***Продажі виробника***.

На формі з'явився елемент керування ComboBox й панель навігатора, а в області компонентів – елемент ***manufacturerBindingSource***.

2.3. Видаліть панель навігатора з форми.

2.4. Змініть властивість **Text** для напису ***Код_виробника***, установивши нове значення ***Виробник***.

2.5. Встановіть такі значення властивостей для елемента ComboBox ***Виробник***:

Властивість	Значення
(DataBidings)-SelectedValue	(нет)
DataSource	manufacturerBindingSource
DisplayMember	Виробник
ValueMember	Код_виробника

2.6. Двічі клацніть у вільному місці форми ***Продажі виробника*** й у вікні коду форми уведіть оператори тіла оброблювача події ***formПродажіВиробника_Load***. Він має такий вигляд:

```
// Спільні об'єкти
ХлібEntities db;

private void formПродажіВиробника_Load(object sender, EventArgs e)
{
    db = new ХлібEntities();

    // Завантажуємо дані для manufacturerBindingSource
    manufacturerBindingSource.DataSource = db.Manufacturers;
}
```

2.7. Перейдіть у вікно конструктора форми ***Хліб*** і додайте код оброблювача події "Клацання кнопки Продажі виробника":

```
private void buttonПродажіВиробника_Click(object sender, EventArgs e)
{
 formПродажіВиробника вікноПродажіВиробника = new formПродажіВиробника();
 вікноПродажіВиробника.ShowDialog();
}
```

2.8. Запустіть програму на виконання й перевірте функціональність поля зі списком ***Виробник*** на формі ***Продажі виробника*** (рис. 7.70).

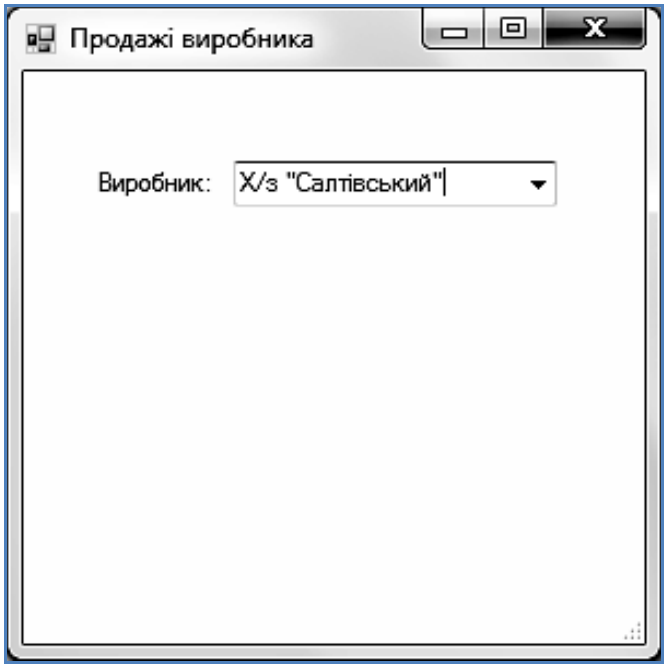


Рис. 7.70. **Форма *Продажі виробника* з полем зі списком *Виробник***

2.9. Закрийте форми ***Продажі виробника*** та ***Хліб*** і збережіть зміни, що зроблені в проекті.

3. Для відображення результатів продажів товарів вибраного виробника у табличній формі виконайте таке:

3.1. Додайте на форму елемент **DataGridView** з ім'ям ***gvПродажіВиробника***.

3.2. Двічі клацніть у вільному місці форми ***Продажі виробника*** й у вікні коду форми введіть у розділ спільних об'єктів класу такий оператор:

```
BindingSource bindingПродажіВиробника = new BindingSource();
```

а в тіло оброблювача події ***formПродажіВиробника_Load*** додайте такі оператори,

```
    // Завантажуємо дані для manufacturerBindingSource
    manufacturerBindingSource.DataSource = db.Manufacturers;

    // Запит для gvПродажіВиробника
    var queryПродажіВиробника =
        from продаж in db.Sales
        where продаж.Код_виробника ==
            (int)код_виробникаComboBox.SelectedValue
        select new
        {
            Дата = продаж.Дата,
            Товар = продаж.Product.Товар,
            Ціна = продаж.Product.Ціна,
            Кількість = продаж.Кількість,
            Вартість = продаж.Product.Ціна * продаж.Кількість
        };

    // Відображаємо дані
    bindingПродажіВиробника.DataSource = queryПродажіВиробника;
    gvПродажіВиробника.DataSource = bindingПродажіВиробника;
    gvПродажіВиробника.AutoResizeColumns();
```

Тепер ці дві частини коду мають такий вигляд:

```
// Спільні об'єкти
ХлібEntities db;
BindingSource bindingПродажіВиробника = new BindingSource();

private void formПродажіВиробника_Load(object sender, EventArgs e)
{
```

```
    db = new Хліб Entities();

    // Завантажуємо дані для manufacturerBindingSource
    manufacturerBindingSource.DataSource = db.Manufacturers;

    // Завантажуємо дані для manufacturerBindingSource
    manufacturerBindingSource.DataSource = db.Manufacturers;

    // Запит для gvПродажіВиробника
    var queryПродажіВиробника =
        from продаж in db.Sales
        where продаж.Код_виробника ==
            (int)код_виробникаComboBox.SelectedValue
        select new
        {
            Дата = продаж.Дата,
            Товар = продаж.Product.Товар,
            Ціна = продаж.Product.Ціна,
            Кількість = продаж.Кількість,
            Вартість = продаж.Product.Ціна * продаж.Кількість
        };

    // Відображаємо дані
    bindingПродажіВиробника.DataSource = queryПродажіВиробника;
    gvПродажіВиробника.DataSource = bindingПродажіВиробника;
    gvПродажіВиробника.AutoResizeColumns();
}
```

3.3. Запустіть програму на виконання й перевірте функціональність поля зі списком ***Виробник*** і елемента DataGridView на формі ***Продажі виробника***. В останньому відображаються результати продажів товарів виробника, що вказаний у полі зі списком (рис. 7.71). Але у разі зміни виробника, відповідні дані не з'являються. Цей недолік ліквідується у подальшому.

3.4. Закрийте форми ***Продажі виробника*** та ***Хліб*** і збережіть зміни, що зроблені в проекті.

4. Для відображення сумарної вартості продажів товарів вибраного виробника виконайте таке:

4.1. Додайте на форму елементи напис і текстове поле.

4.2. Для властивості **Text** напису встановіть значення ***Сума:***, а для властивості **Name** текстового поля – значення ***textBoxСума***.

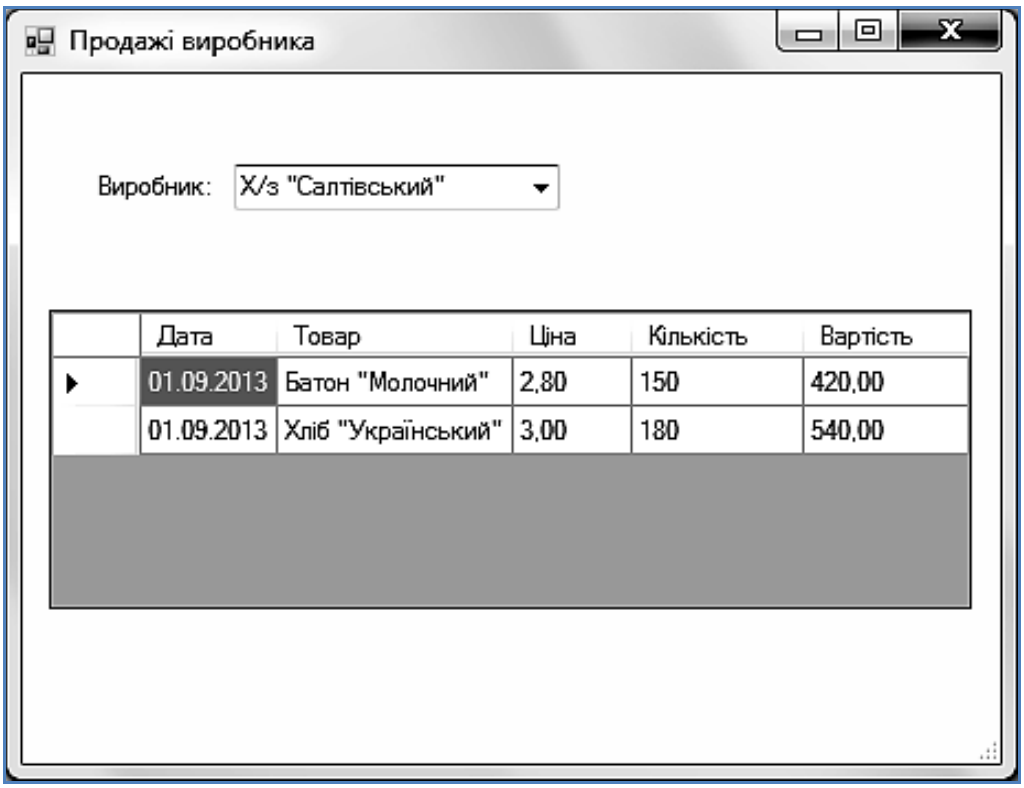


Рис. 7.71. **Форма *Продажі виробника* з результатами продажів товарів виробника**

4.3. Двічі клацніть у вільному місці форми ***Продажі виробника*** й у вікні коду форми додайте в кінець коду оброблювача події ***formПродажіВиробника*** такі оператори:

```
//Сума
decimal Сума = queryПродажіВиробника.Sum(p => p.Вартість);
textBoxСума.Text = Сума.ToString("0.00");
textBoxСума.TextAlign = HorizontalAlignment.Right;
```

4.4. Запустіть програму на виконання й перевірте функціональність текстового поля ***Сума*** на формі ***Продажі виробника***. В ньому відображається сумарна вартість продажів товарів виробника, що вказаний у полі зі списком (рис. 7.72).

4.5. Закрийте форми ***Продажі виробника*** та ***Хліб*** і збережіть зміни, що зроблені в проекті.

5. Для відображення структури продажів товарів вибраного виробника у вигляді об'ємної кругової діаграми виконайте таке:

5.1. Додайте на форму елемент **Chart** з ім'ям ***chartСтруктура***.

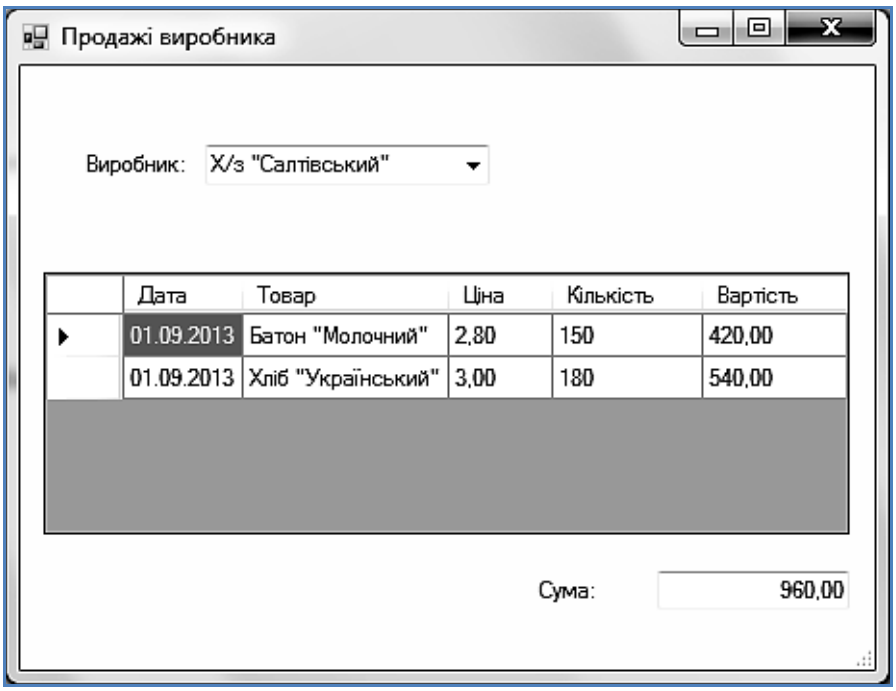


Рис. 7.72. **Форма *Продажі виробника* з полем *Сума***

5.2. Двічі клацніть у вільному місці форми ***Продажі виробника*** й у вікні коду форми додайте у розділ просторів імен такий оператор:

```
using System.Windows.Forms.DataVisualization.Charting;//Для діаграми
```

у розділ спільних об'єктів класу такий оператор:

```
BindingSource bindingСтруктура = new BindingSource();
```

а в кінець коду тіла оброблювача події ***formПродажіВиробника*** такі оператори:

```
    //***********
    // Діаграма *
    //***********

    //Запит
    var queryСтруктура = from товар in queryПродажіВиробника
    group товар by товар.Товар into t
    select new
    {
        Товар = t.Key,
        Вартість = t.Sum(p => p.Вартість)
    };
```

```
//Дані для відображення
bindingСтруктура.DataSource = queryСтруктура;
chartСтруктура.DataSource = bindingСтруктура;
chartСтруктура.Series["Series1"].XValueMember = "Товар";
chartСтруктура.Series["Series1"].YValueMembers = "Вартість";

//Тип диаграми
chartСтруктура.Series["Series1"].ChartType = SeriesChartType.Pie;

//Підписи на диаграмі
chartСтруктура.Series["Series1"].Label = "#PERCENT{P0}";

//Об'ємний варіант (3D)
this.chartСтруктура.ChartAreas[0].Area3DStyle.Enable3D = true;

//Заголовок
chartСтруктура.Titles.Add("Заголовок");
chartСтруктура.Titles[0].Text = "Структура продаж\n щодо виробника";
chartСтруктура.Titles[0].Font= new Font("Times New Roman", 12,
    FontStyle.Bold);
chartСтруктура.Titles[0].ForeColor = Color.Red;

//Легенда
chartСтруктура.Series["Series1"].IsVisibleInLegend = true;
chartСтруктура.Series["Series1"].LegendText = "#VALX";
```

5.3. Запустіть програму на виконання й перевірте функціональність діаграми на формі ***Продажі виробника***. В ній відображається структура продажів товарів вибраного виробника у вигляді об'ємної кругової діаграми (рис. 7.73).

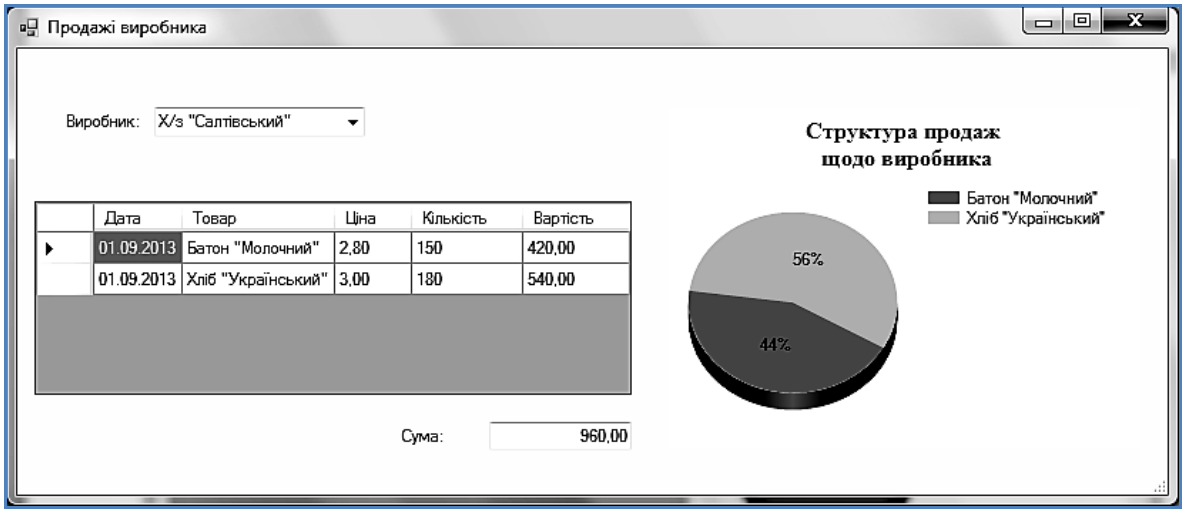


Рис. 7.73. **Форма *Продажі виробника* з діаграмою**

5.4. Закрийте форми ***Продажі виробника*** та ***Хліб*** і збережіть зміни, що зроблені в проекті.

6. Для відображення результатів продажів товарів вибраного виробника у разі його зміни виконайте таке:

6.1. Перейдіть у вікно конструктора форми ***Продажі виробника***.

6.2. Двічі клацніть на полі зі списком ***Виробник*** і у вікні коду форми введіть оператори тіла оброблювача події ***код_виробника ComboBox_SelectedIndexChanged***. Він має такий вигляд:

```
private void код_виробникаComboBox_SelectedIndexChanged(object sender,
        EventArgs e)
{
     if (this.CanFocus) //Після завантаження
     {
         var queryПродажіВиробника =
             from продаж in db.Sales
             where продаж.Код_виробника ==
                 (int)код_виробникаComboBox.SelectedValue
             select new
             {
               Дата = продаж.Дата,
               Товар = продаж.Product.Товар,
               Ціна = продаж.Product.Ціна,
               Кількість = продаж.Кількість,
               Вартість = продаж.Product.Ціна * продаж.Кількість
             };

         // Відображаємо дані
         bindingПродажіВиробника.DataSource = queryПродажіВиробника;
         //Сума
         decimal Сума = (decimal)queryПродажіВиробника.Sum(p =>
             p.Вартість);
         textBoxСума.Text = Сума.ToString("0.00");

         //***********
         // Діаграма *
         //***********

         //Запит
         var queryСтруктура =
             from товар in queryПродажіВиробника
             group товар by товар.Товар into t
             select new
             {
                 Товар = t.Key,
                 Вартість = t.Sum(p => p.Вартість)
```

```
            };
        //Дані для відображення
        bindingСтруктура.DataSource = queryСтруктура;
        if (queryСтруктура.Count() > 0)
        {
            chartСтруктура.DataSource = bindingСтруктура;
            chartСтруктура.Series["Series1"].XValueMember = "Товар";
            chartСтруктура.Series["Series1"].YValueMembers = "Вартість";
            chartСтруктура.Series["Series1"].ChartType =
                SeriesChartType.Pie;
        }
    }
}
```

6.3. Запустіть програму на виконання й перевірте функціональність поля зі списком ***Виробник***. У разі зміни виробника на формі відображаються результати продажів товарів виробника, що вказаний у полі зі списком (рис. 7.74).

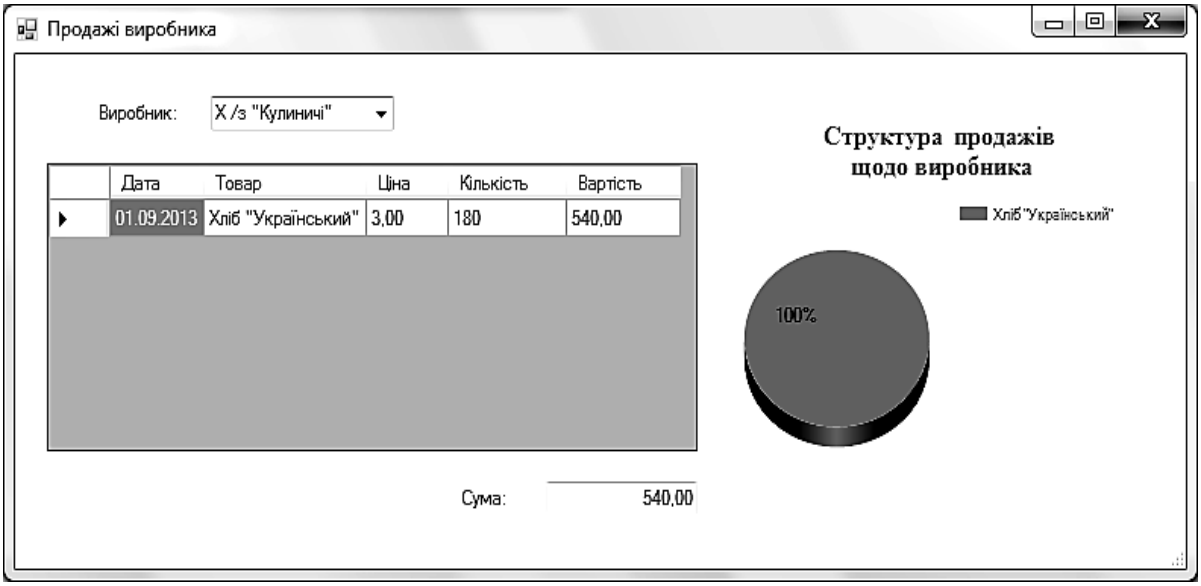


Рис. 7.74. **Форма *Продажі виробника* зі зміненим значенням виробника**

6.4. Закрийте форми ***Продажі виробника*** та ***Хліб*** і збережіть зміни, що зроблені в проекті.

Завдання для самостійного виконання

1. Побудувати концептуальні моделі шляхом модифікування моделі ***ХлібModel***. Моделі задовольняють такі вимоги:

1.1. Магазин фірмовий, тобто продає товари тільки свого виробника (хлібозаводу).

1.2. У базу даних записують дані про продажі, які надходить із декількох магазинів, що утворюють торгівельну мережу.

1.3. Здійснюється продаж кількох видів хліба ("Український", "Родзинка", "Бородинський"), батонів ("Молочний", "Слобожанський", "Нарізний") тощо, тобто товари розподіляються за групами, наприклад хліб, батони, булки.

1.4. У різних виробників закупівельна ціна товарів кожного виду різна.

1.5. У базі даних зберігають усі зміни цін на товари, указуючи дату, з якої починає діяти нова ціна.

1.6. У ході підбиття підсумків дня вказують, хто з продавців працював у цей день. Передбачається, що продавців кілька, але щодня працює тільки один. У даних про продавця зберігають його фотографію.

2. Вибрати одну з побудованих у п. 1 моделей і створити на її основі базу даних.

3. Змінити застосування ***efХліб*** відповідно до бази даних, що отримана у п. 2.

4. Додати в застосування ***efХліб*** такі форми для задач аналізу даних (лаб. робота № 6):

4.1. Форма ***Прайс***.

4.2. Форма ***Продано***.

5. Виконати п.п. 1 – 7 ходу роботи для індивідуальної бази даних.

Висновки

1. Платформа Entity Framework – це набір технологій ADO.NET, що забезпечують розробку застосувань, пов'язаних з обробкою даних на основі концепції об'єктно-реляційного відображення.

2. Метою створення платформи Entity Framework є об'єднання концепцій сховища даних, заснованого на реляційній моделі і розробки застосувань на основі об'єктно-орієнтованого програмування. Її реалізація дозволяє в коді оперувати з даними таблиць бази як з об'єктами програми без використання мови SQL.

3. Сутнісна модель даних EDM (Entity Data Model) складається із трьох компонентів – концептуальної моделі даних, моделі збереження (фізична модель) та опису відповідності між елементами кожної моделі.

4. Концептуальну модель записують засобами мови CSDL (Conceptual Schema Definition Language), модель збереження даних – засобами мови SSDL (Store Schema Definition Language), а відображення однієї схеми на іншу – засобами мови MSL (Mapping Schema Language). Усі три мови є діалектами мови XML.

5. У концептуальній моделі визначають сутності й зв'язку між ними як класи, які використовують у програмі незалежно від того, як дані зберігаються в базі.

6. Основними поняттями концептульної моделі даних є контекст об'єкта, набір сутностей, сутність, властивість і асоціація. Між основними поняттями концептульної і фізичної моделей даних є взаємно однозначна відповідність.

7. Застосування на платформі Entity Framework сторюють при наявності сутнісної моделі даних EDM і бази даних. Для побудови останніх застосовують один зі сценаріїв Model First, DB First чи Code First. Вибір сценарію залежить від того, що вже було створено перед цим та на скільки кретичною є ефективність роботи застосування.

8. Платформа Entity Framework підтримує увесь набір операцій CRUD. Для збереження Для збереження змін у базі даних викликають метод SaveChanges класу ObjectContext.

9. Технологія LINQ to Entities забезпечує підтримку LINQ у запитах до сутностей для реалізації операцій проекція, фільтрація, сортування, агрегування, секціонування, навігація за зв'язками, з'єднання тощо.

10. Відібрані в запиті дані можна відобразити в елементах керування на формі такими способами: вказати запит як джерело даних елемента керування, вказати запит як джерело даних з'єднувача, використати візуальні засоби на основі вікна **Источники данных**.

11. Технологія LINQ to Entities підтримує багаторівневе сортування. Воно полягає в тому, що спочатку виконують сортування за першим критерієм. Якщо є кілька сутностей, у яких однакове значення цього критерію, утворюються групи сутностей з тим самим його значенням. Після цього у кожній групі виконують сортування за другим критерієм і т. д.

12. Відібрані в запиті дані можна відобразити в елементах керування на формі такими способами: вказати запит як джерело даних елемента керування, вказати запит як джерело даних з'єднувача, використати візуальні засоби на основі вікна **Источники данных**.

8. Технологія Code First

8.1. Призначення технології Code First та її переваги

Під час побудови моделі EDM платформою Entity Framework в автоматичному режимі створюється код опису класів сутностей. Він має універсальний характер, щоб задовольняти будь-які потреби розробника, більшість з яких залишається без використання. За універсальність доводиться платити збільшенням обсягу коду, і, особливо, часом його виконання.

Наприклад, у ході використання технології DB First під час формування класу сутності Product для бази даних Хліб платформою Entity Framework генерується такий код опису властивості Товар:

```
[EdmScalarPropertyAttribute(EntityKeyProperty=false, IsNullable=false)]
[DataMemberAttribute()]
public global::System.String Товар
{
    get
    {
        return _Товар;
    }
    set
    {
        OnТоварChanging(value);
        ReportPropertyChanging("Товар");
        _Товар = StructuralObject.SetValidValue(value, false);
        ReportPropertyChanged("Товар");
        OnТоварChanged();
    }
}
private global::System.String _Товар;
partial void OnТоварChanging(global::System.String value);
partial void OnТоварChanged();
```

З іншого боку, та сама властивість у процесі використання технології Code First має набагато простіший вигляд:

```
public string Товар { get; set; }
```

Таке співвідношення між технологіями у поданні класів має місце для будь-якої властивості.

З наведеного прикладу можна зробити такий висновок. Технології Model First та DB First є простішими на етапі побудови, оскільки надають

візуальні засоби, які не вимагають безпосереднього кодування моделей, але вони вимагають значних обчислювальних ресурсів під час виконання застосування. Технологія Code First, що базується на простих класах POCO (Plain Old CLR Objects), навпаки, дає значно економніший код, але потребує "ручного" кодування. Тому технології Model First та DB First рекомендуються для початківців, а Code First – для професійного використання.

З метою отримання більш ефективного коду у технології Code First додано класи DbContext і DbSet з дещо меншою функціональністю, ніж відповідні їм класи ObjectContext і ObjectSet у технологіях Model First та DB First, але вимагають значно менше ресурсів.

Для використання технологій Model First та DB First платформою Entity Framework в автоматичному режимі створюється файл EDMX, в якому зберігаються концептульна і фізична моделі даних і їхнє взаємне відображення. У технології Code First такий файл не зберігається, але відповідні метадані формуються в пам'яті під час виконання застосування на основі описів класів.

За сценаріями використання ці технології відрізняються так. Технологію Model First застосовують, коли ще немає бази даних, технологію Model First, навпаки, коли вона вже існує, а технологію Code First – в обох випадках. Якщо бази даних ще немає, вона створюється автоматично на основі моделі під час виконання застосування. Якщо база даних вже існує, то створюють відповідні класи сутностей та доступу до даних і виконання операцій з цими об'єктами призводить до відповідних дій з даними у базі даних.

На відміну від технологій Model First та DB First у разі внесення змін у схему даних у технології Code First необов'язково заново створювати базу даних і повторно заповнювати даними. Для реалізації можливості плавно змінювати базу даних тут введено засоби її міграції.

Так само, як і в технологіях Model First та DB First, Code First можна використовувати візуальні засоби побудови інтерфейсу користувача на основі вікна DataSource, хоча вони тут мають деякі особливості.

Запитання і завдання

1. Яке призначення має технологія Code First?
2. У чому полягає різниця між технологією Code First та іншими технологіями платформи Entity Framework?

3. У яких випадках краще користуватися технологією Code First, а в яких – технологіями Model First та DB First? Наведіть приклади.

8.2. Класи сутностей. Об'єкти доступу до даних

Для опису предметної області в коді створюють класи сутностей. Кожній сутності відповідає окремий клас. Множина таких класів є концептуальною моделлю предметної області, а кожен клас – відповідно, моделлю сутності, або просто моделлю.

Зазвичай, кожен клас описують в окремому файлі, а всі такі файли поміщають в одну папку, яку найчастіше називають ***Models***.

Оскільки технологія Code First спрямована на максимальну ефективність коду, опис класу сутності (моделі) заснований на класах POCO (Plain Old CLR Objects), які містять тільки властивості. Окрім властивостей, що описують атрибути реального світу (назва товару, його ціна тощо) модель обов'язково має ключову властивість, значення якої однозначно визначає екземпляр сутності.

Якщо сутність пов'язана з іншою сутністю, то в описі моделі вказують властивість навігації. Причому для батьківської сутності зазначають колекцію дочірніх сутностей, а для дочірньої, навпаки, – один екземпляр батьківської сутності.

Приклад 8.1. Опис батьківської сутності ***Product***.

```
public class Product
     {
    public int ProductID { get; set; }
    public string Товар { get; set; }
    public Decimal Ціна { get; set; }
    public Decimal Ціна_закупівлі { get; set; }

    // Властивість навігації
public virtual ICollection<Sale> Sales { get; set; }
}
```

Приклад 8.2. Опис дочірньої сутності ***Sale***.

```
public class Sale
     {
    public int SaleID { get; set; }
    public DateTime Дата { get; set; }
    public int ProductID { get; set; }

    public int ManufacturerID { get; set; }
    public Int16 Кількість { get; set; }
```

```
    // Властивості навігації
    public virtual Product Product { get; set; }
    public virtual Manufacturer Manufacturer { get; set; }
}
```

Примітки. 1. Так само, як і в технології Model First, бажано давати імена сутностям англійською мовою. Тоді ім'я таблиці у базі даних, що автоматично створюється засобами Entity Framework, отримає те саме, що й сутність, але у формі множини (наприклад, Products, Sales). Хоча в Code First є засоби налаштування для роботи з будь-якими іменами таблиць.

2. Ключовій властивості бажано давати ім'я, яке складається з імені сутності (класу), за яким йдуть символи ***ID***. Це полегшить задавання первинних та зовнішніх ключів і створення зв'язків між таблицями бази даних. Хоча в Code First є засоби налаштування для роботи з будь-якими іменами ключових властивостей.

3. Модифікатор ***virtual*** у властивостях навігації вмикає функцію відкладеного завантаження LINQ, яка читає значення властивості з бази даних тільки при спробі її використання.

Опис класу сутності дає можливість працювати в коді з одним рядком таблиці. Сама ж таблиця є набором сутностей. Тому для звертання до таблиці потрібно описати колекцію сутностей. Для кожної таблиці вказують свою колекцію. Опис усіх колекцій сутностей поміщають у клас, що є похідним від класу System.Data.Entity.DbContext. У ньому вказують колекції сутностей, що відповідають тим таблицям бази даних, з якими будуть виконуватися операції в даному проекті. Це не обов'язково можуть бути всі таблиці бази даних. Тому цей клас називають контекстом, підкреслюючи той факт, що операції виконуватимуться із зазначеною частиною бази даних.

Клас контексту використовують для доступу до бази даних. Тому найчастіше його файл поміщають у папку з ім'ям ***DataAccess*** або ***DAL*** (від англ. *Data Access Layer* – шар доступу до даних).

Приклад 8.3. Опис контексту для виконання операцій з продажами, дані про які містяться у базі даних ***Хліб***.

```
public class BreadContext : DbContext
{
    public DbSet<Product> Products { get; set; }
```

```
    public DbSet<Manufacturer> Manufacturers { get; set; }
    public DbSet<Sale> Sales { get; set; }
}
```

У прикладі 8.3 зазначено тільки три колекції, хоча в базі даних може міститися набагато більше таблиць.

Опис класів сутностей і контексту дає можливість виконувати операції CRUD так само, як це описано у п. 7.4.

Запитання і завдання

1. Як описують класи сутностей в Code First? Наведіть приклади.
2. Якими засобами описують зв'язки між таблицями в Code First?
3. З якою метою використовують модифікатор ***virtual*** у властивостях навігації?
4. Яке призначення має клас контексту?
5. Запишіть класи сутностей і контексту для виконання операцій з надходженням товарів, дані про які містяться у таблицях ***Накладні*** та ***Товари_накладних*** бази даних ***Хліб***.

8.3. Бібліотеки Entity Framework

Для виконання операцій з базою даних у коді необхідно попередньо надати доступ до простору імен **System.Data.Entity**, що міститься у стандартній збірці, а також посилання на останній випуск бібліотеки **EntityFramework.dll**. Для роботи з базою даних SQL Server на платформі Entity Framework починаючи з версії 6 потрібно також підключити бібліотеку **EntityFramework.SqlServer.dll**. На ці бібліотеки роблять посилання в розділі **References** проекту, попередньо завантаживши останні версії обох бібліотек із сайта Microsoft.

Операції завантаження бібліотек, встановлення посилань на них та внесення даних у файл конфігурації проекту можна виконати за допомогою відповідного майстра, який називається **NuGet Package Manager** (скорочено **NuGet**). Цей майстер входить до складу Visual Studio, починаючи з версії 2012. Для Visual Studio 2010 його можна встановити, виконавши такі команди:

1. Відкрити Visual Studio 2010 від імені адміністратора.
2. Виконати команду **Tools – Extension Manager**. З'явиться вікно **Extension Manager** (рис. 8.1).

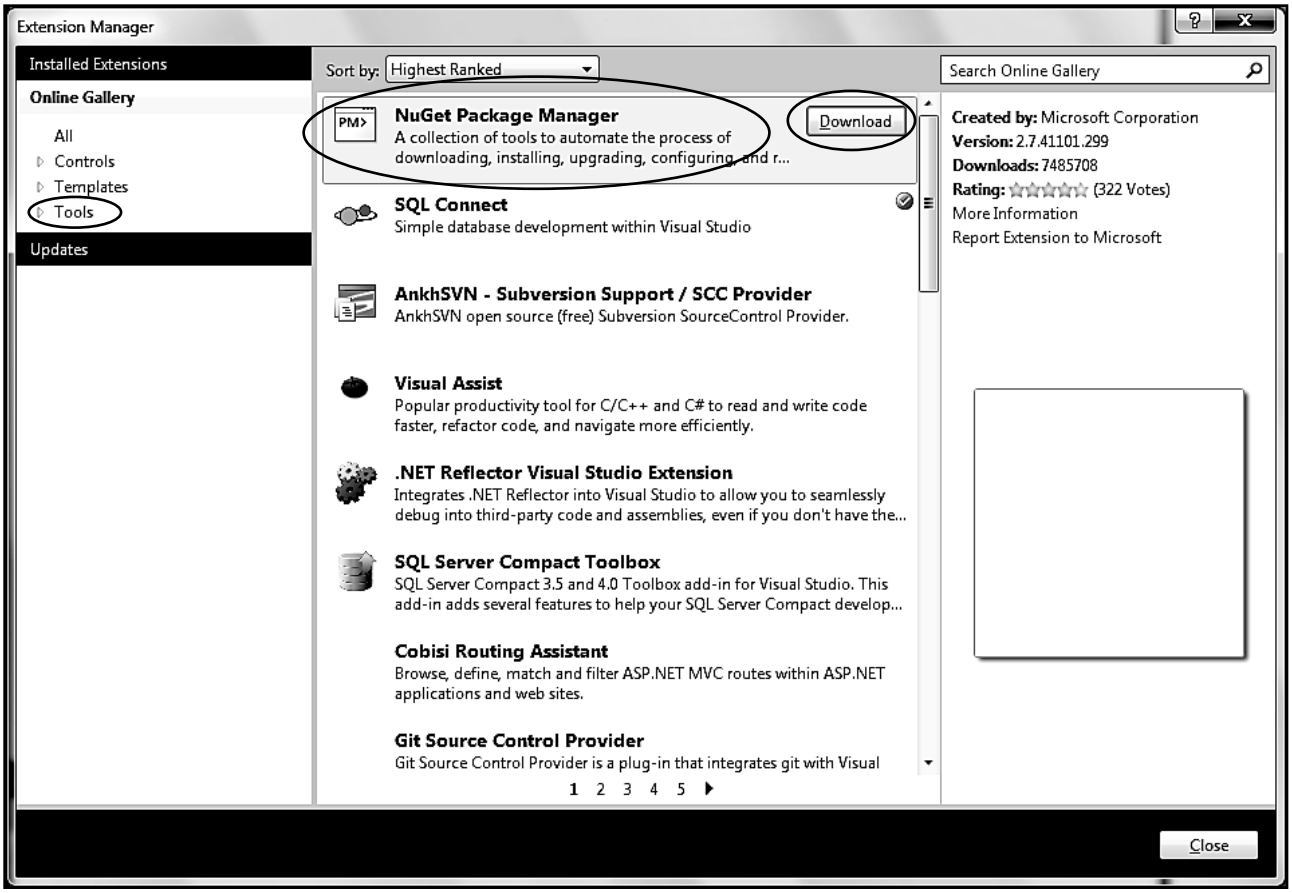


Рис. 8.1. **Вікно *Extension Manager***

2. Вибрати елемент **Online Gallery – Tools** у лівому списку.

3. Вибрати елемент **NuGet Package Manager** у центральному списку і клацнути кнопку **Download**.

4. Дочекатися завершення процесів завантаження і встановлення майстра.

5. Щоб можна було скористатися встановленим майстром NuGet, потрібно перезавантажити VisualStudio.

Щоб завантажити в проект бібліотеки **EntityFramework.dll** та **EntityFramework.SqlServer.dll** за допомогою майстра NuGet, потрібно виконати таке:

1. Виконати команду **Project – Manage NuGet Packages**. З'явиться вікно ManageNuGetPackages.

2. Вибрати елемент **Online** в лівому списку.

3. Вибрати елемент **Entity Framework** у центральному списку і клацнути кнопку **Install** (рис. 8.2).

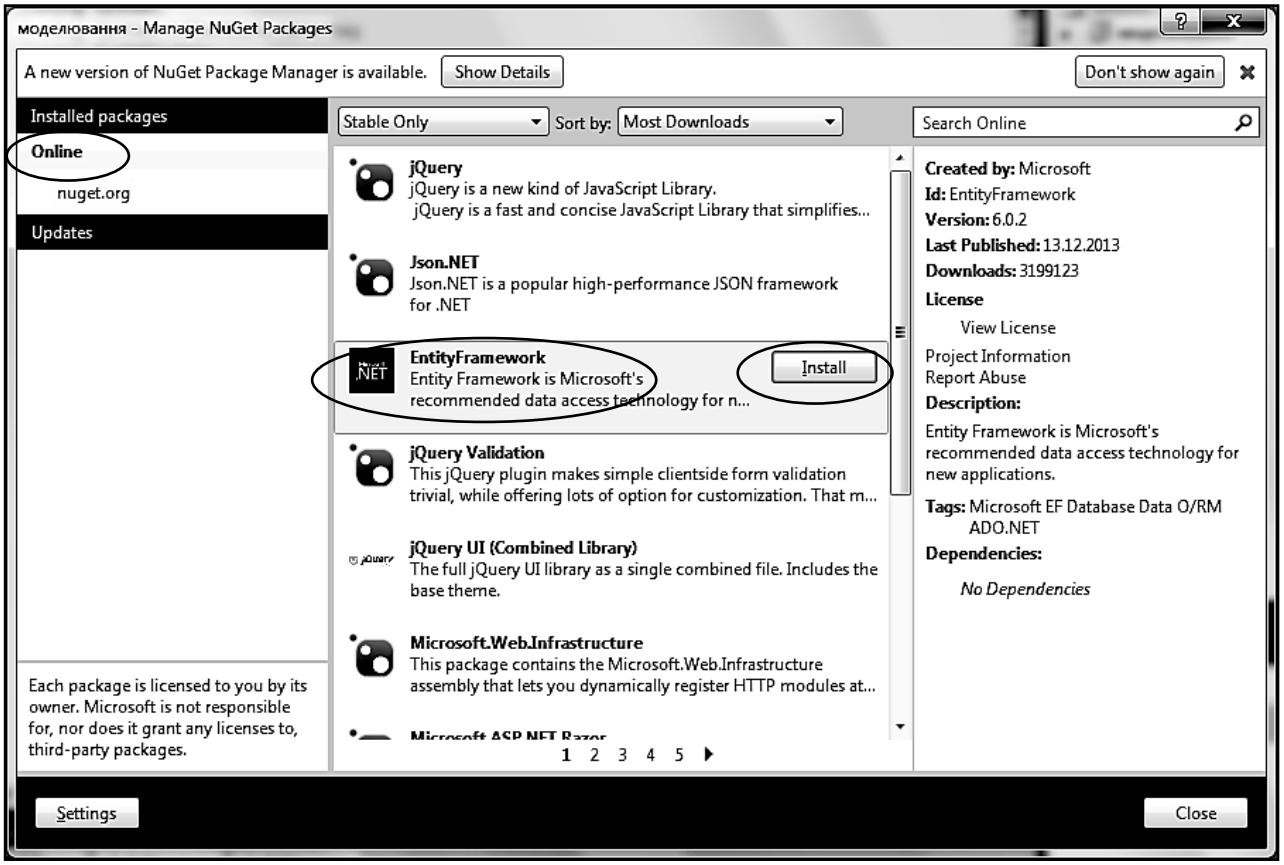


Рис. 8.2. **Додавання останньої версії пакета EntityFramework**

4. Дочекатися, поки завершиться встановлення пакета Entity Framework, клацнувши кнопку **I Accept** у кінці.

У папці рішення поряд із папкою проекту з'явиться папка ***packages***. У ній містяться два файли **EntityFramework.dll** та **EntityFramework.SqlServer.dll**.

Примітки. 1. Шар доступу до даних найчастіше виділяють в окремий проект. Тому під час створення застосування рішенню дають ім'я застосування, а проекту – пов'язане з моделлю даних, наприклад, ***моделювання***. Причому останній створюють як консольне застосування.

2. Для кожного застосування завантажують свій елемент **EntityFramework**. Як правило, це остання версія, що міститься на сайті Microsoft.

Якщо рішення складається з кількох проектів, можлива ситуація, коли в різних проектах містяться різні версії бібліотеки. Вони можуть конфліктувати між собою. Наприклад, у результаті завантаження елемента **EntityFramework**, у версії 6 отримують два файли **EntityFra-**

mework.dll та **EntityFramework.SqlServer.dll**, а у версії 5 – тільки перший. Тому за допомогою майстра NuGet завантажують елемент **Entity Framework** тільки у перший проект, а в решті проектів роблять посилання на результати першого завантаження. Щоб дізнатися адресу для посилань, використовують властивість **Path** у папці **References** першого проекту.

Запитання і завдання

1. Які бібліотеки потрібно підключити до проекту, що використовує технологію Code First?

2 Опишіть призначення майстра NuGet.

3. Опишіть алгоритм завантаження в проект бібліотеки EntityFramework.dll та EntityFramework.SqlServer.dll за допомогою майстра NuGet.

8.4. Створення бази даних

Якщо база даних ще не існує, вона створюється автоматично на сервері SQLEXPRESS (для VisualStudio 2010) або localDB (для Visual Studio 2012 та наступних версій). При цьому їй дається ім'я, яке складається з трьох частин – імені проекту, імені папки, в якій міститься клас контексту та імені самого контексту, наприклад

моделювання.DataAccess.BreadContext

Це ім'я вважається іменем за замовчуванням, тобто, якщо не задано ConnectionString, то для створення і подальшої роботи з базою даних використовується саме це ім'я.

У разі, коли потрібно створити базу даних з якимсь певним ім'ям або підключитися до вже існуючої бази даних, використовують рядок під'єднання у файлі конфігурації. Причому він повинен мати ім'я класу контексту.

Приклад 8.4. Задавання рядка під'єднання у файлі ***App.config*** для бази даних ***Хліб***, що розташована на сервері SQLEXPRESS (в коді клас контексту має ім'я ***BreadContext***).

```
<connectionStrings>
<add name="BreadContext" connectionString="Data Source=.\sqlexpress;
Initial Catalog=Хліб; Integrated Security=True;"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

Щоб під час першого виконання застосування база даних створилася, необхідно виконати будь-які операції з її даними. Наприклад, достатньо додати дані в одну таблицю, щоб створилася база даних з усіма таблицями.

Приклад 8.5. Створення бази даних ***Хліб*** під час заповнення даними таблиці ***Products***.

```
// Стратегія роботи з базою даних
Database.SetInitializer(
                    new DropCreateDatabaseAlways<BreadContext>());

var products = new List<Product>
{
    new Product { Товар = @"Хліб ""Український""", Ціна = 3.00M,
        Ціна_закупівлі =2.50M},
    new Product { Товар = @"Батон ""Молочний""", Ціна = 2.80M,
        Ціна_закупівлі =2.50M},
    new Product { Товар = @"Булка з маком", Ціна = 1.98M,
        Ціна_закупівлі =1.85M}
};
using (var context = new BreadContext())
{
    products.ForEach(p => context.Products.Add(p));
    context.SaveChanges();
}
```

Варто звернути увагу на початок прикладу 8.5. Там вказано стратегію роботи в даному застосуванні, а саме метод **DropCreateDatabaseAlways**, який завжди спочатку видаляє базу даних, а потім створює її заново. Такої стратегії слід дотримуватися на початку розроблення застосування, коли модель даних постійно змінюється. Ця стратегія також позбавляє проблеми повторення даних, якщо зазначений у прикладі 8.5 фрагмент коду повторюється кожний раз під час запуску програми на виконання.

Окрім стратегії **DropCreateDatabaseAlways** в Code First передбачені ще й такі:

DropCreateDatabaseIfModelChanges<TContext> – забезпечує автоматичне видалення бази даних у разі будь-яких змін хоча б в одному класі моделі;

CreateDatabaseIfNotExists<TContext> – створює базу даних і заповнює її значеннями тільки в тому разі, коли вона відсутня.

Тут **TContext** – тип контексту бази даних.

Запитання і завдання

1. Яке ім'я за замовчуванням отримує нова база даних?
2. Які засоби використовують для задавання імені базі даних з або підключення до вже існуючої бази даних?
3. Які стратегії роботи з базою даних у застосуванні вам відомі? Опишіть їхнє призначення.

8.5. Домовленості в Code First

Для створення бази даних у технології Code First використовується низка домовленостей за замовчуванням. Вони визначають правила створення таблиць і зв'язків між ними. Якщо ж створена за цими правилами база даних не повністю задовольняє потреби розробника, в Code First є можливості налаштувати процес її генерування так, щоб властивості таблиць і зв'язків між ними мали потрібні значення. В даному розділі буде розглянуто домовленості, при виконанні яких, властивості бази даних матимуть відповідні значення, а в наступному – як змінити ці значення.

Ім'я бази даних. Якщо серед рядків з'єднань немає рядка з іменем, що співпадає з іменем контексту, платформа Entity Framework створює базу даних, яка має ім'я, що складається з трьох частин – імені проекту, імені папки, в якій міститься клас контексту, та імені самого контексту і розміщується на сервері SQLEXPRESS (для Visual Studio 2010) або localDB (для Visual Studio наступних версій), наприклад

моделювання.DataAccess.BreadContext

Ім'я таблиці. Таблиці дається ім'я, що співпадає з іменем класу сутності. Причому якщо клас сутності складається з одного слова ім'я англійською мовою, то таблиця отримує теж саме ім'я, але в множині, в іншому разі – імена таблиці та класу сутності співпадають. Наприклад, для сутності ***Manufacturer*** створюється таблиця ***Manufacturers***, а для сутності ***Manufacturer_1*** – таблиця ***Manufacturer_1***.

Якщо ім'я класу сутності складається більш ніж з одного слова англійською мовою, щоб таблиця отримала ім'я у формі множини, потрібно всі слова писати разом і починати останнє слово з великої літери. На-

приклад, для сутності ***InvoiceProduct*** створюється таблиця ***Invoice-Products***, а для сутності ***Invoice_Product*** – таблиця ***Invoice_Product***.

Кількість таблиць. У базі даних створюються тільки ті таблиці, для яких є класи сутностей і ці класи вживаються як елементи колекцій у класі DbContext. Наприклад, якщо описати п'ять класів ***Product***, ***Manufacturer***, ***Sale***, ***Invoice*** та ***InvoiceProduct***, а в класі ***BreadContext*** описати тільки властивості ***Products***, ***Manufacturers*** та ***Sales***, то в базі даних буде створено тільки три таблиці ***Products***, ***Manufacturers*** та ***Sales***.

Первинний ключ. Якщо в класі сутності є властивість з ім'ям ***ID*** або ***ім'ясутності*** + ***ID***, така властивість відображається як ключ таблиці. Наприклад, якщо сутність ***Manufacturer***має властивість ***ManufacturerID***, то в таблиці ***Manufacturers*** поле ***ManufacturerID*** позначається як первинний ключ.

Зовнішній ключ. Найпростіше створити зовнішній ключ для зв'язку з іншою таблицею, якщо до складу сутності ввести властивість, яка має ім'я первинного ключа зв'язаної сутності. Наприклад, властивість ***ManufacturerID***, що входить до складу властивостей сутності ***Sale***, відображається як зовнішній ключ таблиці ***Sales*** для зв'язку з таблицею ***Manufacturers***.

Кратність відношення. Для відношення між таблицями встановлюється певна кратність залежно від властивостей навігації, що вказані у відповідних класах сутностей за правилами, що подані у табл. 8.1.

Примітка. Для встановлення відношення один-до-одного потрібно додатково додати інформацію про те, яка з сутностей є головною, а яка залежною за допомогою анотацій даних.

Якщо в сутності задано властивість, яка визначає зовнішній ключ, і вона може мати значення NULL, то відношення між таблицями має тип один-або нуль-до-багатьох чи один-або нуль-до-одного.

Каскадне видалення. Якщо зовнішній ключ дочірньої сутності забороняє значення NULL, Code First встановлює каскадне видалення для зв'язку і навпаки. У разі відсутності каскадного видалення, під час видалення батьківського об'єкта зовнішньому ключу надається значення NULL.

Типи даних стовпців. Під час створення таблиці її стовпцю встановлюється тип даних відповідно до типу даних властивості сутності. У табл. 8.2 подано відповідність між типом даних властивості і типом даних стовпця в базі даних SQL Sever.

Таблиця 8.1.

Правила кратності відношень

Перша сутність	Друга сутність	Кратність відношення між таблицями
Має властивість навігації як посилання на об'єкт другої сутності	Має властивість навігації як посилання на колекцію об'єктів першої сутності	Один-до-багатьох
Має властивість навігації як посилання на об'єкт другої сутності	Не має властивості навігації як посилання на об'єкти першої сутності	Один-до-багатьох
Не має властивості навігації як посилання на об'єкти другої сутності	Має властивість навігації як посилання на колекцію об'єктів першої сутності	Один-до-багатьох
Має властивість навігації як посилання на колекцію об'єктів другої сутності	Має властивість навігації як посилання на колекцію об'єктів першої сутності	Багато-до-багатьох
Має властивість навігації як посилання на об'єкт другої сутності	Має властивість навігації як посилання на об'єкт першої сутності	Один-до-одного

Таблиця 8.2

Відповідності між типами даних властивості і стовпця

Тип даних властивості	Тип даних стовпця в SQL Sever
System.Boolean	BIT
System.Byte	TINYINT
System.Int16	SMALLINT
System.Int32	INT
System.Int64	BIGINT
System.SByte	SMALLINT
System.UInt16	INT
System.UInt32	BIGINT
System.UInt64	DECIMAL(20)
System.Decimal	DECIMAL(29,4)
System.Single	REAL
System.Double	FLOAT
System.Char	NCHAR(1)
System.String	NVARCHAR(4000)
System.Char []	NVARCHAR(4000)
System.DateTime	DATETIME
System.DateTimeOffset	DATETIMEOFFSET
System.TimeSpan	TIME

Запитання і завдання

1. Яке ім'я за замовчуванням дається таблиці?
2. Як враховуються описи колекцій у класі DbContext під час створення таблиць? Наведіть приклади.
3. За якими правилами в Code First визначається кратність відношень між таблицями?
4. Як в Code First визначають первинний і зовнішній ключі?

8.6. Налаштування моделі даних засобами Data Annotations та Fluent API

Якщо розробника застосування не влаштовують значення властивостей бази даних, що отримані за замовчуванням, можна використати засоби Data Annotations або Fluent API для налаштування взаємного відображення між моделлю та базою даних. Це набуває особливо важливого значення у тому разі, коли база даних вже існує. В ній таблиці, первинні та зовнішні ключі мають певні імена, які не співпадають з тими, що визначаються домовленостями Code First. Далі слід розглянути засоби Data Annotations, які дозволяють налаштувати відображення класів сутностей на об'єкти бази даних, а потім – аналогічні засоби Fluent API. Такі налаштування бази даних називають *конфігуруванням* бази даних.

Примітки 1. Щоб використовувати засоби Data Annotations потрібно в класі сутності додати до описів просторів імен ще й такий

```
using System.ComponentModel.DataAnnotations;
```

2. Щоб змінити властивості якогось елемента в описі класу, перед ним поміщають відповідну анотацію, яку беруть в квадратні дужки.

Ім'я таблиці. Якщо потрібно, щоб ім'я таблиці не співпадало з іменем класу сутності, використовують анотацію

[Table("Ім'я таблиці")]

Її поміщають перед описом класу сутності.

Приклад 8.6. Відображення класу сутності ***Manufacturer*** на таблицю ***Виробники***.

```
[Table("Виробники")]
public class Manufacturer
{
```

```
    public int ManufacturerID { get; set; }
    public string Виробник { get; set; }
    public string Адреса { get; set; }
    public string Телефон { get; set; }
    public string Контактна_особа { get; set; }

    // Властивості навігації
    public virtual ICollection<Sale> Sales { get; set; }
}
```

Примітки. 1. Якщо бази даних не існувало, то створиться база даних з таблицею ***Виробники*** (рис. 8.3).

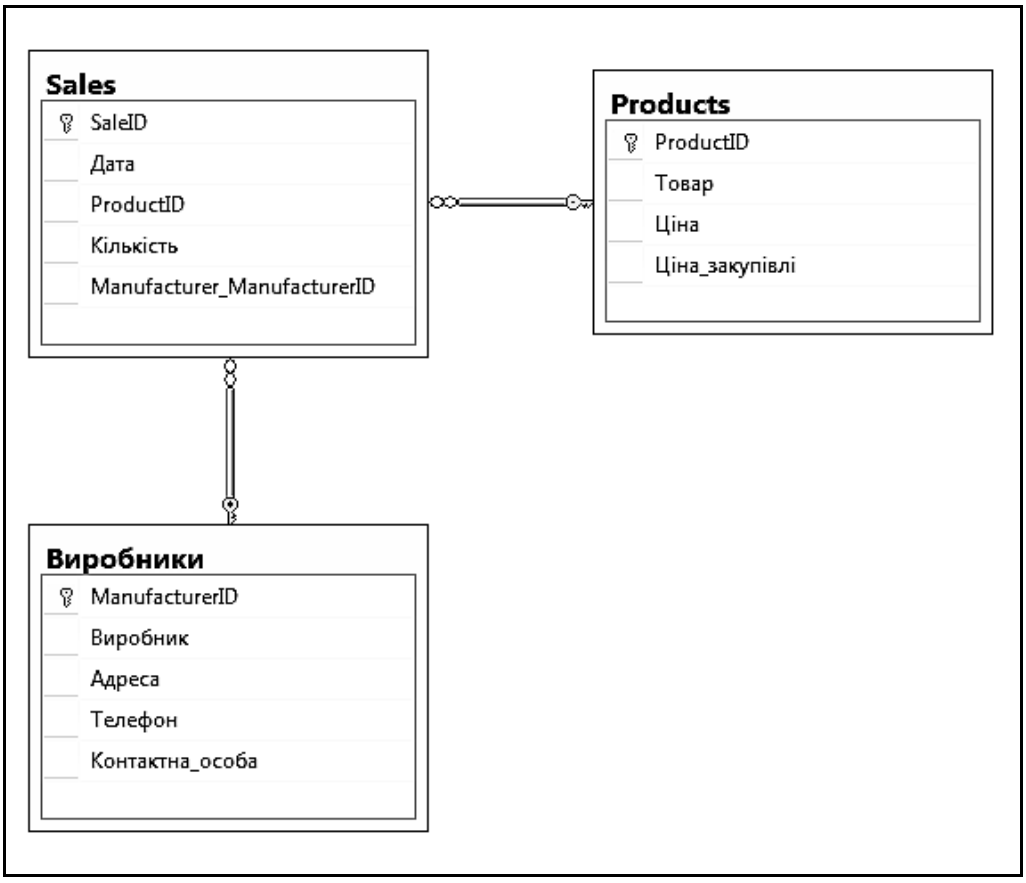


Рис. 8.3. **Схема бази даних після задавання імені таблиці *Виробники***

2. Операції із сутністю ***Manufacturer*** у коді відповідають операціям з таблицею ***Виробники*** в базі даних.

3. Для використання анотації [Table("*Ім'я таблиці*")] потрібно в класі сутності додати до описів просторів імен ще й такий

```
using System.ComponentModel.DataAnnotations.Schema;
```

Первинний ключ. Якщо ім'я ключового поля в базі даних не відповідає домовленостям Code First, то в описі класу вказують ім'я властиво-

сті, що співпадає з іменем ключового поля в базі даних, перед яким записують анотацію

[Key]

Приклад 8.7. Задавання атрибута *первинний ключ* для поля ***Код_виробника*** у таблиці ***Manufacturers***.

```
public class Manufacturer
{
    [Key]
    public int Код_виробника{ get; set; }
    public string Виробник { get; set; }
    public string Адреса { get; set; }
    public string Телефон { get; set; }
    public string Контактна_особа { get; set; }

    // Властивості навігації
    public virtual ICollection<Sale> Sales { get; set; }
}
```

Примітки. 1. Якщо бази даних не існувало, то створиться база даних із таблицею ***Manufacturers***, в якій поле ***Код_виробника*** позначається як первинний ключ.

2. У коді для сутності ***Manufacturer*** властивість ***Код_виробника*** є ключовою.

3. У таблиці ***Sales*** замість зовнішнього ключа ***ManufacturerID*** завдяки властивостям навігації створився зовнішній ключ ***Manufacturer_Код_виробника*** (рис. 8.4).

Зовнішній ключ. Якщо ім'я поля зовнішнього ключа в базі даних не відповідає домовленостям Code First, то в описі дочірньої сутності вказують ім'я властивості, що співпадає з іменем зовнішнього ключа в базі даних, перед яким записують анотацію

[ForeignKey("Ім'я батьківської сутності")]

Приклад 8.8. Задавання атрибута *зовнішній ключ* для поля ***Код_виробника*** у дочірній таблиці ***Sales*** для встановлення зв'язку з батьківською таблицею ***Manufacturers***.

```
public class Sale
{
    public int SaleID { get; set; }
    public DateTime Дата { get; set; }
    public int ProductID { get; set; }
```

```
    [ForeignKey("Manufacturer")]
    public int Код_виробника { get; set; }
    public Int16 Кількість { get; set; }
    // Властивості навігації
    public virtual Product Product { get; set; }
    public virtual Manufacturer Manufacturer { get; set; }
}
```

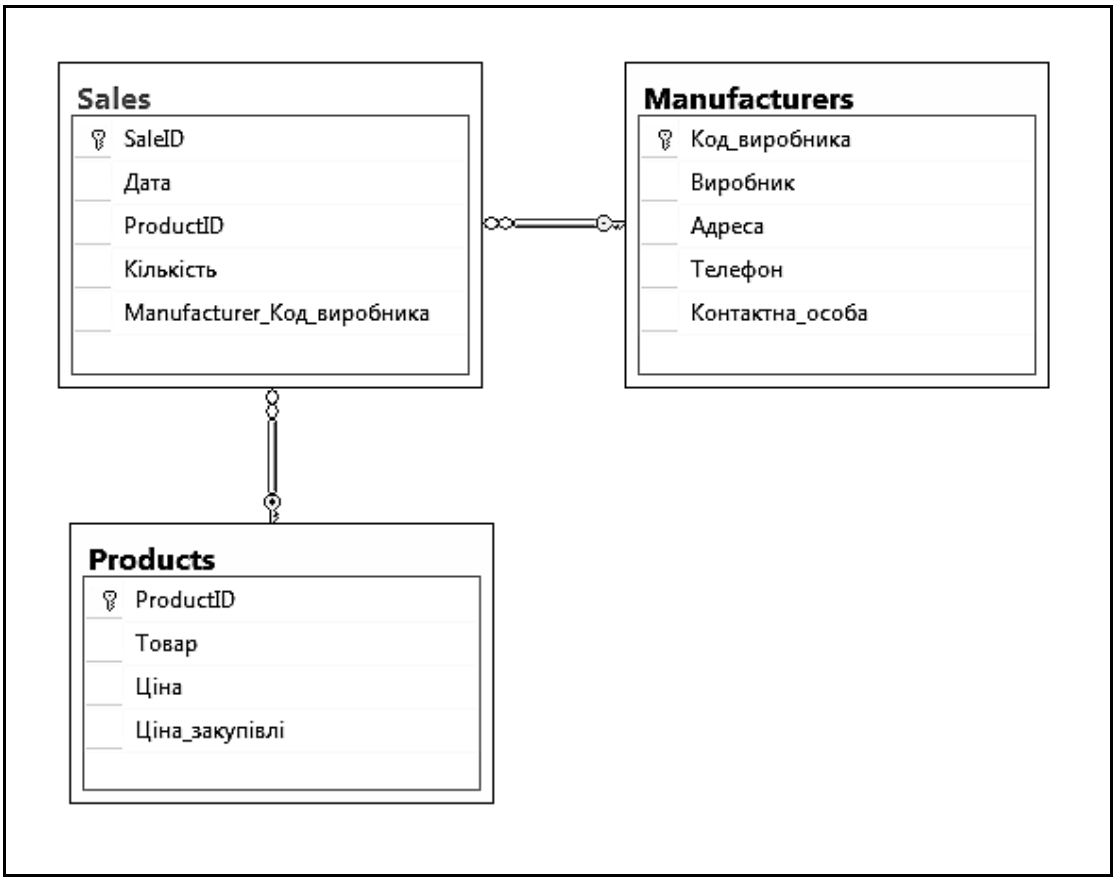


Рис. 8.4. **Схема бази даних після задавання атрибута *первинний ключ* для поля *Код_виробника* у таблиці *Manufacturers***

Примітки. 1. Для використання анотації [ForeignKey("*Ім'я батьківської сутності*")] потрібно в класі сутності додати до описів просторів імен ще й такий

```
using System.ComponentModel.DataAnnotations.Schema;
```

2. Якщо бази даних не існувало, то створиться база даних, схему якої подано на рис. 8.5.

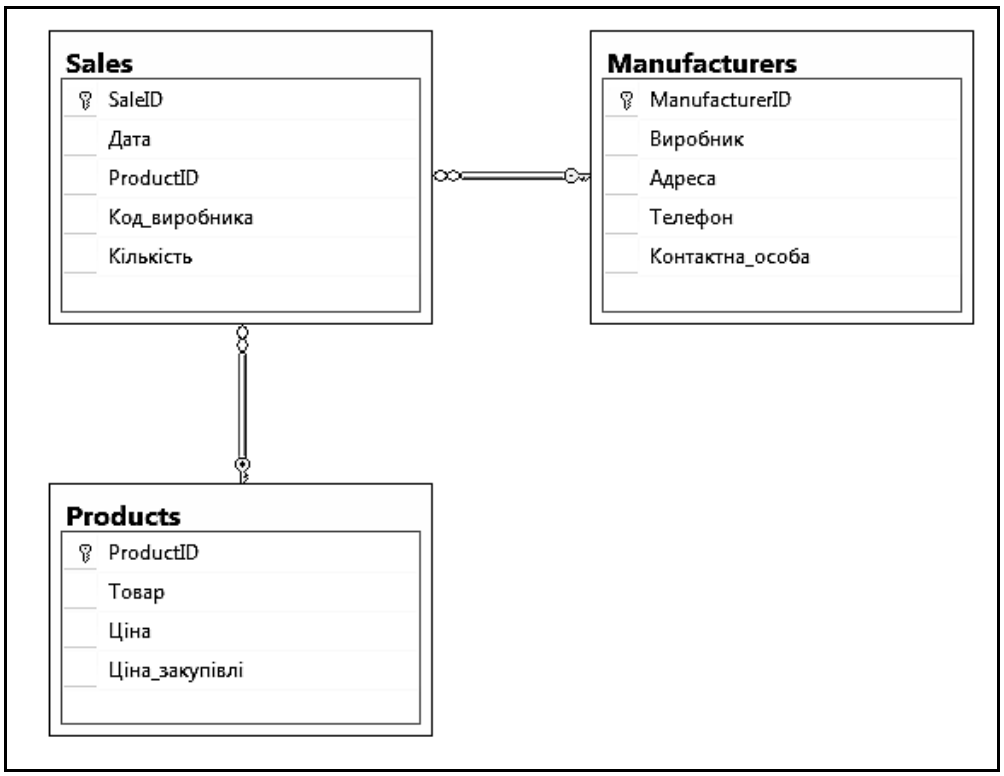


Рис. 8.5. **Схема бази даних після задавання атрибута *зовнішній ключ* для поля *Код_виробника* у таблиці *Sales***

Заборона Null. Усі поля крім первинних та зовнішніх ключів за замовчуванням можуть мати значення Null. Якщо ж потрібно заборонити таку можливість (тобто полю обов'язково потрібно задавати хоч якесь значення), то в описі класу перед іменем відповідної властивості вказують анотацію

[Required]

Приклад 8.9. Задавання атрибута *заборона Null* для поля ***Виробник*** у таблиці ***Manufacturers***.

```
public class Manufacturer
{
    public int ManufacturerID { get; set; }
    [Required]
    public string Виробник { get; set; }
    public string Адреса { get; set; }
    public string Телефон { get; set; }
    public string Контактна_особа { get; set; }

    // Властивості навігації
    public virtual ICollection<Sale> Sales { get; set; }
}
```

Примітка. Якщо бази даних не існувало, то створиться база даних з таблицею ***Manufacturers***, в якій поле ***Виробник*** не має дозволу отримувати значення Null (рис. 8.6).

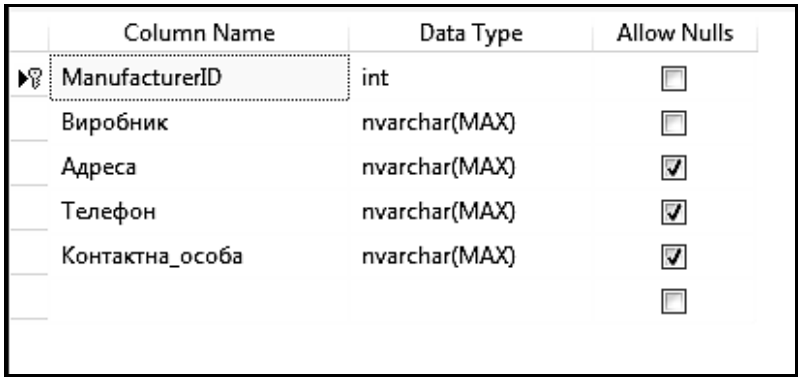


Рис. 8.6. **Схема таблиці *Manufacturers* після задавання атрибута *заборона Null* для поля *Виробник***

Довжина рядка. Якщо поле в базі даних має тип nvarchar, то його довжину можна обмежити за допомогою анотацій

[MinLength(nn)]

[MaxLength(nn)]

Приклад 8.10. Задавання обмежень на довжину текстових полів у таблиці ***Manufacturers***.

```
public class Manufacturer
{
    public int ManufacturerID { get; set; }
    [MinLength(3)]
    [MaxLength(20)]
    public string Виробник { get; set; }
    [MaxLength(30)]
    public string Адреса { get; set; }
    [MaxLength(15)]
    public string Телефон { get; set; }
    [MaxLength(20)]
    public string Контактна_особа { get; set; }

    // Властивості навігації
    public virtual ICollection<Sale> Sales { get; set; }
}
```

Примітка. Якщо бази даних не існувало, то створиться база даних з таблицею ***Manufacturers***, в якій текстові поля мають обмеження довжини (рис. 8.7).

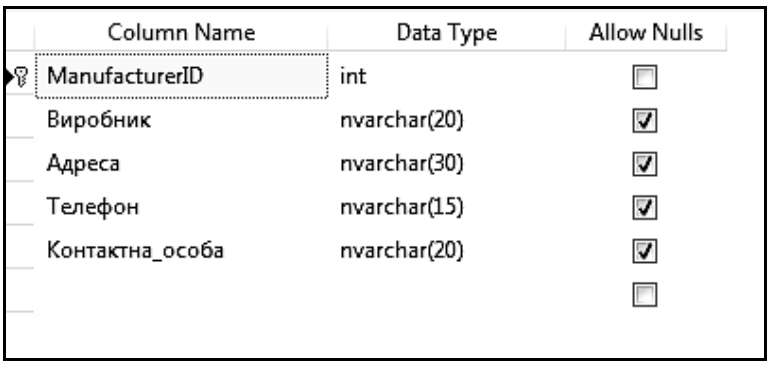


Рис. 8.7. **Схема таблиці *Manufacturers* після задавання обмежень на довжину текстових полів**

Заборона відображення. Якщо потрібно заборонити відображати якусь властивість чи клас в базу даних, перед ним записують анотацію

[NotMapped]

Приклад 8.10. Заборона переносити властивість ***Вартість*** як поле у таблиці ***Sales***.

```
public class Sale
{
    public int SaleID { get; set; }
    public DateTime Дата { get; set; }
    public int ProductID { get; set; }
    public int ManufacturerID { get; set; }
    public Int16 Кількість { get; set; }
    [NotMapped]
    public decimal Вартість { get; set; }

    // Властивості навігації
    public virtual Product Product { get; set; }
    public virtual Manufacturer Manufacturer { get; set; }
}
```

Ім'я поля. Якщо потрібно, щоб ім'я поля не співпадало з іменем властивості класу сутності, використовують анотацію

[Column("*Ім'я поля*")]

Її поміщають перед описом властивості сутності.

Приклад 8.11. Відображення властивості ***Виробник*** на поле ***Хлібозавод***.

```
public class Manufacturer
{
    public int ManufacturerID { get; set; }
    [Column("Хлібозавод")]
```

```
    public string Виробник { get; set; }
    public string Адреса { get; set; }
    public string Телефон { get; set; }
    public string Контактна_особа { get; set; }

    // Властивості навігації
    public virtual ICollection<Sale> Sales { get; set; }
}
```

Примітки. 1. Для використання анотації [Column("*Ім'я поля*")] потрібно в класі сутності додати до описів просторів імен ще й такий

```
using System.ComponentModel.DataAnnotations.Schema;
```

2. Якщо бази даних не існувало, то створиться база даних із таблицею ***Manufacturers***, в якій є поле ***Хлібозавод*** (рис. 8.8).

Налаштування відображення між концептуальною моделлю і базою даних засобами Data Annotations здійснюється шляхом внесення змін у модель. Це ускладнює її розуміння.

На відміну від Data Annotation використання Fluent API дозволяє налаштувати сутності, залишивши незмінним код моделі. Налаштування відображення в цьому разі здійснюється під час формування об'єкта класу DbContext.

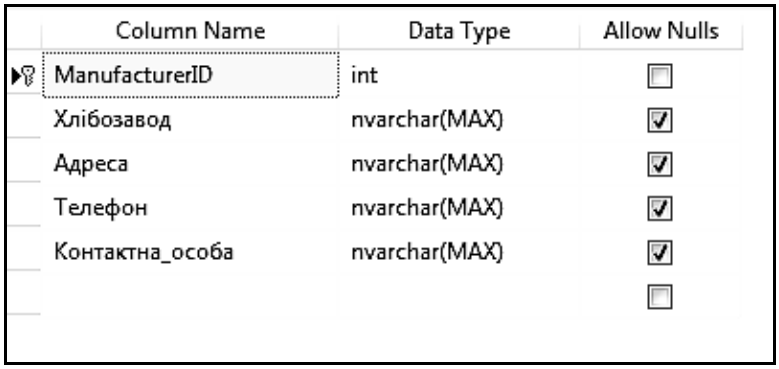


Рис. 8.8. **Схема таблиці *Manufacturers* з полем *Хлібозавод***

Налаштування відображення за допомогою Fluent API здійснюється у такій послідовності. Для кожної сутності, яку потрібно налаштувати, створюють клас конфігурації EntityTypeConfiguration, в якому вказують потрібні зміни. Ці вказівки використовуються будівником моделі класу DbModelBuilder. Він викликається перед побудовою моделі. В оброблювачі події OnModelCreating будівника DbModelBuilder додають до його колекції конфігурацій класи, в яких описано налаштування кожної сутності.

У табл. 8.3 подано методи EntityTypeConfiguration, які найчастіше використовують для налаштування відображення між концептуальною моделлю і базою даних засобами Fluent API.

Таблиця 8.3

Основні засоби Fluent API

Об'єкт налаштування	Засіб Fluent API
Ім'я таблиці	Entity<T>.ToTable("TableName")
Первинний ключ	Entity<T>.HasKey(t=>t.PropertyName)
Зовнішній ключ	Entity<T>.HasRequired(c => c.PropertyName1). WithMany(t => t. PropertyName2).Map(m => m.MapKey("ColumnName"))
Заборона Null	Entity<T>.Property(t=>t.PropertyName).IsRequired
Довжина рядка	Entity<T>.Property(t=>t.PropertyName).HasMaxLength(nn)
Кількість цифр і десятко-вих цифр у Decimal	Entity<T>.Property(t=>t.PropertyName).HasPrecision(n,m)
Заборона відображення сутності	Ignore<T>()
Заборона відображення властивості	Entity<T>.Ignore(t => t.PropertyName);
Ім'я поля	Entity<T>.Property(t => t.PropertyName). HasColumnName("ColumnName");

Приклад 8.12. Налаштування сутностей Product, Manufacturer та Sales засобами Fluent API.

Опис класу ProductConfiguration.

```
class ProductConfiguration : EntityTypeConfiguration<Product>
{
    public ProductConfiguration()
    {
        ToTable("Products");
        HasKey(p => p.ProductID);
        Property(p => p.ProductID).IsRequired().
            HasDatabaseGeneratedOption((DatabaseGeneratedOption.
            Identity));
        Property(p => p.Товар).IsRequired().HasMaxLength(25);
        Property(p => p.Ціна).IsOptional().HasPrecision(5, 2);
         Property(p => p.Ціна_закупівлі).IsRequired().HasPrecision(5,2);
    }
}
```

Опис класу ManufacturerConfiguration.

```
class ManufacturerConfiguration:EntityTypeConfiguration<Manufacturer>
{
    public ManufacturerConfiguration()
    {
        Property(m => m.Виробник).IsRequired().HasMaxLength(20);
        Property(m => m.Адреса).IsRequired().HasColumnName("Адреса").
            HasMaxLength(30);
        Property(m => m.Телефон).IsRequired().
            HasColumnType("nvarchar").HasMaxLength(15);
        Property(m => m.Контактна_особа).IsOptional().
            HasMaxLength(20);
    }
}
```

Опис класу SaleConfiguration.

```
class SaleConfiguration : EntityTypeConfiguration<Sale>
{
    public SaleConfiguration()
    {
        // Ключ
        HasKey(s => s.SaleID);
        // Властивості
        Property(s => s.SaleID).HasColumnType("int").IsRequired();
        Property(s => s.Дата).HasColumnType("date").IsRequired();
        Property(s => s.ProductID).IsRequired();
        Property(s => s.ManufacturerID).IsRequired();
        Property(s => s.Кількість).HasColumnType("smallint").
            IsRequired();
        // Не відображати в БД
        Ignore(s => s.Вартість);
        // Зовнішні ключі
        HasRequired(s => s.Product).WithMany(s => s.Sales).
            HasForeignKey(s =>s.ProductID).WillCascadeOnDelete(true);
        HasRequired(s => s.Manufacturer).WithMany(s => s.Sales).
            HasForeignKey(s=>s.ManufacturerID).
            WillCascadeOnDelete(true);
    }
}
```

Опис методу ***OnModelCreating*** в класі ***BreadContext***. У ньому додаються об'єкти конфігурацій до колекції будівника моделі.

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Configurations.Add(new ProductConfiguration());
    modelBuilder.Configurations.Add(new ManufacturerConfiguration());
    modelBuilder.Configurations.Add(new SaleConfiguration());
}
```

Запитання і завдання

1. В яких випадках використовують засоби Data Annotations чи Fluent API?
2. Які засоби Data Annotations вам відомі. Опишіть їхнє призначення.
3. Додайте до таблиці "Основні засоби Fluent API" стовпець "Засіб Data Annotations", в якому вкажіть відповідну анотацію.

8.7. Удосконалення моделі (міграції)

У процесі експлуатації бази даних може виявитися необхідність її змінити (додати нову таблицю чи стовпець, змінити властивості якогось стовпця, наприклад, збільшити розмір текстового поля). У таких випадках застосування розглянутих засобів призведе до видалення бази даних і повторного її створення. При цьому втрачаються дані, які зберігалися в базі, що значно ускладнюється у тому разі, коли база даних вже тривалий час використовується на підприємстві. Ця проблема вирішується в CodeFirst за допомогою засобів міграцій.

Керування міграціями виконується в середовищі Visual Studio у вікні **Package Manager Console**. Воно відкривається за допомогою однойменної команди, що міститься у списку команди **View – Other Windows** (рис. 8.9).

У цьому вікні в режимі консолі вводять команди, які спричиняють виконання міграцій.

Примітка. Щоб у вікні **Package Manager Console** виконати міграцію з базою даних, не обов'язково її переводити у стан відключення від сервера, як це вимагалося перед запуском програми на виконання, коли створювалася база даних.

Щоб розпочати міграції виконують команду **Enable-Migrations**. У результаті її виконання в проекті створюється папка **Migrations** з двома файлами: ***xxx_InitialCreate.cs*** та ***Configuration.cs*** (рис. 8.10).

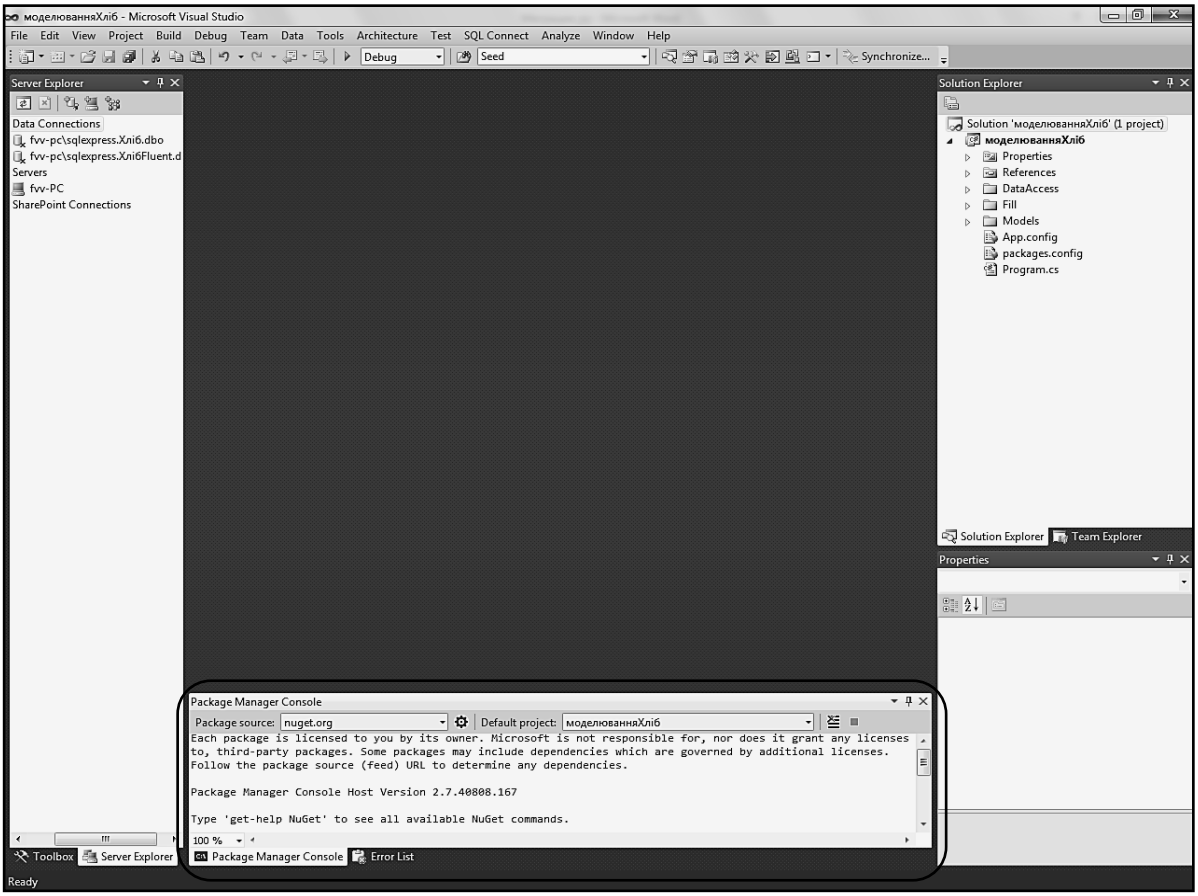


Рис. 8.9. **Вікно *Package Manager Console* в середовищі Visual Studio**

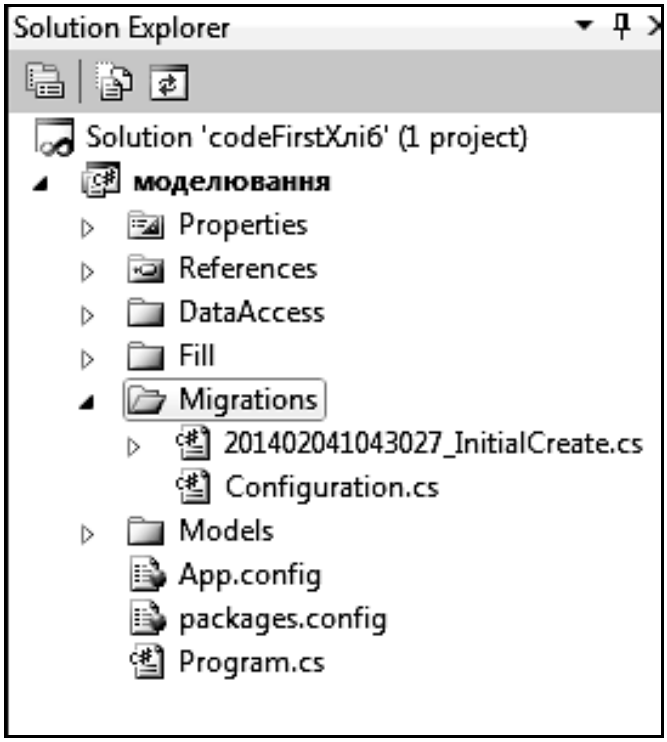


Рис. 8.10. **Нова папка *Migrations* в проекті**

Файл ***Configuration.cs*** містить опис однойменного класу. У ньому міститься конструктор класу, який дає можливість керувати автоматичним виконанням міграцій, і метод ***Seed***. Цей метод застосовують для заповнення даними таблиць. Оскільки метод ***Seed*** викликається автоматично після виконання кожної нової міграції, потрібно потурбуватися про те, щоб повторно не записувалися в базу даних ті самі дані. Тому в методі ***Seed*** перед збереженням даних виконують перевірку на їхню неповторюваність. У найпростішому випадку, коли перевірка здійснюється за значеннями тільки однієї властивості сутності, використовують метод AddOrUpdate цієї сутності.

Приклад 8.13. Заповнення даними таблиці ***Manufacturer*** з перевіркою на недублювання записів.

```
protected override void Seed(моделювання.DataAccess.BreadContext context)
{
    var manufacturers = new List<Manufacturer>
    {
new Manufacturer
{ Виробник = @"Х/з ""Салтівський""",
Адреса = "вул. Гв. Широнінців, 1",
Телефон ="(057)710-50-40",
Контактна_особа="Іванов І. І." },
new Manufacturer
{ Виробник = @"Х /з ""Кулиничі""",
Адреса = "смт Кулиничі, вул. Шкільна, 18",
Телефон ="(0572)62-51-37",
Контактна_особа="Петренко П. П." }
};
// Формуємо колекцію нових виробників без повторювань
    manufacturers.ForEach(m => context.Manufacturers.AddOrUpdate
(v => v.Виробник, m));
context.SaveChanges();
}
```

Файл ***xxx_InitialCreate.cs*** як і файли подальших міграцій має префікс, в якому зазначено дату і час створення міграції. Наприклад, префікс ***201402041043027*** означає, що міграцію створено ***04.02.2014 р.*** о ***10:43:027 год***. Завдяки префіксам міграції впорядковуються за часом їхнього виконання.

Кожна міграція є шаблоном (за термінологією Code First – *scaffold*). Він містить два методи – ***Up()*** і ***Down()***. У методі ***Up()*** зазначено дії з базою даних, які має виконати міграція, а в методі ***Down()*** – скасувати дії, що виконані міграцією.

Приклад 8.14. Міграція ***InitialCreate***.

```
public partial class InitialCreate : DbMigration
{
    public override void Up()
    {
        CreateTable(
            "dbo.Manufacturers",
            c => new
                {
                   ManufacturerID=c.Int(nullable:false,identity:true),
                   Виробник=c.String(nullable:false,maxLength:20),
                   Адреса= c.String(nullable: false, maxLength:30),
                   Телефон = c.String(nullable:false,maxLength:15),
                   Контактна_особа = c.String(nullable: false,
                       maxLength: 20),
                })
            .PrimaryKey(t => t.ManufacturerID);

        CreateTable(
            "dbo.Sales",
            c => new
                {
                    SaleID = c.Int(nullable: false, identity: true),
                    Дата = c.DateTime(nullable: false),
                    ProductID = c.Int(nullable: false),
                    ManufacturerID = c.Int(nullable: false),
                    Кількість = c.Short(nullable: false),
                })
            .PrimaryKey(t => t.SaleID)
            .ForeignKey("dbo.Manufacturers", t => t.ManufacturerID,
                 cascadeDelete: true)
            .ForeignKey("dbo.Products", t => t.ProductID,
                 cascadeDelete: true)
            .Index(t => t.ManufacturerID)
            .Index(t => t.ProductID);

        CreateTable(
            "dbo.Products",
            c => new
                {
                    ProductID=c.Int(nullable: false, identity: true),
                    Товар = c.String(nullable: false, maxLength: 25),
                    Ціна = c.Decimal(precision: 18, scale: 2),
                    Ціна_закупівлі = c.Decimal(nullable: false,
                       precision: 18, scale: 2),
                })
```

```
            .PrimaryKey(t => t.ProductID);

    }

    public override void Down()
    {
        DropForeignKey("dbo.Sales", "ProductID", "dbo.Products");
       DropForeignKey("dbo.Sales","ManufacturerID","dbo.Manufacturers");
        DropIndex("dbo.Sales", new[] { "ProductID" });
        DropIndex("dbo.Sales", new[] { "ManufacturerID" });
        DropTable("dbo.Products");
        DropTable("dbo.Sales");
        DropTable("dbo.Manufacturers");
    }
}
```

Примітка. У методі ***Up()*** зазначені дії для створення всіх таблиць, що містилися в базі даних, а в методі ***Down()*** – їхнє видалення.

Для внесення змін у схему бази даних спочатку змінюють концептуальну модель (додають новий клас сутності, додають чи змінюють властивість у якомусь класі сутності тощо). Потім у вікні **Package Manager Console** виконують команду **Add-Migration**. Вона формує шаблон міграції, що відповідає змінам **в** концептуальній моделі. У разі необхідності розробник може додати зміни у цей шаблон. Після цього виконують команду **Update-Database**. Вона викликає на виконання шаблон останньої міграції, щоб реалізувати зміни в базі даних.

Приклад 8.15. Додавання до бази даних нової таблиці ***Invoices***.

Кроки виконання:

1. Опис нового класу сутності ***Invoice***:

```
public class Invoice
{
    public int InvoiceID { get; set; }
    [MaxLength(12)]
    public string Номер_накладної { get; set; }
    public DateTime Дата { get; set; }
    public int ManufacturerID { get; set; }

    // Властивості навігації
    public virtual Manufacturer Manufacturer { get; set; }
}
```

Додана в клас ***BreadContext*** властивість колекції нових сутностей:

```
public DbSet<Invoice> Invoices { get; set; }
```

2. Команда Add-Migration AddInvoice.

3. Команда Update-Database.

Примітка. 1. У команді **Add-Migration** на другому кроці параметр **AddInvoice** є ім'ям міграції, яке довільно дає розробник.

2. Після виконання другого кроку утворюється шаблон міграції, який містить методи ***Up()*** і ***Down()***.

```
public partial class AddInvoice : DbMigration
{
    public override void Up()
    {
        CreateTable(
            "dbo.Invoices",
            c => new
                {
                    InvoiceID=c.Int(nullable: false, identity: true),
                    Номер_накладної = c.String(maxLength: 6),
                    Дата = c.DateTime(nullable: false),
                    ManufacturerID = c.Int(nullable: false),
                })
            .PrimaryKey(t => t.InvoiceID)
            .ForeignKey("dbo.Manufacturers", t => t.ManufacturerID,
                 cascadeDelete: true)
            .Index(t => t.ManufacturerID);
    }

    public override void Down()
    {
        DropForeignKey("dbo.Invoices", "ManufacturerID",
            "dbo.Manufacturers");
        DropIndex("dbo.Invoices", new[] { "ManufacturerID" });
        DropTable("dbo.Invoices");
    }
}
```

3. Після виконання команди **Update-Database** у базі даних створюється нова таблиця ***Invoices*** (рис. 8. 11).

Під час виконання команди **Update-Database**у вікні **Package Manager Console** можна відобразити SQL-команди, якщо цю команду задати у вигляді

Update-Database –Verbose

До шаблону міграції можна безпосереднього додавати SQL-команди. Для цього використовують метод

SQL(рядок)

Тут **рядок** – текстовий рядок, у якому міститься SQL-команда.

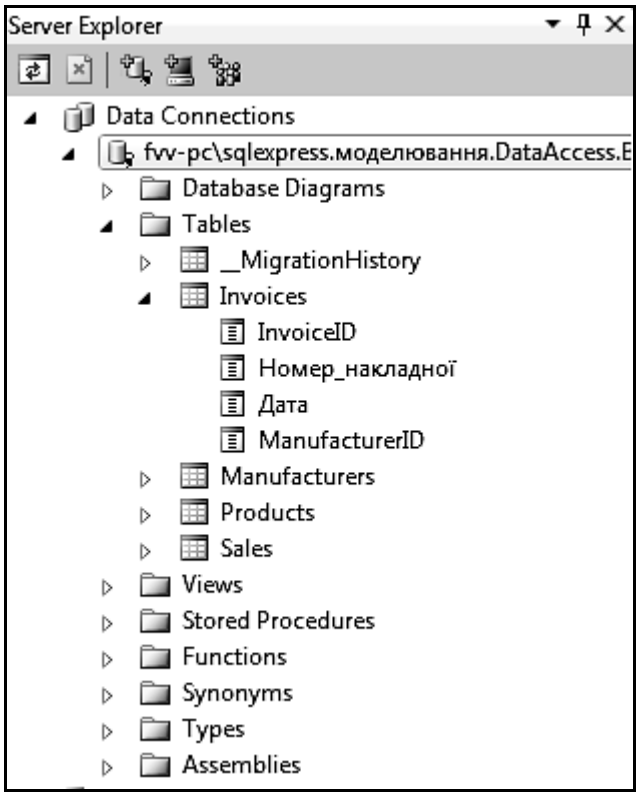


Рис. 8.11. **Нова таблиця *Invoices* у базі даних**

Приклад 8.16. Заміна порожнього значення поля ***Номер_накладної*** у всіх рядках таблиці ***Invoices*** на значення, що обчислюються за таким виразом

ID + "–" + YYYYMMDD,

з використанням значення полів того самого рядка. У виразі прийнято такі скорочення:

ID – значення поля ***ManufacturerID***;

YYYYMMDD – рік, місяць і день у полі ***Дата***.

Зазначені величини розділяються символом "мінус".

Розв'язок. У метод ***Up()*** коду шаблона міграції додається таке викликання методу SQL:

```
Sql("UPDATE Invoices SET Номер_накладної = CONVERT(varchar(3),
    ManufacturerID) + '-'  + CONVERT(varchar(8), Дата, 112)
    WHERE Номер_накладної IS NULL");
```

Для виконання зворотної дії в метод ***Down()*** коду шаблона міграції додається таке викликання методу SQL:

```
Sql("UPDATE Invoices SET Номер_накладної = NULL
    WHERE Номер_накладної=CONVERT(varchar(3), ManufacturerID) + '-'+
    CONVERT(varchar(8), Дата, 112)");
```

Окрім оновлення бази даних до останньої міграції, за допомогою команди **Update-Database** можна перейти до заданої міграції (більш ранньої чи більш пізньої). Для цього використовують параметр

–TargetMigration

Приклад 8.17. Повернення до стану, в якому перебувала база даних після виконання міграції **AddInvoice**

Update- Database - TargetMigration : AddInvoice

Приклад 8.18. Повернення до повністю порожньої бази даних

Update- Database –TargetMigration : $InitialDatabase

Якщо розробник доопрацював базу даних і потрібно зробити такі самі зміни у базі даних підприємства, він може передати відповідний SQL-скрипт адміністратору бази даних цього підприємства. SQL-скрипт, що описує зміни від заданої початкової міграції до заданої кінцевої міграції, можна отримати за допомогою ключа **– Script** і відповідних параметрів початкової та кінцевої міграцій. Команду задають у такому вигляді:

Update-Database -Script -SourceMigration: ***Початкова_міграція*** **-TargetMigration:** ***Кінцева_міграція***

Приклад 8.19. SQL-скрипт, що описує зміни від початку (міграція **InitialDatabase**) до міграції **AddInvoiceProduct**

Update-Database -Script -SourceMigration: $InitialDatabase
-TargetMigration: AddInvoiceProduct

Якщо потрібно створити копію бази даних, яку отримано в результаті виконання усіх міграцій, можна використати проект, що містить міграції. Для цього реєструють ініціалізатор бази даних **MigrateDatabase-ToLatestVersion**. Він використовує міграції, щоб оновити базу даних до останньої версії на початку роботи застосування. Тому цей спосіб роботи

з базою даних називають *автоматичне оновлення під час запуску застосування*.

Щоб виконався процес оновлення потрібно, щоб в застосуванні здійснювалися операції з базою даних (наприклад, читання, додавання, вилучення або змінення даних).

Найчастіше реєстрацію ініціалізатора бази даних і операцію з нею вказують в методі ***Main*** класу **Program**.

Приклад 8.20. Автоматичне оновлення бази даних під час запуску застосування з виведенням назв товарів.

```
static void Main(string[] args)
{
    // Реєстрування ініціалізатора бази даних
    Database.SetInitializer(new
      MigrateDatabaseToLatestVersion<BreadContext, Configuration>());

    using (var context = new BreadContext())
    {
        foreach (var p in context.Products)
        {
            Console.WriteLine(p.Товар);
        }
    }

    Console.WriteLine("Натисніть будь-яку клавішу, щоб вийти...");
    Console.ReadKey();
}
```

Примітки. 1. Команду для автоматичного оновлення бази даних під час запуску застосуванняу середовищі VisualStudio задають не у вікні **Package Manager Console**, а за допомогою меню **Debug – Start Debugging**.

2. Перед запуском на виконання застосування потрібно налаштувати **ConnectionString**, що міститься у файлі **App.config** (ім'я сервера у параметрі **Data Source** та ім'я бази даних у параметрі **Initial Catalog**).

Запитання і завдання

1. Яке призначення мають міграції в Code First?
2. Порівняйте міграції і Fluent API.
3. Які команди використовують для виконання міграції? Опишіть їхнє призначення.
4. Яке призначення мають методи ***Up()*** і ***Down()***? Як вони формуються?

8.8. Побудова застосування з використанням технології Code First

Під час створення застосувань у Code First використовують ті самі технології, що й в інших різновидах Entity Framework – з використанням вікна DataSource і без нього, з використанням класу BindingSource і без нього. Але у Code First ці технології мають свої особливості. Далі зупинимося на них.

По-перше, в CodeFirst є можливість спочатку завантажити дані в пам'ять, а потім користуватися ними для відображення в елементах керування на формі. При цьому використовують такі методи та властивості:

метод **Load** – для завантаження даних у пам'ять (у локальний набір сутностей);

властивість **Local** – для доступу до набору сутностей, що знаходяться в пам'яті;

метод **ToBindingList** – для створення колекції рядків, що прив'язують до інтерфейсу, які синхронізуються з даними ObservableCollection. Тобто тут реалізовано двостороннє прив'язування – зміни у локальному наборі сутностей відображаються в прив'язаному елементі керування і, навпаки, зміни в елементікерування спричиняють до змін у локальному наборі сутностей.

Приклад 8.21. Код оброблювача події ***formТовари_Load*** для відображення даних таблиці ***Products***в елементі керування DataGridView з ім'ям ***gvТовари***.

```
BreadContext context;
private void formТовари_Load(object sender, EventArgs e)
{
    // Створюємо екземпляр класу DbContext
    context = new BreadContext();

    // Додаємо колекцію сутностей до контексту
    context.Products.Load();

    // Прив'язуємо набір сутностей до елемента DataGridView
    gvТовари.DataSource = context.Products.Local.ToBindingList();
```

```
    //Вилучаємо властивості навігації з інтерфейсу
    gvТовари.Columns.Remove("InvoiceProducts");
    gvТовари.Columns.Remove("Sales");
}
```

По-друге, у більш складних випадках, коли на одній формі потрібно подавати дані ієрархічно пов'язаних таблиць, з метою забезпечення двосторонньої прив'язки та сортування необхідно розширити ObservableCollection для додавання функціональності IListSource. Інтерфейс IListSource надає об'єкту функціональні можливості, що дозволяють повертати список, який може бути пов'язаний із джерелом даних. Це вимагає внесення змін до батьківського класу для опису пов'язаної колекції сутностей дочірнього класу.

Приклад 8.22. Реалізація інтерфейсу IListSource для батьківської колекції сутностей ***Invoices***, що оброблятимуться на формі разом із колекціями дочірніх сутностей ***InvoiceProducts***.

Кроки виконання:

1. Описнового класу ***ObservableListSource***, що реалізує IListSource для колекцій:

```
public class ObservableListSource<T> :
ObservableCollection<T>, IListSource
where T : class
{
    private IBindingList _bindingList;
    bool IListSource.ContainsListCollection { get { return false; } }

    IList IListSource.GetList()
    {
        return _bindingList ?? (_bindingList = this.ToBindingList());
    }
}
```

Примітка. Більш детально про використання IListSource під час побудови застосувань можна дізнатися у статті Databinding with WinForms на сайті http://msdn.microsoft.com/ru-ru/data/jj682076.

2. Опис класу ***Invoice***, із зміненою властивістю навігації для колекції дочірніх сутностей ***InvoiceProducts***.

```
public class Invoice
{
    public int InvoiceID { get; set; }
    [MaxLength(12)]
    public string Номер_накладної { get; set; }
    public DateTime Дата { get; set; }
    public int ManufacturerID { get; set; }

    // Властивості навігації
    public virtual Manufacturer Manufacturer { get; set; }
    // Змінена властивість навігації для колекції
    // дочірніх сутностей InvoiceProducts
    private readonly ObservableListSource<InvoiceProduct>
        invoiceProducts = new ObservableListSource<InvoiceProduct>();
    public virtual ObservableListSource<InvoiceProduct>
        InvoiceProducts { get { return invoiceProducts; } }
}
```

Після внесення таких змін можна використовувати візуальні засоби побудови форм на основі вікна **DataSource**, як це описано в інших різновидах Entity Framework.

Приклад 8.23. Код оброблювача події ***formНакладні_Load*** для відображення даних таблиць ***Invoice*** та ***InvoiceProducts***.

```
private void formНакладні_Load(object sender, EventArgs e)
{
    // Створюємо екземпляр класу DbContext
    context = new BreadContext();

    // Завантажуємо дані для manufacturerBindingSource
    manufacturerBindingSource.DataSource =
        context.Manufacturers.ToList();

    // Завантажуємо дані для productBindingSource
    productBindingSource.DataSource = context.Products.ToList();

    // Завантажуємо дані для invoiceBindingSource
    context.Invoices.Load();

    // Задаємо дані для з'єднувача
    invoiceBindingSource.DataSource =
        context.Invoices.Local.ToBindingList();
}
```

Примітка. Дані про товари за накладною, що відображаються в елементі керування DataGridView, задано автоматично в типізованому елементі ***invoiceProductsBindingSource*** під час перетягування підвузла ***InvoiceProducts***, що знаходиться у вузлі ***Invoice*** вікна **Data Sources**, на форму.

Запитання і завдання

1. З якою метою використовують властивість Local у Code First?
2. У чому полягають особливості подання на формі ієрархічно пов'язаних таблиць у Code First?
3. Порівняйте технології створення застосувань у Code First та в інших різновидах Entity Framework.

Лабораторна робота № 8. Реалізація доступу до даних на основі технології Code First

Цілі лабораторної роботи:

1. Набуття практичних навичок побудови класів сутностей та класу DbContext.
2. Освоєння засобів Data Annotations та Fluent API для налаштування сутностей.
3. Оволодіння засобами вдосконалення моделі на основі міграцій.
4. Набуття практичних навичок побудови застосувань WindowsForms на основі технології Code First.
5. Удосконалення навичок роботи з інтегрованим середовищем розробки Visual C# і довідковою системою Microsoft Developer Network (MSDN).

Перед виконанням лабораторної роботи студент повинен знати:

1. Основи створення POCO-класів.
2. Базові домовленості в Code First.
3. Основи проектування реляційних баз даних.
2. Структурні елементи бази даних і їхні властивості.
3. Основи побудови SQL-запитів.
4. Принципи прив'язування даних до елементів інтерфейсу.
5. Основи роботи з ієрархічними даними.
6. Принципи обробки подій в C#-програмі.

Після виконання лабораторної роботи студент повинен вміти:

1. Створювати класи сутностей та DbContext.
2. Налаштовувати сутності засобами Data Annotations та Fluent API.
3. Супроводжувати бази даних засобами міграцій.
2. Самостійно розробляти C#-застосування на основі технології Code First.
3. Використовувати основні бібліотеки .Net Framework під час розробки програм.

Хід роботи

1. Побудова попередньої моделі.
1.1. Установлення пакета Entity Framework.
1.2. Побудова класів сутностей.
1.3. Побудова класу DbContext.
1.4. Заповнення даними.
1.5. Установлення імені бази даних.
1.6. Налаштування сутностей з використанням Data Annotations.
1.7. Використання Fluent API для налаштування сутностей.
2. Розвиток моделі (міграції).
2.1. Включення міграцій.
2.2. Формування методу Seed.
2.3. Додавання нової сутності Invoice.
2.4. Змінення атрибутів властивості. Використання SQL користувача.
2.5. Додавання дочірньої сутності.
2.6. Відкочування до заданої міграції.
2.7. Отримання скрипта SQL.
3. Побудова застосування.
3.1. Додавання проекту Windows Forms.
3.2. Безпосереднє прив'язування сутностей.
3.3. Візуальні засоби побудови інтерфейсу.
3.4. Робота з ієрархічними сутностями.
3.4. Використання технології LINQ to Entity в задачах аналізу даних.

Інструкції

Постановка загальної задачі

Вивчення засобів роботи з даними на основі технології Code First проводиться шляхом створення рішення ***codeFirstХліб***, яке складається з двох проектів – ***моделювання*** та ***winForms***.

Проект ***моделювання*** є консольним, а ***winForms*** – проектом Windows Forms. У проекті ***моделювання*** створюються класи сутностей та клас DbContext. На їхній основі будується база даних, яка потім налаштовується засобами Data Annotations та Fluent API. Після цього вона розвивається з використанням засобів міграцій.

Проект ***winForms*** призначений для побудови інтерфейсу користувача з даними, що зберігаються в базі даних. Під час його розроблення використовуються засоби, які аналогічні до тих, що вживалися під час виконання попередньої лабораторної роботи. Але водночас вони мають свою специфіку. Зверніть увагу на особливості роботи з ієрархічними сутностями.

Отже, консольне застосування використовується для виконання п. п. 1, 2 ходу роботи, а Windows Forms-застосування – п. 3.

1. Побудова попередньої моделі

1.1. Установлення пакета Entity Framework

1. Створіть нове рішення ***codeFirstХліб***, а в ньому консольне застосування ***моделювання***. Для цього:

1.1. Відкрийте Visual Studio.

1.2. Виконайте команду **File – New – Project**.

1.3. Виберіть вид застосування **Console Application** у вікні **New – Project**, а потім введіть проекту ***моделювання*** у текстовому полі **Name** та ім'я рішення ***codeFirstХліб*** у текстовому полі **Solution Name** (рис. 8.12).

2. Додайте останню версію пакета Entity Framework за допомогою засобу NuGet. Для цього:

2.1. Виконайте команду **Project – Manage NuGet Packages**. З'явилося вікно Manage NuGet Packages.

2.2. Виберіть елемент **Online** в лівому списку.

2.3. Виберіть елемент **Entity Framework** у центральному списку і клацніть кнопку **Install** (рис. 8.13).

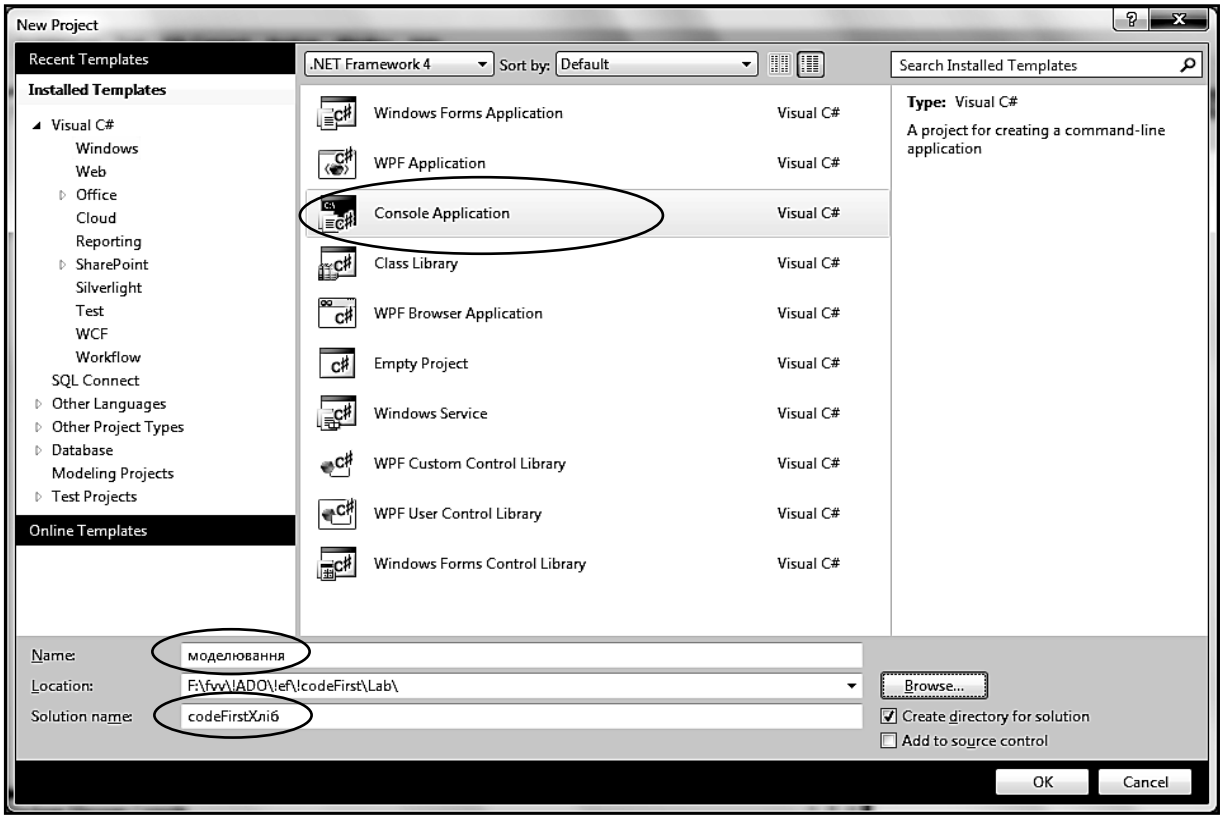


Рис. 8.12. **Побудова рішення *codeFirstХліб* з консольним застосуванням *моделювання***

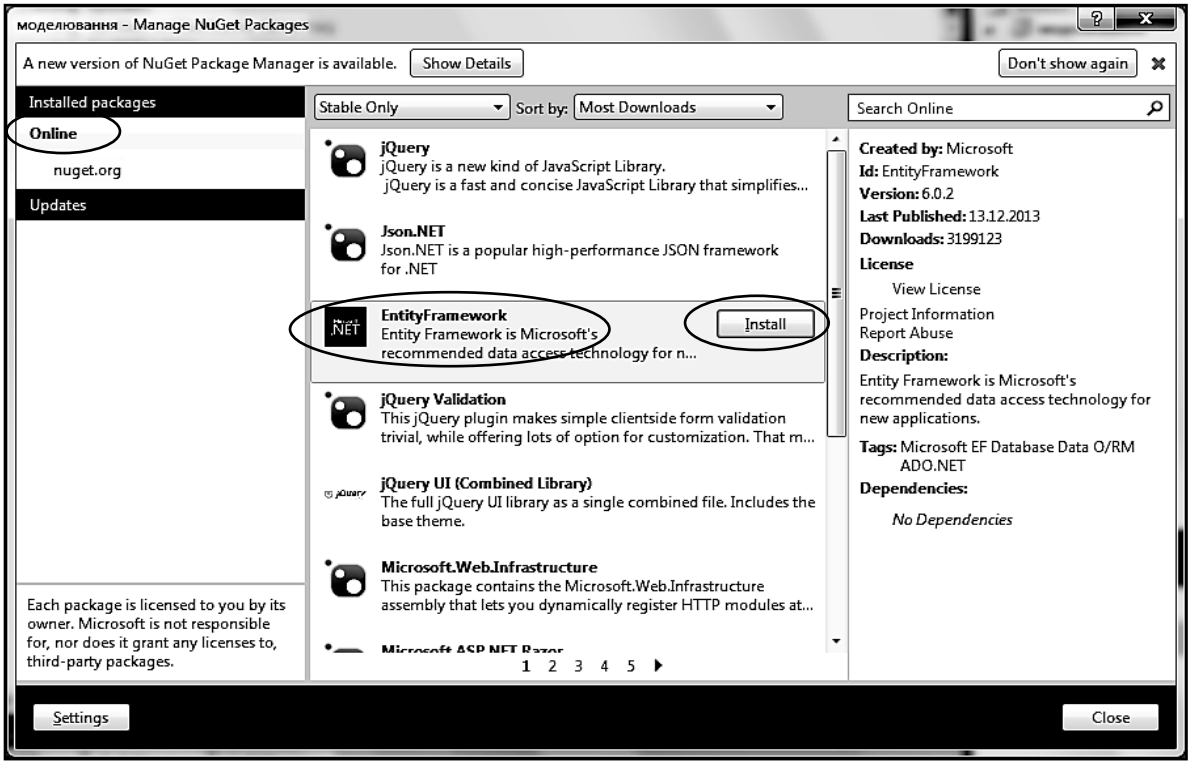


Рис. 8.13. **Додавання останньої версії пакета Entity Framework**

Дочекайтеся, поки завершиться встановлення пакета EntityFramework, клацнувши кнопку **I Accept** у кінці. У папці ***моделюванняХліб*** поряд із файлом рішення з'явилася папка ***packages***. Перевірте її вміст.

Примітка. Якщо лабораторна робота виконується у середовищі Visual Studio, в якій ще не встановлено засіб **NuGet** (у меню **Project** відсутня команда **Manage NuGet Packages**), то перед виконанням п. 2 встановіть його, виконавши таке:

1. Перебуваючи у середовищі Visual Studio, виконайте команду **Tools – Extension Manager**. З'явилося вікно **Extension Manager**.
2. Виберіть елемент **Online Gallery** в лівому списку.
3. Виберіть елемент **NuGet Package Manager** у центральному списку і клацніть кнопку **Download**.
4. Далі відповідайте на запити завантажувача, щоб встановити пакет **NuGet**.

Пакет **NuGet** можна також скачати з поза меж Visual Studio, перейшовши на сайт

http://visualstudiogallery.msdn.microsoft.com/27077b70-9dad-4c64-adcf-c7cf6bc9970c

1.2. Побудова класів сутностей

Завдання

Побудувати класи сутностей, за якими у подальшому буде створено базу даних ***Хліб***. У базі даних міститимуться таблиці аналогічні таблицям ***Товари***, ***Виробники*** та ***Продажі***, які розглядалися у лабораторних роботах до вивчення платформи Entity Framework.

Ідея розв'язку

Клас сутності є відовраженням рядка відповідної таблиці бази даних. Тому в ньому як властивості класу вказують поля таблиці. Типам даних властивостей класу відповідають типи полів у таблиці.

Щоб спростити створення бази даних, класам і таблицям слід дати імена англійською мовою. Причому імена класів матимуть форму однини (***Product***, ***Manufacturer*** та ***Sale***), тоді імена таблиць отримають форму множини (***Products***, ***Manufacturers*** та ***Sales***).

Майже всім властивостям сутностей можна дати ті самі імена українською мовою, що співпадають з іменами полів у таблицях, за винятком

ключових полів (первинних та зовнішніх ключів). Імена первинних ключів слід утворювати за таким шаблоном:

Ім'я_сутностіID,

наприклад, ProductID, ManufacturerID.

Для встановлення зв'язків між таблицями варто зазначити властивості навігації у класах сутностей. Причому в батьківській сутності треба вказати колекцію сутностей дочірньої сутності (наприклад,

public virtual ICollection<Sale> Sales { get; set; }

у сутності ***Product***), а в дочірній сутності, навпаки, – один екземпляр батьківської сутності (наприклад,

public virtual Product Product { get; set; }

у сутності ***Sale***).

Виконання

1. Перейдіть у вікно **Solution Explorer** і додайте до проекту ***моделювання*** нову папку ***Models***.
2. Додайте у папку ***Models*** новий клас ***Product*** і введіть його опис

```
public class Product
{
    public int ProductID { get; set; }
    public string Товар { get; set; }
    public Decimal Ціна { get; set; }
    public Decimal Ціна_закупівлі { get; set; }

    // Властивості навігації
    public virtual ICollection<Sale> Sales { get; set; }
}
```

3. Додайте у папку ***Models*** новий клас ***Manufacturer*** і введіть його опис

```
public class Manufacturer
{
    public int ManufacturerID { get; set; }
    public string Виробник { get; set; }
    public string Адреса { get; set; }
    public string Телефон { get; set; }
```

```
    public string Контактна_особа { get; set; }

    // Властивості навігації
    public virtual ICollection<Sale> Sales { get; set; }
}
```

4. Додайте у папку ***Models*** новий клас ***Sale*** і введіть його опис

```
public class Sale
{
    public int SaleID { get; set; }
    public DateTime Дата { get; set; }
    public int ProductID { get; set; }

    public int ManufacturerID { get; set; }
    public Int16 Кількість { get; set; }

    // Властивості навігації
    public virtual Product Product { get; set; }
    public virtual Manufacturer Manufacturer { get; set; }
}
```

5. Збережіть зміни, що зроблені в проекті.

1.3. Побудова класу DbContext

Завдання

Побудувати клас DbContext, за яким будуть створені таблиці бази даних ***Хліб*** і в подальшому здійснюватиметься доступ до них із застосування через створені раніше класи сутностей. У базі даних міститимуться таблиці, які аналогічні таблицям ***Товари***, ***Виробники*** та ***Продажі***, які розглядалися у лабораторних роботах до вивчення платформи Entity Framework.

Ідея розв'язку

У базі даних міститимуться таблиці, що є аналогічними до таблиць ***Товари***, ***Виробники*** та ***Продажі***. Для доступу до них слід створити клас, у якому вказати як властивості колекції сутностей, що описані у створених вище в класах. Для спрощення відображення цих колекцій у таблиці бази даних треба дати їм такі самі імена, як для таблиць, наприклад, опис властивості

```
public DbSet<Product> Products { get; set; }
```

слугує для відображення колекції ***Products*** сутностей ***Product*** у таблицю ***Products***.

Виконання

1. Додайте до проекту ***моделювання*** нову папку ***DataAccess***.
2. Додайте у папку ***DataAccess*** новий клас ***BreadContext*** і введіть його опис

```
public class BreadContext : DbContext
{
    public DbSet<Product> Products { get; set; }
    public DbSet<Manufacturer> Manufacturers { get; set; }
    public DbSet<Sale> Sales { get; set; }
}
```

3. Щоб забезпечити доступ до опису класів, додайте посилання на простори імен

```
using System.Data.Entity;
using моделювання.Models;
```

1.4. Заповнення даними

Завдання

Створити базу даних ***Хліб*** і заповнити її тестовими даними.

Ідея розв'язку

Щоб створити базу даних відповідно до вказаних описів класів, потрібно запустити застосування на виконання і здійснити будь-яку операцію з її даними. Тому заповнення даними бази попередньо викличе її створення.

Для розвантаження методу ***Main*** у класі ***Program*** винести процес заповнення даними у метод ***FillDB*** окремого класу ***DataAdd***.

Заповнення кожної таблиці бази даних засноване на такому алгоритмі. Спочатку створюються екземпляри сутностей. Потім їхнім властивостям надаються певні значення. Після цього сутності додаються до колекцій. Всі ці дії виконуються безпосередньо в пам'яті. Для збереження даних у базі викликається метод **SaveChanges** класу **DbContext**.

Щоб під час повторного запуску застосування на виконання не було проблем із повторенням даних, слід дотримуватися стратегії видалення вже існуючої бази даних і створення нової. Для цього зазначимо метод **DropCreateDatabaseAlways** в ініціалізаторі бази даних.

Якщо явно не задано ім'я бази даних через рядок підключення, платформа Entity Framework створить базу даних з ім'ям ***моделювання.DataAccess.BreadContext*** на сервері SQLEXPRESS для Visual Studio 2010 чи (localDB)\v.11 для наступних версій Visual Studio. Тому у разі обмеження прав користувача може з'явитися виключення про неможливість створення бази даних. Цю проблему вирішено у наступному пункті.

Виконання

1. Додайте до проекту ***моделювання*** нову папку ***Fill***.
2. Додайте у папку ***Fill*** новий клас ***DataAdd*** і введіть його опис

```
public class DataAdd
{
    public static void FillDB(out int nProduct,
         out int nManufacturer,
         out int nSale)
{
    var products = new List<Product>
    {
        new Product { Товар = @"Хліб ""Український""", Ціна = 3.00M,
            Ціна_закупівлі =2.50M},
        new Product { Товар = @"Батон ""Молочний""", Ціна = 2.80M,
            Ціна_закупівлі =2.50M},
        new Product { Товар = @"Булка з маком", Ціна = 1.98M,
            Ціна_закупівлі =1.85M}
    };
    using (var context = new BreadContext())
    {
        products.ForEach(p => context.Products.Add(p));
        context.SaveChanges();
        // Визначаємо кількість доданих рядків
        nProduct = context.Products.Count();
    }
    var manufacturers = new List<Manufacturer>
    {
        new Manufacturer { Виробник = @"Х/з ""Салтівський""",
            Адреса = "вул. Гв. Широнінців, 1",
            Телефон ="(057)710-50-40",
            Контактна_особа="Іванов І. І." },
        new Manufacturer { Виробник = @"Х/з ""Кулиничі""",
            Адреса = "смт Кулиничі, вул. Шкільна, 18",
            Телефон ="(0572)62-51-37",
            Контактна_особа="Петренко П. П." }
    };
    using (var context = new BreadContext())
```

```
    {
        manufacturers.ForEach(m => context.Manufacturers.Add(m));
        context.SaveChanges();
        // Визначаємо кількість доданих рядків
        nManufacturer = context.Manufacturers.Count();
    }
    var sales = new List<Sale>
    {
        new Sale { Дата = DateTime.Parse("01.09.2014"),ProductID = 1,
            ManufacturerID =1, Кількість=200 },
        new Sale { Дата = DateTime.Parse("01.09.2014"),ProductID = 2,
            ManufacturerID =1, Кількість=150 },
        new Sale { Дата = DateTime.Parse("01.09.2014"),ProductID = 1,
            ManufacturerID =2, Кількість=180 }
    };
    using (var context = new BreadContext())
    {
        sales.ForEach(s => context.Sales.Add(s));
        context.SaveChanges();
        // Визначаємо кількість доданих рядків
        nSale = context.Sales.Count();
    }
}
}
```

3. Щоб забезпечити доступ до опису класів, додайте посилання на простори імен

```
using моделювання.Models;
using моделювання.DataAccess;
```

4. Перейдіть у вікно класу ***Program*** і додайте опис методу ***Main***.

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            // Стратегія роботи з базою даних
            Database.SetInitializer(
                new DropCreateDatabaseAlways<BreadContext>());
            // Описуємо фактичні параметри методу FillDB
            int nProduct;
            int nManufacturer;
```

```
            int nSale;

            // Викликаємо метод
            DataAdd.FillDB(out nProduct, out nManufacturer,
                out nSale);

            // Виводимо результат
            Console.WriteLine(
                "Базу даних на SQL Server створено і заповнено.\n"
                + "У таблиці записано таку кількість рядків:\n"
                + "Products - " + nProduct
                + ", Manufacturers - " + nManufacturer
                + ",  Sales - " + nSale
                + ".\n Перевірте!!!");
        }
        catch (System.Exception ex)
        {
           // Виводимо повідомлення про помилку
           Console.WriteLine("Базу даних не створено. \n Помилка:\n "
           + ex.ToString());
        }

       // Виводимо повідомлення про подальші дії
       Console.WriteLine("Натисніть будь-яку клавішу, щоб вийти...");
       Console.ReadKey();
    }
}
```

5. Щоб забезпечити доступ до опису класів, додайте посилання на простори імен

```
using System.Data.Entity;
using моделювання.DataAccess;
using моделювання.Fill;
```

6. Запустіть програму на виконання і дочекайтеся, поки у вікні консолі з'явиться повідомлення.

7. Знайдіть нову базу даних ***моделюванняХліб.DataAccess.BreadContext*** на сервері SQL Server. Ознайомтеся з властивостями стовпців її таблиць і даними, що містяться в таблицях. Вони відповідають тим, що вказані в методі ***FillDB***.

Ім'я бази даних співпадає з іменем класу **DbContext**. Далі слід потурбуватися про те, щоб під час створення база даних отримувала ім'я, яке вказав користувач.

Примітка. Якщо виконання п. 6 завершилося повідомленням про помилку, що пов'язана з обмеженням прав користувача, відразу переходьте до виконання п. "1.5. Установлення імені бази даних".

1.5. Установлення імені бази даних

Завдання

Створити базу даних з ім'ям ***Хліб*** і заповнити її даними.

Ідея розв'язку

Відповідно до домовленостей Code First треба вибрати ім'я бази даних із рядка під'єднання, який у файлі конфігурації має ім'я класу **DbContext** тобто ***BreadContext***.

Виконання

1. Якщо в результаті виконання попереднього завдання успішно створено базу даних ***моделювання.DataAccess.BreadContext***, видаліть її із сервера SQL Server, наприклад за допомогою запиту

```
DROP DATABASE [моделювання.DataAccess.BreadContext]
```

Примітка. Перед виконанням запиту закрийте з'єднання з базою даних у вікні ServerExplorer.

2. Відкрийте файл конфігурації **App.config** і додайте тег рядка з'єднання, помістивши його перед останнім закриваючим тегом конфігурації. Якщо лабораторна робота виконується на власному комп'ютері, він має такий вигляд:

```
<connectionStrings>
<add name="BreadContext" connectionString="Data Source=.\sqlexpress;
Initial Catalog=Хліб; Integrated Security=True;"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

Якщо лабораторна робота виконується на комп'ютері університету, він має такий вигляд:

```
<connectionStrings>
<add name="BreadContext" connectionString="Data Source=ISServ; Initial
Catalog=Хліб***Прізвище***; Integrated Security=True;"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

Примітка. Замість слова ***Прізвище*** в значенні параметра **Initial Catalog** вставте своє прізвище. Тоді база даних матиме відповідне ім'я, наприклад, ***ХлібПетренко***.

3. Запустіть програму на виконання і дочекайтеся, поки у вікні консолі з'явиться повідомлення.

4. Знайдіть нову базу даних ***Хліб*** на SQLServer. Ознайомтеся з властивостями стовпців її таблиць і даними, що містяться в таблицях (рис. 8.14). Вони відповідають тим, що вказані в методі ***FillDB***.

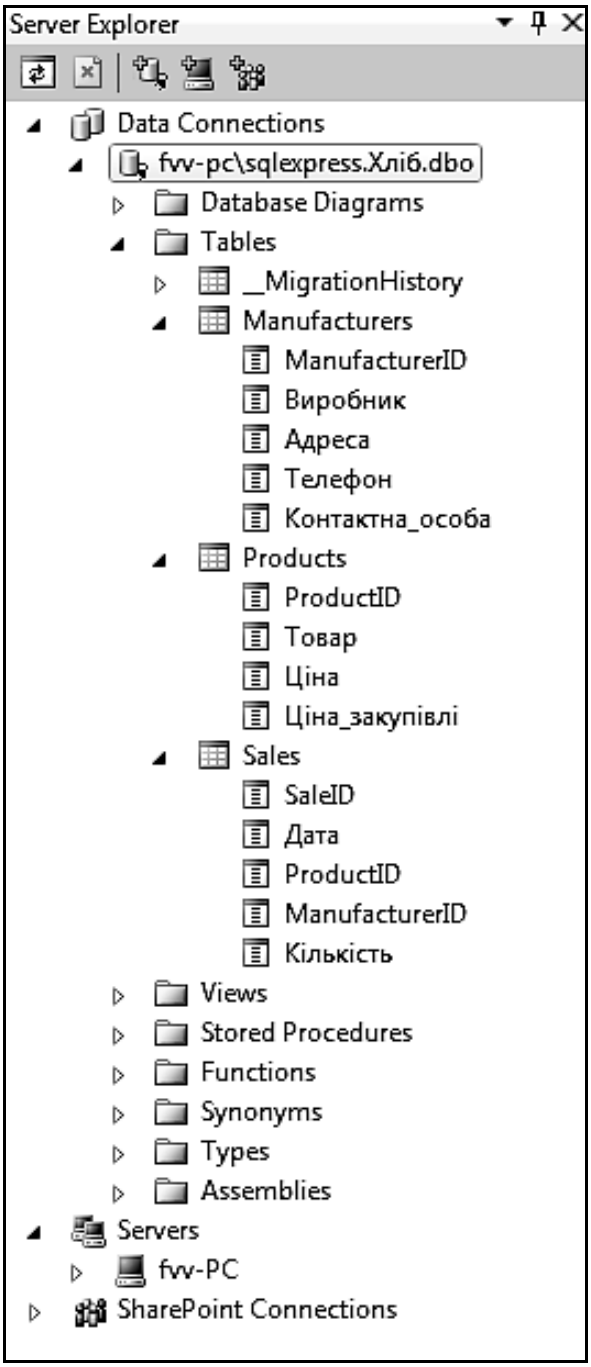


Рис. 8.14. **База даних *Хліб***

5. Закрийте вікно проекту зі збереженням зроблених змін.

Далі займемося налаштуванням сутностей, щоб отримати таблиці з потрібними атрибутами полів.

1.6. Налаштування сутностей з використанням Data Annotations

Завдання 1

Налаштувати властивості сутності ***Products***, щоб вони відповідали властивостям полів таблиці Товари у попередніх проектах. Тобто властивості повинні мати такі атрибути:

Властивість	Атрибути
ProductID	Ключ
Товар	Обов'язкова, довжина не перевищує 25 символів
Ціна	Необов'язкова
Ціна_закупівлі	Обов'язкова

Ідея розв'язку

Для встановлення відповідних атрибутів перед властивостями слід зазначити анотації даних, якщо атрибут не відповідає домовленостям, які прийнято в Code First.

Оскільки ім'я властивості ***ProductID*** складається з імені сутності і символів ***ID***, тому за домовленостями Code First вона є ключем сутності і використання анотації [Key] у цьому разі є надлишковим.

Щоб встановити атрибути ***обов'язкова*** і ***довжина не перевищує 25 символів*** для властивості ***Товар***, треба скористатися анотаціями ***Required*** та ***MaxLength***.

Оскільки властивість ***Ціна*** є необов'язковою, варто зазначити, що вона відноситься до класу **Nullable**, тобто може мати значення ***null***. Якщо цю обставину пропустити, то в разі, коли не задається значення для властивості ***Ціна***, вона отримає значення ***0***, а це не те саме, що ***null***.

Для властивості ***Ціна_закупівлі*** слід зазначити анотацію ***Required***, яка забезпечить атрибут ***обов'язкова***.

Виконання

1. Скопіюйте папку, в якій міститься рішення ***codeFirstХліб*** і новій папці дайте ім'я ***codeFirstХліб_Data_Annotations***. Рішення ***codeFirstХліб*** ще знадобиться для виконання п. 1.7, а також під час захисту лабо-

раторної роботи, щоб можна було продемонструвати початковий варіант проекту і подальше його вдосконалення.

2. Відкрийте рішення ***codeFirstХліб***, що міститься у папці рішення ***codeFirstХліб_Data_Annotations***.

3. Відкрийте визначення полів таблиці ***Products***, двічі клацнувши на її значку у вікні ServerExplorer. На рис. 8.15 подано властивості полів таблиці ***Products***.

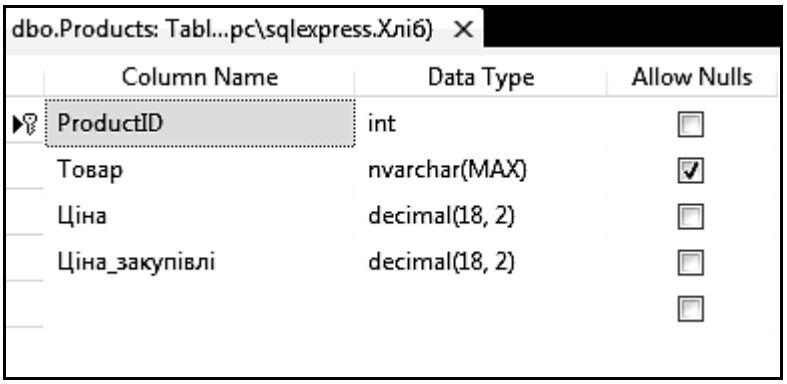


Рис. 8.15. **Властивості полів таблиці *Products***

4. Щоб змінити властивості полів таблиці, відкрийте код класу ***Product*** і вставте анотації перед полями класу. Код класу має такий вигляд:

```
public class Product
{
    public int ProductID { get; set; }
    [Required]
    [MaxLength(25)]
    public string Товар { get; set; }
    public Nullable<Decimal> Ціна { get; set; }
    [Required]
    public Decimal Ціна_закупівлі { get; set; }

    // Властивості навігації
    public virtual ICollection<Sale> Sales { get; set; }
}
```

Щоб забезпечити доступ до опису класів, додайте посилання на простір імен

```
using System.ComponentModel.DataAnnotations;
```

5. Запустіть програму на виконання і дочекайтеся, поки у вікні консолі з'явиться повідомлення. Обновіть вікно **Server Explorer** і перегляньте властивості полів таблиці ***Products*** (рис. 8.16).

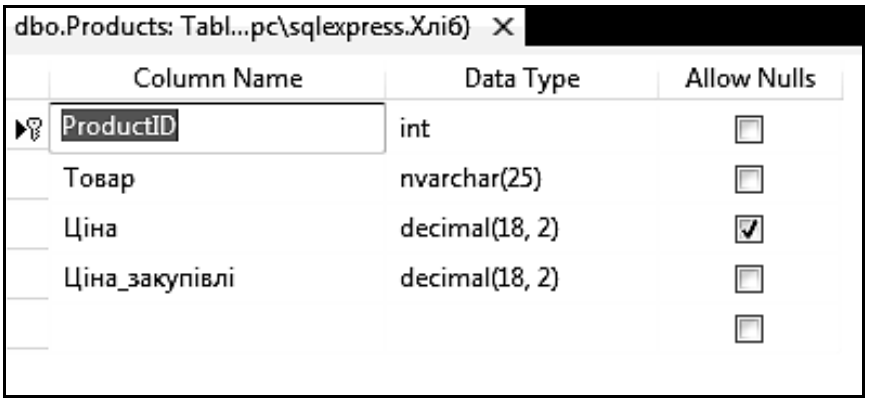


Рис. 8.16. **Властивості полів таблиці *Products* після налаштування**

Примітка. Оскільки під час виконання програми видаляється база даних і створюється нова, слідкуйте за тим, щоб з'єднання з нею було закрите. Це можна зробити контекстною командою бази даних **Close Connection** у вікні **Server Explorer**.

Завдання 2

Налаштувати властивості сутності ***Manufacturer***, щоб вони відповідали властивостям полів таблиці ***Виробники*** у попередніх проектах. Тобто властивості повинні мати такі атрибути:

Властивість	Атрибути
ManufacturerID	Ключ
Виробник	Обов'язкова, довжина не перевищує 20 символів
Адреса	Обов'язкова, довжина не перевищує 30 символів
Телефон	Обов'язкова, довжина не перевищує 15 символів
Контактна_особа	Обов'язкова, але може мати порожнє значення, довжина не перевищує 20 символів

Ідея розв'язку

Встановлення атрибутів властивостей сутності виконується аналогічно описаному для сутності ***Product***.

Для властивості ***Контактна_особа*** слід зазначити анотацію ***Required*** зі значенням параметра ***AllowEmptyStrings = true***.

Виконання

1. Відкрийте визначення полів таблиці ***Manufacturers***, двічі клацнувши на її значку у вікні **Server Explorer**. На рис. 8.17 подано властивості полів таблиці ***Manufacturers***.

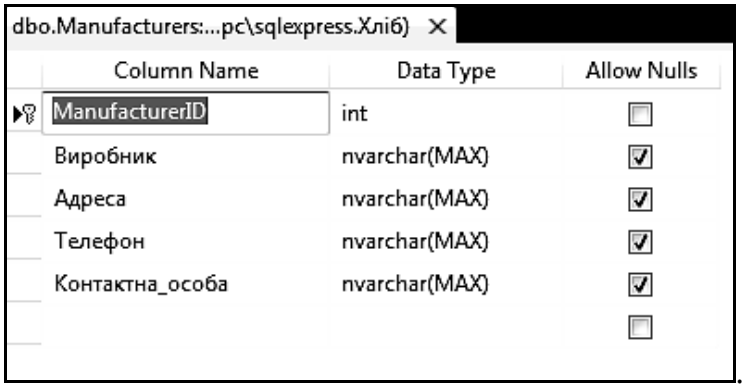


Рис. 8.17. **Властивості полів таблиці *Manufacturers***

2. Щоб змінити властивості полів таблиці, відкрийте код класу ***Manufacturer*** і вставте анотації перед полями класу. Код класу має такий вигляд:

```
public class Manufacturer
{
    public int ManufacturerID { get; set; }
    [Required]
    [MaxLength(20)]
    public string Виробник { get; set; }
    [Required]
    [MaxLength(30)]
    public string Адреса { get; set; }
    [Required]
    [MaxLength(15)]
    public string Телефон { get; set; }
    [Required(AllowEmptyStrings = true)]
    [MaxLength(20)]
    public string Контактна_особа { get; set; }
```

```
    // Властивостінавігації
    public virtual ICollection<Sale> Sales { get; set; }
}
```

Щоб забезпечити доступ до опису класів, додайте посилання на простір імен

```
using System.ComponentModel.DataAnnotations;
```

3. Запустіть програму на виконання і дочекайтеся, поки у вікні консолі з'явиться повідомлення. Обновіть вікно **Server Explorer** і перегляньте властивості полів таблиці ***Manufacturers*** (рис. 8.18).

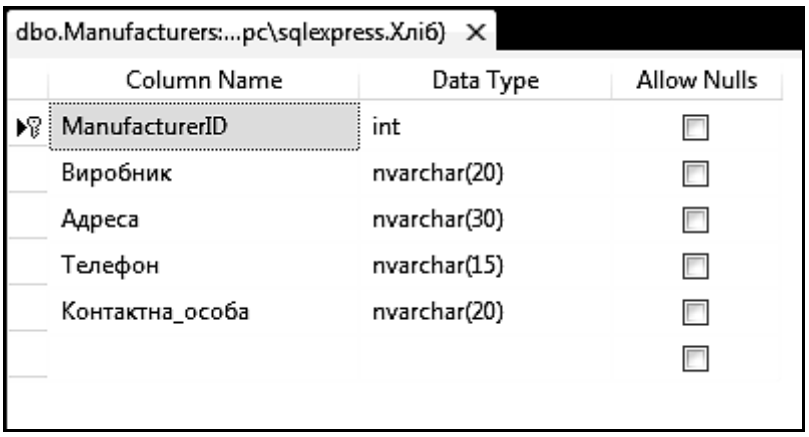


Рис. 8.18. **Властивості полів таблиці *Manufacturers* після налаштування**

Завдання 3

Налаштувати властивості сутності ***Sale***, щоб вони відповідали властивостям полів таблиці ***Продажі*** у попередніх проектах. Тобто властивості повинні мати такі атрибути:

Властивість	Атрибути
SaleID	Ключ
Дата	Обов'язкова, відображається у форматі dd.MM.yyyy
ProductID	Зовнішній ключ для зв'язку із сутністю Product
ManufacturerID	Зовнішній ключ для зв'язку із сутністю Manufacturer
Кількість	Обов'язкова, ціла, змінюється у межах від 0 до 10 000

А також до сутності ***Продажі*** додати обчислювану властивість ***Вартість***, що визначається за формулою

Вартість = Ціна × Кількість,

але в базі даних не зберігається.

Ідея розв'язку

Встановлення атрибутів властивостей сутності виконується аналогічно до описаного для сутності ***Product*** та ***Manufacturer***.

Оскільки імена властивостей ***ProductID*** та ***ManufacturerID*** складається з імен сутностей, з якими встановлюється зв'язок, і символів ***ID***, тому за домовленостями CodeFirst вона є зовнішніми ключами і використання анотації [ForeignKey("Product")] чи [ForeignKey("Manufacturer")] у цьому разі є надлишковим.

Щоб властивість ***Вартість*** не зберігалася в базі даних, треба використати анотацію [NotMapped].

Виконання

1. Відкрийте визначення полів таблиці ***Sales***, двічі клацнувши на її значку у вікні **Server Explorer**. На рис. 8.19 подано властивості полів таблиці ***Sales***.

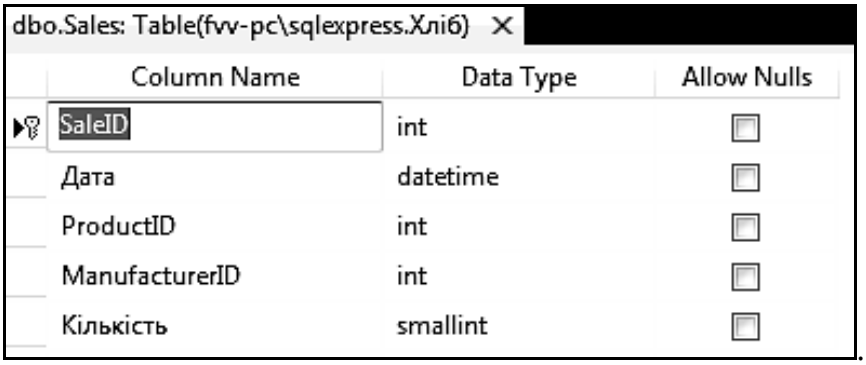


Рис. 8.19. **Властивості полів таблиці *Sales***

2. Щоб змінити властивості полів таблиці, відкрийте код класу ***Sale*** і вставте анотації перед полями класу, а також додайте опис властивості ***Вартість***. Код класу має такий вигляд:

```
public class Sale
{
    public int SaleID { get; set; }
    [Required]
    [DisplayFormat(DataFormatString = "{0:dd.MM.yyyy}",
```

```
        ApplyFormatInEditMode = true)]
    public DateTime Дата { get; set; }
    public int ProductID { get; set; }
    public int ManufacturerID { get; set; }
    [Required]
    public Int16 Кількість { get; set; }

    // Властивості навігації
    public virtual Product Product { get; set; }
    public virtual Manufacturer Manufacturer { get; set; }

    // Обчислювана властивість
    [NotMapped]
    public decimal Вартість
    {
        get
        {
            if (Product != null)
                return (Decimal)(Product.Ціна * Кількість);
            else
                return 0;
        }
    }
}
```

Щоб забезпечити доступ до опису класів, додайте посилання на простір імен

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
```

3. Запустіть програму на виконання і дочекайтеся, поки у вікні консолі з'явиться повідомлення. Обновіть вікно **Server Explorer** і перегляньте властивості полів таблиці ***Sales*** (рис. 8.20).

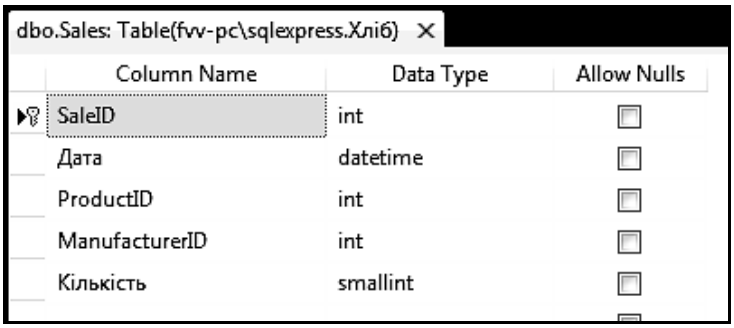


Рис. 8.20. **Властивості полів таблиці *Sales* після налаштування**

4. Закрийте вікно проекту із збереженням зроблених змін.

1.7. Використання Fluent API для налаштування сутностей

Завдання

Налаштувати властивості сутностей ***Product***, ***Manufacturer*** та ***Sale***, щоб вони відповідали властивостям полів таблиць ***Products***, ***Manufacturers*** та ***Sales*** як під час застосування Data Annotation (див. п. 1.6).

Ідея розв'язку

На відміну від Data Annotation застосування Fluent API дозволяє налаштувати сутності, залишивши незмінним код моделі. При цьому використовують класи конфігурацій EntityTypeConfiguration, в екземплярах яких вказують зміни сутностей. Ці зміни використовуються будівником моделі класу DbModelBuilder, який викликається перед побудовою моделі в оброблювачі події OnModelCreating.

Отже, для кожної сутності слід створити свій клас конфігурації, а потім додати їх до колекції конфігурацій будівника моделі в оброблювачі події OnModelCreating, залишивши самі моделі без змін. Налаштування сутностей виконується під час формування об'єкта класу DbContext, тому класи конфігурацій треба помістити у папку ***DataAccess***.

Щоб простежити за результатами налаштувань, створимо нову базу даних ***ХлібFluent***.

Виконання

1. Скопіюйте папку, в якій міститься рішення ***codeFirstХліб*** і новій папці дайте ім'я рішення ***codeFirstХліб_Fluent_API***.

2. Відкрийте рішення ***codeFirstХліб***, що міститься у папці ***codeFirstХліб_Fluent_API***.

3. Додайте у папку ***DataAccess*** проекту ***моделювання*** новий клас ***ProductConfiguration*** і введіть його опис.

```
using System.Data.Entity.ModelConfiguration;
using моделювання.Models;

// Для DatabaseGeneratedOption.Identity
using System.ComponentModel.DataAnnotations.Schema;

namespace моделювання.DataAccess
{
    class ProductConfiguration : EntityTypeConfiguration<Product>
```

```
    {
        public ProductConfiguration()
        {
            ToTable("Products");

            HasKey(p => p.ProductID);

            Property(p => p.ProductID)
               .IsRequired()
               .HasDatabaseGeneratedOption((DatabaseGeneratedOption
                  .Identity));

            Property(p => p.Товар)
                .IsRequired().HasMaxLength(25);

            Property(p => p.Ціна)
                .IsOptional().HasPrecision(5, 2);

            Property(p => p.Ціна_закупівлі)
                .IsRequired().HasPrecision(5, 2);

    // Дослідіть, виклик яких методів можна пропустити
    //  (їхня дія передбачена домовленостями в Code First)
        }
    }
}
```

4. Додайте у папку ***DataAccess*** проекту ***моделювання*** новий клас ***ManufacturerConfiguration*** і введіть його опис.

```
using System.Data.Entity.ModelConfiguration;
using моделювання.Models;

namespace моделювання.DataAccess
{
    class ManufacturerConfiguration :
        EntityTypeConfiguration<Manufacturer>
    {
        public ManufacturerConfiguration()
        {
            Property(m => m.Виробник)
                .IsRequired().HasMaxLength(20);

            Property(m => m.Адреса)
                .IsRequired().HasColumnName("Адреса")
                .HasMaxLength(30);

            Property(m => m.Телефон)
                .IsRequired().HasColumnType("nvarchar")
```

```
                .HasMaxLength(15);

            Property(m => m.Контактна_особа)
                .IsRequired().HasMaxLength(20);

    // Дослідіть, виклик яких методів можна пропустити
    //  (їхня дія передбачена домовленостями в Code First)
        }
    }
}
```

5. Додайте в кінець опису класу ***Sale*** обчислювану властивість ***Вартість***. Після цього опис класу ***Sale*** матиме такий вигляд:

```
public class Sale
{
    public int SaleID { get; set; }
    public DateTime Дата { get; set; }
    public int ProductID { get; set; }

    public int ManufacturerID { get; set; }
    public Int16 Кількість { get; set; }

    // Властивостінавігації
    public virtual Product Product { get; set; }
    public virtual Manufacturer Manufacturer { get; set; }

    // Обчислюванавластивість
    public decimal Вартість
    {
        get
        {
            if (Product != null)
                return (Decimal)(Product.Ціна * Кількість);
            else
                 return 0;
        }
    }
}
```

6. Додайте у папку ***DataAccess*** проекту ***моделювання*** новий клас ***SaleConfiguration*** і введіть його опис

```
using System.Data.Entity.ModelConfiguration;
using моделювання.Models;

namespace моделювання.DataAccess
{
    class SaleConfiguration : EntityTypeConfiguration<Sale>
```

```
    {
        public SaleConfiguration()
        {
            // Ключ
            HasKey(s => s.SaleID);

            // Властивості
            Property(s => s.SaleID)
                .HasColumnType("int")
                .IsRequired();
            Property(s => s.Дата)
                .HasColumnType("date").IsRequired();
            Property(s => s.ProductID)
                .IsRequired();
            Property(s => s.ManufacturerID)
                .IsRequired();
            Property(s => s.Кількість)
                .HasColumnType("smallint")
                .IsRequired();
            // Не відображати в БД
            Ignore(s => s.Вартість);

            // Зовнішні ключі
            HasRequired(s => s.Product)
               .WithMany(s => s.Sales)
               .HasForeignKey(s => s.ProductID)
               .WillCascadeOnDelete(true);
            HasRequired(s => s.Manufacturer)
                .WithMany(s => s.Sales)
                .HasForeignKey(s => s.ManufacturerID)
                .WillCascadeOnDelete(true);

    // Дослідіть, виклик яких методів можна пропустити
    //  (їхня дія передбачена домовленостями в Code First)
        }
    }
}
```

7. Додайте до класу ***BreadContext*** опис методу ***OnModelCreating***, в якому додаються об'єкти конфігурацій до колекції будівника моделі. Клас ***ХлібContext*** набуде такого вигляду:

```
using System.Data.Entity;
using моделювання.Models;

namespace моделювання.DataAccess
{
```

```
    public class BreadContext : DbContext
    {
        public DbSet<Product> Products { get; set; }
        public DbSet<Manufacturer> Manufacturers { get; set; }
        public DbSet<Sale> Sales { get; set; }

        protected override void OnModelCreating
            (DbModelBuilder modelBuilder)
        {
            // Entity Type Configuration
            modelBuilder.Configurations.Add(new
                ProductConfiguration());
            modelBuilder.Configurations.Add(new
                ManufacturerConfiguration());
            modelBuilder.Configurations.Add(new SaleConfiguration());
        }
    }
}
```

8. Відкрийте файл **App.config** і в тезі **add** *з* ім'ям **BreadContext**, що знаходиться в розділі ***connectionStrings***, змініть назву бази даних на ***ХлібFluent*** (параметр **Initial Catalog**). Тепер цей розділ має такий вигляд:

```
<connectionStrings>
<add name="BreadContext" connectionString="Data Source=.\sqlexpress;
Initial Catalog=ХлібFluent ; Integrated Security=True;"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

9. Запустіть програму на виконання і дочекайтеся, поки у вікні консолі з'явиться повідомлення. Обновіть вікно **Server Explorer** і перегляньте властивості полів усіх таблиць у базі даних ***ХлібFluent***.

10. Порівняйте способи налаштування сутностей за допомогою Data Annotation та Fluent API.

11. Закрийте вікно рішення ***codeFirstХліб***, що знаходиться у папці ***codeFirstХліб_Fluent_API*** зі збереженням зроблених змін.

2. Розвиток моделі (міграції)

2.1. Включення міграцій

Завдання

Додати засоби розвитку проекту з можливостями відкочування до певної версії.

Ідея розв'язку

Засоби Data Annotation та Fluent API дозволяють тільки налаштувати сутності. Застосовуючи описані засоби, спочатку видалялася база даних і замість неї створювалася нова. Такі самі кардинальні операції з базою даних виконуються в разі, коли потрібно додавати нові сутності або змінювати вже існуючі. При цьому значно ускладняється можливість повернутися до однієї з попередніх версій моделі і відповідно бази даних. Для вирішення цієї проблеми застосовують міграції.

Для опанування засобами міграцій слід використати рішення, що знаходиться у папці ***codeFirstХліб_Data_Annotation***, хоча з тим самим успіхом можна було б скористатися рішенням, що міститься у папці ***codeFirstХліб_Fluent_API***.

Виконання

1. Скопіюйте папку ***codeFirstХліб_Data_Annotation*** і новій папці дайте ім'я ***codeFirstХліб_Migration***, щоб під час захисту лабораторної роботи можна було продемонструвати попередній варіант проекту і подальший його розвиток.

2. Відкрийте проект ***моделювання***, що міститься у папці ***codeFirstХліб_Migration***.

3. Відкрийте вікно **Package Manager Console** за допомогою однойменної команди, що міститься у списку команди **View – OtherWindows** (рис. 8.21).

4. Введіть команду

Enable-Migrations

у вікні **Package Manager Console** і натисніть клавішу **Enter**.

У вікні **Solution Explorer** з‘явилася папка **Migrations** з двома файлами: ***xxx_InitialCreate.cs*** та ***Configuration.cs***. Причому останній відкрито. Він описує клас конфігурації і надає можливості налаштування її поведінки. У цьому класі також міститься метод ***Seed***, який викликається під час виконання кожної міграції для заповнення таблиць бази даних.

5. Відкрийте файл ***xxx_InitialCreate.cs***, ознайомтеся з його вмістом і визначте призначення методів ***Up***та ***Down*** класу ***InitialCreate***.

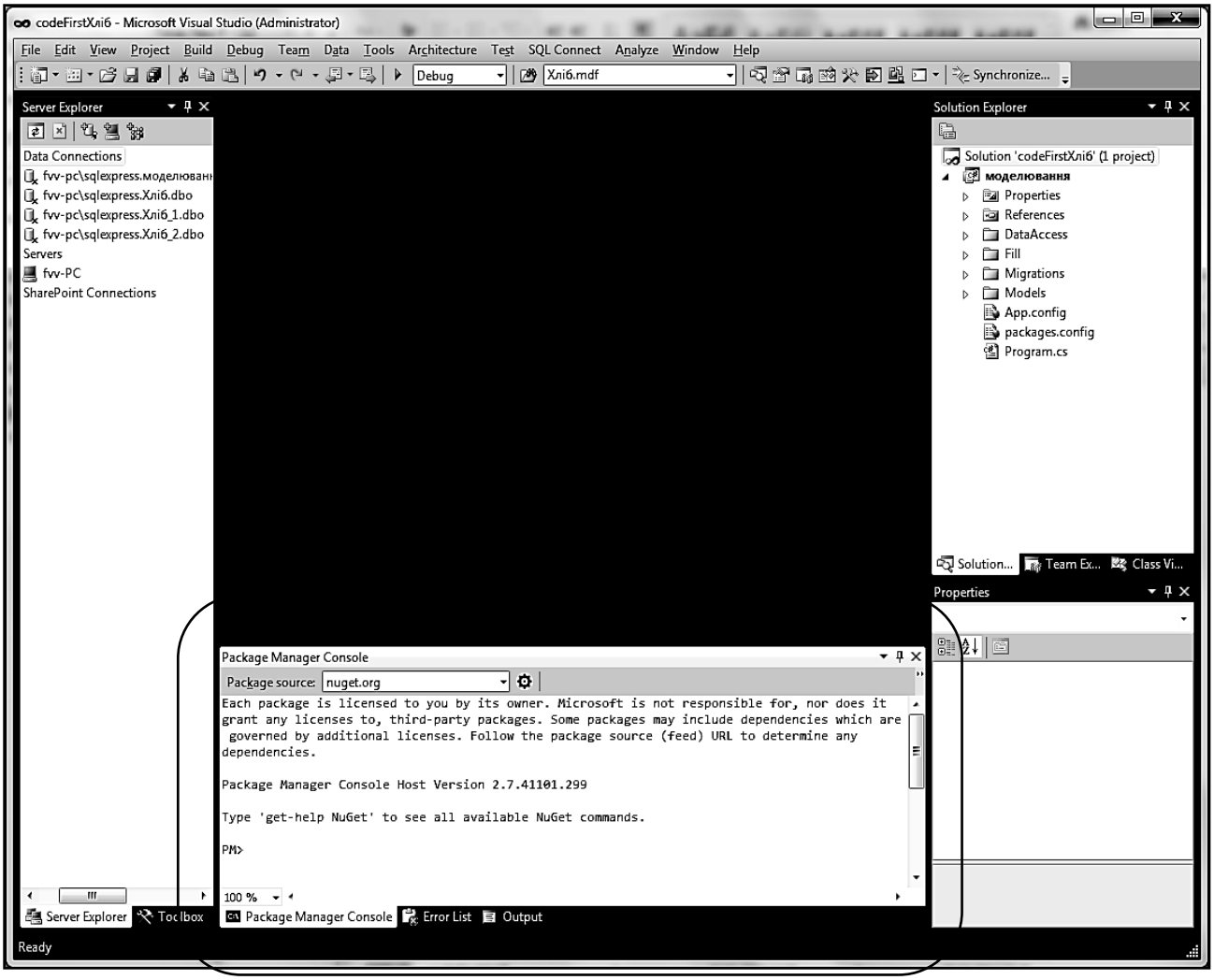


Рис. 8.21. **Вікно *Package Manager Console* в середовищі Visual Studio**

2.2. Формування методу Seed

Завдання

Забезпечити можливість заповнення даними таблиць ***Products***, ***Manufacturers*** та ***Sales***.

Ідея розв'язку

Оскільки під час міграцій база даних не створюється повторно, а метод **Seed** викликається кожний раз під час виконання міграції, то до таблиць можуть повторно додаватися ті самі дані. Щоб запобігти цьому потрібно перевіряти дані, що додаються, на збіг із тими, що вже зберігаються у таблицях.

Найпростіше це робити у довідниках. Там перевірку можна виконувати разом із додаванням до колекції сутностей, які потрібно зберегти у базі даних. Для цього використовують метод ***AddOrUpdate***. У ньому перший параметр вказує назву властивості сутності, за якою виконується перевірка на неповторюваність. Наприклад, для сутності ***Product*** можна застосувати такий алгоритм формування колекції нових даних:

```
products.ForEach(p => context.Products.AddOrUpdate(t => t.Товар, p)).
```

У ньому новими вважаються дані, що мають оригінальну назву товару.

Якщо ж перевірку потрібно виконати за даними кількох властивостей, алгоритм ускладнюється. Наприклад, для сутності ***Sale*** можна застосувати такий алгоритм формування колекції нових даних:

```
foreach (Sale s in sales)
{
    var saleInDB = context.Sales.Where(
        p => p.ProductID == s.ProductID &&
        p.ManufacturerID == s.ManufacturerID &&
        p.Дата == s.Дата).SingleOrDefault();

    if (saleInDB == null)
    {
        context.Sales.Add(s);
    }
}
```

Виконання

1. Відкрийте файл ***DataAdd.cs***, що знаходиться у папці ***Fill*** і скопіюйте з нього тіло методу ***FillDB***, а потім вставте його як тіло методу ***Seed*** у класі ***Configuration***.

2. Відкоригуйте тіло методу ***Seed***, щоб позбавитися дублювання даних під час формування колекції нових даних. Після цього метод ***Seed*** матиме такий вигляд:

```
protected override void Seed(моделювання.DataAccess.BreadContext context)
{
    var products = new List<Product>
    {
         new Product
             { Товар = @"Хліб ""Український""",
```

```
            Ціна = 3.00M,
            Ціна_закупівлі =2.50M },
        new Product
            { Товар = @"Батон ""Молочний""",
            Ціна = 2.80M,
            Ціна_закупівлі =2.50M},
        new Product
            { Товар = @"Булказмаком",
            Ціна = 1.98M,
            Ціна_закупівлі =1.85M}
};

// Формуємо колекцію нових товарів без повторювань
products.ForEach(p => context.Products.AddOrUpdate(t =>
    t.Товар, p));
context.SaveChanges();
var manufacturers = new List<Manufacturer>
{
    new Manufacturer
        { Виробник = @"Х/з ""Салтівський""",
          Адреса = "вул. Гв. Широнінців, 1",
          Телефон ="(057)710-50-40",
          Контактна_особа="Іванов І. І." },
    new Manufacturer
        { Виробник = @"Х /з ""Кулиничі""",
          Адреса = "смт Кулиничі, вул. Шкільна, 18",
          Телефон ="(0572)62-51-37",
          Контактна_особа="Петренко П. П." }
};

// Формуємо колекцію нових виробників без повторювань
manufacturers.ForEach(m => context.Manufacturers.AddOrUpdate
    (v => v.Виробник, m));
context.SaveChanges();

var sales = new List<Sale>
{
    new Sale
        { Дата = DateTime.Parse("01.09.2014"),
          ProductID = 1,
          ManufacturerID =1,
          Кількість=200 },
    new Sale
        { Дата = DateTime.Parse("01.09.2014"),
          ProductID = 2,
          ManufacturerID =1,
```

```
            Кількість=150 },
        new Sale
            { Дата = DateTime.Parse("01.09.2014"),
              ProductID = 1,
              ManufacturerID =2,
              Кількість=180 }
    };

    // Формуємо колекцію нових даних про продажі без повторювань
    foreach (Sale s in sales)
    {
        var saleInDB = context.Sales.Where(
            p => p.ProductID == s.ProductID &&
            p.ManufacturerID == s.ManufacturerID &&
            p.Дата == s.Дата).SingleOrDefault();
        if (saleInDB == null)
        {
            context.Sales.Add(s);
        }
    }
    context.SaveChanges();

}
```

Щоб забезпечити доступ до опису класів, додайте посилання на простори імен

```
using System.Collections.Generic;
using моделювання.Models;
```

3. Щоб перевірити, чи метод ***Seed*** може додавати нові дані без повторювань рядків, спочатку видалимо усі таблиці з бази, потім створимо їх знову і додамо в них дані. Після цього спробуємо повторно додати ті самі дані. Все це зробимо за допомогою міграцій. Для цього виконайте таке:

4.1. Зробіть відкат до порожньої бази даних за допомогою команди

Update-Database –TargetMigration: $InitialDatabase

Перевірте, що в базі даних не залишилося жодної таблиці.

4.2. Виконайте всі відкладені міграції (в даному разі ***InitialDatabase***), щоб повернутися до того стану, в якому була база даних до відкату. Для цього виконайте команду

Update-Database

Зверніть увагу на повідомлення про те, що під час виконання міграції InitialCreate викликався метод Seed.

Перевірте, що в базі даних з'явилися всі заповнені таблиці.

4.3.Додайте порожню міграцію, виконання якої викликатиме тільки метод Seed. Для цього виконайте команду

Add-Migration TestSeed

4.4. Виконайте останню міграцію за допомогою команди

Update-Database

Зверніть увагу на повідомлення про те, що під час виконання міграції TestSeed викликався метод Seed. Перевірте вміст таблиць бази даних. Повторне виконання методу Seed не призвело до появи однакових рядків. Отже, метод Seed працює правильно. Після виконання нової міграції він робить спробу додати в таблиці бази даних тільки ті дані, яких ще там не було.

2.3. Додавання нової сутності Invoice

Завдання

Додати до моделі даних нову сутність ***Invoice***. У базі даних вона реалізується у вигляді таблиці ***Invoices***, що відповідає таблиці Накладні у попередніх роботах.

Ідея розв'язку

У папку ***Models*** додати новий клас ***Invoice***, а до класу ***BreadContext*** – інформацію про відповідний об'єкт класу DbSet. На основі цієї інформації Code First створить таблицю ***Invoices***.

Для виконання операції створення додати два рядки до таблиці ***Invoices***. Причому у першому рядку задамо значення усіх полів, а в другому – пропустити значення поля ***Номер_накладної***, щоб перевірити реакцію на значення Null. У подальшому це знадобиться для демонстрування інших операцій у міграціях.

Щоб не зачіпати об'єкти бази даних, що були створені раніше, для додавання нової сутності слід скористатися механізмом міграцій. Тому додавання рядків до таблиці ***Invoices*** вказати в методі ***Seed***.

Виконання

1. Додайте у папку ***Models*** новий клас ***Invoice*** і введіть його опис

```
public class Invoice
```

```
{
    public int InvoiceID { get; set; }
    [MaxLength(6)]
    public string Номер_накладної { get; set; }
    public DateTime Дата { get; set; }
    public int ManufacturerID { get; set; }

    // Властивості навігації
    public virtual Manufacturer Manufacturer { get; set; }
}
```

2. Додайте в клас ***BreadContext***, що знаходиться у папці ***Data-Access***, властивість колекції нових сутностей, розташувавши її у кінці класу.

```
public DbSet<Invoice> Invoices { get; set; }
```

3. Додайте до методу ***Seed***, що знаходиться в класі ***Configuration***, код для додавання двох рядків до таблиці ***Invoices***, розташувавши його у кінці методу.

```
    //***********
    // Inovices *
    //***********
    // Нові сутності
    var invoices = new List<Invoice>
    {
        new Invoice
        { Номер_накладної ="11",
          Дата=DateTime.Parse("01.09.2014"),
          ManufacturerID=1 },
        new Invoice
        { Дата=DateTime.Parse("01.09.2014"),
          ManufacturerID=2 }
    };

    // Перевіряємо рядки на дублювання накладних
    foreach (Invoice i in invoices)
    {
        var invoiceInDB = context.Invoices.Where(
            n => n.Дата == i.Дата &&
            n.ManufacturerID == i.ManufacturerID
            ).FirstOrDefault();
        if (invoiceInDB == null)
        {
            context.Invoices.Add(i);
```

```
        }
    }

    context.SaveChanges();
```

4. Щоб створити шаблон для міграції, яка реалізує додавання нової сутності, введіть команду

Add-Migration AddInvoice

у вікні **Package Manager Console**.

У папці ***Migrations*** з'явився новий клас ***AddInvoice***. Він містить метод ***Up*** для створення таблиці ***Invoices***, а також метод ***Down*** для її видалення.

5. Щоб виконати міграцію і додати дані в нову таблицю, введіть команду

Update-Database –Verbose

у вікні **Package Manager Console**. Ключ **Verbose** забезпечує виведення відповідних SQL-команд.

6. Перейдіть у вікно **Server Explorer** і перегляньте схему таблиці ***Invoices*** та дані, що зберігаються у ній (рис. 8.22, 8.23).

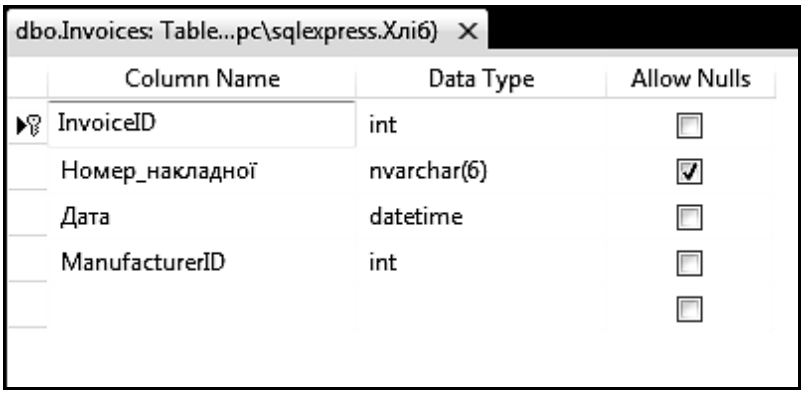


Рис. 8.22. **Схема таблиці *Invoices***

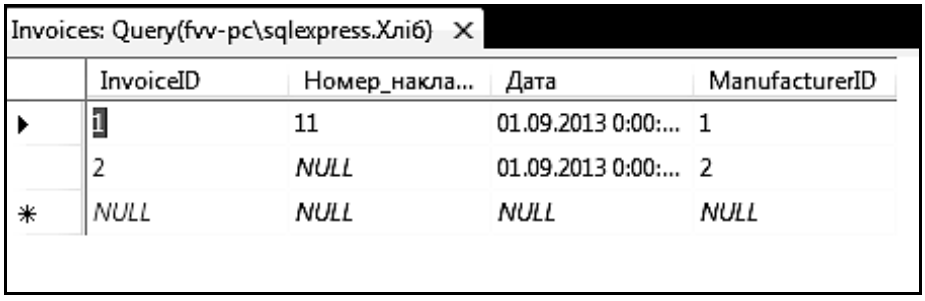


Рис. 8.23. **Дані таблиці *Invoices***

2.4. Змінення атрибутів властивості. Використання SQL користувача

Завдання

Замінити порожні значення поля ***Номер_накладної*** у всіх рядках таблиці ***Invoices*** на значення, що обчислюються за таким виразом

ID + "–" + YYYYMMDD,

використавши значення полів того самого рядка. У виразі прийнято такі скорочення:

ID – значення поля ***ManufacturerID***;

YYYYMMDD – рік, місяць і день у полі ***Дата***.

Зазначені величини розділяються символом "мінус".

Ідея розв'язку

Заміну значень можна виконати за допомогою запиту SQL, що містить такий оператор UPDATE:

```
UPDATE Invoices SET Номер_накладної =CONVERT(varchar(3),
ManufacturerID) + '-'  + CONVERT(varchar(8), Дата, 112)  WHERE Но-
мер_накладної IS NULL
```

В операторі використано функцію CONVERT (varchar(3), ManufacturerID) з огляду на те, що кількість виробників, що постачають товари в кіоск менша 1000, чого більше ніж достатньо.

Проаналізувавши вираз, можна зробити висновок, що максимальна довжина результату може досягати 12 символів (3 символи для ***ID***, 1 символ для "мінусу" і 8 символів для дати). Тому в міграції перед викликом SQL-запиту слід змінити розмір поля ***Номер_накладної***.

Щоб міграція могла мати зворотну дію, потрібно в методі ***Down*** вставити запит

```
UPDATE Invoices SET Номер_накладної = NULL WHERE Но-
мер_накладної=CONVERT(varchar(3), ManufacturerID) + '-'  +
CONVERT(varchar(8), Дата, 112)
```

який повертає попереднє значення поля ***Номер_накладної***.

Виконання

1. Відкрийте файл ***Invoice.cs***, що знаходиться у папці ***Models***.

2. Змініть анотацію даних, що розташована перед властивістю **Номер_накладної** з***MaxLength(6)*** на ***MaxLength(12)***.

```
public class Invoice
{
    public int InvoiceID { get; set; }
    [MaxLength(12)]
    public string Номер_накладної { get; set; }
    public DateTime Дата { get; set; }
    public int ManufacturerID { get; set; }

    // Властивості навігації
    public virtual Manufacturer Manufacturer { get; set; }
}
```

3. Щоб створити шаблон для міграції, яка реалізує змінення поля **Номер_накладної**, введіть у вікні **Package Manager Console** команду

Add-Migration ChangeInvoiceNumber

У папці ***Migrations*** з'явився новий клас ***ChangeInvoiceNumber***. Він містить метод ***Up*** для змінення властивості поля **Номер_накладної** у таблиці ***Invoices***, а також метод ***Down*** для відновлення попереднього значення цієї властивості.

4. Щоб замінити порожні значення поля **Номер_накладної** у всіх рядках таблиці ***Invoices***, додайте звертання до методу ***SQL***, розмістивши його в кінці методу ***Up*** класу ***ChangeInovoiceNumber***. Метод ***Up*** набуде такого вигляду:

```
public override void Up()
{
    AlterColumn("dbo.Invoices", "Номер_накладної", c =>
c.String(maxLength: 12));
    Sql("UPDATE Invoices SET Номер_накладної =CONVERT(varchar(3),
ManufacturerID) +         '-'  + CONVERT(varchar(8), Дата, 112)  WHERE
Номер_накладної IS NULL");
}
```

Зворотна дія забезпечується методом ***Down***, який після додавання запиту набуде такого вигляду:

```
public override void Down()
{
    Sql("UPDATE Invoices SET Номер_накладної = NULL WHERE
       Номер_накладної=CONVERT(varchar(3), ManufacturerID) + '-'
```

```
        +CONVERT(varchar(8), Дата, 112)");
    AlterColumn("dbo.Invoices", "Номер_накладної", c =>
        c.String(maxLength: 6));
}
```

5. Щоб виконати міграцію і додати дані у нову таблицю, введіть у вікні **Package Manager Console** команду

Update-Database –Verbose

6. Перейдіть у вікно **Server Explorer** і перегляньте схему таблиці ***Invoices*** та дані, що зберігаються у ній (рис. 8.24, 8.25).

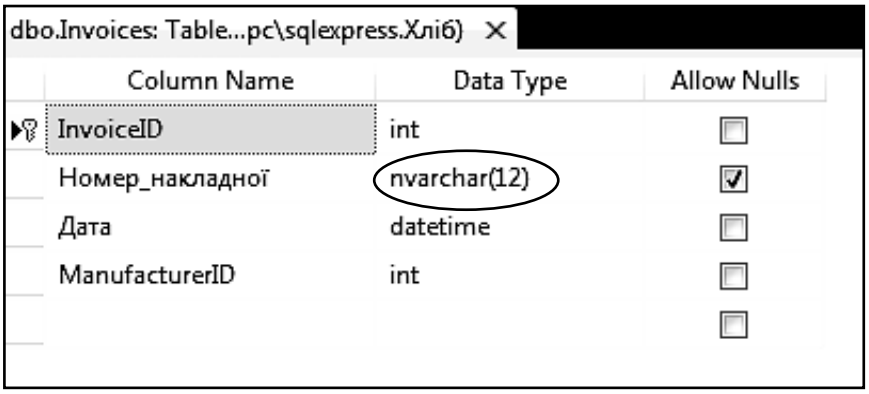


Рис. 8.24. **Схема таблиці *Invoices* зі зміненою властивістю поля Номер_накладної**

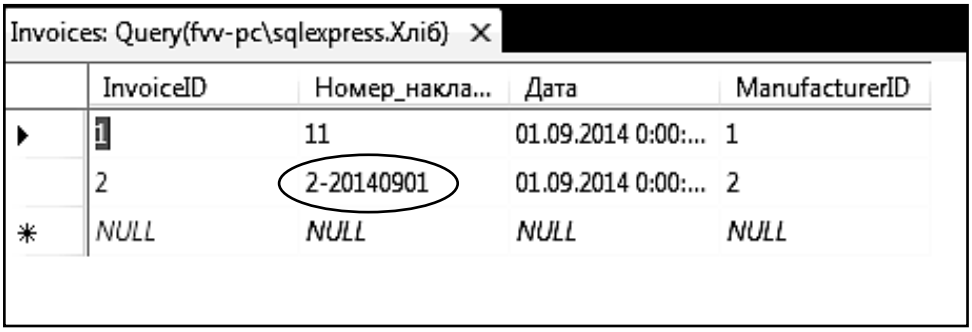


Рис. 8.25. **Дані таблиці *Invoices* зі зміненим значенням поля *Номер_накладної***

2.5. Додавання дочірньої сутності

Завдання

Додати до моделі даних нову сутність ***InvoiceProduct,*** що є дочірньою до сутності ***Invoice***. У базі даних вона реалізується у вигляді таб-

лиці ***InvoiceProducts***, що відповідає таблиці Товари_накладних у попередніх роботах.

Ідея розв'язку

У папку ***Models*** слід додати новий клас ***InvoiceProduct***, до класів ***Invoice*** та ***Product*** – властивості навігації, а до класу ***BreadContext*** – інформацію про відповідний об'єкт класу **DbSet**. На основі цієї інформації Code First створить таблицю ***InvoiceProducts*** і встановить зв'язок з таблицями ***Invoices*** та ***Products***.

Для виконання операції створення таблиці ***InvoiceProducts*** додати дані про товари до накладної, що отримані за накладною з номером ***11*** від виробника з кодом ***1*** (сама накладна вже міститься в таблиці ***Invoices***), а також нову накладну з товарами в ній. Для цього використаємо метод ***Seed***.

Виконання

1. Додайте у папку ***Models*** новий клас ***InvoiceProduct*** і введіть його опис

```
public class InvoiceProduct
{
    public int InvoiceProductID { get; set; }
    public int InvoiceID { get; set; }
    public int ProductID { get; set; }
    public Int16 Кількість { get; set; }
    // Властивості навігації
    public virtual Product Product { get; set; }
    public virtual Invoice Invoice { get; set; }

    // Обчислювана властивість
    [NotMapped]
    public decimal Вартість
    {
        get
        {
            if (Product != null)
                return (Decimal)(Product.Ціна * Кількість);
            else
                return 0;
        }
    }
}
```

2. Додайте до класу ***Invoice*** властивість навігації

```
public virtual ICollection<InvoiceProduct> InvoiceProducts {get; set;}
```

3. Повторіть п. 2 для класу ***Product***.

4. Додайте в клас ***BreadContext***, що знаходиться у папці ***Data-Access***, властивість колекції нових сутностей, розташувавши її в кінці класу.

```
public DbSet<InvoiceProduct> InvoiceProducts { get; set; }
```

5. Додайте до методу ***Seed***, що знаходиться в класі ***Configuration***, код для додавання даних про товари до накладних, що вже містяться в таблиці ***Invoices***, а також нову накладну з товарами в ній, розташувавши його у кінці методу.

```
// 1. Додавання товарів до накладної, що отримані за накладною
// з номером 11 від виробника з кодом 1

// Дізнаємося InoviceID накладної, до якої потрібно додати товари
int inoviceID = context.Invoices.Where(
    n => n.Номер_накладної == "11" && n.ManufacturerID == 1
    ).SingleOrDefault().InvoiceID;
// Додаємотовари
var invoiceProducts = new List<InvoiceProduct>
{
// ТоваринакладнихчерезїхніID
    new InvoiceProduct
    { InvoiceID=inoviceID,
      ProductID=1, Кількість=210},
    new InvoiceProduct
    { InvoiceID=inoviceID,
      ProductID=2, Кількість=155},
};

// Перевіряємо рядки на дублювання товарів в накладній
foreach (InvoiceProduct ip in invoiceProducts)
{
    var invoiceProductInDB = context.InvoiceProducts.Where(
        nt => nt.InvoiceID == ip.InvoiceID &&
        nt.ProductID == ip.ProductID).FirstOrDefault();

    if (invoiceProductInDB == null)
    {
        context.InvoiceProducts.Add(ip);
    }
}
context.SaveChanges();
```

```
//2. Додавання товарів до нової накладної, що отримані
// від виробника з кодом 2.

invoices = new List<Invoice>
{
    new Invoice
    { Номер_накладної="21",
      Дата=DateTime.Parse("02.09.2014"),
      ManufacturerID=2,
      InvoiceProducts=new List<InvoiceProduct>
    // Товари накладних через їхні назви (LINQ)
    {
        new InvoiceProduct
        { ProductID=products.Single( p => p.Товар ==
            @"Хліб ""Український""").ProductID,
            Кількість=190},
        }
    }
};
// Перевіряємо рядки на дублювання накладних
foreach (Invoice i in invoices)
{
    var invoiceInDB = context.Invoices.Where(
        n => n.Дата== i.Дата && n.ManufacturerID == i.ManufacturerID
        ).FirstOrDefault();

    if (invoiceInDB == null)
    {
        context.Invoices.Add(i);
    }
}
context.SaveChanges();
```

6. Щоб створити шаблон для міграції, яка реалізує додавання нової сутності, введіть у вікні **Package Manager Console** команду

Add-Migration AddInvoiceProduct

У папці ***Migrations*** з'явився новий клас ***AddInvoiceProduct***. Він містить метод ***Up*** для створення таблиці ***AddInvoiceProducts***, а також метод ***Down*** для її видалення.

7. Оскільки під час виконання міграції ***AddInvoice_Product*** у методі ***Seed*** додано новий рядок до таблиці ***Invoice***, щоб забезпечити зворотність цієї операції, додайте в кінець методу ***Down*** цієї міграції оператор, що видаляє доданий рядок.

```
Sql("DELETE FROM Invoices WHERE  Номер_накладної= N'21' AND
ManufacturerID=2");
```

8. Щоб виконати міграцію і додати дані у нову таблицю, введіть команду

Update-Database –Verbose

у вікні **Package Manager Console**. Ключ **Verbose** забезпечує виведення відповідних SQL-команд.

9. Перейдіть у вікно **Server Explorer** і перегляньте схему таблиці ***Invoice_Products*** та дані, що зберігаються у ній (рис. 8.26, 8.27).

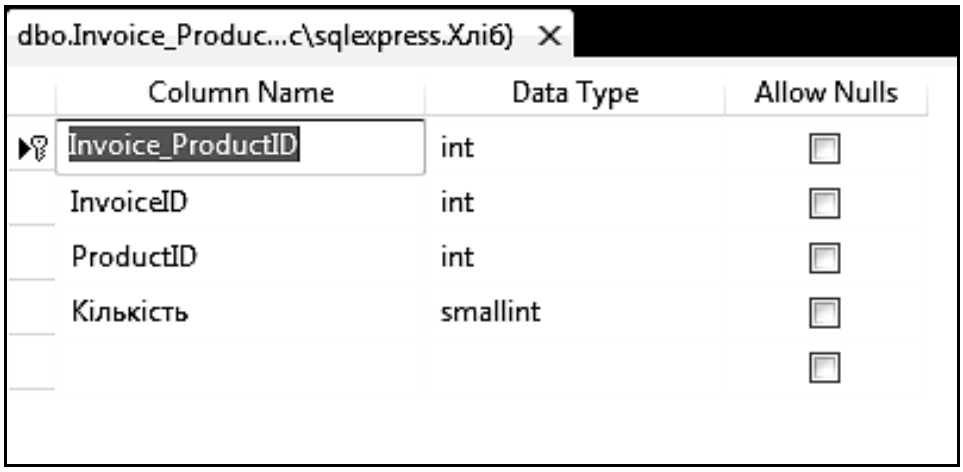


Рис. 8.26**. Схема таблиці *Invoice_Products***

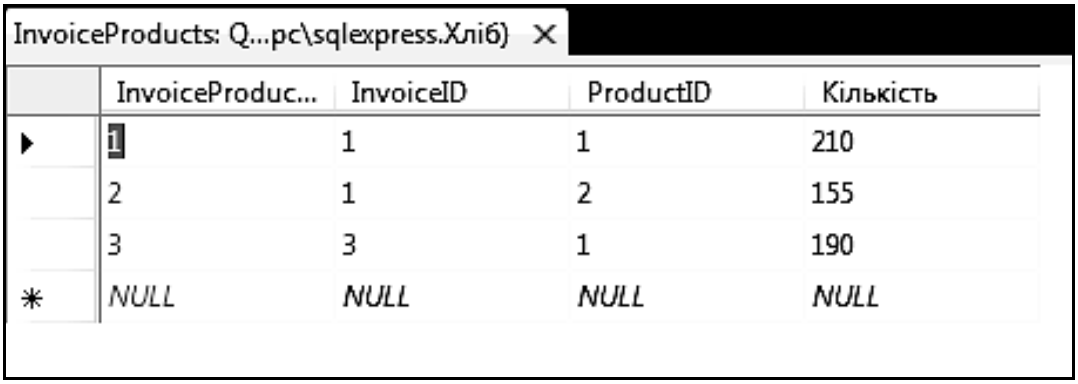


Рис. 8.27. **Дані таблиці *Invoice_Products***

10. З метою закріплення навичок роботи з ієрархічними даними додайте дані про товари до накладної, що отримані за накладною від виробника з кодом 2 за 01.09.2014 (сама накладна вже міститься в таблиці ***Invoices***). Для цього використайте метод ***Seed***.

2.6. Відкочування до заданої міграції

Завдання

Видалити з бази даних таблицю ***InvoiceProducts***, а потім повернутися до останньої міграції.

Ідея розв'язку

Оскільки таблицю ***InvoiceProducts*** додано в базу даних за допомогою міграції ***AddInvoiceProduct***, потрібно відкотити базу даних до стану, який отримано за допомогою попередньої міграції, тобто ***ChangeInvoiceNumber***. Це можна виконати за допомогою команди

Update-Database –TargetMigration: ChangeInvoiceNumber.

Виконання

1. Перейдіть у вікно **Package Manager Console** і введіть команду

Update-Database –TargetMigration: ChangeInvoiceNumber

2. Перейдіть у вікно **Server Explorer**, обновіть таблиці бази даних і впевніться, що з бази даних зникла таблиця ***InvoiceProducts***.
3. Щоб повернутися до останньої міграції, введіть у вікні **Package Manager Console** команду

Update-Database –TargetMigration: AddInvoiceProduct

4. Перейдіть у вікно **Server Explorer**, обновіть таблиці бази даних і впевніться, що в базі даних знову з'явилася таблиця ***InvoiceProducts*** з даними.

2.7. Отримання скрипта SQL

Завдання

Створити SQL-скрипт, за допомогою якого можна побудувати нову БД ***Хліб*** на іншому комп'ютері.

Ідея розв'язку

Code First дозволяє будувати SQL-скрипти, які реалізують зміни в базі даних, що виконуються послідовністю міграцій. Для цього використовують команду **Update-Database** з параметром – **Script**, вказавши початкову та кінцеву міграції за допомогою параметрів – **SourceMigration** та – **TargetMigration** відповідно.

Виконання

1. Перейдіть у вікно **Package Manager Console** і введіть команду

Update–Database –Script –SourceMigration: $InitialDatabase – TargetMigration: AddInvoiceProduct

2. Перейдіть у вікно нового запиту і видаліть всі оператори SQL, що стосуються ***MigrationHistory***. Отримаємо SQL-скрипт, за допомогою якого можна побудувати нову БД ***Хліб***:

```
    CREATE TABLE [dbo].[Manufacturers] (
        [ManufacturerID] [int] NOT NULL IDENTITY,
        [Виробник] [nvarchar](20) NOT NULL,
        [Адреса] [nvarchar](30) NOT NULL,
        [Телефон] [nvarchar](15) NOT NULL,
        [Контактна_особа] [nvarchar](20) NOT NULL,
        CONSTRAINT [PK_dbo.Manufacturers] PRIMARY KEY
([ManufacturerID])
    )
    CREATE TABLE [dbo].[Sales] (
        [SaleID] [int] NOT NULL IDENTITY,
        [Дата] [datetime] NOT NULL,
        [ProductID] [int] NOT NULL,
        [ManufacturerID] [int] NOT NULL,
        [Кількість] [smallint] NOT NULL,
        CONSTRAINT [PK_dbo.Sales] PRIMARY KEY ([SaleID])
    )
    CREATE INDEX [IX_ManufacturerID] ON [dbo].[Sales]([ManufacturerID])
    CREATE INDEX [IX_ProductID] ON [dbo].[Sales]([ProductID])
    CREATE TABLE [dbo].[Products] (
        [ProductID] [int] NOT NULL IDENTITY,
        [Товар] [nvarchar](25) NOT NULL,
        [Ціна] [decimal](18, 2),
        [Ціна_закупівлі] [decimal](18, 2) NOT NULL,
        CONSTRAINT [PK_dbo.Products] PRIMARY KEY ([ProductID])
    )
    ALTER TABLE [dbo].[Sales] ADD CONSTRAINT
[FK_dbo.Sales_dbo.Manufacturers_ManufacturerID] FOREIGN KEY
([ManufacturerID]) REFERENCES [dbo].[Manufacturers] ([ManufacturerID])
ON DELETE CASCADE
    ALTER TABLE [dbo].[Sales] ADD CONSTRAINT
[FK_dbo.Sales_dbo.Products_ProductID] FOREIGN KEY ([ProductID])
```

```
REFERENCES [dbo].[Products] ([ProductID]) ON DELETE CASCADE
    CREATE TABLE [dbo].[Invoices] (
        [InvoiceID] [int] NOT NULL IDENTITY,
        [Номер_накладної] [nvarchar](6),
        [Дата] [datetime] NOT NULL,
        [ManufacturerID] [int] NOT NULL,
        CONSTRAINT [PK_dbo.Invoices] PRIMARY KEY ([InvoiceID])
    )
    CREATE INDEX [IX_ManufacturerID] ON
[dbo].[Invoices]([ManufacturerID])
    ALTER TABLE [dbo].[Invoices] ADD CONSTRAINT
[FK_dbo.Invoices_dbo.Manufacturers_ManufacturerID] FOREIGN KEY
([ManufacturerID]) REFERENCES [dbo].[Manufacturers] ([ManufacturerID])
ON DELETE CASCADE
    ALTER TABLE [dbo].[Invoices] ALTER COLUMN [Номер_накладної]
[nvarchar](12)
    UPDATE Invoices SET Номер_накладної =CONVERT(varchar(3),
ManufacturerID) +         '-'  + CONVERT(varchar(8), Дата, 112)  WHERE
Номер_накладної IS NULL
    CREATE TABLE [dbo].[InvoiceProducts] (
        [InvoiceProductID] [int] NOT NULL IDENTITY,
        [InvoiceID] [int] NOT NULL,
        [ProductID] [int] NOT NULL,
        [Кількість] [smallint] NOT NULL,
        CONSTRAINT [PK_dbo.InvoiceProducts] PRIMARY KEY
([InvoiceProductID])
    )
    CREATE INDEX [IX_InvoiceID] ON [dbo].[InvoiceProducts]([InvoiceID])
    CREATE INDEX [IX_ProductID] ON [dbo].[InvoiceProducts]([ProductID])
    ALTER TABLE [dbo].[InvoiceProducts] ADD CONSTRAINT
[FK_dbo.InvoiceProducts_dbo.Invoices_InvoiceID] FOREIGN KEY
([InvoiceID]) REFERENCES [dbo].[Invoices] ([InvoiceID]) ON DELETE
CASCADE
    ALTER TABLE [dbo].[InvoiceProducts] ADD CONSTRAINT
[FK_dbo.InvoiceProducts_dbo.Products_ProductID] FOREIGN KEY
([ProductID]) REFERENCES [dbo].[Products] ([ProductID]) ON DELETE
CASCADE
```

3. Збережіть SQL-скрипт у файлі ***SqlQuery_Хліб*** у папці рішення.

2.8. Автоматичне оновлення бази даних під час запуску застосування

Завдання

Створити нову базу даних із урахуванням усіх міграцій.

Ідея розв'язку

За допомогою тих міграцій, що містяться в застосуванні, можна створити нову базу даних на рівні останньої міграції. Для цього потрібно задати нове ім'я бази даних у рядку ConnectionString, потім зареєструвати ініціалізатор бази даних **MigrateDatabaseToLatestVersion** і запустити застосування на виконання. При цьому в застосуванні повинна бути хоч одна операція з базою даних, наприклад, читання даних.

Виконання

1. Відкрийте файл **App.config** і замініть значення параметра Initial Catalog у ConnectionString із Хліб***Прізвище*** на Хліб***Прізвище***_1. Потім закрийте файл **App.config** зі збереженням зроблених змін.

2. Відкрийте файл **Program.cs** і в методі ***Main*** закоментуйте всі оператори, що входять до гілки **try**, а під ними вставте такі:

```
// Реєстрування ініціалізатора бази даних
Database.SetInitializer(new
    MigrateDatabaseToLatestVersion<BreadContext, Configuration>());

using (var context = new BreadContext())
{
    foreach (var p in context.Products)
    {
        Console.WriteLine(p.Товар);
    }
}
```

У результаті метод ***Main*** набуде такого вигляду:

```
static void Main(string[] args)
{
    try
    {
        /*
        // Стратегія роботи з базою даних
        Database.SetInitializer(
                new DropCreateDatabaseAlways<BreadContext>());
        int nProduct;
        int nManufacturer;
        int nSale;
        DataAdd.FillDB(out nProduct, out nManufacturer,
        out nSale);
        Console.WriteLine(
            "Базу даних на SQL Server створено і заповнено.\n"
            + "У таблиці записано таку кількість рядків:\n"
```

```
            + "Products - " + nProduct
            + ", Manufacturers - " + nManufacturer
            + ",  Sales - " + nSale
            + ".\n Перевірте!!!");
        */
        // Реєстрування ініціалізатора бази даних
        Database.SetInitializer(new
      MigrateDatabaseToLatestVersion<BreadContext, Configuration>());
        using (var context = new BreadContext())
        {
            foreach (var p in context.Products)
            {
                Console.WriteLine(p.Товар);
            }
        }
    }
    catch (System.Exception ex)
    {
        Console.WriteLine("Базу даних не створено. \n Помилка:\n "
            + ex.ToString());
    }
    Console.WriteLine("Натисніть будь-яку клавішу, щоб вийти...");
    Console.ReadKey();
}
```

3. Щоб забезпечити доступ до опису класів, додайте посилання на простори імен

```
using моделювання.Migrations;
using моделювання.Models;
```

4. Запустіть проект на виконання. Дочекайтеся закінчення операцій з базою даних і завершіть виконання проекту.

5. Знайдіть нову базу даних Хліб***Прізвище***_1 на SQL Server. Ознайомтеся з властивостями стовпців її таблиць і даними, що містяться в таблицях.

6. Закрийте вікно проекту із збереженням зроблених змін.

3. Побудова застосування

3.1. Додавання проекту Windows Forms

Завдання

Додати до рішення ***codeFirstХліб*** новий проект Windows Forms з ім'ям ***winForms*** і створити в ньому кнопкову форму для керування основними функціями проекту.

Виконання

1. Скопіюйте папку ***codeFirstХліб_Migration*** і новій папці дайте ім'я ***codeFirstХліб_App***, щоб під час захисту лабораторної роботи можна було продемонструвати попередній варіант проекту і подальший його розвиток.

2. Відкрийте рішення ***codeFirstХліб***, додайте до нього новий проект Windows Forms з ім'ям ***winForms***.

3. Встановіть покажчик миші на імені нового проекту і з контекстового меню виберіть команду **Set as StartUp Project**.

4. Встановіть покажчик миші на значку **References** нового проекту у вікні **Solution Explorer**, з контекстового меню виберіть команду **Add Reference**, а потім додайте такі посилання (рис. 8.28):

System.Data.Entity (вкладка .NET);

EntityFramework (вкладка **Browse**, шлях \packages\EntityFramework.***x.x.x***\lib\net40 у папці рішення);

EntityFramework.SqlServer (вкладка **Browse**, шлях \packages\EntityFramework.***x.x.x***\lib\net40 у папці рішення);

проект ***моделювання*** (вкладка **Projects**).

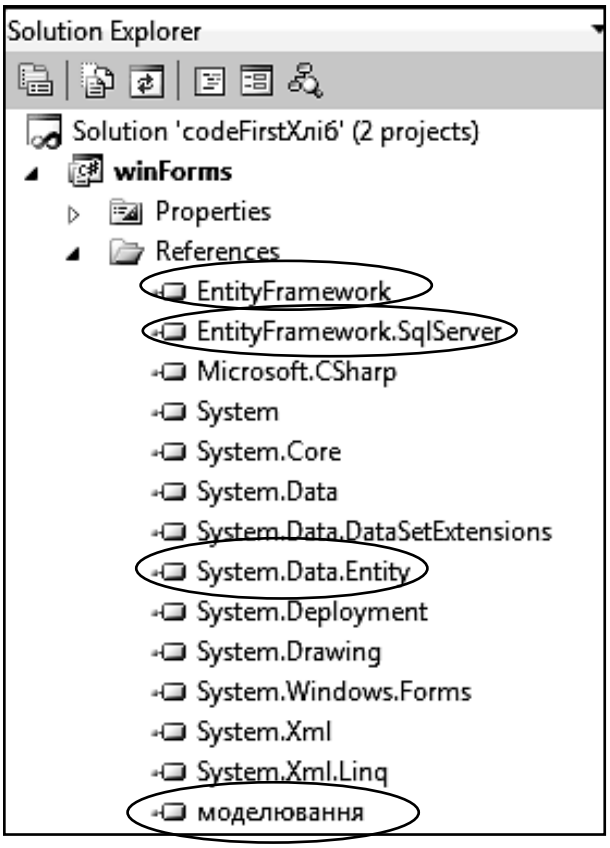


Рис. 8.28. **Нові посилання в проекті**

Примітка. Під час додавання посилань на **EntityFramework** та **EntityFramework.SqlServer** слід вибирати ту саму бібліотеку **EntityFramework.dll** та **EntityFramework.SqlServer.dll**, що й у проекті ***моделювання***. Про шлях до них можна дізнатися з властивості **Path** у папці **References** проекту ***моделювання***.

5. Скопіюйте файл ***App.config*** з проекту ***моделювання*** у ***winForms***. Для цього перетягніть його значок із першого проекту в другий.

6. Перейменуйте файл ***Form1.cs*** у проекті ***winForms*** на ***formХліб.cs*** і для властивості **Text** цієї форми задайте значення ***Хліб***.

7. Додайте елементи керування на форму ***Хліб***, щоб вона набула такого вигляду (рис. 8.29):

При цьому встановіть такі значення властивостей кнопок:

Група	Кнопка	Властивість	Значення
Оперативна інформація	1	Text	Накладні
		Name	buttonНакладні
	2	Text	Продажі
		Name	buttonПродажі
Довідники	1	Text	Товари
		Name	buttonТовари
	2	Text	Виробники
		Name	buttonВиробники
Аналіз	1	Text	Продажі виробника
		Name	buttonПродажі_виробника

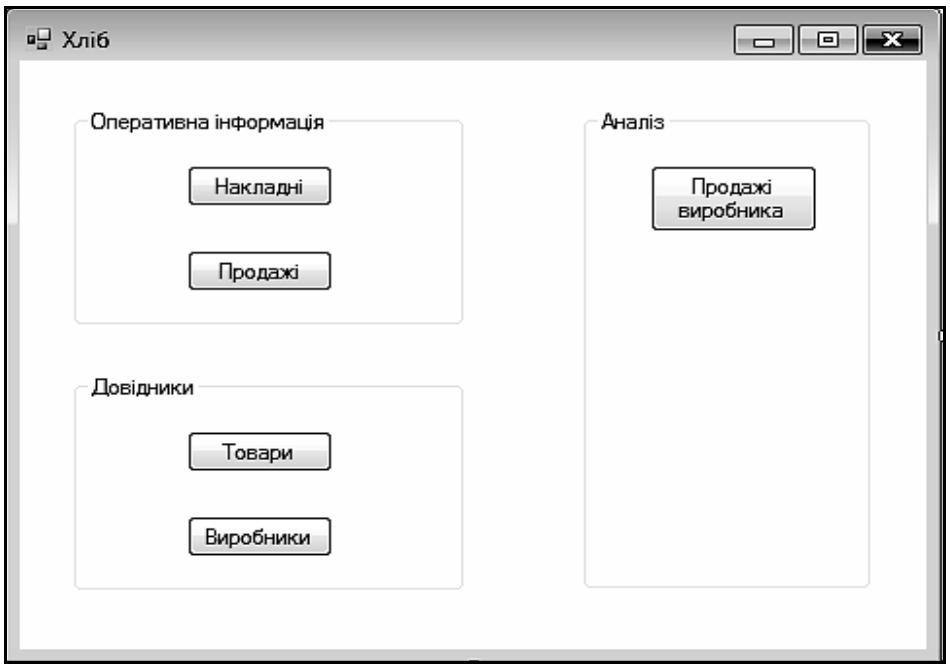


Рис. 8.29. **Форма *Хліб***

8. Збережіть зміни, що зроблені в рішенні.

3.2. Безпосереднє прив'язування сутностей

Завдання 1

Створити форму **Товари** для виконання операцій CRUD з таблицею ***Products*** у базі даних ***Хліб***.

Ідея розв'язку

Оскільки таблиця ***Products*** є батьківською і довідниковою, для виконання операцій CRUD з нею достатньо використати елемент керування DataGridView, а для фіксування зроблених змін – кнопку ***Зберегти***.

Для відображення даних у DataGridView використати такі методи та властивості:

Load – для додавання сутності до контексту;

Local – для подання всіх доданих, модифікованих і незмінених об'єктів у локальному наборі;

ToBindingList – для створення колекції рядків, що прив'язують до інтерфейсу, які синхронізуються з даними ObservableCollection.

Для фіксування змін із даними, що виконані користувачем, застосовуються такі методи:

SaveChanges – для збереження в базі даних усіх змін, що зроблені в контексті;

Refresh – для оновлення даних, що відображаються в DataGridView.

Виконання

1. Додайте до проекту ***winForms*** нову форму з ім'ям файла ***formТовари.cs*** і значенням ***Товари*** для властивості **Text**.

2. Додайте на форму елемент **DataGridView** з ім'ям ***gvТовари***, для відображення набору сутностей ***Products*** (рис. 8.30).

3. Двічі клацніть у вільному місці форми ***Товари*** й у вікні коду форми введіть оператори тіла оброблювача події ***formТовари_Load***. Він має такий вигляд:

```
BreadContext context;
private void formТовари_Load(object sender, EventArgs e)
{
```

```
    // Створюємо екземпляр класу DbContext
    context = new BreadContext();

    // Додаємо колекцію сутностей до контексту
    context.Products.Load();

    // Прив'язуємо набір сутностей до елемента DataGridView
    gvТовари.DataSource = context.Products.Local.ToBindingList();

    //Вилучаємо властивості навігації з інтерфейсу
    gvТовари.Columns.Remove("InvoiceProducts");
    gvТовари.Columns.Remove("Sales");
}
```

У розділ опису просторів імен додайте ще й такі:

```
using System.Data.Entity;
using моделювання.DataAccess;
```

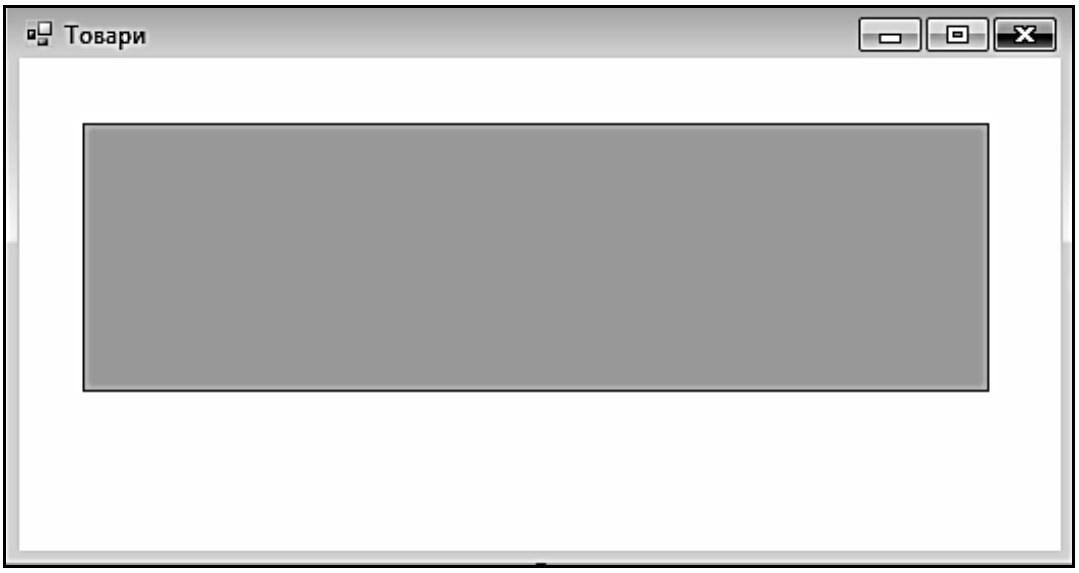


Рис. 8.30. **Елемент DataGridView на формі *Товари***

4. Перейдіть у вікно конструктора форми ***formХліб***, двічі клацніть кнопку ***Товари*** й у вікні коду введіть код оброблювача події "Клацання кнопки Товари".

```
private void buttonТовари_Click(object sender, EventArgs e)
{
    formТовари вікноТовари = new formТовари();
    вікноТовари.ShowDialog();
}
```

5. Перевірте функціональність форми. Після її завантаження повинні відображатися всі дані таблиці ***Products***.

6. Для збереження змін у базі даних додайте на форму ***formТовари*** кнопку ***Зберегти*** і код її оброблювача.

```
private void buttonЗберегти_Click(object sender, EventArgs e)
{
    context.SaveChanges();
    gvТовари.Refresh();
}
```

7. Перевірте функціональність кнопки ***Зберегти***, додавши дані про товари і зберігши їх в базі даних. Потрібно ввести такі дані:

Товар	Ціна	Ціна_закупівлі
Батон "Нарізний"	3,00	2,50

Поясніть дію методу **Refresh** на цій формі.

8. Збережіть зміни, що зроблені в проекті.

Завдання 2

Додати до проекту ***winForms*** форму ***Виробники***, у якій будуть відображатися й змінюватися дані таблиці ***Manufacturers***.

Виконання

Оскільки форма ***Виробники*** будується аналогічно формі ***Товари***, створіть її самостійно.

3.3. Візуальні засоби побудови інтерфейсу

Завдання

Створити форму ***Продажі*** для виконання операцій CRUD з таблицею ***Sales*** у базі даних Хліб (рис. 8.31).

Ідея розв'язку

Для побудови форми слід застосувати візуальні засоби на основі вікна **Data Sources**. Побудова форми виконується в 2 етапи:

1. Додавання джерела даних на основі об'єкта.
2. Створення форми візуальними засобами.

У результаті виконання цих етапів буде отримано форму ***Продажі***, в якій можна переглядати, додавати, видаляти і змінювати дані сутності ***Sale***.

Оскільки таблиця ***Sales*** містить поля ***ProductID*** та ***ManufacturerID*** для зв'язку з відповідними батьківськими таблицями, на формі ***Продажі*** використати поля підстановки у вигляді елементів ComboBox.

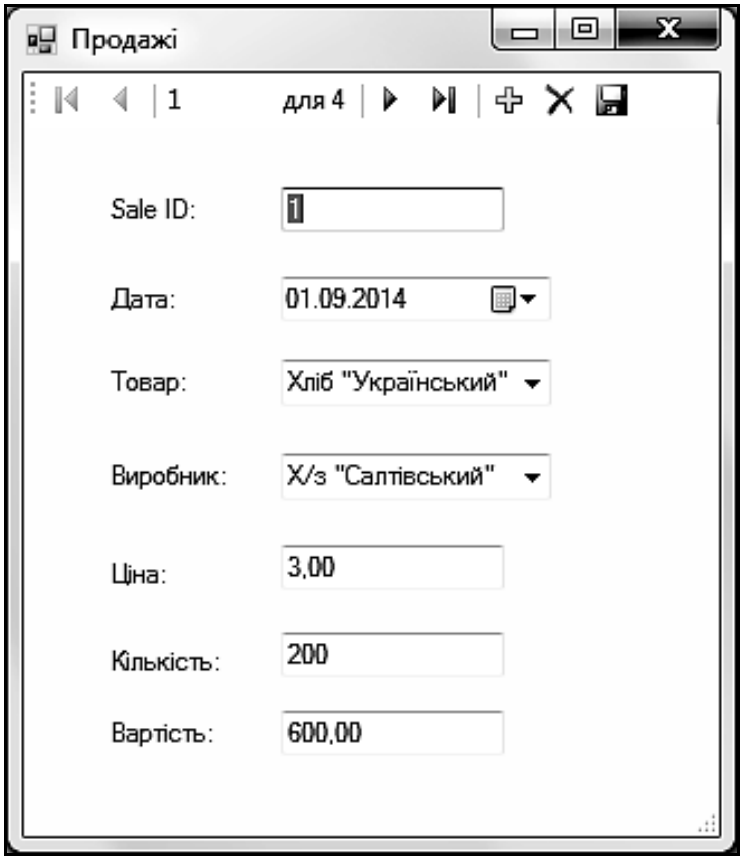


Рис. 8.31. **Форма *Продажі***

Завдання 1

Додати в застосування джерела даних на основі моделі даних Code First.

Виконання

1. Викличте майстра настроювання джерела даних, вибравши команду **Add New Data Source**, що знаходиться в меню **Data**.

2. Виберіть значок **Object** у вікні **Choose a Data Source Type** і клацніть кнопку **Next** (рис. 8.32).

3. Розкрийте вузол ***моделювання*** і виберіть сутності ***Invoice***, ***InvoiceProduct***, ***Manufacturer***, ***Product*** і ***Sale*** з підвузла ***моделювання.Models***, дані яких знадобляться для відображення на формах, у вікні **Select the Data Objects** і клацніть кнопку **Finish** (рис. 8. 33).

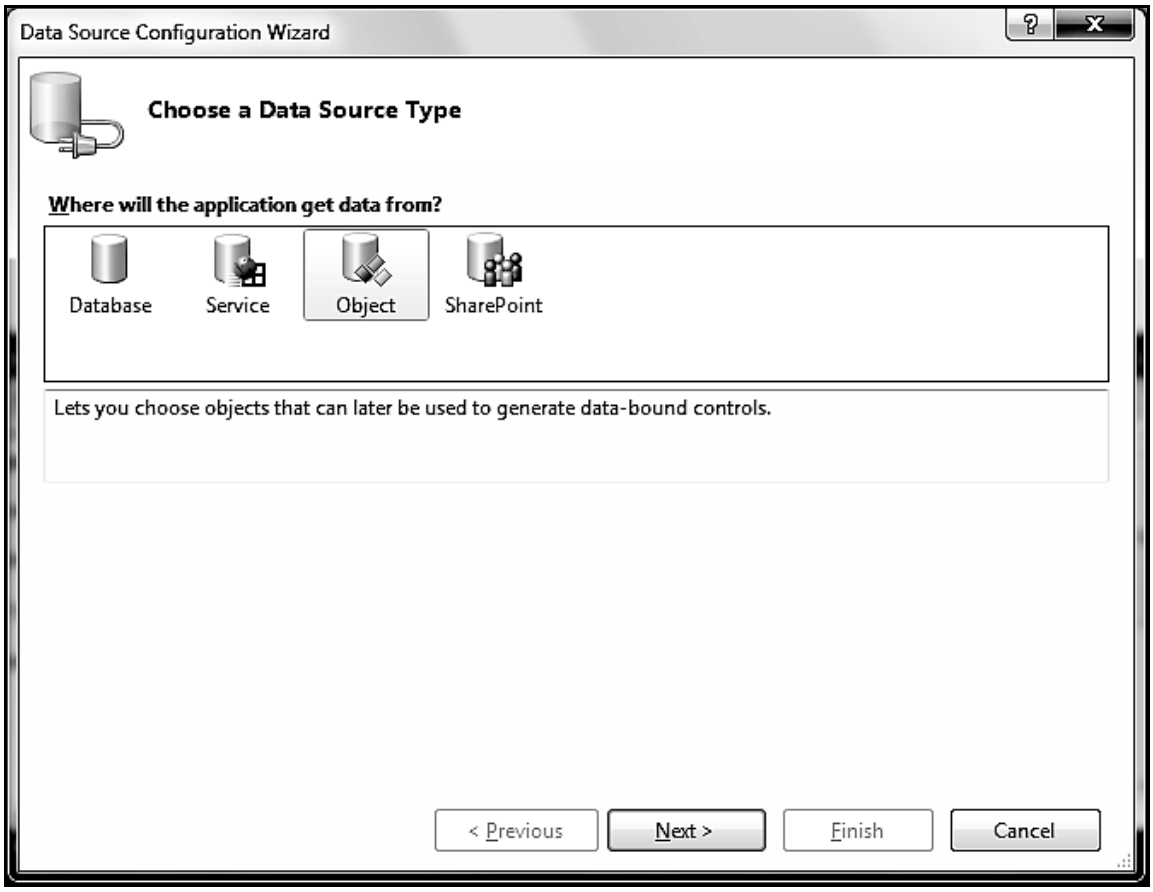


Рис. 8.32. **Вікно *Choose a Data Source Type***

4. Відкрийте вікно **DataSources** командою **Data – Show Data-Sources**.

У результаті роботи майстра у вікні **Data Sources** з'явилося зображення вибраних сутностей. Кожна сутність тут подана вузлом у вигляді значка. Елементами вузла є імена властивостей. Якщо сутність має асоціацію з іншою сутністю, то остання відображається у вигляді підвузла. На рис. 8.34 у вузлі сутності ***Sale*** відображаються її властивості та дві батьківські сутності – ***Manufacturer*** і ***Product***. Сутність ***Product*** буде використовуватися під час побудови форми для відображення ціни товару. Сутності ***Manufacturer*** і ***Product***, що подані вузлами першого рівня, використовуватимуться під час побудови відповідних елементів ComboBox для заповнення їх даними.

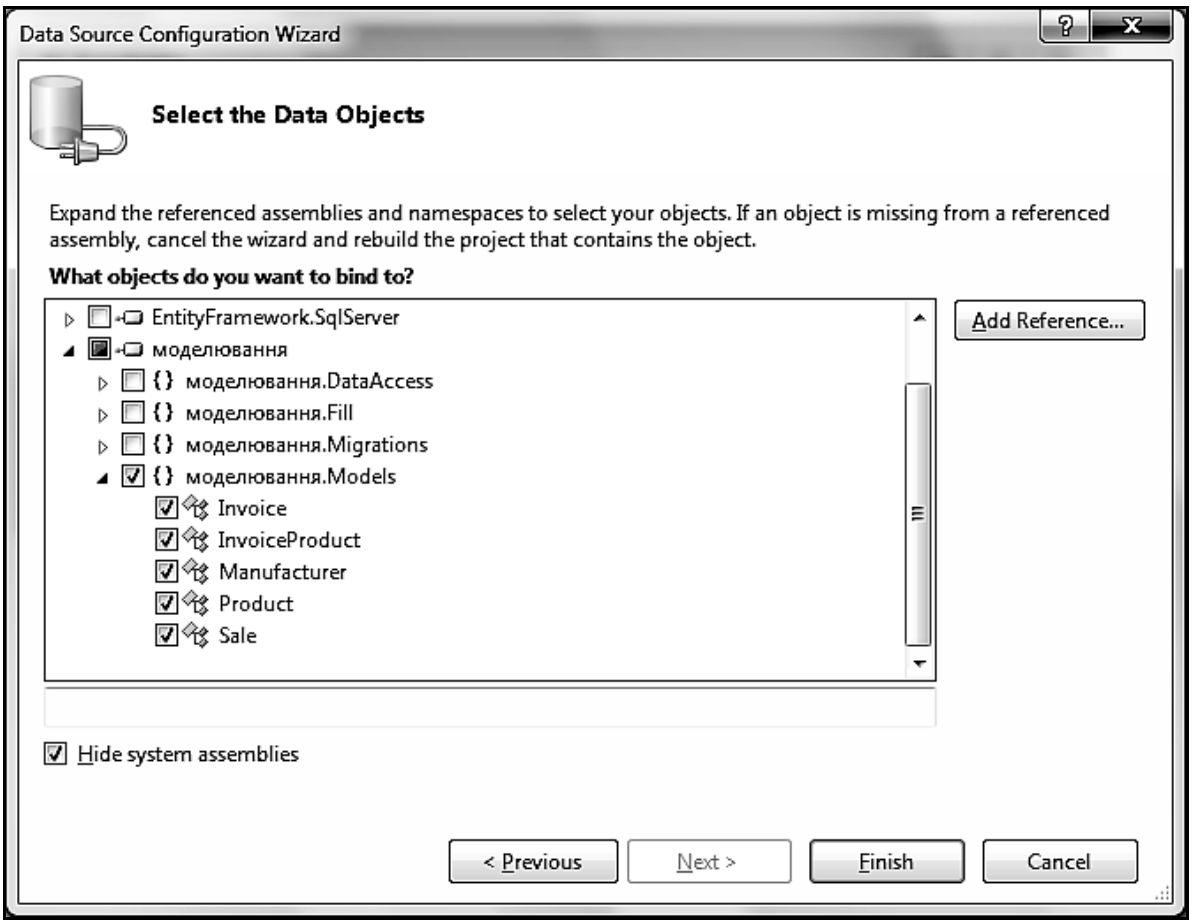


Рис. 8.33. Вікно **Select the DataObjects**

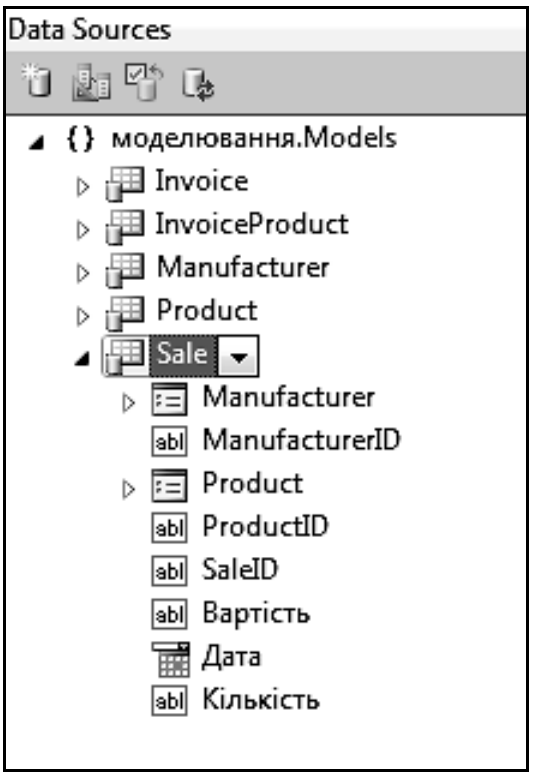


Рис. 8.34. **Вікно *Data Sources***

Завдання 2

Створити форму ***Продажі*** з використанням візуальних засобів (рис. 8.35).

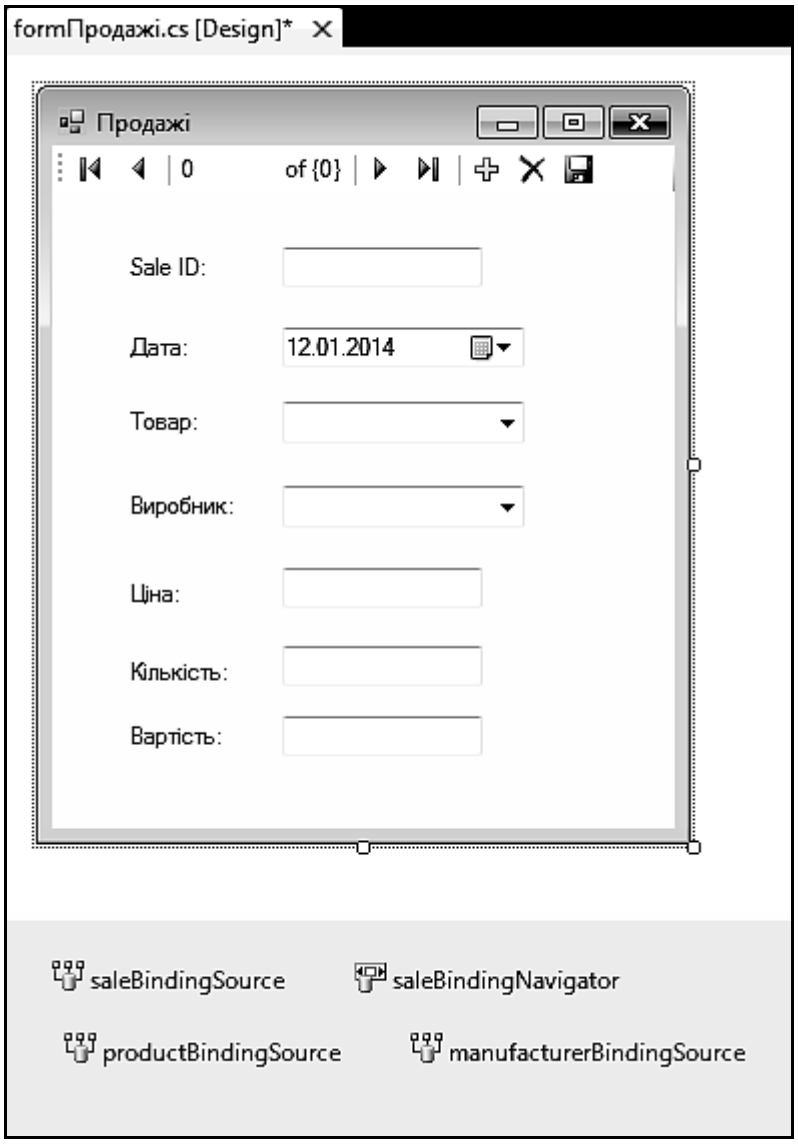


Рис. 8.35. **Форма *Продажі* в конструкторі**

Виконання

1. Додайте до проекту ***winForms*** нову форму, з ім'ям файла ***formПродажі.cs*** і значенням ***Продажі*** для властивості **Text**.

2. Клацніть вузол сутності ***Sale*** у вікні **Data Sources** і з її списку, що розкривається, виберіть подання **Details**.

3. Виберіть подання **ComboBox** у вузлі ***Sale*** для властивості ***ManufacturerID*** у списку, що розкривається.

4. Повторіть п. 3 для властивості ***ProductID***.

5. Перетягніть з вузла ***Sale*** на форму ***Продажі*** по черзі такі властивості:

SaleID,

Дата,

ProductID,

ManufacturerID,

Product.Ціна,

Кількість,

Вартість.

На формі з'явилися елементи керування для відображення даних й панель навігатора, а в області компонентів – ряд об'єктів, які забезпечують роботу з даними сутності (рис. 8.36). Ознайомтеся з ними й визначте призначення кожного.

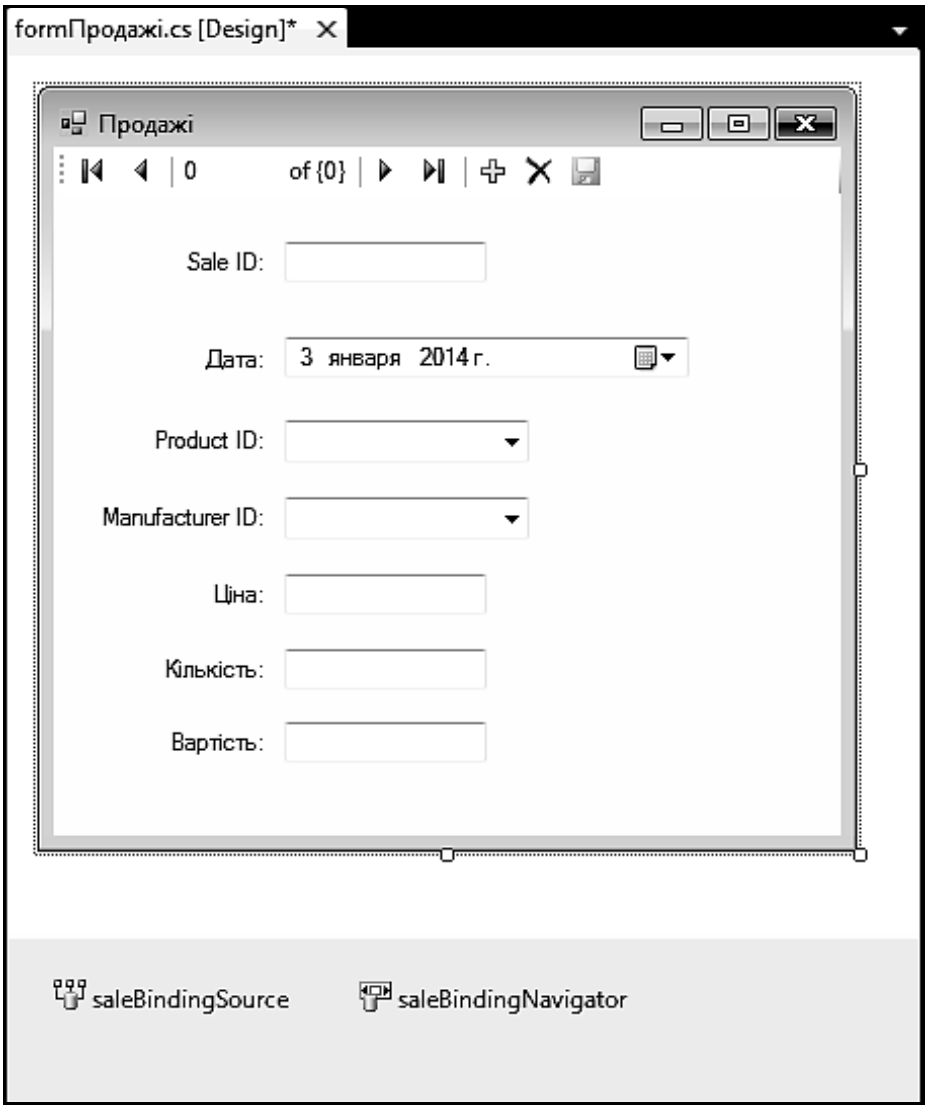


Рис. 8.36. **Вікно форми *Продажі* після додавання сутності *Sale***

6. Клацніть елемент DateTimePicker і для його властивості **Format** встановіть значення ***Short***.

7. Змініть властивість **Text** для написів ***ProductID*** й ***ManufacturerID***, установивши нові значення ***Товар*** і ***Виробник*** відповідно. Потім вирівняйте всі елементи Label по лівому краю.

8. Щоб в елементі ComboBox для товару відображалися назви товарів, перетягніть вузол сутності ***Product*** з вікна ***Data Sources*** на цей ComboBox і відпустіть. Потім для елемента ComboBox ***Товар*** встановіть такі значення властивостей:

Властивість	Значення
DataSource	productBindingSource
DisplayMember	Товар
ValueMember	ProductID
SelectedValue	saleBindingSource - ProductID

Примітка. Значення цих чотирьох властивостей краще встановлювати у вікні задач ComboBox (рис. 8.37).

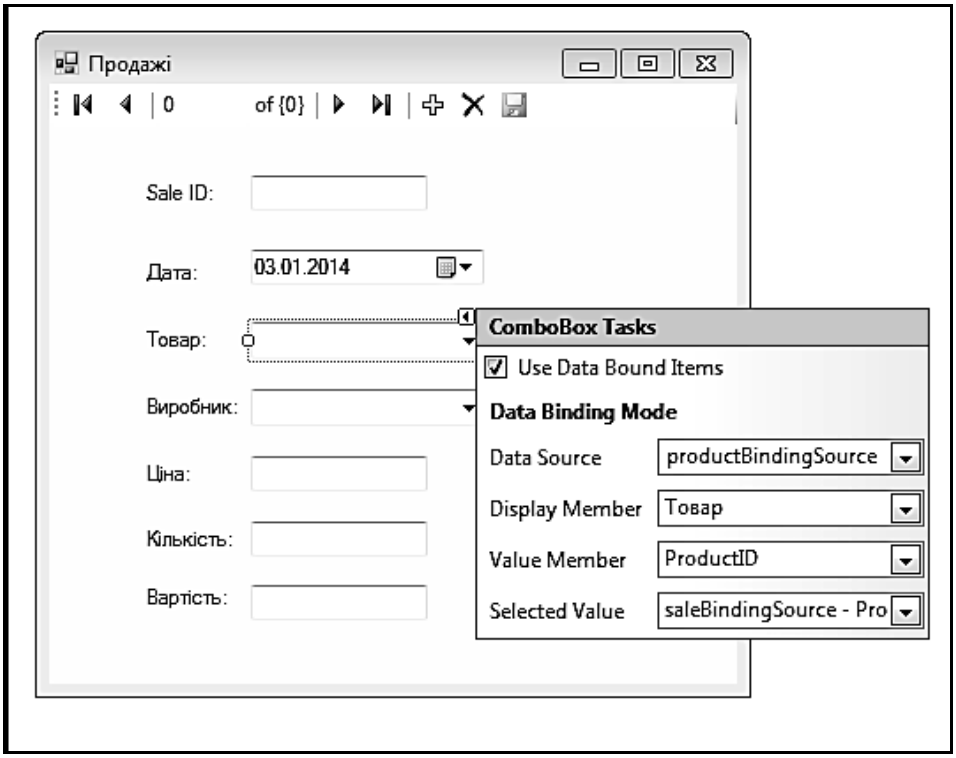


Рис. 8.37**. Встановлення значень властивостей прив'язки для елемента ComboBox**

9. Повторіть п. 8 для ComboBox ***Виробник***.

Примітка. В область компонентів форми автоматично додалися компоненти BindingSource для сутностей ***Product*** й ***Manufacturer***.

10. Двічі клацніть у вільному місці форми ***Продажі*** й у вікні коду форми введіть оператори тіла оброблювача події ***formПродажі_Load***. Він має такий вигляд:

```
BreadContext context;

private void formПродажі_Load(object sender, EventArgs e)
{
    // Створюємо екземпляр класу DbContext
    context = new BreadContext();

    // Завантажуємо дані для productBindingSource
    // та manufacturerBindingSource
    productBindingSource.DataSource = context.Products.ToList();
    manufacturerBindingSource.DataSource =
        context.Manufacturers.ToList();

    // Завантажуємо дані для saleBindingSource
    context.Sales.Load();

    saleBindingSource.DataSource =
        context.Sales.Local.ToBindingList();
}
```

У розділ опису просторів імен додайте ще й такі:

```
using System.Data.Entity;
using моделювання.DataAccess;
```

11. Перейдіть у вікно конструктора форми ***Продажі***, клацніть правою кнопкою миші на кнопці ***Зберегти***, що розташована елементі ***saleBindingNavigator*** і виберіть значення **Enable** з контекстового меню. Потім додайте код оброблювача події "Клацання кнопки Зберегти".

```
private void saleBindingNavigatorSaveItem_Click(object sender,
    EventArgs e)
{
    saleBindingSource.EndEdit();
    context.SaveChanges();
}
```

12. Перейдіть у вікно конструктора форми ***Хліб*** і додайте код оброблювача події "Клацання кнопки Продажі":

```
private void buttonПродажі_Click(object sender, EventArgs e)
{
    formПродажі вікноПродажі = new formПродажі();
    вікноПродажі.ShowDialog();
}
```

13. Запустіть програму на виконання й перевірте функціональність форми ***Продажі***, додавши дані про продаж одного товару і зберігши їх (рис. 8.38).

Рис. 8.38. **Додавання даних про продаж одного товару**

14. Закрийте форми ***Продажі*** й ***Хліб*** і збережіть зміни, що зроблені в проекті.

3.4. Робота з ієрархічними сутностями

Завдання

Додати в застосування форму ***Накладні***, у якій будуть відображатися й змінюватися дані сутностей ***Invoice*** та ***InvoiceProduct***, що пов'язані відношенням один-до-багатьох (рис. 8.39).

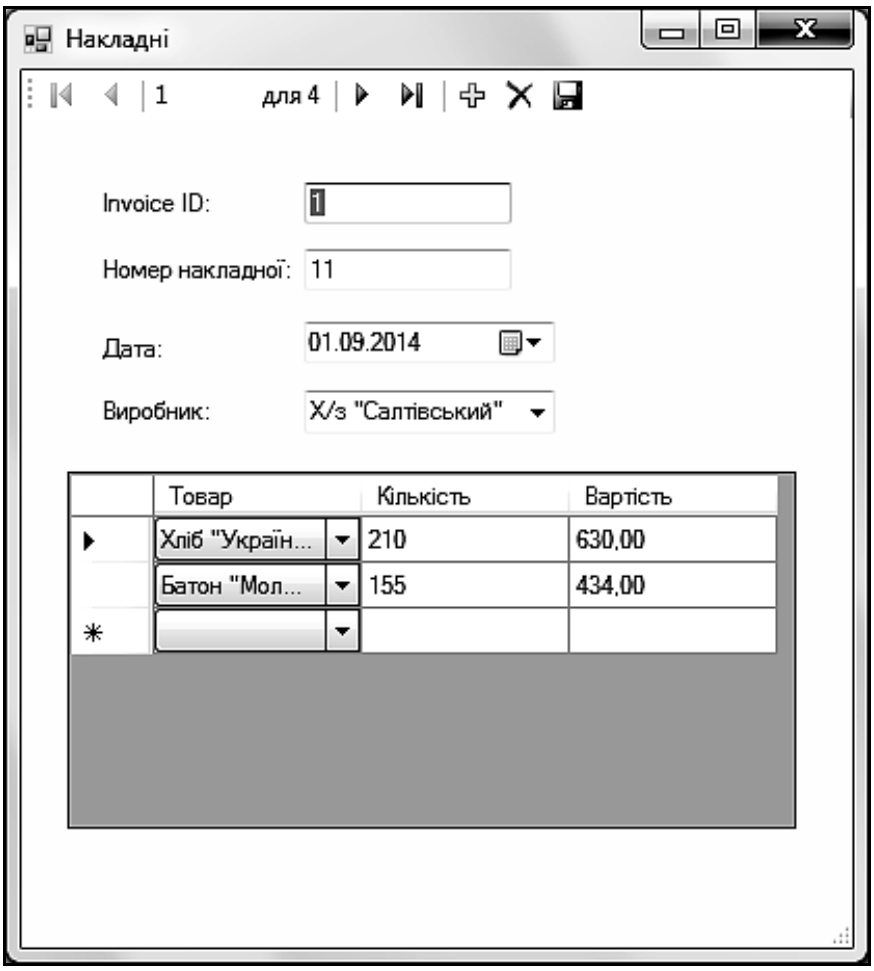


Рис. 8.39. **Форма *Накладні***

Ідея розв'язку

Оскільки сутності ***Invoice*** та ***InvoiceProduct*** пов'язані асоціацією один-до-багатьох, то з метою забезпечення двосторонньої прив'язки та сортування необхідно розширити ObservableCollection для додавання функціональності IListSource. Варто нагадати, що інтерфейс IListSource надає об'єкту функціональні можливості, що дозволяють повертати список, який може бути пов'язаний з джерелом даних. Це вимагає внесення змін до батьківського класу ***Invoice*** для опису пов'язаної колекції сутностей ***InvoiceProducts***. Тому виконання завдання складається з двох етапів:

1. Реалізація IListSource для колекцій.
2. Створення форми.

У результаті виконання цих етапів буде отримано форму ***Накладні***, в якій можна переглядати, додавати, видаляти і змінювати дані ієрархічних сутностей ***Invoice*** та ***InvoiceProduct***.

Завдання 1

Реалізувати інтерфейс IListSource для колекцій сутностей ***Invoices***.

Виконання

1. Додайте у папку ***Models*** проекту ***моделювання*** новий клас ***ObservableListSource*** і введіть його опис.

```
using System.Collections;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Diagnostics.CodeAnalysis;
using System.Data.Entity;

namespace моделювання.Models
{
    public class ObservableListSource<T> : ObservableCollection<T>,
        IListSource where T : class
    {
        private IBindingList _bindingList;

        bool IListSource.ContainsListCollection {get {return false; }}

        IList IListSource.GetList()
        {
           return _bindingList ?? (_bindingList=this.ToBindingList());
        }
    }
}
```

2. Відкрийте клас ***Invoice***, що знаходиться у папці ***Models*** проекту ***моделювання***, закоментуйте останній рядок опису класу, що стосується колекції ***InvoiceProducts***, і додайте після нього такі рядки:

```
private readonly ObservableListSource<InvoiceProduct>
    invoiceProducts = new ObservableListSource<InvoiceProduct>();
public virtual ObservableListSource<InvoiceProduct>
    InvoiceProducts { get { return invoiceProducts; } }
```

Після цього опис класу набуде такого вигляду:

```
public class Invoice
{
    public int InvoiceID { get; set; }
    [MaxLength(12)]
    public string Номер_накладної { get; set; }
```

```
    public DateTime Дата { get; set; }
    public int ManufacturerID { get; set; }

    // Властивості навігації
    public virtual Manufacturer Manufacturer { get; set; }
  //public virtual ICollection<InvoiceProduct> InvoiceProducts {get; set;}

    private readonly ObservableListSource<InvoiceProduct>
        invoiceProducts = new ObservableListSource<InvoiceProduct>();
    public virtual ObservableListSource<InvoiceProduct>
        InvoiceProducts { get { return invoiceProducts; } }
    }
```

3. Щоб позбавитися від помилок, що пов'язані із додаванням даних у методі ***Seed***, відкрийте файл ***Configuration.cs***, що міститься у папці ***Migrations*** проекту ***моделювання***, і закоментуйте рядки методу ***Seed***, які стосуються додавання товарів до нової накладної, що отримані від виробника з кодом 2 (позначені коментарем //2. Додавання товарів …). Ці рядки можна відновити, якщо знову знадобляться міграції.

4. Перевірте рішення на відсутність помилок, виконавши команду **Build – Rebuild Solution**.

Завдання 2

Створити форму ***Накладні*** з використанням візуальних засобів (рис. 8.39).

Виконання

1. Додайте до проекту ***winForms*** нову форму, давши їй ім'я ***formНакладні***.

2. Клацніть вузол сутності ***Invoice*** у вікні **Data Sources** і з її списку, що розкривається, виберіть подання **Details**.

3. Виберіть подання **ComboBox** у вузлі ***Invoice*** для властивості ***ManufacturerID*** у списку, що розкривається.

4. Перетягніть з вузла ***Invoice*** на форму ***Накладні*** по черзі такі властивості:

InvoiceID,

Номер_накладної,

Дата,

ManufacturerID.

На формі з'явилися елементи керування для відображення даних й панель навігатора, а в області компонентів – об'єкти, які забезпечують роботу з даними сутності (рис. 8.40).

Рис. 8.40. **Вікно форми *Накладні* після додавання сутності *Invoice***

5. Налаштуйте елемент DateTimePicker, встановивши для його властивості **Format** значення ***Short***, а також змініть значення властивості **Text** для напису ***ManufacturerID***, установивши нове значення ***Виробник***.

6. Щоб в елементі ComboBox відображалися назви виробників, перетягніть вузол сутності ***Manufacturer*** з вікна **Data Sources** на цей ComboBox і відпустіть. Потім для елемента ComboBox ***Виробник*** встановіть такі значення властивостей:

Властивість	Значення
DataSource	manufacturerBindingSource
DisplayMember	Виробник
ValueMember	ManufacturerID
SelectedValue	invoiceBindingSource - ManufacturerID

7. Двічі клацніть у вільному місці форми ***Накладні*** й у вікні коду форми уведіть оператори тіла оброблювача події ***formНакладні_Load***. Він має такий вигляд:

```
BreadContext context;

private void formНакладні_Load(object sender, EventArgs e)
{
    // Створюємо екземпляр класу DbContext
    context = new BreadContext();

    // Завантажуємо дані для manufacturerBindingSource
    manufacturerBindingSource.DataSource =
        context.Manufacturers.ToList();

    // Завантажуємо дані для invoiceBindingSource
    context.Invoices.Load();

    invoiceBindingSource.DataSource =
        context.Invoices.Local.ToBindingList();
}
```

У розділ опису просторів імен додайте ще й такі:

```
using System.Data.Entity;
using моделювання.DataAccess;
```

8. Перейдіть у вікно конструктора форми ***Накладні***, клацніть правою клавішею миші на кнопці ***Зберегти***, що розташована елементі ***invoiceBindingNavigator*** і виберіть значення **Enable** з контекстового меню. Потім додайте код оброблювача події "Клацання кнопки Зберегти".

```
private void invoiceBindingNavigatorSaveItem_Click(object sender,
    EventArgs e)
{
    invoiceBindingSource.EndEdit();
    context.SaveChanges();
}
```

9. Перейдіть у вікно конструктора форми ***Хліб*** і додайте код оброблювача події "Клацання кнопки Накладні":

```
private void buttonНакладні_Click(object sender, EventArgs e)
{
    formНакладні вікноНакладні = new formНакладні();
    вікноНакладні.ShowDialog();
}
```

10. Запустіть програму на виконання й перевірте функціональність форми ***Накладні***, додавши дані про накладну і зберігши їх (рис. 8.41).

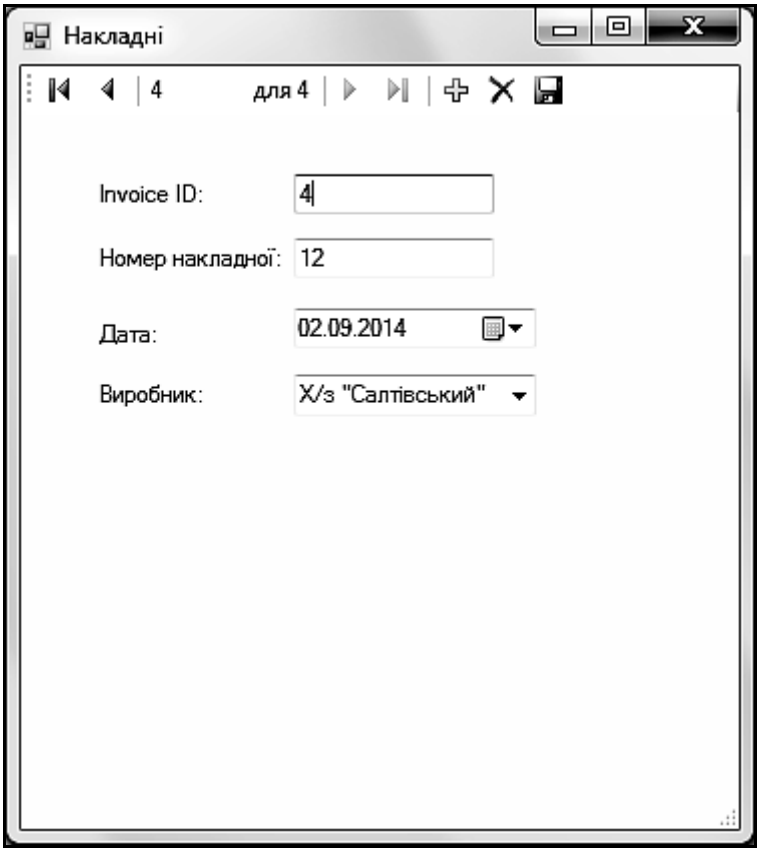


Рис. 8.41. **Додавання даних про накладну**

11. Закрийте форми ***Накладні*** та ***Хліб*** і збережіть зміни, що зроблені в проекті.

12. Перетягніть підвузол ***InvoiceProducts***, що знаходиться у вузлі ***Invoice*** вікна **DataSources**, в нижню частину форми ***Накладні***. Перевірте функціональність форми ***Накладні*** (рис. 8.42).

13. Налаштуйте відображення елемента ***invoiceProductsDataGridView***, зробивши невидимими стовпці ***InvoiceProductID*** і ***InvoiceID*** та

видаливши стовпці ***Product*** та ***Invoice***. Для цього можна використати властивість **Columns**.

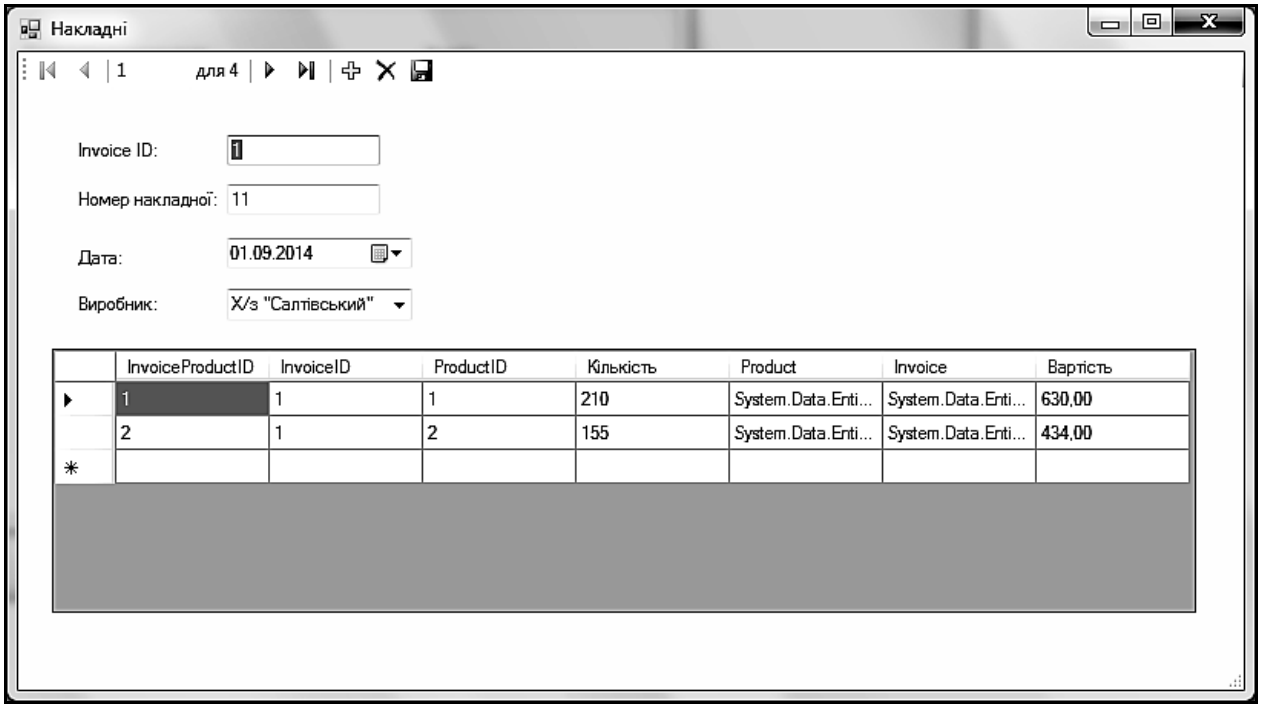


Рис. 8.42. **Форма *Накладні* з доданими товарами накладних**

14. Налаштуйте властивість ***ProductID*** у вигляді елемента ComboBox. Для цього:

14.1. Перетягніть на форму яке-небудь поле (наприклад, ***Товар***) з вузла ***Products*** вікна **Data Sources** у вільне місце форми й відразу вилучіть його. У нижній частині вікна конструктора залишиться елемент ***productBindingSource***.

14.2. Перейдіть у вікно **Edit Columns** за допомогою властивості **Columns** для елемента ***invoiceProductsDataGridView***.

14.3. Змініть тип стовпця ***ProductID*** на **DataGridViewComboBoxColumn**, дайте йому ім'я ***productIDDataGridViewComboBoxColumn*** і заголовок ***Товар***. Тут також установіть значення властивостей у групі **Data**:

Властивість	Значення
DataPropertyName	ProductID
DataSource	productBindingSource
DisplayMember	Товар
ValueMember	ProductID

Вікно **Edit Columns** зі встановленими значеннями властивостей у групі **Data** подано на рис. 8.43.

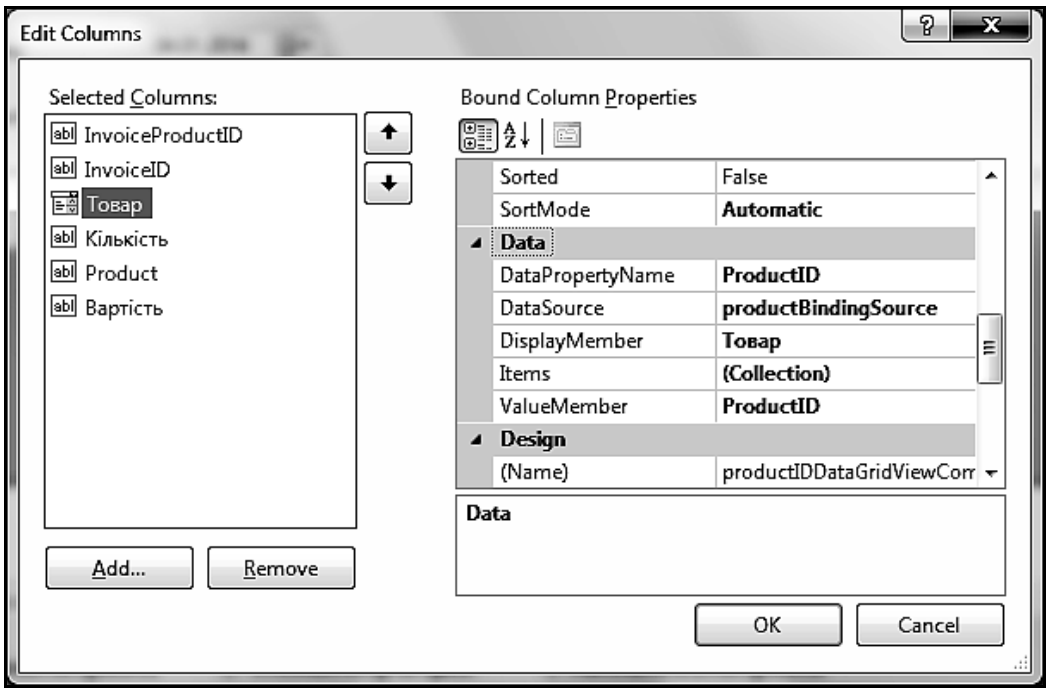


Рис. 8.43. **Встановлення значень властивостей у групі *Data***

14.4. Перейдіть у вікно коду форми ***Накладні*** й додайте оператор для завантаження даних для ***productBindingSource*** в тілі оброблювача події ***formНакладні_Load***:

```
   // Завантажуємо дані для productBindingSource
   productBindingSource.DataSource = context.Products;
```

Після цього оброблювач набуде такого вигляду:

```
private void formНакладні_Load(object sender, EventArgs e)
{
    // Створюємо екземпляр класу DbContext
    context = new BreadContext();

    // Завантажуємо дані для manufacturerBindingSource
    manufacturerBindingSource.DataSource =
        context.Manufacturers.ToList();
    // Завантажуємо дані для productBindingSource
    productBindingSource.DataSource = context.Products.ToList();

    // Завантажуємо дані для invoiceBindingSource
    context.Invoices.Load();

    invoiceBindingSource.DataSource =
        context.Invoices.Local.ToBindingList();
}
```

14.5. Додайте оператор завершення редагування ***invoiceProducts BindingSource*** в тілі оброблювача події ***invoiceBindingNavigatorSave Item_Click***:

```
invoiceProductsBindingSource.EndEdit();
```

та переміщення записами, щоб забезпечити оновлення даних на формі.

Після цього оброблювач набуде такого вигляду:

```
private void invoiceBindingNavigatorSaveItem_Click(object sender, EventArgs e)
{
    invoiceProductsBindingSource.EndEdit();
    invoiceBindingSource.EndEdit();
    context.SaveChanges();

    // Оновлення даних на формі
    invoiceBindingSource.MovePrevious();
    invoiceBindingSource.MoveNext();
}
```

15. Встановіть значення за замовчуванням для стовпця ***Товар***. Для цього:

15.1. Виділіть елемент invoiceProductsDataGridView.

15.2. Перейдіть у вікно властивостей і встановіть режим відображення подій.

15.3. Двічі клацніть подію **DefaultValuesNeeded**.

15.4. Введіть тіло оброблювача події **DefaultValuesNeeded**. Він набуде такого вигляду:

```
private void invoiceProductsDataGridView_DefaultValuesNeeded(object
sender, DataGridViewRowEventArgs e)
{
    //Значення за замовчанням для стовпця Товар
    e.Row.Cells["productIDDataGridViewComboBoxColumn"].Value =
        context.Products.Min(t => t.ProductID);
}
```

16. Запустіть програму на виконання й перевірте функціональність форми ***Накладні***, додавши дані про надходження кількох товарів і зберігши їх (рис. 8.44).

17. Закрийте форми ***Накладні*** й ***Хліб***.

18. Збережіть зміни, що зроблені в рішенні.

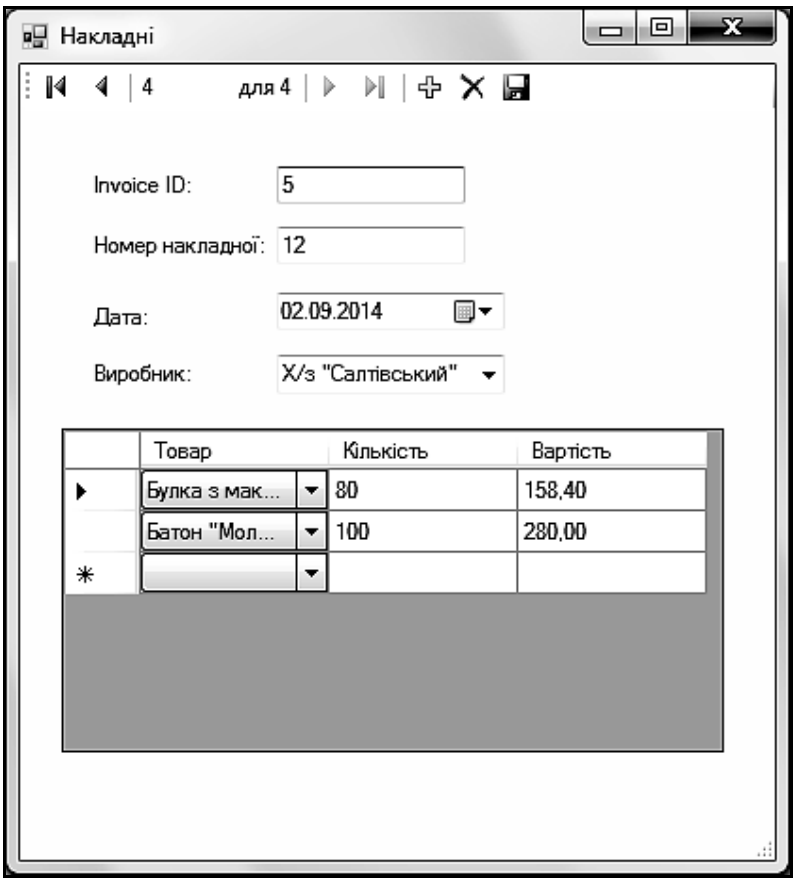


Рис. 8.44. **Форма *Накладні* з відображенням властивості *ProductID* у вигляді ComboBox**

3.5. Використання технології LINQ to Entity в задачах аналізу даних

Завдання

Додати в застосування форму ***Продажі виробника***. У ній відображаються дані про результати продажів товарів вибраного виробника в табличній формі та у вигляді об'ємної кругової діаграми (рис. 8.45).

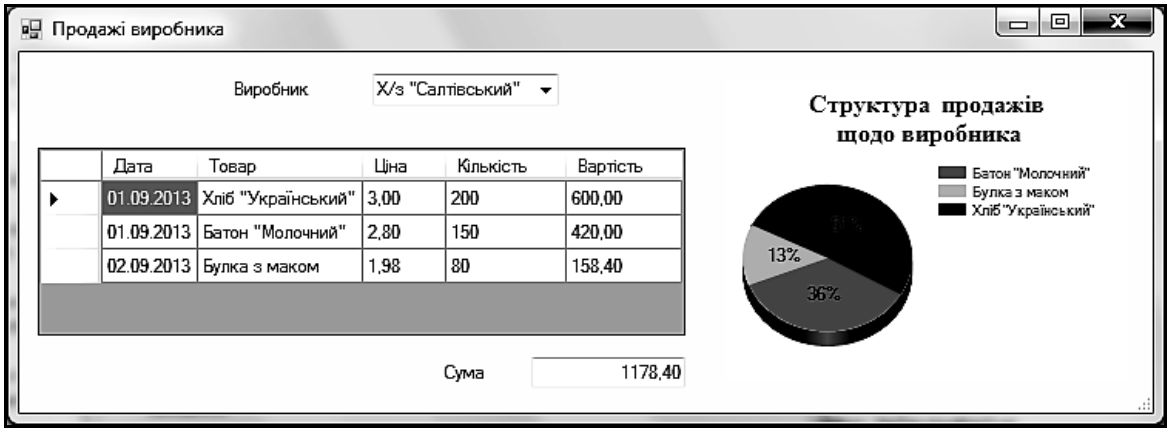


Рис. 8.45. **Форма *Продажі виробника***

Ідея розв'язку

Дані, що відображаються в елементі DataGridView, отримуються із сутностей ***Sale*** та ***Product***, які пов'язані між собою асоціацією. Тому для відбору таких комбінацій даних доцільно використати запит LINQ.

Більш того, набір сутностей, що відображаються, залежить від вибору виробника в елементі ComboBox. Такий відбір даних у LINQ-запиті реалізується за допомогою речення Where.

Для відображення загальної вартості проданих товарів вибраного виробника слід скористатися агрегатною функцією Sum в LINQ-запиті.

Дані, що використовуються для побудови діаграми, також отримані з LINQ-запиту, до якого додано речення **group**. Воно забезпечує групування даних за назвами товарів з обчисленням суми вартостей у кожній групі.

Для реакції на зміну виробника в ComboBox треба використати оброблювача події з цим елементом. Він містить аналогічні запити.

Виконання

1. Додайте в застосування нову форму, давши їй ім'я ***formПродажіВиробника***, а для властивості **Text** встановіть значення ***Продажі виробника***.

2. Додайте на форму поле зі списком ***Виробник***. Для цього:

2.1. Виберіть у вікні **Data Sources** подання **ComboBox** для властивості ***ManufacturerID*** у вузлі ***Manufacturer***.

2.2. Перетягніть властивість ***ManufacturerID*** з вузла ***Manufacturer*** у вікні ***Data Sources*** на форму ***Продажі виробника***.

На формі з'явився елемент керування ComboBox й панель навігатора, а в області компонентів – елемент ***manufacturerBindingSource***.

2.3. Видаліть панель навігатора з форми.

2.4. Змініть властивість **Text** для напису ***ManufacturerID***, установивши нове значення ***Виробник***.

2.5. Встановіть такі значення властивостей для елемента ComboBox ***Виробник***:

Властивість	Значення
DataSource	manufacturerBindingSource
DisplayMember	Виробник
ValueMember	ManufacturerID
SelectedValue	(none)

2.6. Двічі клацніть у вільному місці форми ***Продажі виробника*** й у вікні коду форми введіть оператори тіла оброблювача події ***formПродажіВиробника_Load***. Він має такий вигляд:

```
// Спільні об'єкти
BreadContext context;

private void formПродажіВиробника_Load(object sender, EventArgs e)
{
    // Створюємо екземпляр класу DbContext
    context = new BreadContext();

    // Завантажуємо дані для manufacturerBindingSource
    manufacturerBindingSource.DataSource =
        context.Manufacturers.ToList();
}
```

У розділ опису просторів імен додайте ще й такі:

```
using System.Data.Entity;
using моделювання.DataAccess;
```

2.7. Перейдіть у вікно конструктора форми ***Хліб*** і додайте код оброблювача події "Клацання кнопки Продажі виробника":

```
private void buttonПродажіВиробника_Click(object sender, EventArgs e)
{
    formПродажіВиробника вікноПродажіВиробника =
        new formПродажіВиробника();
    вікноПродажіВиробника.ShowDialog();
}
```

2.8. Запустіть програму на виконання й перевірте функціональність поля зі списком ***Виробник*** на формі ***Продажі виробника*** (рис. 8.46).

2.9. Закрийте форми ***Продажі виробника*** та ***Хліб*** і збережіть зміни, що зроблені в проекті.

3. Для відображення результатів продажів товарів вибраного виробника у табличній формі виконайте таке:

3.1. Додайте на форму елемент **DataGridView** з ім'ям ***gvПродажіВиробника***.

3.2. Двічі клацніть у вільному місці форми ***Продажі виробника*** й у вікні коду форми введіть у розділ спільних об'єктів класу такий оператор:

```
BindingSource bindingПродажіВиробника = new BindingSource();
```

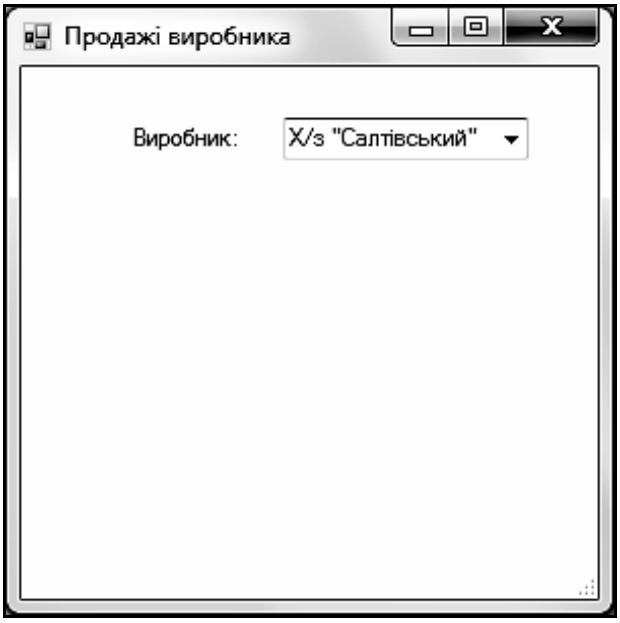


Рис. 8.46. **Форма *Продажі виробника* з полем зі списком *Виробник***

а в тіло оброблювача події ***formПродажіВиробника_Load*** додайте такі оператори:

```
// Запит для gvПродажіВиробника
var queryПродажіВиробника = from продаж in context.Sales
    where продаж.ManufacturerID ==
(int)manufacturerIDComboBox.SelectedValue
select new
    {
        Дата = продаж.Дата,
        Товар = продаж.Product.Товар,
        Ціна = продаж.Product.Ціна,
        Кількість = продаж.Кількість,
        Вартість = продаж.Product.Ціна * продаж.Кількість
    };
// Відображаємо дані
bindingПродажіВиробника.DataSource = queryПродажіВиробника.ToList();
gvПродажіВиробника.DataSource = bindingПродажіВиробника;
gvПродажіВиробника.AutoResizeColumns();
```

Тепер ці дві частини коду мають такий вигляд:

```
// Спільні об'єкти
BreadContext context;
BindingSource bindingПродажіВиробника = new BindingSource();

private void formПродажіВиробника_Load(object sender, EventArgs e)
{
    // Створюємо екземпляр класу DbContext
    context = new BreadContext();

    // Завантажуємо дані для manufacturerBindingSource
    manufacturerBindingSource.DataSource =
        context.Manufacturers.ToList();

    // Запитдля gvПродажіВиробника
    var queryПродажіВиробника = from продаж in context.Sales
        where продаж.ManufacturerID ==
        (int)manufacturerIDComboBox.SelectedValue
        select new
        {
            Дата = продаж.Дата,
            Товар = продаж.Product.Товар,
            Ціна = продаж.Product.Ціна,
            Кількість = продаж.Кількість,
            Вартість = продаж.Product.Ціна * продаж.Кількість
        };

    // Відображаємо дані
    bindingПродажіВиробника.DataSource =
        queryПродажіВиробника.ToList();
    gvПродажіВиробника.DataSource = bindingПродажіВиробника;
    gvПродажіВиробника.AutoResizeColumns();
}
```

3.3. Запустіть програму на виконання й перевірте функціональність поля зі списком ***Виробник*** і елемента DataGridView на формі ***Продажі виробника***. В останньому відображаються результати продажів товарів виробника, що вказаний у полі зі списком (рис. 8.47). Але в разі зміни виробника, відповідні дані не з'являються. Цей недолік ліквідується у подальшому за допомогою оброблювача події.

3.4. Закрийте форми ***Продажі виробника*** та ***Хліб*** і збережіть зміни, що зроблені в проекті.

4. Для відображення сумарної вартості продажів товарів вибраного виробника виконайте таке:

4.1. Додайте на форму елементи напис і текстове поле.

4.2. Для властивості **Text** напису встановіть значення ***Сума:***, а для властивості **Name** текстового поля – значення ***textBoxСума***.

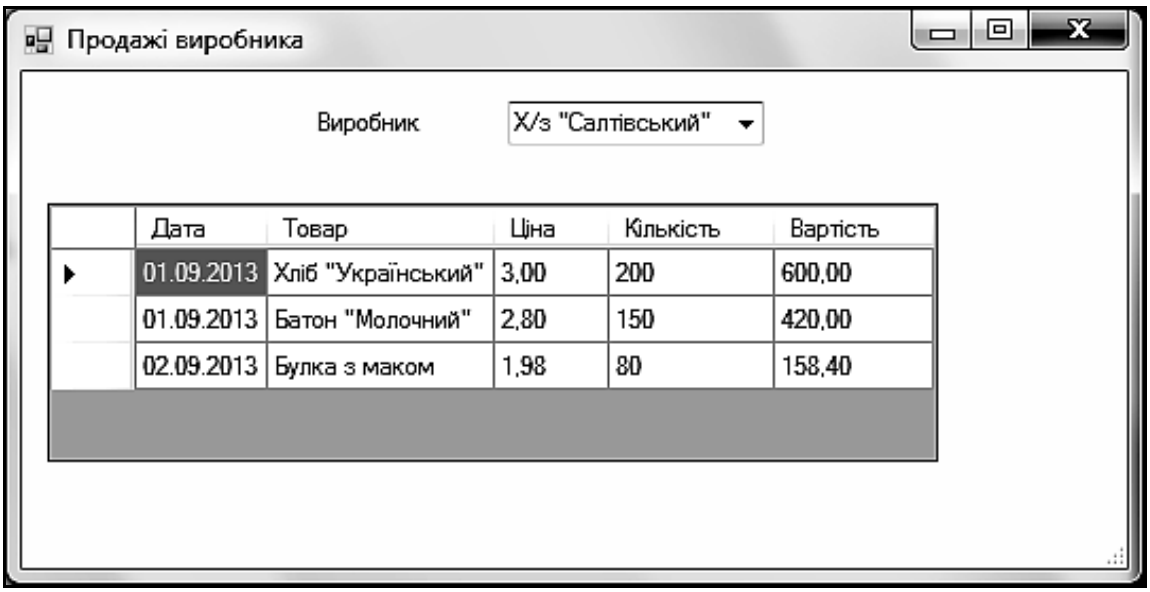


Рис. 8.47. **Форма *Продажі виробника* з результатами продажів товарів виробника**

4.3. Двічі клацніть у вільному місці форми ***Продажі виробника*** й у вікні коду форми додайте в кінець коду оброблювача події ***formПродажіВиробника*** такі оператори:

```
//Сума
decimal Сума= (decimal)queryПродажіВиробника.Sum(p => p.Вартість);
textBoxСума.Text = Сума.ToString("0.00");
textBoxСума.TextAlign = HorizontalAlignment.Right;
```

4.4. Запустіть програму на виконання й перевірте функціональність текстового поля ***Сума*** на формі ***Продажі виробника***. В ньому відображається сумарна вартість продажів товарів виробника, що вказаний у полі зі списком (рис. 8.48).

4.5. Закрийте форми ***Продажі виробника*** та ***Хліб*** і збережіть зміни, що зроблені в проекті.

5. Для відображення структури продажів товарів вибраного виробника у вигляді об'ємної кругової діаграми виконайте таке:

5.1. Додайте на форму елемент **Chart** з ім'ям ***chartСтруктура***.

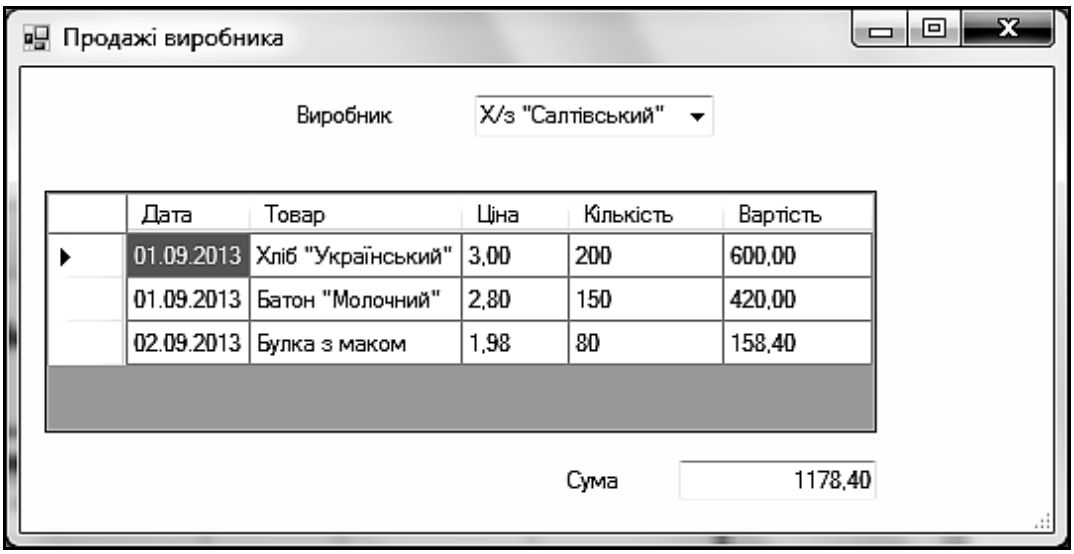


Рис. 8.48. **Форма *Продажі виробника* з полем *Сума***

5.2. Двічі клацніть у вільному місці форми ***Продажі виробника*** й у вікні коду форми додайте у розділ просторів імен такий оператор:

```
using System.Windows.Forms.DataVisualization.Charting;//Для діаграми
```

у розділ спільних об'єктів класу такий оператор:

```
BindingSource bindingСтруктура = new BindingSource();
```

а в кінець коду тіла оброблювача події ***formПродажіВиробника*** такі оператори:

```
    //***********
    // Діаграма *
    //***********

    //Запит
    var queryСтруктура = from товар in queryПродажіВиробника
    group товар by товар.Товар into t
    select new
    {
        Товар = t.Key,
        Вартість = t.Sum(p => p.Вартість)
    };

    //Дані для відображення
    bindingСтруктура.DataSource = queryСтруктура.ToList();
    chartСтруктура.DataSource = bindingСтруктура;
    chartСтруктура.Series["Series1"].XValueMember = "Товар";
```

```
chartСтруктура.Series["Series1"].YValueMembers = "Вартість";

//Тип диаграми
chartСтруктура.Series["Series1"].ChartType = SeriesChartType.Pie;

//Підписи на диаграмі
chartСтруктура.Series["Series1"].Label = "#PERCENT{P0}";

//Об'ємний варіант (3D)
this.chartСтруктура.ChartAreas[0].Area3DStyle.Enable3D = true;

//Заголовок
chartСтруктура.Titles.Add("Заголовок");
chartСтруктура.Titles[0].Text = "Структура продажів\n щодо виробника";
chartСтруктура.Titles[0].Font = new Font("Times New Roman", 12,
    FontStyle.Bold);
chartСтруктура.Titles[0].ForeColor = Color.Red;

//Легенда
chartСтруктура.Series["Series1"].IsVisibleInLegend = true;
chartСтруктура.Series["Series1"].LegendText = "#VALX";
```

5.3. Запустіть програму на виконання й перевірте функціональність діаграми на формі ***Продажі виробника***. В ній відображається структура продажів товарів вибраного виробника у вигляді об'ємної кругової діаграми (рис. 8.49).

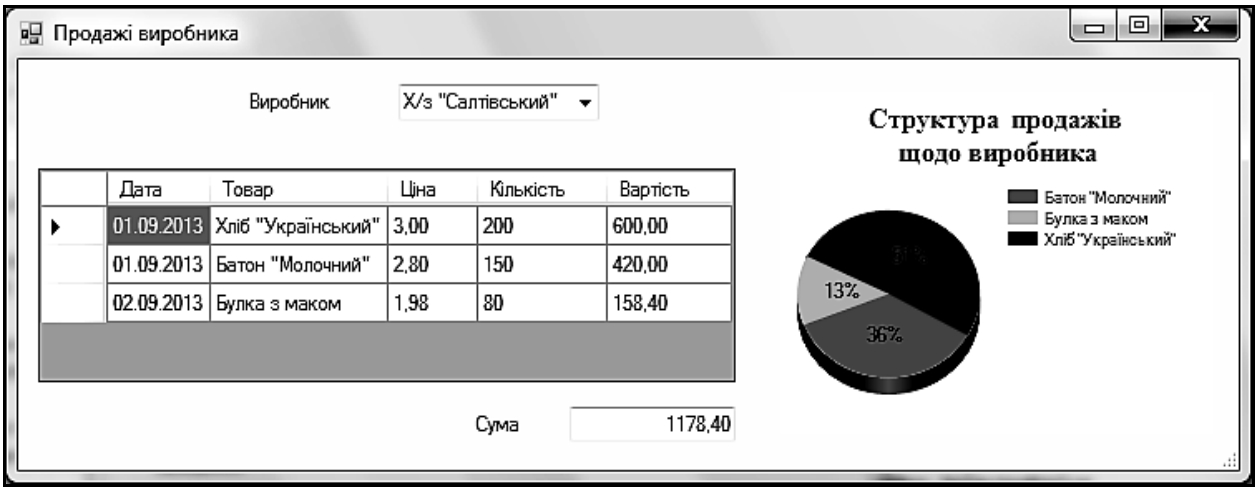


Рис. 8.49. **Форма *Продажі виробника* з діаграмою**

5.4. Закрийте форми ***Продажі виробника*** та ***Хліб*** і збережіть зміни, що зроблені в проекті.

6. Для відображення результатів продажів товарів вибраного виробника у разі його зміни виконайте таке:

6.1. Перейдіть у вікно конструктора форми ***Продажі виробника***.

6.2. Двічі клацніть на полі зі списком ***Виробник*** і у вікні коду форми введіть оператори тіла оброблювача події ***ManufacturerID ComboBox_SelectedIndexChanged***. Він має такий вигляд:

```
private void manufacturerIDComboBox_SelectedIndexChanged(object sender,
        EventArgs e)
{
     if (this.CanFocus) //Після завантаження
     {
         var queryПродажіВиробника =
             from продаж in context.Sales
             where продаж.ManufacturerID ==
                 (int)manufacturerIDComboBox.SelectedValue
             select new
             {
               Дата = продаж.Дата,
               Товар = продаж.Product.Товар,
               Ціна = продаж.Product.Ціна,
               Кількість = продаж.Кількість,
               Вартість = продаж.Product.Ціна * продаж.Кількість
             };

         // Відображаємо дані
         bindingПродажіВиробника.DataSource =
             queryПродажіВиробника.ToList();

         //Сума
         decimal Сума = (decimal)queryПродажіВиробника.Sum(p =>
             p.Вартість);
         textBoxСума.Text = Сума.ToString("0.00");

         //***********
         // Діаграма *
         //***********

         //Запит
         var queryСтруктура =
             from товар in queryПродажіВиробника
             group товар by товар.Товар into t
             select new
             {
                 Товар = t.Key,
                 Вартість = t.Sum(p => p.Вартість)
             };
         //Дані для відображення
         bindingСтруктура.DataSource = queryСтруктура.ToList();
         if (queryСтруктура.Count() > 0)
```

```
        {
            chartСтруктура.DataSource = bindingСтруктура;
            chartСтруктура.Series["Series1"].XValueMember = "Товар";
            chartСтруктура.Series["Series1"].YValueMembers = "Вартість";
            chartСтруктура.Series["Series1"].ChartType = SeriesChartType.Pie;
        }
    }
}
```

6.3. Запустіть програму на виконання й перевірте функціональність поля зі списком ***Виробник***. У разі зміни виробника на формі відображаються результати продажів товарів виробника, що вказаний у полі зі списком.

6.4. Закрийте форми ***Продажі виробника*** та ***Хліб*** і збережіть зміни, що зроблені в проекті.

Завдання для самостійного виконання

1. Проаналізувати проект ***моделювання*** після додавання проекту ***winForms***. Які елементи з першого проекту можна видалити без втрати функціонування другого проекту? Перевірити експериментальним шляхом зроблені висновки.

2. Додати в застосування форму ***Виробники***, у якій будуть відображатися й змінюватися дані таблиці ***Manufacturers***.

3. Додати в базу таблицю ***Групи_товарів*** із полями ***Група_товарівID*** та ***Група_товарів***, пов'язавши її з таблицею ***Товари*** відношенням один-до-багатьох. Використати для цього засоби міграції.

4. Додати стовпець ***Ціна*** в елемент ***invoiceProductsDataGridView*** на формі ***Накладні***.

5. Додати в проект ***winForms*** такі форми для задач аналізу даних ***Прайс*** та ***Продано*** (лаб. робота № 6) .

6. Додати на форму ***Продажі виробника*** стовпчикову діаграму для візуального аналізу динаміки продажів товарів кожного виробника. На горизонтальній осі подати дати продажів, а на вертикальній – вартості.

7. Виконати п.п. 1 – 3 ходу роботи для бази даних, що зберігається в окремому mdf-файлі.

8. Виконати п.п. 1 – 3 ходу роботи для індивідуальної бази даних.

Висновки

1. Технологія Code First, що базується на простих класах POCO (Plain Old CLR Objects), дає економний код, але потребує "ручного" кодування. Тому технології Model First та DB First рекомендуються для початківців, а Code First – для професійного використання.

2. З метою отримання більш ефективного коду у технології Code First додано класи DbContext і DbSet з дещо меншою функціональністю ніж відповідні їм класи ObjectContext і ObjectSet в технологіях Model First та DB First, але вимагають значно менше ресурсів.

3. Технологію Code First використовують для роботи з вже існючою і новою базою даних. Якщо бази даних ще немає, вона створюється автоматично на основі моделі під час виконання застосування. Якщо база даних вже існує, то створюють відповідні класи сутностей та доступу до даних і виконання операцій з цими об'єктами призводить до відповідних дій з даними у базі даних.

4. Операції завантаження бібліотек EntityFramework.dll та EntityFramework.SqlServer.dll, встановлення посилань на них та внесення даних у файл конфігурації проекту можна виконати за допомогою відповідного майстра, який називається NuGet Package Manager.

5. Якщо розробника застосування не влаштовують значення властивостей бази даних, що отримані за замовчуванням, можна використати засоби Data Annotations або Fluent API для налаштування взаємного відображення між моделлю та базою даних.

6. У процесі експлуатації бази даних може виявитися необхідність її змінити (додати нову таблицю чи стовпець, змінити властивості якогось стовпця, наприклад, збільшити розмір текстового поля). Щоб не втрачати дані в Code First використовують засобів міграції.

7. Під час створення застосувань в Code First використовують ті самі технології, що й в інших різновидах Entity Framework – з використанням вікна Data Source і без нього, з використанням класу BindingSource і без нього. Але в Code First ці технології мають свої особливості.

8. Коли на одній формі потрібно подавати дані ієрархічно пов'язаних таблиць, необхідно розширити ObservableCollection для додавання функціональності IListSource. Це вимагає доповнення до батьківського класу описом пов'язаної колекції сутностей дочірнього класу.

9. Використання збережених процедур у ADO.NET

9.1. Створення збережених процедур в ADO.NET та в ADO.NET Entity Framework

Збережені процедури становлять набір команд, що складається з декількох операторів SQL або функцій та зберігається в базі даних як компільований модуль.

Збережені процедури дуже схожі на звичайні процедури мов високого рівня, у них можуть бути вхідні і вихідні параметри і локальні змінні, в них можуть вироблятися числові обчислення й операції над символьними даними, результати яких можуть присвоюватися змінним і параметрам. У збережених процедурах можуть виконуватися стандартні операції з базами даних (як DDL, так і DML). Крім того, в збережених процедурах можливі цикли і розгалуження, тобто в них можуть використовуватися інструкції управління потоком.

Існує кілька видів збережених процедур.

Системні збережен іпроцедури призначені для виконання різних адміністративних дій. Практично всідії з адміністрування сервера виконуються з їх допомогою. Системні збережені процедури мають префікс sp_, містяться в системній базі даних і можуть бути викликані в контексті будь-якої іншої базиданих.

Користувальницькі збережені процедури реалізують ті чи інші дії. Збережені процедури – повноцінний об'єкт бази даних. Внаслідок цього кожна збережена процедура розташовується в конкретній базіданих, де і виконується.

Тимчасові збережені процедури існують лише деякий час, після чого автоматично знищуються сервером. Вони діляться на локальні та глобальні. Локальні тимчасові збережені процедури можуть бути викликані тільки з того з'єднання, в якому створені. При створенні такої процедури, їй необхідно дати ім'я, що починається з одного символу #. Як і всі тимчасові об'єкти, збережені процедури цього типу автоматично видаляються при відключенні користувача, перезапуск або зупинку сервера. Глобальні тимчасові збережені процедури доступні для будь-яких з'єднань сервера, на якому є така ж процедура. Для її визначення достатньо дати їй ім'я, що починається з символів ##. Видаляються ці процедури

при перезапуску або зупинці сервера, а також при закритті з'єднання, в контексті якого вони були створені.

Використання збереженої процедури можливе при наявності її в базі даних. Якщо програмний продукт базується на технології ADO.NET, то вже можна викликати збережену процедуру. Якщо доступ до даних здійснюється за допомогою ADO.NET Entity Framework, то спочатку необхідно додати її до ***Entity Model*** та ***EntityContainer***. Таким чином, процес створення збереженої процедури містить такі етапи:

створення збереженої процедури в базі даних.

додавання збереженої процедури до ***Entity Model*** (***EntityModel.Store***).

імпорт процедури до ***EntityContainer***.

Запитання і завдання

1. Опишіть, які види збережених процедур використовуються в програмних продуктах.

2. Який порядок створення і використання збережених процедур в ADO.NET Entity Framework?

9.2. Виклик збережених процедур

Створені збережені процедури використовуються з допомогою викликів, які можуть здійснюватись як в базі даних (в іншій процедурі або тригері), так і в різноманітних програмних продуктах.

Слід розглянути приклади виклику збережених процедур.

Приклад 9.1. Визначення данних про суму грошей, яка отримана при продажі товарів кожного дня.

```
CREATE PROCEDURE dbo.Продажі_товарів
AS
SELECT  Sales.Дата, SUM(Sales.Кількість*Products.Ціна) AS Сума
FROM Sales, Products
    WHERE (Sales.Код_товару = Products.Код_товару)
Groupby Дата
         RETURN
```

Виклик збереженої процедури здійснюється за її ім'ям. Для звернення до процедури ***Продажі_товарів*** можна використовувати таку команду:

```
EXEC Продажі_товарів
```

При необхідності виклику збереженої процедури з програмного продукту, який використовує Entity Data Model можна скористатися таким кодом:

```
ПродуктEntities db = new ПродуктEntities();
……
this.dataGridView_Продажі_товарів.DataSource = db.Продажі_товарів();
```

У даному фрагменті коду здійснюється виклик збереженої процедури ***Продажі_товарів,*** результат виконання відображується в компоненті ***dataGridView_Продажі_товарів.***

Запитання і завдання

1. Наведіть приклади запитів до індивідуальної бази даних, які краще виконувати за допомогою збережених процедур.

2. Виконайте запит із програмного продукту з використанням збереженої процедури для визначення найкращого товару кожного дня.

9.3. Параметри збережених процедур

У процесі розробки програмних продуктів часто виникає необхідність використовувати параметри збережених процедур. При цьому параметри можуть використовуватись як для передачі даних до збереженої процедури, так і для повернення результатів до застосування, яке цю процедуру викликало.

Слід розглянути приклади виклику збереженої процедури з вхідними та вихідними параметрами.

Приклад 9.2. Зменшення ціни товару на задану величину.

```
CREATE PROCEDURE Зменшення_ціни
@tovar VARCHAR(20),
@p DOUBLE
  AS
UPDATE Products SET Ціна=Ціна-@p
WHERE Товар= @tovar
```

Збережена процедура ***Зменшення_ціни*** для виконання обчислень використовує два вхідні параметри ***@Tovar*** і ***@p***. Для звернення до процедури можна скористатись будь-якою з команд:

```
EXEC my_proc4 ‘Батон',0.35
EXEC my_proc4 @tovar = ‘Батон', @p=0.35
```

При необхідності виклику збереженій процедури з програмного продукту, який використовує EntityDataModel, можна скористатися таким кодом:

```
ПродуктEntities db = new ПродуктEntities();
……
string tovar = "Батон";
double cina = 0.35;
dataGridView_Продажі.DataSource = db.Зменшення_ціни(tovar, cina);
```

Приклад 9.3. Визначення товару, який найбільше продається.

```
ALTER PROCEDURE  dbo.Кращій_товар
@tovar VARCHAR (20) OUTPUT
  AS
SELECT  @Name = Products.Товар
FROM Products
WHERE Products.Код_товаруIN
         (SELECT Sales.Код_товару
              FROM Sales
              GROUP BY Код_товару
                HAVING SUM(Кількість) =
                   (SELECT MAX(Summa)
                    FROM (SELECT SUM(Кількість) AS Summa
                       FROM Sales
                       GROUP BY Код_товару) AS Таблиця))
                        RETURN
```

Збережена процедура ***Кращий_товар*** повертає один параметр ***tovar***, який містить ім'я товару.

Для виклику збереженої процедури з програмного продукту, який використовує Entity Data Model, можна скористатися таким кодом:

```
ПродуктEntities db = new ПродуктEntities();
……
ObjectParameter name = newObjectParameter("Name", typeof(String));
    db.Кращій_товар(name);
this.textBox_Кращій_товар.Text = name.Value.ToString();
```

У даному фрагменті коду створюється параметр ***name*** для отримання імені товару з процедури ***Кращій_товар,*** результат виконання відображується в компоненті ***textBox_Кращій_товар.***

Запитання і завдання

1. У яких випадках використовують збережені процедури з параметрами?

2. Опишіть особливості використання збережених процедур із вхідними і вихідними параметрами в програмних продуктах.

9.4. Особливості оновлення даних із використанням збережених процедур

Платформа Entity Framework дозволяє використовувати збережені процедури в Entity Data Model для виконання операцій додавання, оновлення та видалення інформації з таблиць бази даних. При цьому, існуючу в базі процедуру оновлення даних, спочатку необхідно додати до ***EntityModel***. Потім необхідно встановити відповідності між параметрами процедури та стовпцями сутності ***EntityModel***. Таким чином, процес створення збереженої процедури містить три такі етапи:

1. Створення збереженої процедури в базі даних.
2. Додавання процедури до ***EntityModel (EntityModel.Store***).
3. Зіставлення сутності ***EntityModel*** зі збереженою процедурою.

Приклад 9.4. Процедура додавання запису до таблиці ***Sales.***

```
ALTER PROCEDURE dbo.insert_Продажі
    @data date,
    @cod_tovar int,
    @cod_virob int,
    @kilkist smallint
  AS
INSERT INTO Sales(Дата, Код_товару, Код_виробника, Кількість)
            values(@data, @cod_tovar, @cod_virob, @ kilkist)
if @@rowcount=0 return 1
         SELECT   SCOPE_IDENTITY() Код_продажі
         RETURN 0
```

Збережена процедура виконує такі функції:

обчислює значення ***Код_продажі***, що становить наступне число;

виконує вставку нового рядка;

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

У процесі оновлення даних у таблиці не слід спеціально викликати процедуру додавання або видалення інформації. Ці дії виконає за користувача ***Entity Model*** автоматично. Достатньо тільки скористатися методом ***SaveChanges()***:

```
ПродуктEntities db = new ПродуктEntities();
……
    db.SaveChanges();
```

У ході використання збережених процедур для оновлення інформації в таблиці бази даних слід мати на увазі, що необхідно створювати три процедури – додавання, оновлення та видалення даних (якщо створена одна з процедур, то обов'язково створити і останні дві процедури).

Запитання і завдання

1. Які особливості має програмний продукт, що використовує збережені процедури для оновлення інформації в базі даних?

2. Створіть збережені процедури, що реалізують видалення та оновлення даних у таблиці індивідуальної бази даних.

9.5. Переваги та недоліки використання збережених процедур

Виконання в базі даних збережених процедур дає користувачеві такі переваги:

усі оператор ипроцедури, що зберігається пройшли етап синтаксичного аналізу і знаходяться у готовому для виконання форматі;

перед запуском збереженої процедури SQL Server генерує для неї план реалізації, виконує її оптимізацію та компіляцію;

збережені процедури підтримують модульне програмування, оскільки дозволяють розбивати великі завдання на самостійні, більш дрібні та зручні в управлінні частини;

збережені процедури можуть викликати інші збережені процедури і функції;

збережені процедури можуть бути викликані з прикладних програм інших типів;

як правило, збережен іпроцедури виконуються швидше, ніж запити, які формуються в коді застосування;

збережені процедури простіше використовувати – вони можуть складатися з десятків і сотень команд, але для їх запуску достатньо вказати всього лише ім'я необхідної процедури, це дозволяє зменшити розмір запиту, що посилається від клієнта на сервер, а значить, і навантаження на мережу.

Збережені процедури існують незалежно від таблиць або яких-небудь інших об'єктів баз даних. Вони можуть викликатися клієнтською програмою, другою збереженою процедурою або тригером.

З метою поліпшення безпеки роботи програми вся обробка даних ведеться шляхом виклику збережених процедур. Подібний підхід забезпечує простоту модифікації алгоритмів обробки даних і дає можливість розширення програмного продукту без внесення зміни до самого застосування.

Розробник збереженої процедури може управляти правами доступу до неї – дозволяти або забороняти її виконання. Змінювати код збереженої процедури може тільки її власник або користувач, який володіє тими ж правами. При необхідності можна передавати права володіння збереженої процедурою від одного користувача до іншого.

Платформа Entity Framework дозволяє використовувати збережені процедури в Entity Data Model, замість або в поєднанні, з автоматичним створенням команд. При розробці моделі можна:

вказати, що Entity Data Model повинна використовувати існуючі збережені процедури для вставки, оновлення або видалення об'єктів;

створити шаблонні функції для виклику збереженої процедури, які повертають об'єкти або результати, що не відповідають жодному із суб'єктів моделі;

визначити збережену процедуру для усунення необхідності прямого доступу до таблиці;

забезпечити найбільш швидку перевірку оновлюваних даних на паралелізм.

Запитання і завдання

1. Які переваги має програмний продукт, що використовує збережені процедури?

2. Які можливості забезпечує використання збережених процедур в Entity Data Model?

9.6. Особливості використання збережених процедур для контролю конфліктів паралельної обробки даних

Коли велика кількість застосувань і процесів намагаються одночасно модифікувати інформацію в таблицях бази даних, необхідно використовувати механізм контролю, здатний вирішувати конфлікти й ізолювати один від одного конкуруючі запити.

Розробнику необхідно перерахувати різні конфліктні ситуації, які можуть виникнути при оновленні даних, наприклад:

рядок, який необхідно оновити, може бути видалений іншим користувачем;

у рядку, що треба ввести, наявне відношення за зовнішнім ключем з іншим рядком, який до того часу може бути видалений іншим користувачем;

рядок, який треба оновити, вже оновлений іншим користувачем, але не оновлено стовпець, що цікавить саме вас, чи потрібно виконувати оновлення або заборонити його?

Для того щоб вирішити, який підхід підходить найкраще, в конкретній проблемній ситуації, розробнику програмного забезпечення необхідно подбати про методологію виявлення конфліктів.

Якщо програмний продукт потребує запобігання випадкової втрати даних у результаті конфліктів одночасного доступу, одним із методів вирішення проблеми є блокування таблиць. Це називається песимістичним паралелізмом (pessimistic concurrency).

Управління блокуваннями має свої недоліки. Програмування може бути занадто складним, блокування потребують серйозних ресурсів бази даних, і накладні витрати з завантаження зростають в міру зростання кількості користувачів бази даних. У зв'язку з цим не всі СУБД підтримують песимістичний паралелізм.

Блокування рядків суперечить загальній філософії ADO.NET, де підключаться потрібно якомога пізніше, а відключатися – якомога раніше. Тепер базі даних доведеться виконувати не тільки "захист рядка" (або сторінки, або таблиці), а й постійно очікувати запит розблокування цього рядка.

Таким чином, ADO.NET заохочує використання автономної архітектури, яка ніяк не в'яжеться з блокуванням ресурсів на рівні бази даних.

Хоча за допомогою об'єкта Command можна виконати специфічну для СУБД команду на зразок такої (команда SQL Server):

```
Select * from Виробник HOLDLOCK where Код_виробника = 10
```

Використовуючи в транзакції таку команду, просто буде блоковано рядок і перешкоджено всім іншим користувачам оновити його, поки не буде зафіксовано або відкочено транзакцію.

Більше того, може виникнути серйозна проблема, коли одне блокування призводить до іншого блокування. Можна отримати взаємне блокування рядків, що потребує втручання адміністратора бази даних, який почистить і знищить такі транзакції для всіх інших користувачів.

Як альтернатива песимістичному паралелізму виступає оптимістичний паралелізм (optimistic concurrency). Optimistic concurrency дозволяє конфлікту одночасного доступу трапитися, але допомагає адекватно зреагувати на подібні ситуації.

Оптимістичний паралелізм передбачає, що в блокуванні ресурсу з метою запобігання порушення цілісності даних немає необхідності; замість цього він покладається на різні схеми перевірки даних перед фактичним оновленням, видаленням або вставкою. Якщо рядок був змінений, оновлення або видалення завершується невдачею, і необхідно повторити спробу. Рядок може бути заблокований на короткий час виконання команди, але це не так погано, як песимістичне блокування, при якому рядки блокуються від первісної вибірки і до остаточного поновлення, введення або видалення.

Існує кілька варіантів оптимістичного паралелізму:

залишається остання зміна – в базі даних запам'ятовується останнє оновлення, це найпростіша схема оптимістичного паралелізму, і для її виконання нічого не потрібно робити;

перевірка всіх стовпців перед оновленням – перед збереженням змін у базі даних слід перевіряти, чи виконав поновлення будь-який інший користувач у проміжку часу між вибіркою даних та їх збереженням, але такий запит може бути ресурсоємним;

перевірка перед оновленням тільки змінених стовпців і первинних ключів, при цьому необхідно щоразу заново формулювати запит;

перевірка відміток часу – в таблицю в базі даних додається стовпець, який контролює момент поновлення даних, змінюючись кожен раз при виконанні над рядком операції DML. Тип даних такого стовпця зазвичай ***timestamp***, але насправді він не зберігає дату або час. Замість цього, значення дорівнює цифрі, що збільшується на одиницю при кожному оновленні даних. Контроль змін стовпця типу ***timestamp*** можна покласти на збережену процедуру поновлення даних таблиці.

Запитання і завдання

1. Які переваги і недоліки має програмний продукт, що використовує песимістичний паралелізм?
2. Наведіть приклади ситуацій, в яких можливе блокування даних при використанні песимістичного паралелізму.
3. Наведіть приклади варіантів використання оптимістичного паралелізму.

9.7. Тригери та їх властивості

Тригер – це процедура, що зберігається, виконання якої обумовлено настанням певних подій всередині реляційної бази даних.

Запускання тригера відбувається при виконанні заздалегідь визначеного оператора мови маніпулювання даними (DML) таблиці. Тригери використовуються для перевірки цілісності даних, а також для відкату транзакцій.

Застосування тригерів здебільшого вельми зручно для користувачів бази даних. І все ж їх використання часто пов'язано з додатковими витратами ресурсів на операції введення/виведення. У тому випадку, коли тих же результатів (з набагато меншими непродуктивними витратами ресурсів) можна домогтися за допомогою збережених процедур або прикладних програм застосування тригерів недоцільно.

Відмінні особливості тригерів:

кожен тригер прив'язується до конкретної таблиці;

тригер є спеціальним типом збережених процедур, що запускаються сервером автоматично при спробі зміни даних у таблицях, з якими тригери пов'язані;

всі модифікації даних, в яких він "бере участь", розглядаються як одна транзакція, в разі виявлення помилки або порушення цілісності да-

них, відбувається відкат цієї транзакції, тим самим внесення змін забороняється, скасовуються також всі зміни, які вже зроблені тригером;

створює тригер тільки власник бази даних, це обмеження дозволяє уникнути випадкового зміни структури таблиць, а також способів зв'язку з ними інших об'єктів.

Тригер є дуже корисним і водночас небезпечним засобом. Так, при неправильній логіці його роботи можна легко знищити цілу базу даних, тому тригери необхідно дуже ретельно налагоджувати.

На відміну від звичайної підпрограми, тригер виконується неявно в кожному випадку виникнення події, яка його запускає, до того ж він не має аргументів. За допомогою тригерів виконуються такі завдання:

перевірка коректності введених даних і виконання складних обмежень цілісності даних, які важко підтримувати за допомогою обмежень цілісності, що встановлені для таблиці;

видача попереджень , що нагадують про необхідність обов'язкового виконання деяких дій у процесі оновлення таблиці;

накопичення аудиторської інформації за допомогою фіксації відомостей про внесені зміни і тих особах, які їх виконали.

Існує два параметри, що визначають вигляд і поведінку тригерів:

AFTER-тригер виконується після успішного завершення команд, які його викликали, якщо ж команди з якоїсь причини не можуть бути успішно завершені, тригер не виконується. Можна визначити кілька AFTER-тригерів для кожної операції (INSERT, UPDATE, DELETE). Якщо для таблиці передбачено виконання кількох AFTER-тригерів, то за допомогою системної збереженої процедури sp_settriggerorder можна вказати, який з них буде виконуватися першим, а який останнім. За замовчуванням в SQL Server всі тригери є AFTER-тригерами.

INSTEAD OF-тригер викликається замість виконання команд. На відміну від AFTER-тригера – INSTEAD OF-тригер може бути визначений як для таблиці, так і для подання. Для кожної операції INSERT, UPDATE, DELETE можна визначити тільки один INSTEAD OF-тригер.

Слід зазначити, що зміни даних у результаті виконання запиту користувача і виконання тригера здійснюється в тілі однієї транзакції: якщо відбудеться відкочування тригера, то будуть відхилені і зміни користувача.

Тригери розрізняють за типом команд, на які вони реагують. Існує три види тригерів:

INSERT TRIGGER – запускаються при спробі вставки даних за допомогою команди INSERT;

UPDATE TRIGGER – запускаються при спробі зміни даних за допомогою команди UPDATE;

DELETE TRIGGER – запускаються при спробі видалення даних за допомогою команди DELETE.

При створенні тригера повинна бути вказана хоча б одна команда, на яку він буде реагувати. Допускається створення тригера, що реагує на дві або на всі три команди.

Усередині тригера не допускається виконання ряду операцій створення, зміни та видалення бази даних, а також відновлення резервної копії бази даних або журналу транзакцій.

У процесі виконання команд додавання, зміни та видалення записів сервер створює дві спеціальні таблиці inserted і deleted. У них містяться списки рядків, які будуть вставлені або видалені по завершенні транзакції. Структура таблиць inserted і deleted ідентична структурі таблиць, для якої визначається тригер. Для кожного тригера створюється свій комплект таблиць inserted і deleted, тому до будь-якої іншої тригер не зможе отримати доступ. Залежно від типу операції, що викликала виконання тригера, вміст таблиць inserted і deleted може бути різним:

команда INSERT – у таблиці inserted містяться всі рядки, які користувач намагається вставити в таблицю, в таблиці deleted не буде ні одного рядка, після завершення тригера всі рядки з таблиці inserted перемістяться у вихідну таблицю;

команда DELETE – у таблиці deleted будуть міститися всі рядки, які користувач спробує видалити, тригер може перевірити кожен рядок і визначити, чи дозволено її видалення, у таблиці inserted не опиниться жодного рядка;

команда UPDATE – при її виконанні в таблиці deleted знаходяться старі значення рядків, які будуть видалені при успішному завершенні тригера, нові значення рядків містяться в таблиці inserted, ці рядки додадуться у вихідну таблицю після успішного виконання тригера.

За умови правильного використання, тригери можуть стати дуже потужним механізмом. Основна їх перевага полягає в тому, що стандартні функції зберігаються всередині бази даних і узгоджено активізуються при кожному її відновленні. Це може істотно спростити додатки. Проте слід знати про притаманні тригеру недоліки:

під час переміщення деяких функцій в базу даних ускладнюються завдання її проектування, реалізації та адміністрування;

перенесення частини функцій в базу даних і збереження їх у вигляді одного або декількох тригерів іноді призводить до приховування від користувача деяких функціональних можливостей. Хоча це певною мірою спрощує його роботу, але може стати причиною шкідливих побічних ефектів, оскільки в цьому випадку користувач не в змозі контролювати всі процеси, що відбуваються в базі даних;

перед виконанням кожної команди щодо зміни стану бази даних, СУБД повинна перевірити тригерні умови з метою з'ясування необхідності запуску тригера для цієї команди. Виконання подібних обчислень позначається на загальній продуктивності СУБД, а в моменти пікового навантаження її зниження може стати особливо помітним. Очевидно, що при зростанні кількості тригерів збільшуються і накладні витрати, пов'язані з такими операціями.

Неправильно написані тригери можуть призвести до серйозних проблем, таких, наприклад, як поява "мертвих" блокувань. Тригери здатні тривалий час блокувати безліч ресурсів, тому слід звернути особливу увагу на зведення до мінімуму конфліктів доступу.

Приклад 9.5. Контроль додавання запису в таблицю **Products**.

Команда додавання запису в таблицю **Products** може бути такою:

```
INSERT INTO Products VALUES (‘Батон', 7.95, 7.55)
```

Тригер обробляє тільки один запис, який додається до таблиці. При цьому він контролює ціну закупки батонів, яка не може перевищувати 7 грн.

```
CREATE TRIGGER Триггер_ins
ON Products FOR INSERT
AS
IF @@ROWCOUNT=1
BEGIN
 IF NOT EXISTS(SELECT * FROM inserted WHERE inserted.Ціна_закупівліки
>=7
    AND inserted.Товар = ‘Батон')
```

```
    BEGIN
       ROLLBACK
       PRINT  'Ціна батонів значно завищена'
    END
END
```

Запитання і завдання

1. У яких випадках зручно використовувати тригери? Наведіть приклади.

2. Опишіть, які види тригерів використовуються в базах даних.

3. Наведіть приклади ситуацій, в яких можливе блокування даних при використанні тригерів.

Лабораторна робота № 9. Розробка програм взаємодії з базами даних із використанням збережених процедур

Цілі лабораторної роботи:

1. Набуття практичних навичок використання концептуальних моделей.

2. Набуття практичних навичок створення збережених процедур за допомогою платформи Entity Framework.

3. Оволодіння засобами обробки конфліктів паралелізму на основі платформи Entity Framework.

4. Удосконалення практичних навичок відображення даних у застосуваннях на основі платформи Entity Framework.

5. Удосконалення навичок роботи з інтегрованим середовищем розробки Visual C# і довідковою системою Microsoft Developer Network (MSDN).

Перед виконанням лабораторної роботи студент повинен знати:

1. Основи проектування реляційних баз даних.

2. Основи побудови SQL-запитів.

3. Основи роботи з концептуальними моделями засобами Microsoft Visual Studio.

4. Основи роботи з технологією LINQ to Entities.

5. Принципи обробки подій в C#-програмі.

Після виконання лабораторної роботи студент повинен вміти:

1. Використовувати концептуальні моделі засобами Microsoft Visual Studio.

2. Самостійно розробляти C#-застосування на основі платформи Entity Framework з застосуванням процедур, що зберігаються.

3. Використовувати основні бібліотеки .Net Framework під час розробки програм.

Хід роботи

1. Створення і використання збереженої процедури з параметрами.

1.1. Створення збереженої процедури "Продаж товарів виробника".

1.2. Додавання збереженої процедури до Entity Data Model.

1.3. Імпорт процедури до Entity Container: ХлібEntities.

2. Використання можливостей ADO NET EF для контролю конфліктів паралелізму.

2.1. Додавання до таблиці Sales стовпця типу timestamp.

2.2. Оновлення моделі Entity Data Model у зв'язку із зміною таблиці Sales.

2.3. Створення збереженої процедури "update_Продажі".

2.4. Додавання процедури "update_Продажі" до Entity Model.

2.5. Зіставлення сутності Sales зі збереженою процедурою "*update_Продажі*".

2.6. Створення форми "Обробка конфліктів оновлення даних" для роботи з результатами виконання процедури.

2.7. Відображення даних таблиць Products, Sales на формі "Обробка конфліктів оновлення даних".

2.8. Додавання функції оновлення даних у БД.

2.9. Додавання функції обробки конфліктів паралелізму.

3**.** Використання тригера для контролю змінення даних у таблиці.

3.1. Створення таблиці Gurnal.

3.2. Оновлення моделі Entity Data Model у зв'язку з додаванням таблиці Gurnal*.*

3.3. Створення тригера для контролю оновлення записів у таблиці Invoices.

3.4. Створення збереженої процедури для запису в журнал інформації про оновлення даних у таблиці Invoices.

3.5. Створення форми Журнал для відображення даних із таблиці Gurnal.

3.6. Відображення даних таблиці Gurnal на формі Журнал.

3.7. Додавання коду формування повідомлень користувачу про успішність оновлення даних.

Інструкції

1. Створення і використання збереженої процедури з параметрами

Постановка задачі

Однією з ключових особливостей платформи Entity Framework є те, що за допомогою EDM доступ до даних здійснюється також просто, як до властивостей класу. Інший – те, що можна автоматично генерувати команди бази даних для виконання запитів, а також змін у базі даних. Це дозволяє істотно підвищити продуктивність при розробці програмних продуктів, але далеко не завжди приводить до підвищення продуктивності самого програмного продукту. Для усунення цього недоліку, особливо при виконанні складних запитів, використовуються збережені процедури, при запуску яких достатньо вказати всього лише її ім'я і параметри збереженої процедури, що дозволяє зменшити розмір запиту, що посилається від клієнта на сервер.

Вивчення засобів використання збережених процедур у ADO.NET EF проводиться шляхом виконання запиту до бази даних ***Хліб.mdf***, який дозволяє отримати інформацію про продажі обраного товару заданого виробника.

На початковому етапі в програмному продукті виконуються операції з таблицями ***Manufacturers***, ***Products*** та ***Sales***. При цьому застосування складається із двох форм:

форма ***Хліб*** (рис. 9.1), яка слугує для керування роботою застосування, на ній розміщаються кнопки для виклику функціональних форм;

форма ***Продаж товарів виробника***, на якій відображаються дані про продаж кожного дня товару виробника (рис. 9.2).

Щоб змінити або додати дані про продажі, необхідно вибрати товар, вибрати виробника, виконати необхідні дії в таблиці форми та натиснути кнопку ***Зберегти***.

У даній частині лабораторної роботи створення застосування складається з таких завдань:

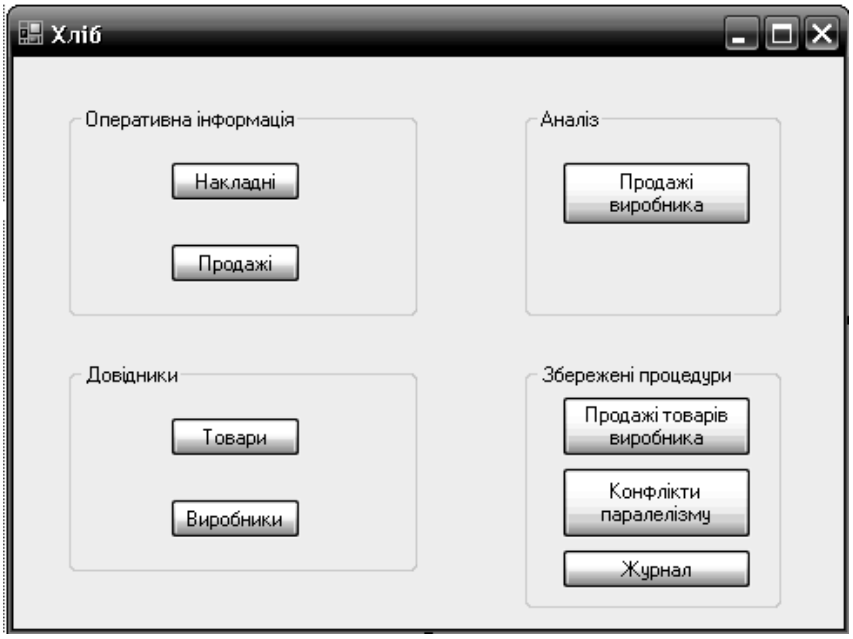


Рис. 9.1. **Форма *Хліб***

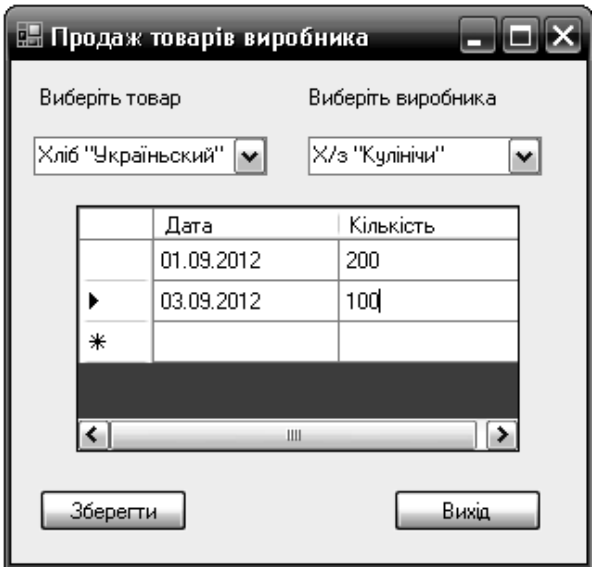


Рис. 9.2. **Форма *Продаж товарів виробника***

4. Створення збереженої процедури ***Продаж_товарів_виробника***.

5. Додавання збереженої процедури до Entity Model (ХлібModel.Store).

6. Імпорт процедури до ***EntityContainer: ХлібEntities***.

7. Створення форми ***Продаж_товарів_виробника*** для роботи з результатами виконання процедури.

8. Відображення результатів виконання процедури на формі ***Продаж_товарів_виробника***.

9. Створення оброблювача події для збереження змін даних.

10. Створення оброблювача події для закриття форми.

1.1. Створення збереженої процедури *Продаж_товарів_виробника*

Виконання

1. Відкрийте проект Windows Forms з ім'ям ***winХліб***.

2. У проекті ***winХліб*** відкрийте вкладку **Server Explorer**, а в ньому папку ***Хліб.mdf***.

3. У відкритому списку натисніть правою кнопкою миші на папці **Stored Procedures** та виберіть пункт меню **Add New Stored Procedure** (рис. 9.3).

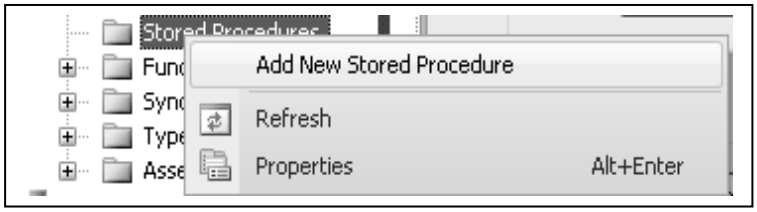


Рис. 9.3. **Контекстне меню для створення збереженої процедури**

4. У відкритому вікні введіть такий код збереженої процедури:

```
ALTER PROCEDURE Продаж_товарів_виробника
    @Cod_tov int,
    @Cod_proiz int
    AS
        Select *
             From Sales
             Where (Sales.Код_товару = @Cod_tov)
            AND(Sales.Код_виробника = @Cod_proiz)
    RETURN
```

5. Закрийте вікно створення процедури та поверніться у вікно проекту. У папці **Stored Procedures** вікна **Server Explorer** відображається ім'я процедури ***Продаж_товарів_виробника*.**

1.2. Додавання збереженої процедури до Entity Data Model

Виконання

1. У вікні **Solution Explorer** відкрийте папку ***ХлібModel.edmx***.

2. У вікні зображення моделі бази даних натисніть праву кнопку миші та виберіть пункт **Update Model from Database**.

3. У вкладці **Add** відкритого вікна **Update Wizard** виберіть список **Stored Procedures**, а в ньому ім'я процедури ***Продаж_товарів_виробника*** і натисніть кнопку **Finish**. При успішному додаванні процедури у вікні ***Output*** з'явиться таке повідомлення:

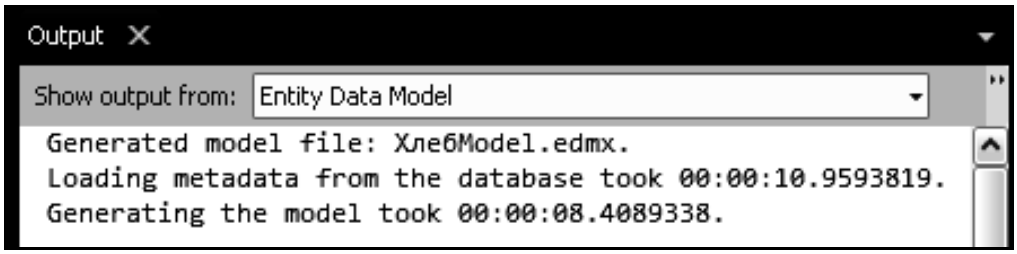


Рис. 9.4. **Повідомлення про успішне додавання процедури до моделі**

У папці **Stored Procedures** розділу ***ХлібModel.Store*** вікна ***Model Browser*** відображається ім'я процедури ***Продаж_товарів_виробника*** (рис. 9.5).

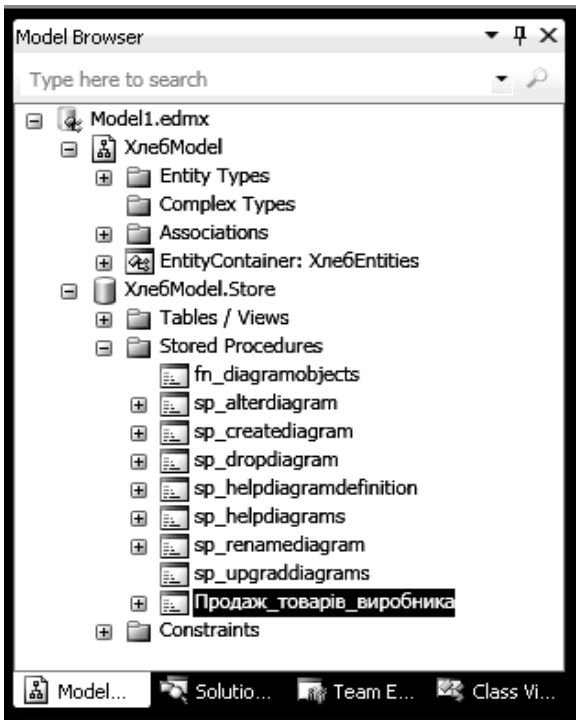


Рис. 9.5. **Вікно *Model Browser***

1.3. Імпорт процедури до EntityContainer: ХлібEntities

Виконання

1. У вікні ***Model Browser*** (рис. 9.5) відкрийте список об'єкта ***EntityContainer: ХлібEntities*** і натисніть праву кнопку на папці **Function Imports.**

2. У розкритому меню виберіть пункт **Add Function Import**.

3. Вікно **Add Function Import** містить компонент **ComboBox** (заголовок **Stored Procedure Name**), у якому потрібно вибрати ім'я процедури ***Продаж_товарів_виробника*** (рис. 9.6).

4. У компоненті **TextBox** (заголовок **Function Import Name**) задається ім'я функції, яка імпортується (рис. 9.6).

5. У компоненті **GroupBox** з ім'ям **Returns a Collection Off** виберіть тип, який повертає процедура (один із компонентів **RadioButton**). У даному випадку збережена процедура повертає різноманітні дані, тому обирається **RadioButton** з ім'ям **Entities**. У вікні поруч із цим компонентом оберіть тип сутності. В прикладі обрано тип ***Sales*** (рис. 9.6).

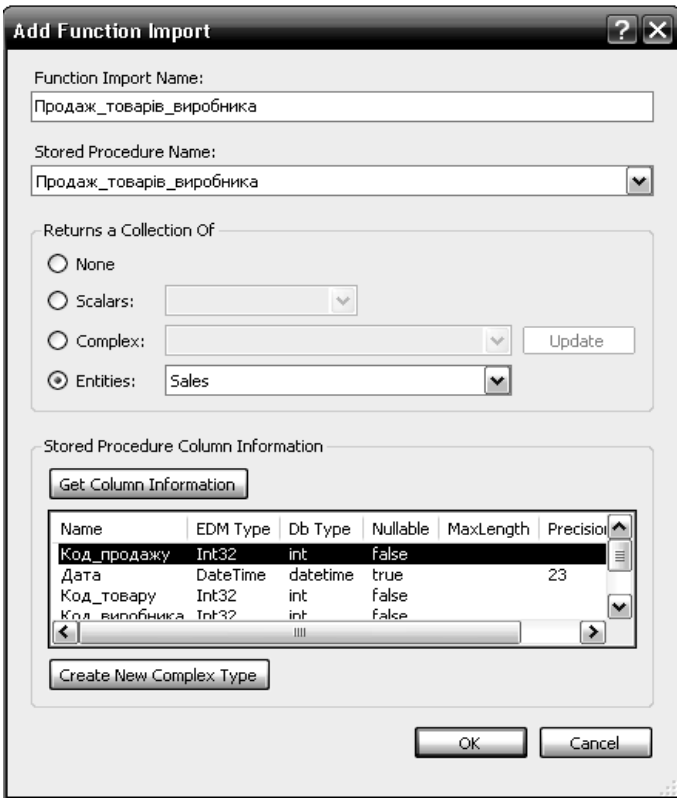


Рис. 9.6. **Вікно для додавання функції, що імпортується**

6. Натисніть кнопку **Get Column Information** для отримання інформації про дані, які повертаються (рис. 9.6).

Імпорт збереженої процедури завершується натисненням кнопки **OK** (рис. 9.6). У папці **Function Imports** об'єкта ***EntityContainer: ХлібEntities*** вікна ***Model Browser*** відображається ім'я процедури ***Продаж_товарів_виробника*** (рис. 9.7).

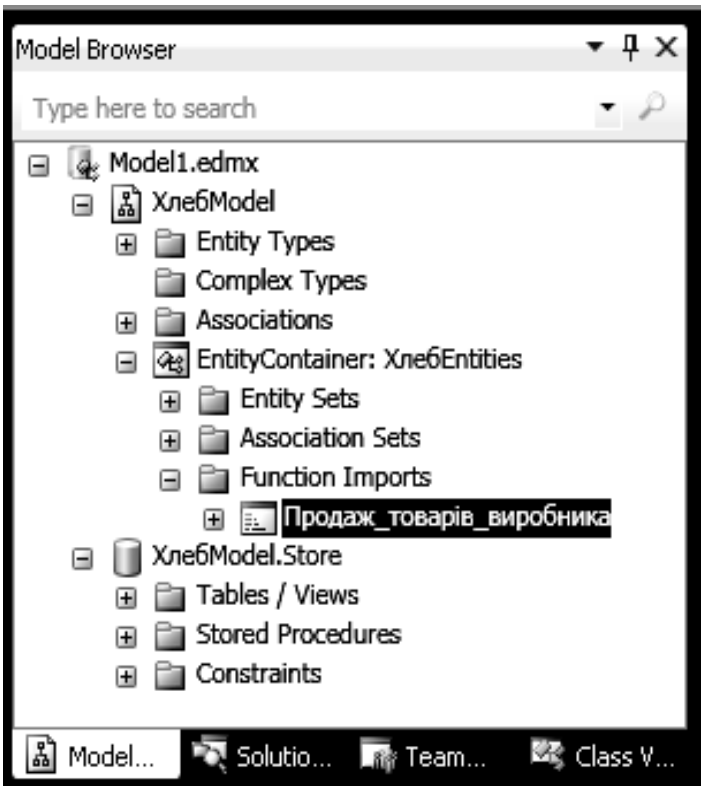


Рис. 9.7. **Результат додавання імпортованої функції у вікні *Model Browser***

Завдання 4. Створення форми *Продаж_товарів_виробника*

Виконання

1. У вікні ***Solution Explorer*** додайте форму з ім'ям ***Продаж_товарів_виробника***.

2. Додайте на форму два елементи **ComboBox** з ім'ям ***comboBox_Товар*** і ***comboBox_Виробник***.

3. Додайте елемент **DataGridView** з ім'ям ***dataGridView_Продажі*** на форму.

4. Додайте два елементи **Label** та установіть їх властивості **Text** відповідно ***Виберіть товар*** і ***Виберіть виробника***.

5. Додайте два елементи **Button**. Задайте кожному з них ім'я ***button_Зберегти*** і ***button_Вихід*** та установіть їх властивості **Text** відповідно ***Зберегти*** і ***Вихід***.

6. Розмістіть елементи на формі ***Обробка конфліктів оновлення даних*** відповідно до рис. 9.8.

7. У вікні **Solution Explorer** відкрийте конструктор форми ***Хліб*** і додайте кнопку для виклику форми ***Продаж_товарів_виробника***.

Завдання 5. Відображення результатів виконання процедури на формі *Продаж_товарів_виробника_Form*

Виконання

1. Для того, щоб посилатись на модель, яка створена на основі бази даних ***Хліб***, та простір імен сутностей до коду форми, додайте такі інструкції:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.Objects;
using System.Data.Objects.DataClasses;
using System.Data.EntityClient;

namespace winХліб
{
    public partial class Продаж_товарів_виробника_Form : Form
    {
        ХлібEntities db;
        // Змінні для завдання коду товару та коду виробника
        int cod_Tovara=2, cod_Virob=2;
        public Продаж_товарів_виробника_Form()
        {
            InitializeComponent();
        }
    }
```

2. Додайте код оброблювача події "Завантаження форми", щоб відображати в елементах ***comboBox_Товар*** і ***comboBox_Виробник*** відповідно до списку товарів та списку виробників під час відкривання форми.

2.1. Перейдіть у вікно конструктора форми ***Продаж_товарів_виробника*** й двічі клацніть у вільному місці форми.

2.2. Уведіть оператори тіла оброблювача події ***Продаж_товарів_виробника_Form_Load***:

```
private void Продаж_товарів_виробника_Form_Load (object sender,
    EventArgs e)
{
```

```
    db = new ХлебEntities();
    var продуктQuery = from d in db.Products select d;
    var виробникQuery = from c in db.Manufacturers select c;
        try
        {
        // Інструкції для виконання запиту і відображення найменування
        // товару в компонентах ComboBox форми

            this.comboBox_Товар.ValueMember = "Код_товару";
            this. comboBox_Товар.DisplayMember = "Товар";
            this. comboBox_Товар.DataSource = продуктQuery;
            this.comboBox_Виробник.ValueMember = "Код_виробника";
            this. comboBox_Виробник.DisplayMember = "Виробник";
            this. comboBox_Виробник.DataSource = виробникQuery;

            cod_Tovara = (int) comboBox_Товар.SelectedValue;
            cod_Virob = (int) comboBox_Виробник.SelectedValue;
            // Прив'язуємо результати запиту до елемента DataGridView
            dataGridView_Продажі.DataSource =
            db.Продаж_товарів_виробника(cod_Tovara, cod_Virob);
            dataGridView_Продажі.Columns["Код_продажу"].Visible = false;
            dataGridView_Продажі.Columns["Код_товару"].Visible = false;
            dataGridView_Продажі.Columns["Код_виробника"].Visible= false;
            dataGridView_Продажі.Columns["Manufacturers"].Visible= false;
            dataGridView_Продажі.Columns["Products"].Visible = false;
        }
        catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

3. Додайте код оброблювача події ***comboBox_Товар_SelectedIndexChanged***, щоб відображати в елементі ***DataGridView*** записи з таблиці ***Sales*** під час вибору назви товару в компоненті ***comboBox_Товар*** форми.

3.1. Перейдіть у вікно конструктора форми та двічі натисніть ліву кнопку миші на компоненті ***comboBox_Товар*** форми.

3.2. Додайте такий код до оброблювача події ***comboBox_Товар_SelectedIndexChanged***:

```
private void comboBox_Товар _SelectedIndexChanged(object sender,
    EventArgs e)
{
    try
    {
    // Наступний код виконується після завантаження
```

```
    // елемента управління, який може отримати фокус
        if (this.CanFocus)
        {
            cod_Tovara = (int) comboBox_Товар.SelectedValue;
            cod_Virob = (int) comboBox_Виробник.SelectedValue;
            // Прив'язуємо результати запиту до елемента DataGridView
            dataGridView_Продажі.DataSource =
                 db.Продаж_товарів_виробника(cod_Tovara, cod_Virob);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

4. Додайте код оброблювача події ***comboBox_Виробник_SelectedIndexChanged***, щоб відображати в елементі ***DataGridView*** записи з таблиці ***Sales*** під час вибору назви виробника в компоненті ***comboBox_Виробник*** форми.

4.1. Перейдіть у вікно конструктора форми та двічі натисніть ліву кнопку миші на компоненті ***comboBox_Виробник*** форми.

4.2. Додайте такий код до оброблювача події ***comboBox_Виробник _SelectedIndexChanged***:

```
private void comboBox_Виробник _SelectedIndexChanged(object sender,
    EventArgs e)
{
    try
    {
    // Наступний код виконується після завантаження
    // елементу управління, який може отримати фокус
        if (this.CanFocus)
        {
           cod_Tovara = (int) comboBox_Товар.SelectedValue;
           cod_Virob = (int) comboBox_Виробник.SelectedValue;
           // Прив'язуємо результати запиту до елемента DataGridView
           dataGridView_Продажі.DataSource =
               db.Продаж_товарів_виробника(cod_Tovara, cod_Virob);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Завдання 6. Створення оброблювача події для збереження змін даних

Виконання

1. У конструкторі форми ***Продаж_товарів_виробника_Form*** двічі клацніть на кнопці ***Зберегти***.

2. У оброблювач події ***зберегти_Click*** додайте такий код:

```
private void зберегти_Click(object sender, EventArgs e)
{
    try
    {
        db.SaveChanges();
        this.Refresh();
        MessageBox.Show("Зміни записані");
    }
    catch (UpdateException ex)
    {
        MessageBox.Show(ex.InnerException.Message);
    }
}
```

3. Для забезпечення врахування за замовчуванням значень ***Код_товару*** та ***Код_виробника*** з елементів ***comboBox_Товар*** і ***comboBox_Виробник*** у відповідних стовпцях ***dataGridView_Продажі*** створіть оброблювач події **DefaultValuesNeeded**.

3.1. У конструкторі форми ***Продаж_товарів_виробника_Form*** клацніть правою кнопкою миші на елементі ***dataGridView_Продажі***.

3.2. У контекстному меню виберіть вкладку **Властивості** (**Properties**).

3.3. У відкритому вікні виберіть події елемента ***dataGridView_Продажі***, клацнувши піктограму із зображенням блискавки.

3.4. Із списку подій вибрати ***DefaultValuesNeeded*** та записати навпроти нього ім'я події ***dataGridView_Продажі _DefaultValuesNeeded***.

3.5. У місті оброблювача події ***dataGridView_Продажі_Default-ValuesNeeded*** додати такий код:

```
private void dataGridView_Продажі_DefaultValuesNeeded(object sender,
    DataGridViewRowEventArgs e)
{
    //  Значення за замовчанням
    e.Row.Cells["Код_товару"].Value =
        (int)comboBox_Товар.SelectedValue;
```

```
    e.Row.Cells["Код_виробника"].Value =
        (int) comboBox_Виробник.SelectedValue;
}
```

Завдання 7. Створення оброблювача події для закриття форми

Виконання

1. У конструкторі форми ***Продаж_товарів_виробника_Form*** двічі клацніть на кнопці ***Зберегти***.

2. У оброблювачі події ***Вихід_Click*** додайте такий код:

```
private void Вихід_Click(object sender, EventArgs e)
        {
            this.Close();
        }
```

3. Перевірте функціональність форми ***Продаж_товарів_виробника.*** Після її завантаження в елементах ***comboBox_Товар*** і ***comboBox_Виробник*** повинні відповідно відображатися назва товару та назва виробника, а в елементі ***dataGridView_Продажі*** – дані про продаж товару, який обрано у ***comboBox_Товар***, що був зроблений виробником, обраним у ***comboBox_Виробник***. У ході додавання рядка або зміні інформації в елементі ***dataGridView_Продажі*** відображається вікно з повідомленням "Зміни записані".

2. Використання можливостей ADO NET EF для контролю конфліктів паралелізму

Постановка задачі

Конфлікт одночасного доступу виникає, коли один користувач переглядає дані про одну сутність і далі редагує її , в цей же час інший користувач оновлює ті ж дані перед тим, як зміни, внесені першим користувачем, зберігаються в базу.

Завдання полягає в тому, щоб користувач, який намагається оновити рядок таблиці, зміг отримати інформацію про те, що цей рядок, із моменту останнього його читання з бази, вже був змінений іншим користувачем. Тільки після цього можливо приймати рішення про необхідність оновлення рядка або новому його перегляді.

Основна ідея рішення задачі

За замовчуванням в платформі Entity Framework реалізується модель оптимістичного паралелізму. Це означає, що між процесами запиту і поновлення даних їх блокування не підтримуються. Entity Framework зберігає зроблені в об'єкті зміни в базі даних, не перевіряючи їх на паралелізм.

Можна дозволяти подібні конфлікти обробкою виключень OptimisticConcurrencyException, які формуються EF. Для того щоб дізнатися, коли сформувати дане виключення, EF повинен вміти визначати момент виникнення конфлікту. Тому необхідно правильно налаштувати базу даних і модель даних. Можна скористатися таким варіантом для подібного налаштування: в таблиці в базі даних включити стовпець, який можна використовувати для визначення моменту зміни запису. Потім включати цей стовпець в операторі Where запитів Update або Delete. Тип даних такого стовпця зазвичай timestamp. Контроль змін стовпця типу timestamp можна покласти на збережену процедуру поновлення даних таблиці.

Слід створити програмний продукт, який демонструє використання можливостей ADO NET EF при виникненні конфліктів у процесі паралельної обробки даних декількома користувачами.

Треба припустити, що операція оновлення даних виконується тільки в таблиці ***Sales***. При цьому програмний продукт містить одну форму (рис. 9.8).

На формі ***Обробка конфліктів оновлення даних*** відображаються дані таблиці ***Products*** – у компоненті ***ComboBox_Товар***, всі записи таблиці ***Sales***, які є залежними від обраного запису в компоненті ***ComboBox_Товар*** відображаються в елементі форми ***DataGrid-View_Продажі*** (рис. 9.8).

Щоб проглянути дані про продаж якого-небудь товару, його вибирають у списку компонента ***ComboBox_Товар,*** в елементі форми ***DataGridView_Продажі*** відображаються дані з обраного товару. Тут їх можна змінити. Щоб зафіксувати зроблені зміни в базі даних, треба натиснути кнопку ***Зберегти зміни***. Програмний продукт повинен відстежувати зміну таких даних, зроблену іншими користувачами. При виникненні конфлікту паралелізму користувач повинен мати можливість примусово зберегти змінені дані. Для цього призначена кнопка ***Примусово зберегти зміни***.

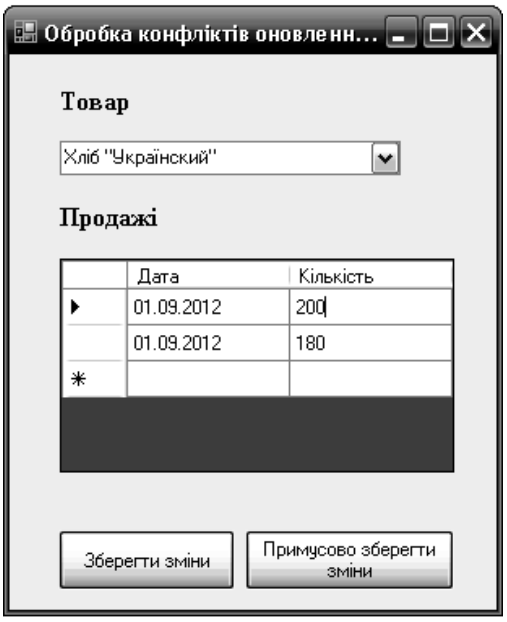


Рис. 9.8. **Форма *Обробка конфліктів оновлення даних***

У даній частині лабораторної роботи створення програмного продукту складається з таких етапів:

1. Додавання до таблиці ***Sales*** стовпця типу ***timestamp***.
2. Оновлення моделі ***Entity Data Model*** у зв'язку зі зміною таблиці ***Sales***.
3. Створення збереженої процедури (***update_Продажі***).
4. Додавання процедури ***update_Продажі*** до **Entity Model *(ХлібModel.Store***).
5. Зіставлення сутності ***Sales*** зі збереженою процедурою ***update_Продажі***.
6. Створення форми ***Обробка конфліктів оновлення даних*** для роботи з результатами виконання процедури.
7. Відображення даних таблиць ***Products***, ***Sales*** на формі ***Обробка конфліктів оновлення даних***.
8. Додавання функції оновлення даних у базі даних.
9. Додавання функції обробки конфліктів паралелізму.

2.1. Додавання до таблиці *Sales* стовпця типу timestamp

Виконання

Створення стовпця типу **timestamp** в таблиці ***Sales*** та збереження змін у базі даних ***Хлеб.mdf*** виконується самостійно, наприклад, у вікні **Server Explorer**. В результаті визначення таблиці ***Sales*** має вигляд, що зображений на рис. 9.9.

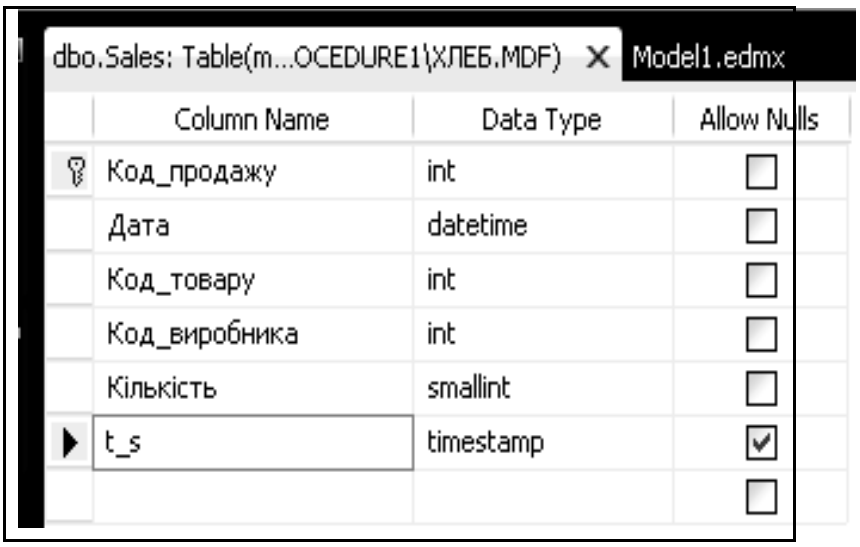


Рис. 9.9. **Схема таблиці *Sales***

2.2. Оновлення моделі Entity Data Model у зв'язку зі зміною таблиці *Sales*

Виконання

1. У вікні зображення моделі бази даних натисніть праву кнопку миші та виберіть пункт **Update Model from Database**.

2. У відкритому вікні **Update Wizard** виберіть вкладку **Refresh**, а в ній таблицю ***Sales*** та натисніть кнопку **Finish** (рис. 9.10).

У вікні ***Model*** в таблицю ***Sales*** буде додано стовпець ***t_s*** типу timestamp (рис. 9.10).

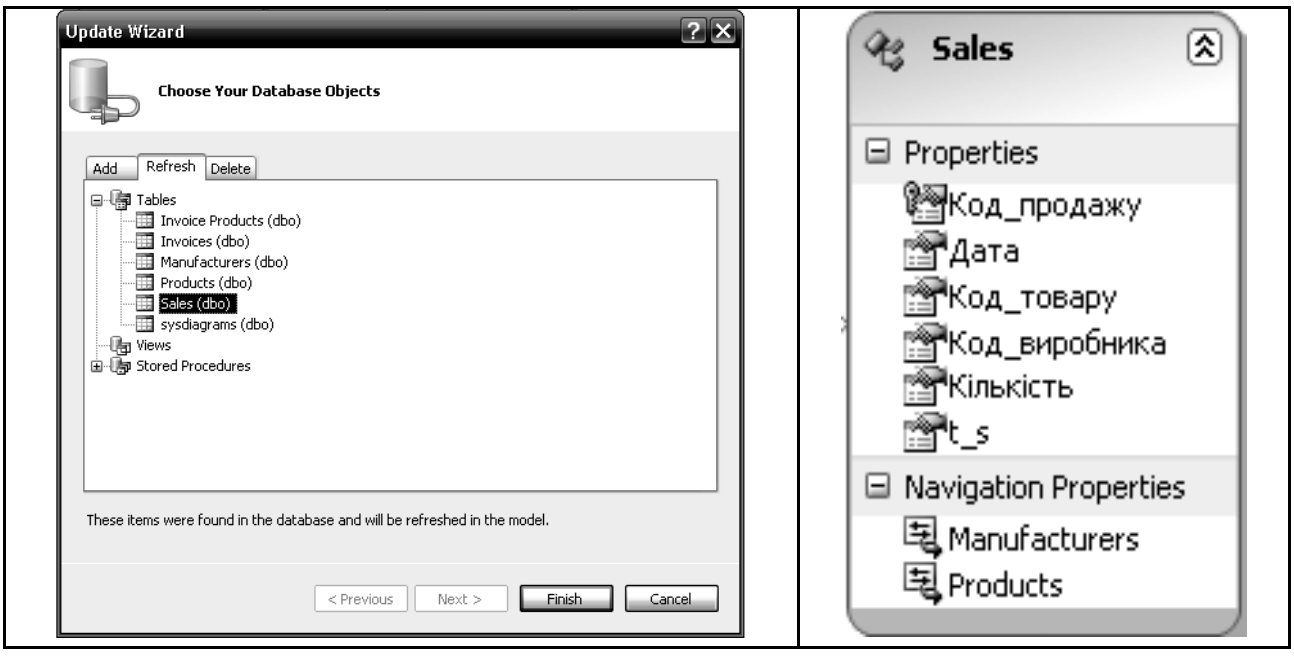


Рис. 9.10. **Вікно *Update Wizard* та сутність *Sales***

2.3. Створення збереженої процедури *update_Продажі*

Виконання

Створення і зберігання в базі даних процедури ***update_Продажі*** виконується самостійно. При цьому слід використовувати такий код процедури:

```
ALTER PROCEDURE dbo.update_Продажі
   @Код_продажу int,
   @Дата datetime,
   @Код_товару int,
   @Код_виробника int,
   @Кількість smallint,
   @t_s timestamp
as
  update Sales set Дата=@Дата, Код_товару=@Код_товару,
  Код_ виробника = @Код_виробника, Кількість =@Кількість
  where  Код_продажу = @Код_продажу AND [t_s]=@t_s;
           IF @@ROWCOUNT > 0
           BEGIN
                SELECT [t_s] FROM Sales
                WHERE Код_продажу=@Код_продажу;
           END
```

2.4. Додавання процедури *update_Продажі* до Entity Model (ХлебModel.Store)

Виконання

1. У вікні **Solution Explorer** відкрийте папку ***ХлібModel.edmx***.

2. У вікні зображення моделі бази даних натисніть праву кнопку миші та виберіть пункт **Update Model from Database**.

3. У вкладці **Add** відкритого вікна **Update Wizard** виберіть список **Stored Procedures**, а в ньому ім'я процедури ***update_Продажі*** і натисніть кнопку **Finish**.

2.5. Зіставлення сутності *Sales* зі збереженою процедурою *update_Продажі*

Виконання

1. У вікні моделі ***ХлібModel*** натисніть праву кнопку миші на типі сутності ***Sales*** та виберіть пункт **Stored Procedure Mapping**. Відкриється вікно **Mapping Details-Sales** (рис. 9.11).

Рис. 9.11. **Вікно *Mapping Details-Sales* при виборі процедури *update_Продажі***

2. Натисніть елемент **<Select Update Function>** та виберіть команду ***update_Продажі*** зі списку. З'являться зіставлення параметрів процедури з властивостями сутності, які визначаються за замовчуванням (рис. 9.12).

3. Виберіть поле в стовпці **Property**, яке відповідає параметру ***t_s:timestamp*** та оберіть із розкритого списку **Timestamp**.

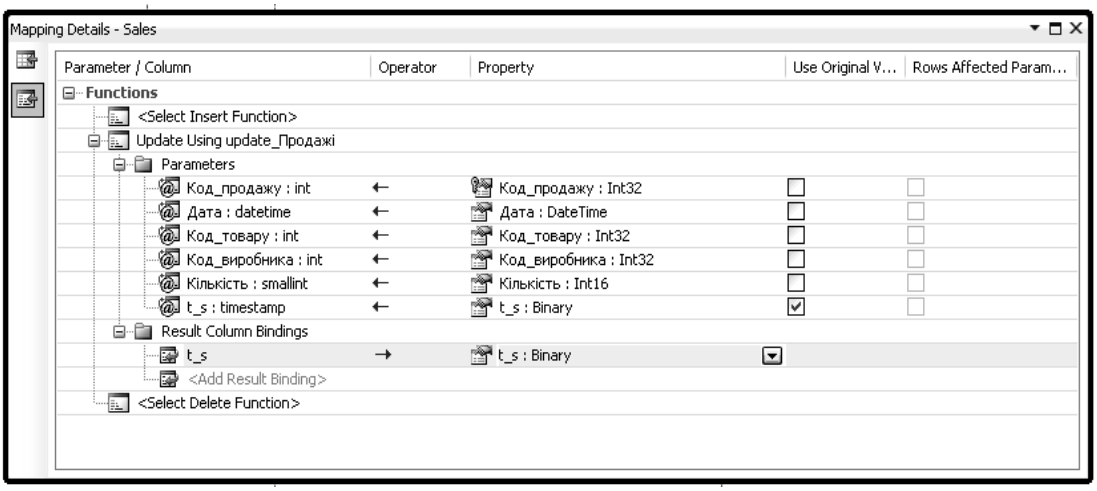


Рис. 9.12. **Вікно *Mapping Details-Sales* при зіставленні параметрів процедури *update_Продажі***

4. Оберіть прапорець у стовпчику **Use Original Value**, який відповідає властивості ***Timestamp*** (рис. 9.12).

5. Натисніть елемент **<Result Column Bindings>**, тоді можна вибрати необхідний параметр. Обираємо параметр ***t_s*** типу ***Timestamp***.

6. Натисніть порожнє поле **Property** поряд із ***Timestamp***. Поле стає списком властивостей, з якими можна зіставити стовпець результату, який повертається процедурою ***update_Продажі***.

7. Виберіть пункт **Timestamp** з розкритого списку.

У процесі обробки даних значення властивості **Timestamp** зчитується з бази та використовується при оновленні даних. Якщо отримане значення не узгоджується із значенням у базі даних, то активізується виключення **OptimisticConcurrencyException**.

2.6. Створення форми *Обробка конфліктів оновлення даних* для роботи з результатами виконання процедури

Виконання

1. У вікні **Solution Explorer** додайте форму з ім'ям ***Обробка конфліктів оновлення даних***.
2. Додайте на форму елемент **ComboBox** із ім'ям ***comboBox_Товар***.
3. Додайте елемент **DataGridView** із ім'ям ***dataGridView_Продажі*** на форму.
4. Додайте два елементи **Label** та установіть їх властивості **Text** відповідно ***Товар*** і ***Продажі***.
5. Додайте два елементи **Button.** Задайте кожному з них ім'я ***button_Зберегти_зміни*** і ***button_Примусово_зберегти_зміни*** та установіть їх властивості **Text** відповідно ***Зберегти_зміни*** і ***Примусово_зберегти_зміни***.
6. Розмістіть елементи на формі ***Обробка конфліктів оновлення даних*** відповідно до рис. 9.8.
7. У вікні **Solution Explorer** відкрийте конструктор форми ***Хліб*** і додайте кнопку ***Конфлікти паралелізму*** для виклику форми ***Обробка конфліктів оновлення даних*** (рис. 9.1).
8. У конструкторі форми ***Хліб*** двічі клацніть на кнопці ***Конфлікти паралелізму***.
9. У оброблювачі події ***Конфлікти_паралелізму _Click*** додайте такий код:

```
private void Конфлікти_паралелізму _Click(object sender, EventArgs e)
  {
      Обробка_конфліктів_оновлення_даних Конфлікти_паралелізму =
            new Обробка_конфліктів_оновлення_даних ();
      Конфлікти_паралелізму.ShowDialog();
  }
```

2.7. Відображення даних таблиць *Products*, *Sales* на формі *Обробка конфліктів оновлення даних*

Виконання

1. Для того, щоб посилатись на модель, яка створена на основі бази даних ***Хліб***, та простір імен сутностей до коду форми, додайте такі інструкції:

```
using System.Data.Objects.DataClasses;
using System.Data.Objects;

namespace winХліб
{
    public partial class Обробка_конфліктів_оновлення_даних : Form
    {
        ХлібEntities db;
        public Form_Продажі_товарів()
        {
            InitializeComponent();
        }
    }
}
```

2. Створіть метод для завантаження даних із таблиць ***Products***, ***Sales***. Для цього використовується такий код:

```
private void ExecuteПродажQuery()
    {
     // Інструкції для запиту
            var prodajQuery = from d in хлебContext.Products
            orderby d.Товар
                 select d;
     // Інструкції для виконання запиту і відображення
     // найменування товару в компоненті ComboBox форми
        comboBox_Товар.DataSource = prodajQuery;
        comboBox_Товар.DisplayMember = "Товар";
    }
```

3. Додайте код оброблювача події "Завантаження форми".

3.1. Перейдіть у вікно конструктора форми ***Обробка конфліктів оновлення даних*** й двічі клацніть у вільнім місці форми.

3.2. Уведіть оператори тіла оброблювача події ***Обробка конфліктів оновлення даних_Load***:

```
private void Обробка_конфліктів_оновлення даних _Load(object sender,
    EventArgs e)
    {
```

```
        хліbContext = new ХліbEntities();
        ExecuteПродажаQuery();
        button2.Enabled = false;
    }
```

4. Додайте код оброблювача події ***comboBox_Товар_SelectedIndexChanged***, щоб відображати в елементі ***DataGridView_Продажі*** записи з таблиці ***Sales*** під час вибору назви товару в компоненті ***comboBox_Товар*** форми.

4.1. Перейдіть у вікно конструктора форми та двічі натисніть ліву кнопку миші на компоненті ***comboBox_Товар*** форми.

4.2. Додайте такий код до оброблювача події ***comboBox_Товар _SelectedIndexChanged*:**

```
private void comboBox_Товар _SelectedIndexChanged(object sender,
    EventArgs e)
    {
        Products prod = (Products)this.comboBox_Товар.SelectedItem;
        dataGridView_Продажі.DataSource = prod.Sales;
        dataGridView_Продажі.Columns["Код_продажи"].Visible = false;
        dataGridView_Продажі.Columns["Код_товара"].Visible = false;
        dataGridView_Продажі.Columns["t_s"].Visible = false;
        dataGridView_Продажі.Columns["Manufacturers"].Visible = false;
        dataGridView_Продажі.Columns["Products"].Visible = false
    }
```

2.8. Додавання функції оновлення даних у БД

Виконання

1. Перейдіть у вікно конструктора форми та двічі натисніть ліву кнопку миші на компоненті ***button_Зберегти_зміни***.

2. Додайте такий код до оброблювача події ***button_Зберегти_зміни_Click***:

```
private void button_Зберегти_зміни_Click(object sender, EventArgs e)
{
    try
    {
        //  Зберігаємо зміни
        хліbContext.SaveChanges();
        MessageBox.Show("Зміни записані в базу.");
        this.Refresh();
    }
    catch (OptimisticConcurrencyException oce)
    {
```

```
        //Контролюємо конфлікт з визначенням таблиці та значенням ключа
        MessageBox.Show(oce.Message + " Конфлікт "
             + "виникає на " + oce.StateEntries[0].Entity
             + " при значенні ключа " + oce.StateEntries[0].
             EntityKey.EntityKeyValues[0].Value);
             button_Зберегти_зміни.Enabled = false;
             button_Примусово_зберегти_зміни.Enabled = true;
     }
     catch (UpdateException ue)
     {
         MessageBox.Show(ue.Message + " Натисніть ОК для отримання"
             + "останніх даних з бази.");
         ExecuteПродажаQuery();
         this.Refresh();
     }
}
```

2.9. Додавання функції обробки конфліктів паралелізму

Виконання

1. Перейдіть у вікно конструктора форми та двічі натисніть ліву кнопку миші на компоненті ***button_Примусово_зберегти_зміни*** форми.

2. Додайте такий код до оброблювача події ***button_Примусово_зберегти_зміни_Click*****:**

```
private void button_Примусово_зберегти_зміни _Click(object sender,
    EventArgs e)
{
    Products prod = (Products)this.comboBox1.SelectedItem;
    try
    {
        // Використання RefreshMode.ClientWins вимикає перевірку
        // оптимистичним паралелізмом
        хлі6Context.Refresh(RefreshMode.ClientWins, prod.Sales);
        хлі6Context.SaveChanges();
        MessageBox.Show("Зміни записані в базу.");
        button_Примусово_зберегти_зміни.Enabled = false;
        button_Зберегти_зміни.Enabled = true;
    }
    catch (InvalidOperationException ioe)
        {
             MessageBox.Show(ioe.Message + " Натисніть ОК для отримання "
                 + " останніх даних з бази.");
             ExecuteПродажаQuery();
             this.Refresh();
```

```
        }
}
```

При спостереженні того, як розроблений програмний продукт контролює виникнення конфлікту паралелізму, буде проходити моделювання одночасної роботи з базою даних декількох користувачів. Тому перевірка роботи програми включає такі етапи:

1. Двічі запустити програму на виконання.
2. В обох вікнах під заголовком ***Товар*** вибрати один і той же запис (наприклад ***Хліб "Український"***).
3. В одному з вікон у таблиці під заголовком ***Продажі*** (елемент ***dataGridView_Продажі***) у стовпці ***Кількість*** одного з рядків змінити число (наприклад 234 на 240).
4. У другому з вікон у таблиці під заголовком ***Продажі*** (елемент ***dataGridView_Продажі***) у стовпці ***Кількість*** того ж рядка змінити число (наприклад 234 на 245).
5. У першому вікні натиснути кнопку ***Зберегти зміни***. При цьому буде отримано повідомлення про збереження змін у базі даних.
6. У другому вікні натисніть кнопку "***Зберегти зміни***". При цьому буде отримано повідомлення про виникнення конфлікту паралелізму (рис. 9.13).

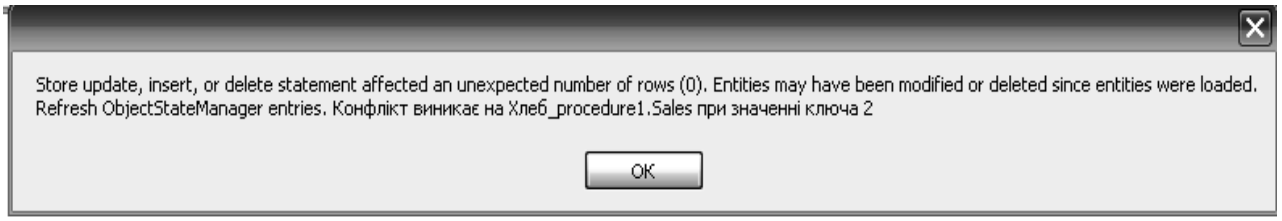


Рис. 9.13. **Повідомлення про виникнення конфлікту паралелізму**

7. На вікні повідомлення натисніть кнопку **ОК**. При цьому на другому вікні програми стане активною кнопка ***Примусово зберегти зміни***.
8. При натисненні кнопки ***Примусово зберегти зміни*** нові дані (дані, які змінив другий користувач) будуть збережені в базі даних і з'явиться вікно, яке повідомляє про це (рис. 9.14).

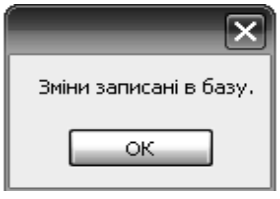


Рис. 9.14. **Вікно повідомлення про збереження даних**

3. Використання тригера для контролю змінення даних у таблиці

Постановка задачі

Робота тригера ініціюється при виконанні для таблиці якого-небудь оператора мови маніпулювання даними. Тому вивчення засобів використання тригерів при зміненні даних проводиться на основі контролю оновлення записів у таблиці ***Invoices*** бази даних ***Хліб.mdf.*** При цьому слід скористатися готовим програмним продуктом ***winХліб,*** який дозволяє вести облік накладних про продаж товарів. Слід контролювати стовпець ***Дата*** так, щоб неможливо було встановлювати його значення окрім сьогоднішнього. Для реєстрації оновлень треба вести журнал, у якому записувати всі вдалі та невдалі спроби змін даних у таблиці ***Invoices.*** Невдалою спробою оновлення запису слід вважати встановлення не сьогоднішньої дати для накладної.

Проблемним питанням є реєстрація невдалої спроби змін даних у таблиці ***Invoices,*** оскільки при цьому необхідно відкочувати транзакцію в тригері і будь-які зміни в базі даних стають неможливими.

Основна ідея рішення задачі

Задача реєстрації невдалої спроби змін даних у таблиці ***Invoices*** вирішується шляхом виклику збереженої процедури з тригера відразу після відкочування транзакції. Процедура повинна записати в журнал інформацію про невдале оновлення даних у таблиці ***Invoices.***

Застосування складається із таких форм:

форма ***Хліб*** (рис. 9.1), яка слугує для керування роботою застосування, на ній розміщуються кнопки для виклику функціональних форм;

форма "***Накладна***", на якої відображаються дані про продаж товарів (рис. 9.15);

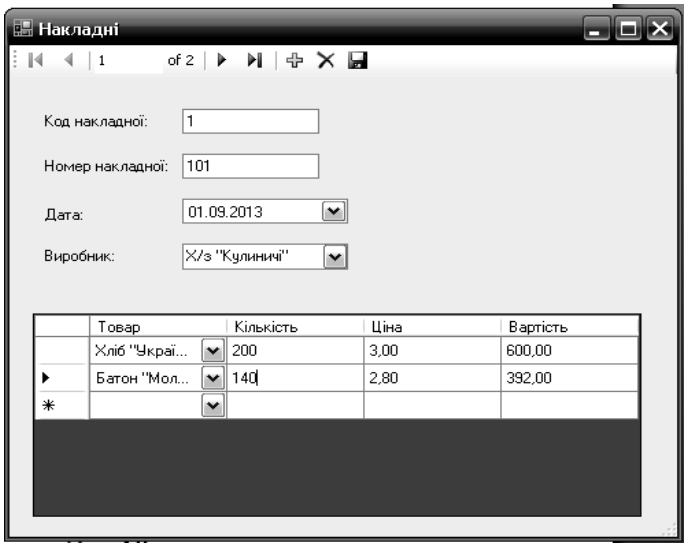


Рис. 9.15. **Форма *Накладна***

форма ***Журнал***, на якій відображаються дані про кількість вдало виконаних оновлень у таблиці ***Invoices,*** кількість помилково виконаних оновлень, а також реєструється дата – коли ці оновлення виконувались (рис. 9.16).

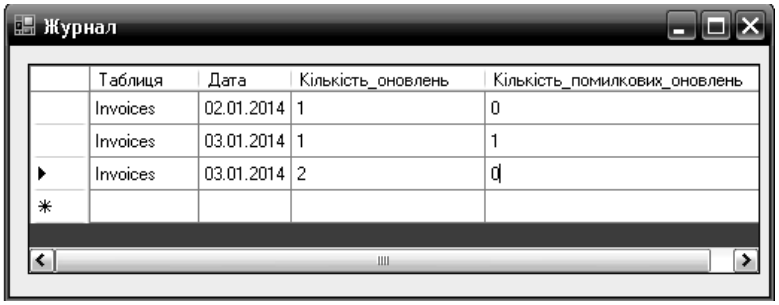


Рис. 9.16. **Форма *Журнал***

У даній частині лабораторної роботи процес рішення задачі складається з таких завдань:

1. Створення таблиці ***Gurnal***.
2. Оновлення моделі ***Entity Data Model*** у зв'язку з додаванням таблиці ***Gurnal***.
3. Створення тригера для контролю оновлення записів у таблиці ***Invoices***.
4. Створення збереженої процедури для запису в журнал інформації про оновлення даних у таблиці ***Invoices***.
5. Створення форми "***Журнал***" для відображення даних із таблиці ***Gurnal***.
6. Відображення даних таблиці ***Gurnal*** на формі ***Журнал***.
7. Додавання коду формування повідомлень користувачу про успішність оновлення даних.

3.1. Створення таблиці Gurnal

Виконання

До бази даних створюється і додається таблиця ***Gurnal***, схема якої відображена на рис. 9.17.

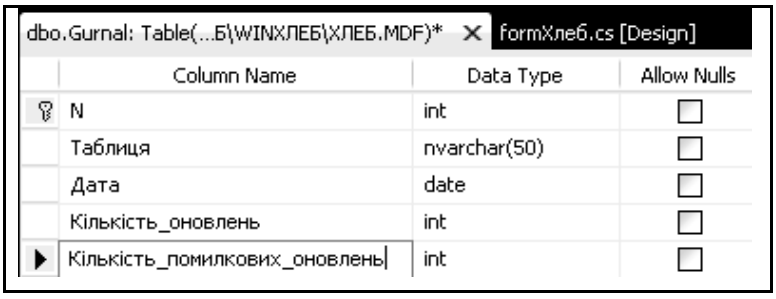


Рис. 9.17. **Схема таблиці *Gurnal***

3.2. Оновлення моделі *Entity Data Model* у зв'язку з додаванням таблиці *Gurnal*

Виконання

Entity Data Model оновлюється відповідно до завдання 2 задачі 1 поточної лабораторної роботи. В результаті оновлення моделі буде отримано сутність ***Gurnal*** (рис. 9.18).

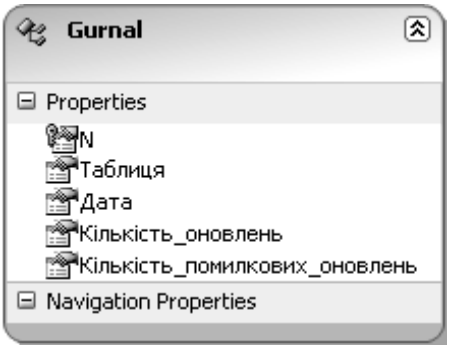


Рис. 9.18. **Сутність *Gurnal* в *Entity Data Model***

3.3. Створення тригера для контролю оновлення записів у таблиці *Invoices*

Виконання

1. У відкритому списку папки **Tables** натисніть правою кнопкою миші на папці ***Invoices*** та виберіть пункт меню **Add New Trigger** (рис. 9.19).

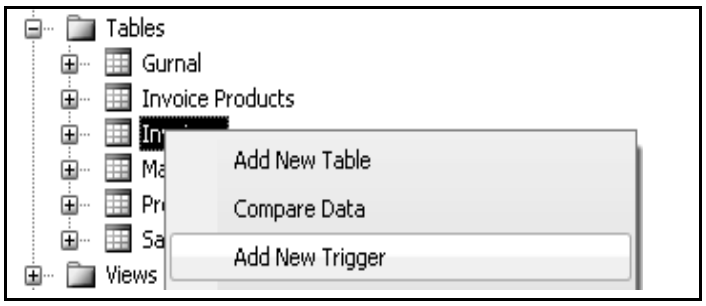


Рис. 9.19. **Контекстне меню для додавання тригера до бази даних**

2. У відкритому вікні введіть такий код тригера:

```
ALTER TRIGGER Trigger_Up_Invoices
ON dbo.Invoices
FOR UPDATE
AS
BEGIN
DECLARE @D date

SET @D=CONVERT(date,GETDATE())  // Отримання сьогоднішньої дати і
                                // перетворення її до типу date
```

```
IF (SELECT CONVERT(date,Дата) FROM inserted) != @D
   BEGIN
       //якщо дата накладної не відповідає сьогоднішній
       //транзакція відкочується
   ROLLBACK
       //викликається збережена процедура
       EXEC Запис_до_журналу 'Invoices'
   END
ELSE
    BEGIN
        IF NOT EXISTS (SELECT * FROM Gurnal WHERE (Gurnal.Таблиця =
                          'Invoices' AND Gurnal.Дата=@D))
     //Якщо на поточну дату записів про оновлення таблиці Invoices не
     //зареєстровано, то виконуємо додавання запису до таблиці Gurnal
            INSERT INTO Gurnal VALUES('Invoices',@D,1,0)
        ELSE
     //Якщо на поточну дату запис про оновлення таблиці Invoices є, то
     // виконуємо оновлення цього запису в таблиці Gurnal
           UPDATE Gurnal SET Кількість_оновлень = Кількість_оновлень+1
           WHERE (Gurnal.Таблиця = 'Invoices' AND Gurnal.Дата=@D)
     END
END
```

3. Закрийте вікно створення процедури та поверніться у вікно проекту. У папці ***Invoices*** вікна **Server Explorer** відображається ім'я тригера ***Trigger_Up_Invoices***.

3.4. Створення збереженої процедури для запису в журнал інформації про оновлення даних у таблиці *Invoices*

Виконання

1. Виконайте додавання нової збереженої процедури, натиснувши правою кнопкою миші на папці ***Stored Procedures*** та вибравши пункт меню **Add New Stored Procedure**.

2. У відкритому вікні введіть такий код збереженої процедури:

```
ALTER PROCEDURE dbo.Запис_до_журналу
    @tabl VARCHAR(50)
AS
    DECLARE @D date
    SET @D=CONVERT(date,GETDATE())
IF NOT EXISTS
    (SELECT * FROM Gurnal
         WHERE (Gurnal.Таблиця = @tabl AND Gurnal.Дата=@D))
```

```
         //Якщо на поточну дату записів про оновлення таблиці Invoices
         //не зареєстровано, то виконуємо додавання запису до таблиці
         //Gurnal
         INSERT INTO Gurnal VALUES(@tabl,@D,0,1)
ELSE
    // Якщо на поточну дату запис про оновлення таблиці Invoices є, то
    // виконуємо оновлення цього запису в таблиці Gurnal

    UPDATE Gurnal SET Кількість_помилкових_оновлень =
        Кількість_помилкових_оновлень +1
    WHERE (Gurnal.Таблиця = @tabl AND Gurnal.Дата=@D)
RETURN
```

3. Закрийте вікно створення процедури та поверніться у вікно проекту. В папці **Stored Procedures** вікна **Server Explorer** відображається ім'я процедури ***Запис_до_журналу***.

3.5. Створення форми Журнал для відображення даних із таблиці *Gurnal*

Виконання

1. У вікні **Solution Explorer** додайте форму з ім'ям ***Журнал***.
2. Додайте елемент **DataGrid** на форму ***Журнал***.
3. У вікні **Solution Explorer** відкрийте конструктор форми ***Хліб*** і додайте кнопку для виклику форми ***Журнал***.

3.6. Відображення даних таблиці *Gurnal* на формі *Журнал*

Виконання

1. У вікні **Solution Explorer** відкрийте конструктор форми ***Журнал***.
2. Додайте код оброблювача події "Завантаження форми", щоб відображати в елементі **DataGrid** дані таблиці ***Gurnal*** під час відкривання форми.

2.1. У вікні конструктора форми "***Журнал***" двічі клацніть на вільному місці форми.

2.2. Уведіть оператори тіла оброблювача події ***Журнал_Form1_Load***:

```
private void Журнал_Form1_Load(object sender, EventArgs e)
{
    db = new ХлебEntities();
    dataGridView_Журнал.DataSource = db.Gurnal;
    //Форматируем DataGridView
    dataGridView_Журнал.AutoResizeColumns();
    dataGridView_Журнал.Columns["N"].Visible = false;
}
```

3.7. Додавання коду формування повідомлень користувачу про успішність оновлення даних

Виконання

1. У вікні конструктора форми ***Накладна*** двічі клацніть на вільному місці форми.

2. Додайте оператор формування повідомлення. Для цього треба змінити код оброблювача події для збереження змін ***invoiceBindingNavigatorSaveItem_Click*** на такий:

```
try
{
   db.SaveChanges();
   MessageBox.Show(" Зміни записані в базу даних ");
}
```

3. Додайте блок перехоплення помилкових ситуацій, які виникають при оновленні даних.

```
try
{
    db.SaveChanges();
    MessageBox.Show("Зміни записані в базу даних ");
}
catch (UpdateException)
{
    MessageBox.Show(" УВАГА! Перевірте  дату накладної ");
}
```

Завдання для самостійного виконання

1. Отримати інформацію про накладні, в яких зареєстровані продажі обраного товару заданого виробника в поточному місяці.

2. Виконати обробку конфлікту паралелізму, який виникає при видаленні даних із таблиці ***Sales***.

3. Створити збережену процедуру для додавання записів до таблиці ***Sales.***

4. Створити тригери, які контролюють додавання та видалення даних із таблиці.

5. У проекті з індивідуальною базою даних застосувати прийоми, які розглянуті в базовій частині лабораторної роботи, а також у завданнях для самостійного виконання.

Висновки

1. Збережені процедури дозволяють ефективно виконувати будь-які складні запити до таблиць бази даних. Для їх запуску достатньо вказати всього лише ім'я необхідної процедури, це дозволяє зменшити розмір запиту, що посилається від клієнта на сервер.

2. Збережені процедури можуть бути викликані з прикладних програм різноманітних типів, забезпечують простоту модифікації алгоритмів обробки даних і дають можливість розширення програмного продукту без внесення зміни до самого застосування.

3. Збережені процедури існують незалежно від таблиць або яких-небудь інших об'єктів баз даних, пройшли етап синтаксичного аналізу і знаходяться у готовому для виконання форматі. Процедури можуть викликатись клієнтською програмою, другою збереженою процедурою або тригером.

4. Результатом виконання збереженої процедури може бути суттєвість, записи якої можна змінять, додавати або видаляти і ці зміни зберігатись у базі даних.

5. За допомогою збережених процедур можливо ефективно вирішувати конфлікти, які виникають у ході паралельної обробки даних. Для цього в таблицю додають стовпець типу timestamp, який фіксує момент зміни запису таблиці. Контроль за цим стовпцем виконується збереженими процедурами оновлення та видалення даних.

6. Тригер – це спеціальний тип збережених процедур, що запускаються сервером автоматично при спробі зміни даних у таблицях, з якими тригери пов'язані.

7. Усі модифікації даних, в яких він "бере участь", розглядаються як одна транзакція, в разі виявлення помилки або порушення цілісності даних, відбувається відкат цієї транзакції, тим самим внесення змін забороняється, скасовуються також всі зміни, які вже зроблені тригером.

Рекомендована література

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

2. Галузевий стандарт вищої освіти України з напряму підготовки 6.050101 "Комп'ютерні науки" // Збірник нормативних документів вищої освіти. – К. : Видавнича група BHV, 2011. – 85 с.

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

4. Робоча програма навчальної дисципліни "Організація баз даних та знань" / О. В. Тарасов, В. В. Федько, М. Ю. Лосєв. – Х. : Вид. ХНЕУ ім. С. Кузнеця, 2014. – 108 с.

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

6. Тарасов О. В. Проектування баз даних : навч. посібн. / О. В. Тарасов, В. В. Федько, М. Ю. Лосєв. – Х. : Вид. ХНЕУ, 2011. – 200 с.

7. Тарасов О. В. Використання мови SQL для роботи з сучасними системами керування базами даних. Практикум з навчальної дисципліни "Організація баз даних та знань" : навч. посібн. / О. В. Тарасов, В. В. Федько, М. Ю. Лосєв. – Х. : Вид. ХНЕУ, 2013. – 349 с.

8. Троелсен Э. C# и платформа NET. Библиотека программиста / Э.Троелсен. – СПб. : Питер, 2007. – С. 629–691.

9. Федько В. В. Лабораторний практикум з модуля "Основи баз даних та знань" навчальної дисципліни "Організація баз даних та знань" / В. В. Федько, О. В. Тарасов, М. Ю. Лосєв. – Х. : Вид. ХНЕУ, 2011. – 192 с.

10. Федько В. В. Організація баз даних та знань : навч.-прак. посібн. / В. В. Федько, О. В. Тарасов, М. Ю. Лосєв. – Х. : Вид. ХНЕУ, 2013. – 200 с.

11. Johnson G. Exam 70-516: TS: Accessing Data with Microsoft .NET Framework 4 / G. Johnson. – Microsoft Press, 2011. – 671 p.

12. Lerman J. Programming Entity Framework, Second Edition. / J. Lerman. – O'Reilly, 2010. – 914 p.

13. Lerman J. Programming Entity Framework: Code First / J. Lerman, R. Miller. – O'Reilly Media, 2011. – 194 p.

14. Lerman J. Programming Entity Framework: DbContext / J. Lerman, R. Miller. – O'Reilly Media, 2012. – 258 p.

15. Rattz J. Pro LINQ: Language Integrated Query in C# 2010 / J.Rattz, A. Freeman. – Apress, 2010. – 656 p.

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

Федько Віктор Васильович
Тарасов Олександр Васильович
Лосєв Михайло Юрійович

СУЧАСНІ ЗАСОБИ ДОСТУПУ ДО ДАНИХ

Навчальний посібник
для самостійної роботи студентів
з навчальної дисципліни
"ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ"
для студентів напряму підготовки
6.050101 "Комп'ютерні науки"

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

Відповідальний редактор **Сєдова Л. М.**

Редактор **Бутенко В. О.**

Коректор **Бриль В. О.**

План 2014 р. Поз. № 34-П.
Підп. до друку Формат 60 х 90 1/16. Папір MultiCopy. Друк Riso.
Ум.-друк. арк. 20,5. Обл.-вид. арк. 25,63. Тираж прим. Зам. №

Видавець і виготівник – видавництво ХНЕУ ім. С. Кузнеця, 61166, м. Харків, пр. Леніна, 9-А

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