



РОБОТА З ДАНИМИ ЗА ДОПОМОГОЮ ТЕХНОЛОГІЙ ENTITY FRAMEWORK CORE ТА LINQ

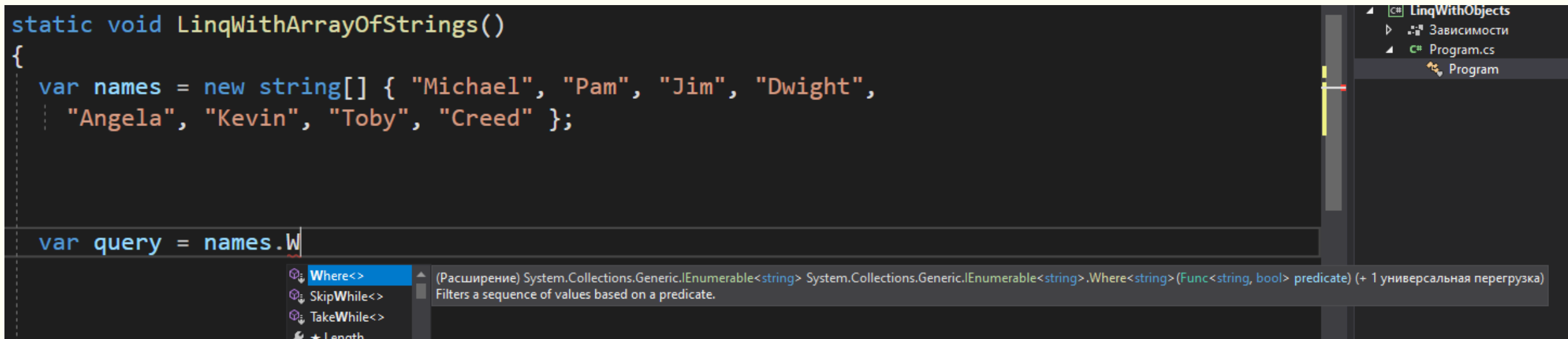
Питання 1.5.

Створення запитів та керування даними за допомогою LINQ

- LINQ складається з кількох частин:
 - **Методи розширення (обов'язкові):** набір методів включає Where, OrderBy, Select та ін. Ці методи представляють функціональність LINQ.
 - **Постачальники даних LINQ (обов'язкові):** набір включає LINQ to Objects, LINQ to Entities, LINQ to XML, LINQ to OData, LINQ to Amazon, а також інші постачальники даних LINQ. Вони конвертують стандартні операції LINQ у команди, характерні для різних видів даних.
 - **Лямбда-вирази (додаткові):** можуть застосовуватись замість іменованих методів для спрощення викликів методів розширення LINQ.
 - **Зрозумілий синтаксис запитів LINQ (додатковий):** цей синтаксис включає такі ключові слова, як from, in, where, orderby, descending, select та ін.
 - Перелічені ключові слова є псевдонімами для деяких методів розширення LINQ.
- Такі методи розширення, як Where і Select, підставляються до будь-якого *туну* за допомогою класу Enumerable, який називають **послідовністю (sequence)** та реалізує інтерфейс IEnumerable<T>.
 - Наприклад, масив будь-якого типу автоматично реалізує IEnumerable<T>, тобто всі масиви підтримують створення запитів та керування за допомогою LINQ.
 - Всі універсальні колекції також реалізують IEnumerable<T>, як і колекція DbSet<T>.

Фільтрування елементів (entities) за допомогою Where (LINQ with Objects)

- Найбільш розповсюджена причина застосування LINQ — можливість фільтрації елементів у послідовності завдяки методу розширення Where.
 - Доступним для IntelliSense цей метод стане після імпорту простору імен System.Linq.



```
static void LinqWithArrayOfStrings()
{
    var names = new string[] { "Michael", "Pam", "Jim", "Dwight",
        "Angela", "Kevin", "Toby", "Creed" };

    var query = names.W
```

The screenshot shows the Visual Studio IDE. The main editor displays a C# method `LinqWithArrayOfStrings()` that creates an array of names. The cursor is at the end of `names.W`, and the IntelliSense dropdown is open, showing the `Where<>` method as the top suggestion. A tooltip for `Where<>` is visible, explaining its function: "(Расширение) System.Collections.Generic.IEnumerable<string> System.Collections.Generic.IEnumerable<string>.Where<string>(Func<string, bool> predicate) (+ 1 универсальная перегрузка) Filters a sequence of values based on a predicate."

- Система підказує, що для виклику методу Where потрібно передати в нього екземпляр делегата `Func<string, bool>`: для кожного переданого рядка метод повинен повертати булеве значення.

Посилання (targeting) на іменовані методи

```
static bool NameLongerThanFour(string name)
{
    return name.Length > 4;
}

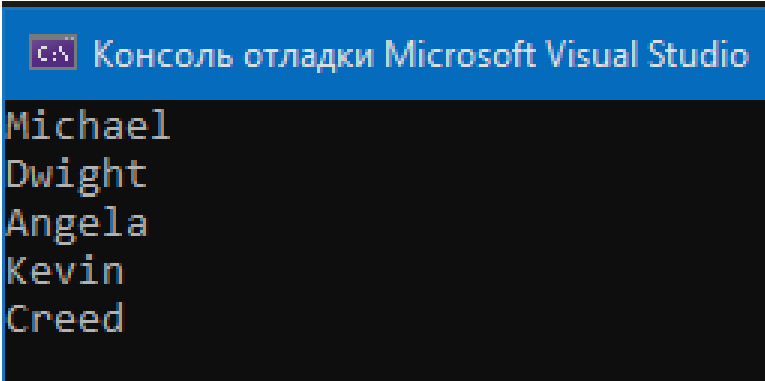
static void LinqWithArrayOfStrings()
{
    var names = new string[] { "Michael", "Pam", "Jim", "Dwight",
        "Angela", "Kevin", "Toby", "Creed" };

    var query = names.Where(
        new Func<string, bool>(NameLongerThanFour));

    foreach (string item in query)
    {
        WriteLine(item);
    }
}

static void Main(string[] args)
{
    LinqWithArrayOfStrings();
}
```

- Відберемо всі імена, довжина яких більше 4 літер.



```
Консоль отладки Microsoft Visual Studio
Michael
Dwight
Angela
Kevin
Creed
```

Спрощення коду

- (1) Видаляємо явне створення екземплярів делегатів

```
// var query = names.Where(  
//     new Func<string, bool>(NameLongerThanFour));  
  
var query = names.Where(NameLongerThanFour);
```

- (2) Посилання (targeting) на лямбда-вираз

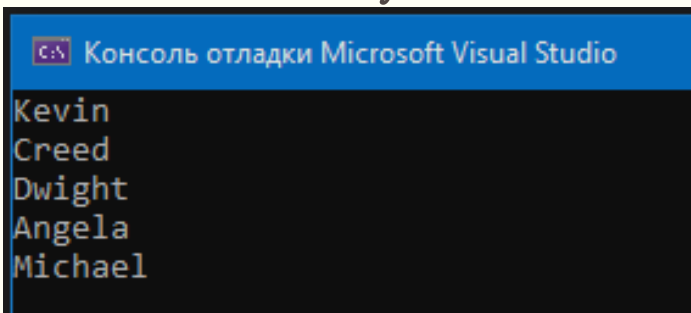
```
// var query = names.Where(NameLongerThanFour);  
  
var query = names  
    .Where(name => name.Length > 4);
```

- Метод Where() — лише один з близько 30 методів розширення, які надає тип Enumerable.
- З методів розширення може складатись ланцюжок, якщо попередній метод повертає ще одну послідовність, тобто, тип, що реалізує інтерфейс IEnumerable<T>.

Сортування елементів

- (1) Сортування окремих елементів за допомогою OrderBy():

```
var query = names
    .Where(name => name.Length > 4)
    .OrderBy(name => name.Length);
```

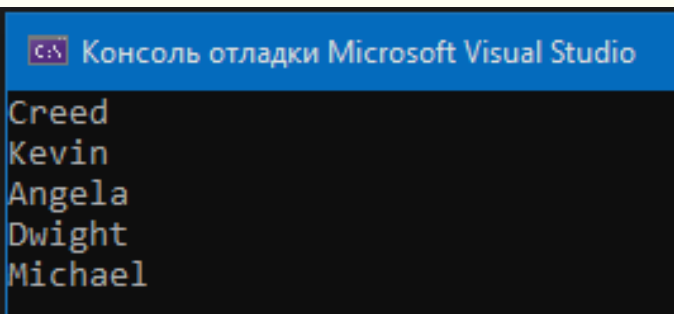


Консоль отладки Microsoft Visual Studio

```
Kevin
Creed
Dwight
Angela
Michael
```

- (2) Сортування за кількома властивостями за допомогою ThenBy:

```
var query = names
    .Where(name => name.Length > 4)
    .OrderBy(name => name.Length)
    .ThenBy(name => name);
```



Консоль отладки Microsoft Visual Studio

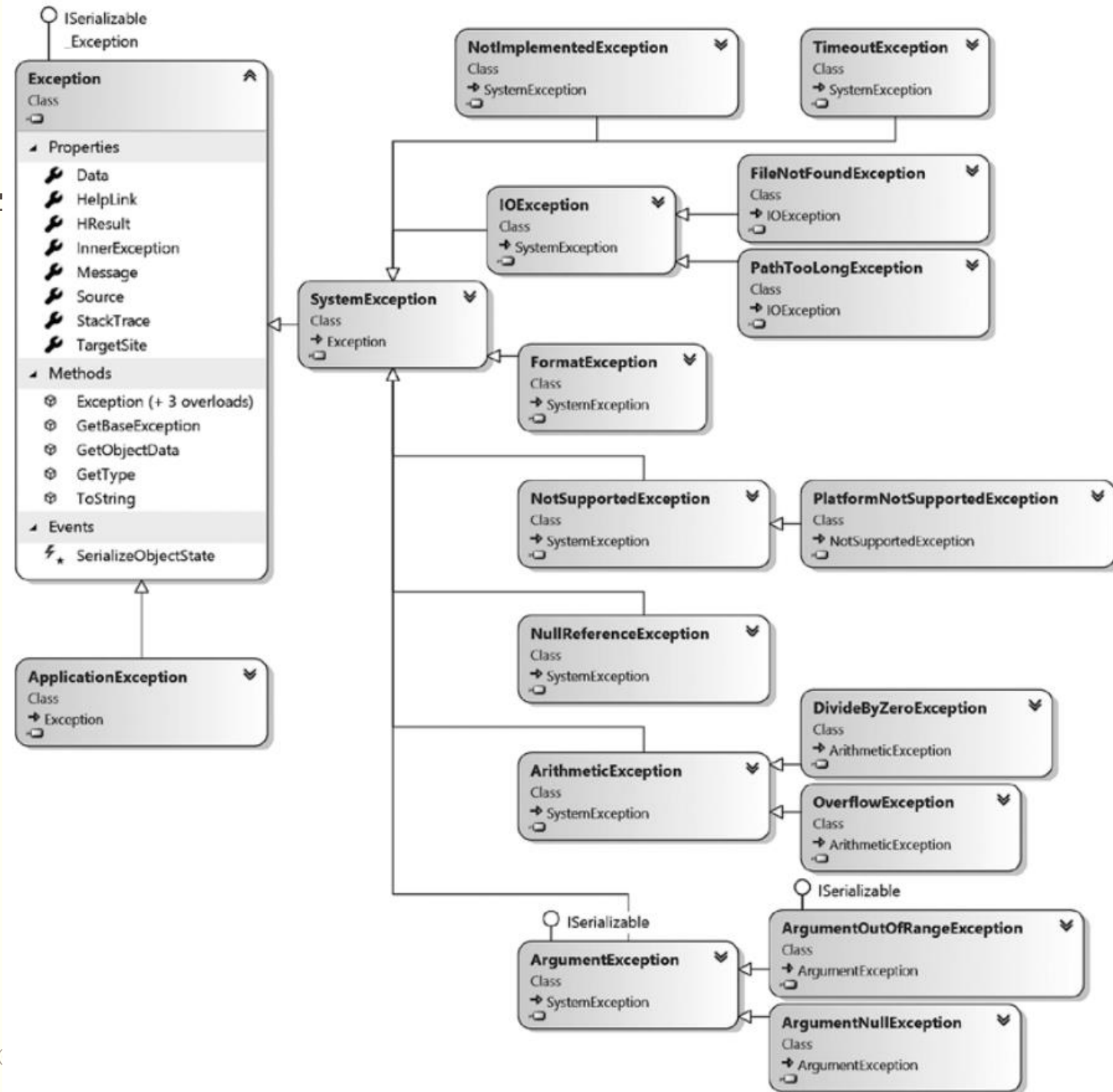
```
Creed
Kevin
Angela
Dwight
Michael
```

Фільтрування за типом

- Нехай існує послідовність винятків:

```
var errors = new Exception[]  
{  
    new ArgumentException(),  
    new SystemException(),  
    new IndexOutOfRangeException(),  
    new InvalidOperationException(),  
    new NullReferenceException(),  
    new InvalidCastException(),  
    new OverflowException(),  
    new DivideByZeroException(),  
    new ApplicationException()  
};
```

- Для фільтрації за типом можна написати інструкції за допомогою методу розширення `OfType<T>`



Фільтрування за типом

```
static void LinqWithArrayOfExceptions()
{
    var errors = new Exception[]
    {
        new ArgumentException(),
        new SystemException(),
        new IndexOutOfRangeException(),
        new InvalidOperationException(),
        new NullReferenceException(),
        new InvalidCastException(),
        new OverflowException(),
        new DivideByZeroException(),
        new ApplicationException()
    };

    var numberErrors = errors.OfType<ArithmeticException>();

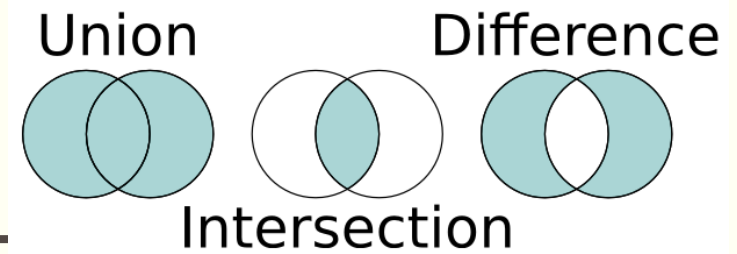
    foreach (var error in numberErrors)
    {
        WriteLine(error);
    }
}
```

Консоль отладки Microsoft Visual Studio

```
System.OverflowException: Arithmetic operation resulted in an overflow.
System.DivideByZeroException: Attempted to divide by zero.
```

```
static void Main(string[] args)
{
    // LinqWithArrayOfStrings();
    LinqWithArrayOfExceptions();
}
```


Робота з множинами (multiset, bag) (LINQ with Sets)



- Універсальний набір операцій включає *переріз* та *об'єднання* множин.

- В прикладі визначимо 3 рядкових масиви для груп студентів, а потім виконаємо з цими масивами кілька загальних операцій.

```
using static System.Console;
using System.Collections.Generic; // for IEnumerable<T>
using System.Linq; // for LINQ extension methods
```

```
namespace LinqWithSets
{
    class Program
    {
        static void Output(IEnumerable<string> cohort,
            string description = "")
        {
            if (!string.IsNullOrEmpty(description))
            {
                WriteLine(description);
            }
            Write(" ");
            WriteLine(string.Join(", ", cohort.ToArray()));
        }
    }
}
```

```
static void Main(string[] args)
{
    var cohort1 = new string[] { "Rachel", "Gareth", "Jonathan", "George" };
    var cohort2 = new string[] { "Jack", "Stephen", "Daniel", "Jack", "Jared" };
    var cohort3 = new string[] { "Declan", "Jack", "Jack", "Jasmine", "Conor" };

    Output(cohort1, "Cohort 1");
    Output(cohort2, "Cohort 2");
    Output(cohort3, "Cohort 3");
    WriteLine();
    Output(cohort2.Distinct(), "cohort2.Distinct():");
    WriteLine();
    Output(cohort2.Union(cohort3), "cohort2.Union(cohort3):");
    WriteLine();
    Output(cohort2.Concat(cohort3), "cohort2.Concat(cohort3):");
    WriteLine();
    Output(cohort2.Intersect(cohort3), "cohort2.Intersect(cohort3):");
    WriteLine();
    Output(cohort2.Except(cohort3), "cohort2.Except(cohort3):");
    WriteLine();
    Output(cohort1.Zip(cohort2, (c1, c2) => $"{c1} matched with {c2}"),
        "cohort1.Zip(cohort2):");
}
}
```

Результати виводу

```
Консоль отладки Microsoft Visual Studio

Rachel, Gareth, Jonathan, George
Cohort 2
  Jack, Stephen, Daniel, Jack, Jared
Cohort 3
  Declan, Jack, Jack, Jasmine, Conor

cohort2.Distinct():
  Jack, Stephen, Daniel, Jared

cohort2.Union(cohort3):
  Jack, Stephen, Daniel, Jared, Declan, Jasmine, Conor

cohort2.Concat(cohort3):
  Jack, Stephen, Daniel, Jack, Jared, Declan, Jack, Jack, Jasmine, Conor

cohort2.Intersect(cohort3):
  Jack

cohort2.Except(cohort3):
  Stephen, Daniel, Jared

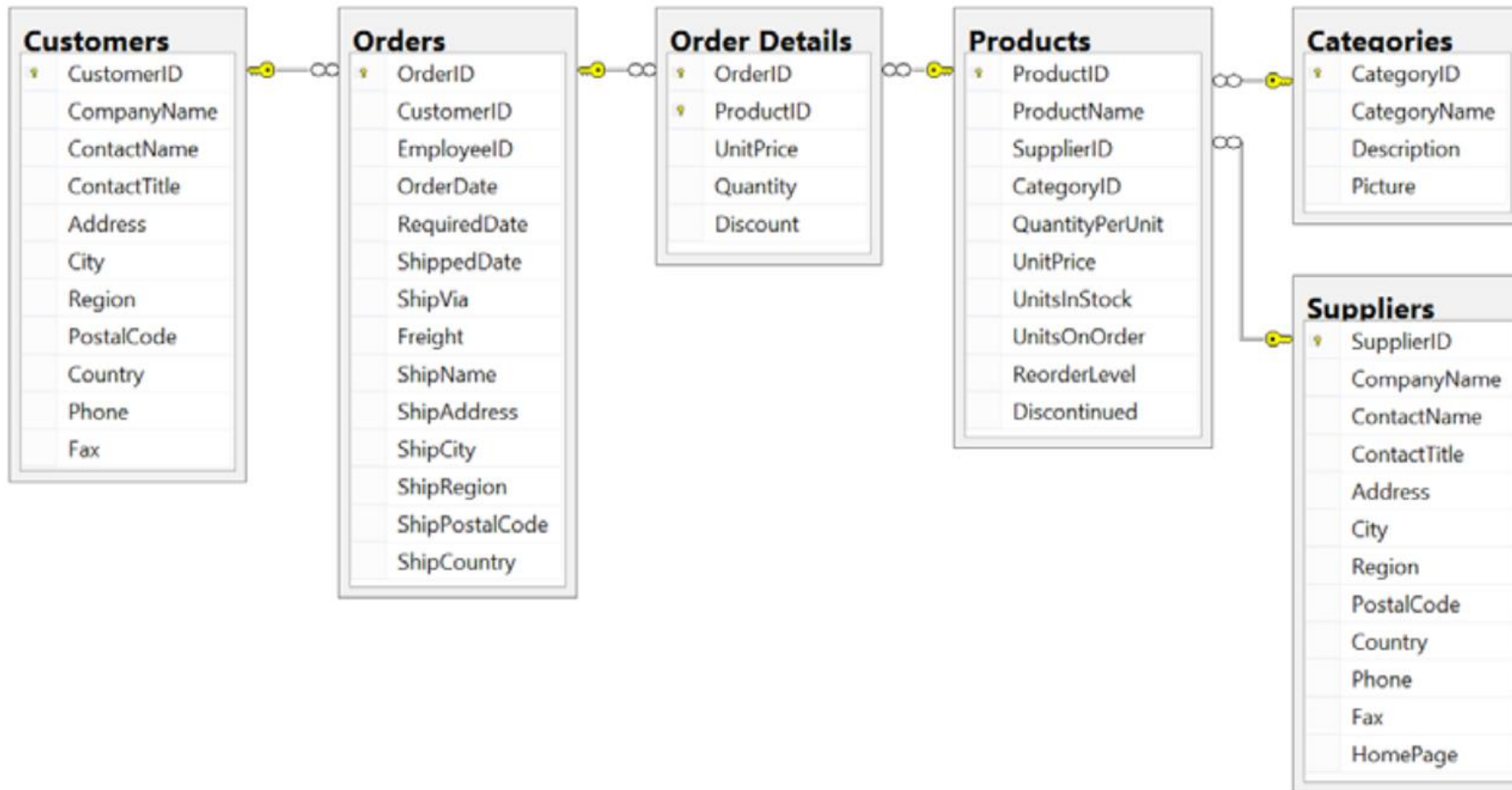
cohort1.Zip(cohort2):
  Rachel matched with Jack, Gareth matched with Stephen, Jonathan matched with Daniel, George matched with Jack
```

Технологія Entity Framework

- Найбільш поширені сховища даних:
 - реляційні СКБД (Microsoft SQL Server, PostgreSQL, MySQL, SQLite та ін.);
 - NoSQL-сховища даних (Microsoft Azure Cosmos DB, Redis, MongoDB, Apache Cassandra тощо).
- ***Entity Framework (EF)*** – фреймворк для об'єктно-реляційного відображення даних (object-relational mapping (ORM)).
 - Остання версія EF у складі .NET Framework – Entity Framework 6, вважається legacy-технологією.
 - EF 6.3 підтримується в .NET Core 3.0 для портування старих проектів, які реалізують веб-додатки та служби.
- ***Entity Framework Core*** – кросплатормна версія з відкритим кодом та широкою підтримкою сховищ даних.
 - Крім традиційних СКБД, підтримує хмарні та нереляційні сховища даних.

Зразок бази даних для роботи

- Microsoft пропонує кілька демонстраційних баз даних. Використаємо Northwind.



- Кожна категорія має унікальний ідентифікатор, назву, опис та зображення.
- Кожний товар має унікальний ідентифікатор, назву, ціну за одиницю, кількість у продажу та ін.
- Кожний товар пов'язаний з категорією, зберігаючи унікальний ІД категорії.
- Зв'язок між Categories та Products – 1:M, кожна категорія може мати 0 або більше товарів.

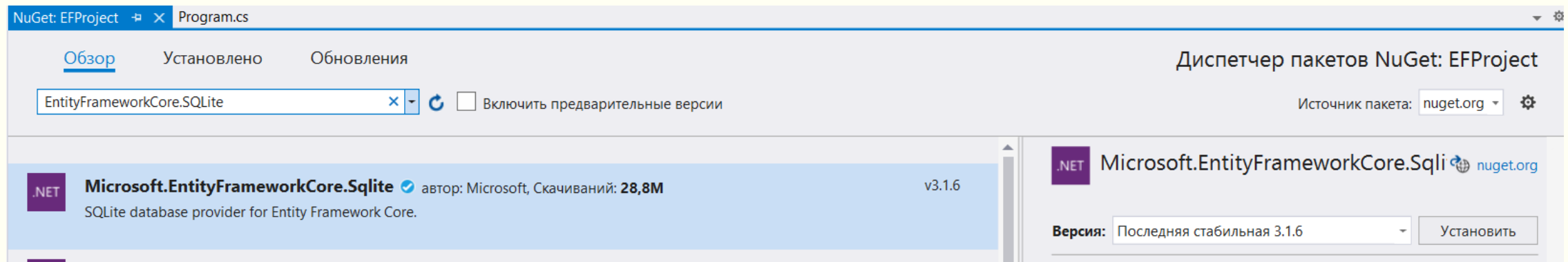
СКБД SQLite та її інсталяція (Windows)

- Сторінка завантаження – <https://www.sqlite.org/download.html>.
 - Обирайте секцію **Precompiled Binaries for Windows**.
 - Розпакуйте завантажений архів, бажано в C:\Sqlite\, та додайте цей шлях до змінної середовища Path.
- Завантажте скрипт для створення БД - <https://github.com/markjprice/cs8dotnetcore3/blob/master/sql-scripts/Northwind.sql> .
 - Скопіюйте його в папку з майбутнім проектом.
 - Перейдіть у цю папку в командному рядку та виконайте команду
 - `sqlite3 Northwind.db < Northwind.sql`
- Редагувати утворену БД (Northwind.db) можна за допомогою [SQLiteStudio](#).

Налаштування EF Core

- *Вибір провайдера даних – набору класів для взаємодії з конкретними сховищами даних.*
 - Постачаються як NuGet-пакети.

СКБД	NuGet-пакет для інсталяції
Microsoft SQL Server 2008 or later	Microsoft.EntityFrameworkCore.SqlServer
SQLite 3.7 or later	Microsoft.EntityFrameworkCore.SQLite
MySQL	MySQL.Data.EntityFrameworkCore
In-memory	Microsoft.EntityFrameworkCore.InMemory



Моделі EF Core

- EF Core застосовує комбінацію угод (conventions), атрибутів анотації та інструкцій Fluent API для побудови entity model at runtime, тому будь-які дії, виконані з класами, можуть пізніше автоматично транслюватись у дії з БД.
 - An entity class represents the structure of a table and an instance of the class represents a row in that table.
- Подальший код передбачатиме наступні угоди:
 - Назва таблиці має відповідати назві властивості DbSet<T> у класі DbContext, наприклад, Products.
 - Назви стовпців мають відповідати назвам властивостей у класі, наприклад, ProductID.
 - Передбачається, що .NET-тип string відповідає типу nvarchar у БД.
 - Тип int відповідає типу int у БД.
 - Властивість ID для класу комбінується з його назвою, наприклад для класу Product – ProductID. Ця властивість передбачається первинним ключем. Якщо вона цілочисельного типу або типу Guid, тоді вона також вважається IDENTITY (стовпчиковий тип, який автоматично присвоює значення при додаванні рядка).

Атрибути анотації EF Core

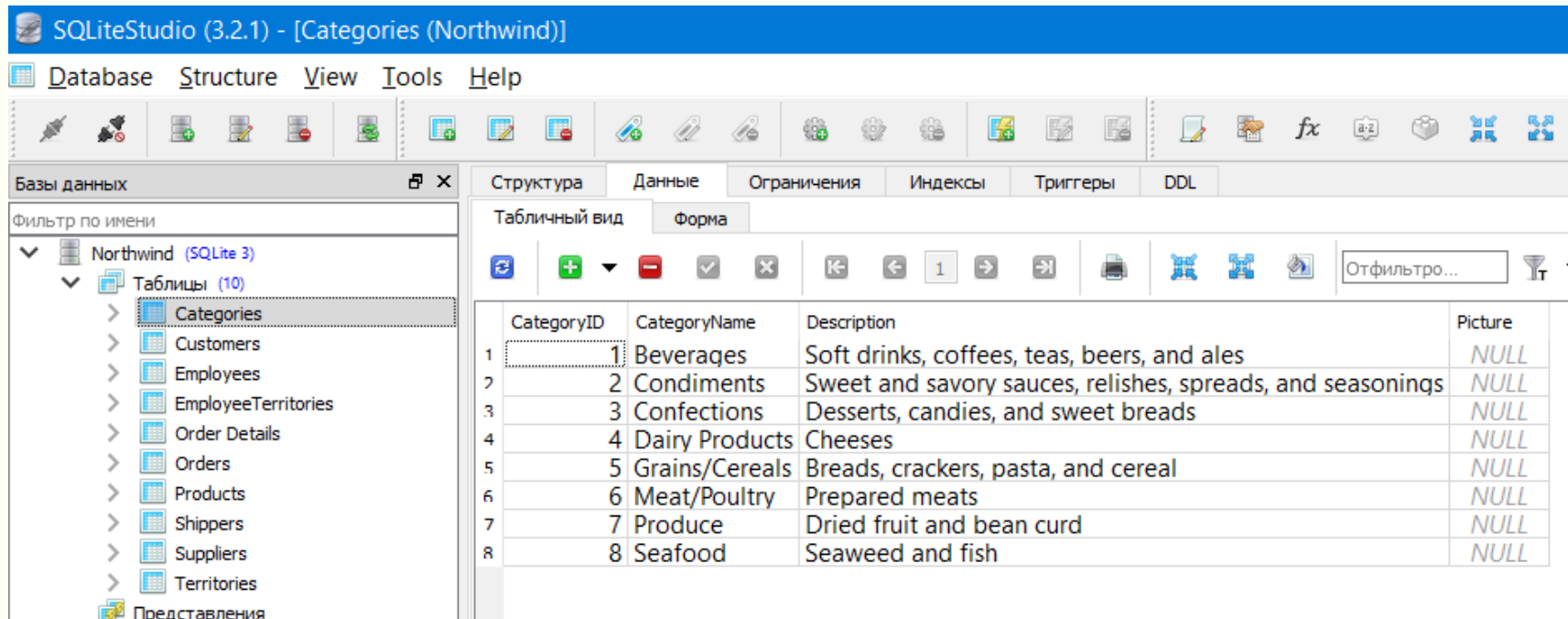
- Угод зазвичай не вистачає для повного відображення класів на об'єкти БД.
 - Наприклад, у БД максимальна довжина назви товару – 40 і значення не може бути null.
 - У класі Product можна застосувати відповідні атрибути:
 - `[Required]`
`[StringLength(40)]`
`public string ProductName { get; set; }`
- Для очевидного відображення між .NET-типами та типами БД може застосовуватись атрибут.
 - Наприклад, у БД тип стовпця UnitPrice таблиці Products table – money.
 - .NET Core не має типу money, тому застосовується тип decimal:
 - `[Column(TypeName = "money")]`
`public decimal? UnitPrice { get; set; }`
- У таблиці Category стовпчик Description може бути довшим за 8000 символів, що можна зберегти в змінній типу nvarchar, тому його потрібно відобразити в тип ntext:
 - `[Column(TypeName = "ntext")]`
`public string Description { get; set; }`

EF Core Fluent API (замість або разом з атрибутами)

- Повернемось до атрибутів у класі Product:
 - `[Required]`
`[StringLength(40)]`
`public string ProductName { get; set; }`
- Для спрощення атрибуту можна видалити з класу та замінити на еквівалентні інструкції Fluent API в методі `OnModelCreating()` з класу контексту БД:
 - `modelBuilder.Entity<Product>()
 .Property(product => product.ProductName)
 .IsRequired()
 .HasMaxLength(40);`
- Fluent API підтримує заповнення БД початковими даними (data seeding).
 - EF Core автоматично works out what insert, update, or delete operations must be executed.
 - Якщо потрібно переконатись, що нова БД має принаймні один рядок у таблиці Product, можемо викликати метод `HasData()`:
 - `modelBuilder.Entity<Product>()
 .HasData(new Product {
 ProductID = 1,
 ProductName = "Chai",
 UnitPrice = 8.99M
 });`

Створення моделі EF Core (1)

- Додамо в проект 3 файли з порожніми класами: Northwind.cs, Category.cs та Product.cs.
 - Category буде використовуватись для представлення таблиці Categories.
 - Стовпчик Picture відображати не будемо.
 - Для відображення стовпця Description з коректним типом даних БД необхідно декорувати рядкову властивість атрибутом Column.



Створення моделі EF Core (1 – Category.cs)

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;

namespace EFProject
{
    class Category
    {
        // these properties map to columns in the database
        public int CategoryID { get; set; }
        public string CategoryName { get; set; }
        [Column(TypeName = "ntext")]
        public string Description { get; set; }
        // defines a navigation property for related rows
        public virtual ICollection<Product> Products { get; set; }

        public Category()
        {
            // to enable developers to add products to a Category we must
            // initialize the navigation property to an empty list
            this.Products = new List<Product>();
        }
    }
}
```

- Product буде використовуватись для представлення рядка в таблиці Products, яка має 10 стовпців.
 - Можна не включати всі стовпці як властивості в клас, візьмемо лише 6: ProductID, ProductName, UnitPrice, UnitsInStock, Discontinued і CategoryID.
 - Невідображені у властивості стовпці не можуть зчитуватись чи записуватись за допомогою екземплярів класу.
 - При створенні об'єкту класу новий рядок у таблиці міститиме NULL чи інше значення за умовчанням для невідображених стовпців.
 - Тут рядки вже мають дані, тому можна відображати не всі стовпці.
 - Можемо перейменувати стовпчик, визначаючи властивість з іншою назвою, як Cost, а потім декоруючи властивість атрибутом [Column] з назвою стовпця, як UnitPrice.

Створення моделі EF Core (2 – Product.cs)

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
```

```
namespace EFProject
{
    class Product
    {
        public int ProductID { get; set; }
        [Required]
        [StringLength(40)]
        public string ProductName { get; set; }
        [Column("UnitPrice", TypeName = "money")]
        public decimal? Cost { get; set; }
        [Column("UnitsInStock")]
        public short? Stock { get; set; }
        public bool Discontinued { get; set; }
        // these two define the foreign key relationship
        // to the Categories table
        public int CategoryID { get; set; }
        public virtual Category Category { get; set; }
    }
}
```

- 2 властивості, які пов'язують 2 сутності, - Category.Products та Product.
- Обидві властивості Category позначені як virtual.
- Це дозволяє EF Core успадковувати та заміщати властивості, доповнюючи їх новою функціональністю, на зразок лінивого завантаження (lazy loading – не доступне до .NET Core 2.1).

Табличний вид Форма

Отфилтровано... Всего загружено строк: 77

	ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInSto	UnitsOnOr	ReorderLe	Discontin
1	1	Chai	1	1	10 boxes x 20 bags	18	39	0	10	0
2	2	Chang	1	1	24 - 12 oz bottles	19	17	40	25	0
3	3	Aniseed Syrup	1	2	12 - 550 ml bottles	10	13	70	25	0
4	4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22	53	0	0	0
5	5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35	0	0	0	1
6	6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25	120	0	25	0
7	7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30	15	0	10	0
8	8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40	6	0	0	0
9	9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97	29	0	0	1
10	10	Ikura	4	8	12 - 200 ml jars	31	31	0	0	0
11	11	Queso Cabrales	5	4	1 kg pkg.	21	22	30	30	0
12	12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs.	38	86	0	0	0
13	13	Konbu	6	8	2 kg box	6	24	0	5	0
14	14	Tofu	6	7	40 - 100 g pkgs.	23.25	35	0	0	0
15	15	Genen Shouyu	6	2	24 - 250 ml bottles	15.5	39	0	5	0
16	16	Pavlova	7	3	32 - 500 g boxes	17.45	29	0	10	0
17	17	Alice Mutton	7	6	20 - 1 kg tins	39	0	0	0	1
18	18	Carnarvon Tigers	7	8	16 kg pkg.	62.5	42	0	0	0
19	19	Teatime Chocolate Biscuits	8	3	10 boxes x 12 pieces	9.2	25	0	5	0
20	20	Sir Rodney's Marmalade	8	3	30 gift boxes	81	40	0	0	0
21	21	Sir Rodney's Scones	8	2	24 pkgs x 4 pieces	10	2	40	5	0

```

public class Northwind : DbContext {
    public DbSet<Category> Categories { get; set; }
    public DbSet<Product> Products { get; set; }

    protected override void OnConfiguring(
        DbContextOptionsBuilder optionsBuilder) {
        string path = System.IO.Path.Combine(
            System.Environment.CurrentDirectory, "Northwind.db");

        optionsBuilder.UseSqlite($"Filename={path}");
    }

    protected override void OnModelCreating(
        ModelBuilder modelBuilder) {
        // приклад застосування Fluent API для обмеження
        // назви категорії 15ма символами
        modelBuilder.Entity<Category>()
            .Property(category => category.CategoryName)
            .IsRequired() // NOT NULL
            .HasMaxLength(15);

        // додано для «виправлення» підтримки decimal в SQLite
        modelBuilder.Entity<Product>()
            .Property(product => product.Cost)
            .HasConversion<double>();
    }
}

```

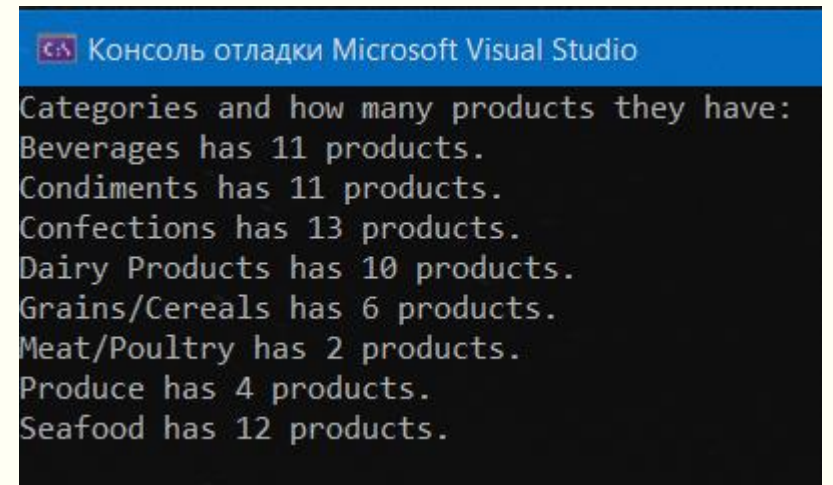
Визначення класу контексту БД – Northwind (3)

- Представлятиме та розумітиме, як працювати з БД, динамічно генеруватиме команди SQL та маніпулюватиме даними.
 - Для використання EF Core клас повинен наслідуватись від DbContext.
- Всередині необхідно визначити принаймні одну властивість типу DbSet<T>, яка представлятиме таблицю.
- Бажано мати переозначений метод OnConfiguring(), який задає connection string для БД.
 - Опційно може переозначатись метод OnModelCreating(), в якому можна записувати інструкції Fluent API в якості альтернативи до декорування атрибутами entity-класів.

Формування запитів до моделей EF Core

- Запишемо прості LINQ-запити, детальніше про LINQ буде пізніше.
 - У класі Program визначимо метод QueryingCategories() та додамо інструкції для виконання:
 - Створення екземпляру класу Northwind, який управлятиме БД.
 - Створення запиту для всіх категорій, який включає пов'язані з ними товари.
 - Обхід категорій з виводом назви та кількості товарів для кожної з них.

```
static void QueryingCategories() {  
    using (var db = new Northwind()) {  
        WriteLine("Categories and how many products they have:");  
  
        // запит на отримання всіх категорій та пов'язаних з ними товарів  
        IQueryable<Category> cats = db.Categories  
            .Include(c => c.Products);  
  
        foreach (Category c in cats) {  
            WriteLine($"{c.CategoryName} has {c.Products.Count} products.");  
        }  
    }  
}  
  
static void Main(string[] args) {  
    QueryingCategories();  
}
```



Консоль отладки Microsoft Visual Studio

Categories and how many products they have:
Beverages has 11 products.
Condiments has 11 products.
Confections has 13 products.
Dairy Products has 10 products.
Grains/Cereals has 6 products.
Meat/Poultry has 2 products.
Produce has 4 products.
Seafood has 12 products.

Вирішення SQLite Error 1

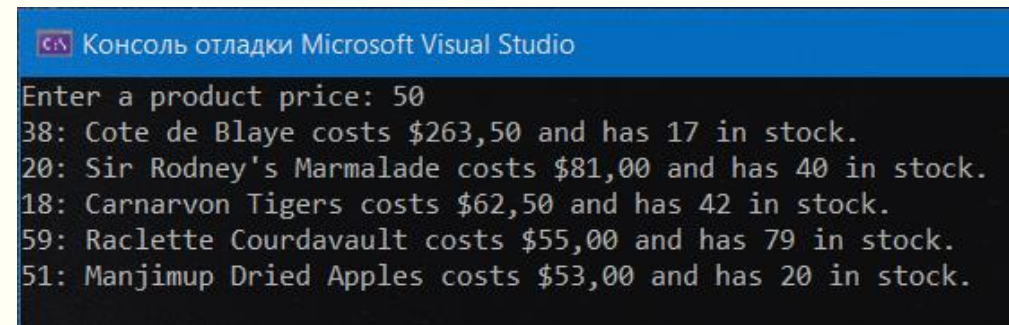
Фільтрування та сортування товарів (метод QueryingProducts())

```
static void QueryingProducts() {  
    using (var db = new Northwind()) {  
        WriteLine("Products that cost more " +  
            "than a price, highest at top.");  
        string input;  
        decimal price;  
        do {  
            Write("Enter a product price: ");  
            input = ReadLine();  
        } while (!decimal.TryParse(input, out price));
```

```
        IQueryable<Product> prods = db.Products  
            .Where(product => product.Cost > price)  
            .OrderByDescending(product => product.Cost);
```

```
        foreach (Product item in prods) {  
            WriteLine(  
                "{0}: {1} costs {2:$#,##0.00} and has {3} in stock.",  
                item.ProductID, item.ProductName, item.Cost, item.Stock);  
        }  
    }  
}
```

- Додамо інструкції, які виконуватимуть наступне:
 - Створять екземпляр класу Northwind, який керуватиме БД.
 - Запросить користувача ввести ціни на товари.
 - Створить запит щодо товарів, які коштують дорожче заданої ціни (за допомогою LINQ).
 - Пройти результати, виводячи ID, назву, ціну (в доларах) та кількість штук у продажу.
- Додамо виклик методу в Main(), закомментувавши QueryingCategories();



```
Консоль отладки Microsoft Visual Studio  
Enter a product price: 50  
38: Cote de Blaye costs $263,50 and has 17 in stock.  
20: Sir Rodney's Marmalade costs $81,00 and has 40 in stock.  
18: Carnarvon Tigers costs $62,50 and has 42 in stock.  
59: Raclette Courdavault costs $55,00 and has 79 in stock.  
51: Manjimup Dried Apples costs $53,00 and has 20 in stock.
```

Логування EF Core

- Моніторинг взаємодії між EF Core та БД передбачає логування, яке вимагає виконання наступних завдань:
 - Реєстрацію постачальника логера (logging provider).
 - Реалізацію логера.
- Додамо в проект файл ConsoleLogger.cs з відповідними класами:
 - ConsoleLoggerProvider повертає екземпляр ConsoleLogger та не потребує unmanaged-ресурсів, тому метод Dispose нічого не робить, проте повинен існувати
 - ConsoleLogger відключений для рівнів логування None, Trace та Information і включений для решти рівнів.

```
public class ConsoleLoggerProvider : ILoggerProvider
{
    public ILogger CreateLogger(string categoryName)
    {
        return new ConsoleLogger();
    }

    // if your logger uses unmanaged resources,
    // you can release the memory here
    public void Dispose() { }
}
```

```
public class ConsoleLogger : ILogger {
    // if your logger uses unmanaged resources, you can
    // return the class that implements IDisposable here
    public IDisposable BeginScope<TState>(TState state) { return null; }
    // to avoid overlogging, you can filter on the log level
    public bool IsEnabled(LogLevel logLevel) {
        switch (logLevel) {
            case LogLevel.Trace:
            case LogLevel.Information:
            case LogLevel.None:
                return false;
            case LogLevel.Debug:
            case LogLevel.Warning:
            case LogLevel.Error:
            case LogLevel.Critical:
            default:
                return true;
        }
    };
}

public void Log<TState>(LogLevel logLevel,
    EventId eventId, TState state, Exception exception,
    Func<TState, Exception, string> formatter)
{
    // log the level and event identifier
    Write($"Level: {logLevel}, Event ID: {eventId.Id}");

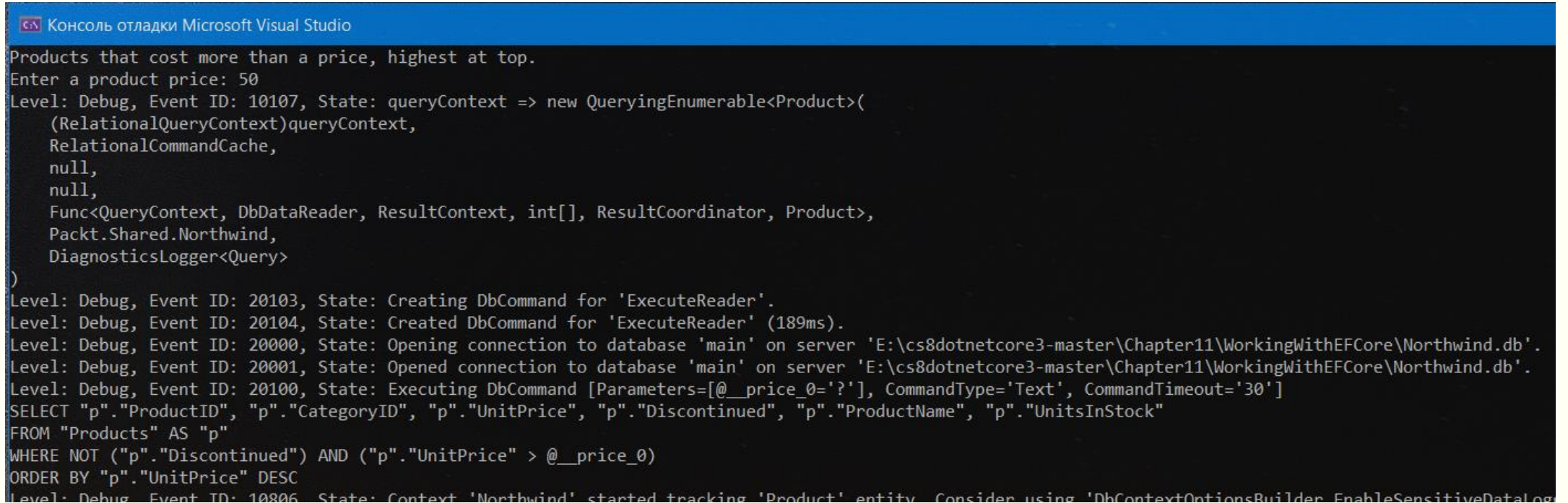
    // only output the state or exception if it exists
    if (state != null) { Write($"", State: {state}); }
    if (exception != null) {
        Write($"", Exception: {exception.Message});
    }
    WriteLine();
}
}
```


Логування EF Core

- У методи `QueryingCategories()` та `QueryingProducts()` додамо інструкції логування в `using`-блок для контексту БД Northwind:

```
using (var db = new Northwind()) {  
    var loggerFactory = db.GetService<ILoggerFactory>();  
    loggerFactory.AddProvider(new ConsoleLoggerProvider());  
    .....
```

- Вивід для `QueryingProducts()`:

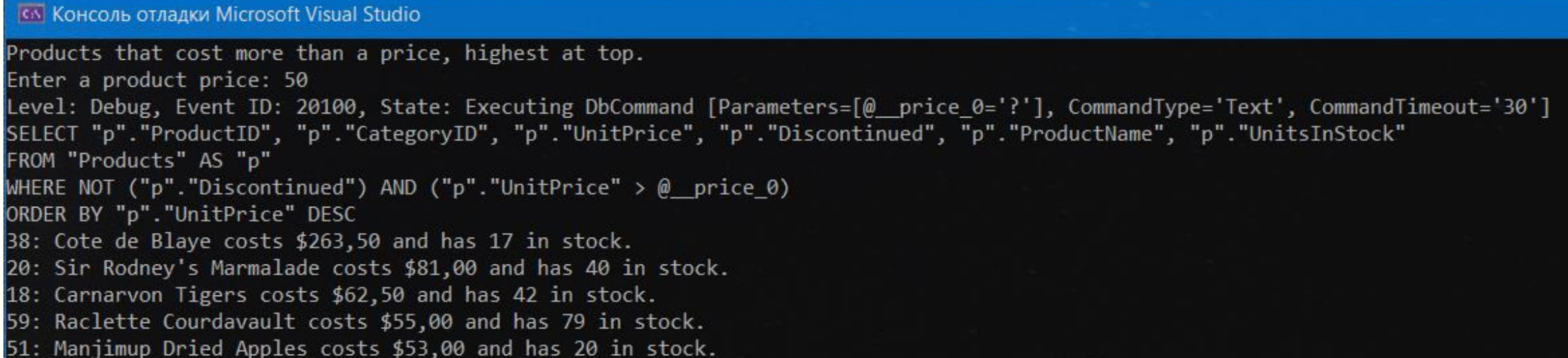


```
Консоль отладки Microsoft Visual Studio  
Products that cost more than a price, highest at top.  
Enter a product price: 50  
Level: Debug, Event ID: 10107, State: queryContext => new QueryingEnumerable<Product>(  
    (RelationalQueryContext)queryContext,  
    RelationalCommandCache,  
    null,  
    null,  
    Func<QueryContext, DbDataReader, ResultContext, int[], ResultCoordinator, Product>,  
    Packt.Shared.Northwind,  
    DiagnosticsLogger<Query>  
)  
Level: Debug, Event ID: 20103, State: Creating DbCommand for 'ExecuteReader'.  
Level: Debug, Event ID: 20104, State: Created DbCommand for 'ExecuteReader' (189ms).  
Level: Debug, Event ID: 20000, State: Opening connection to database 'main' on server 'E:\cs8dotnetcore3-master\Chapter11\WorkingWithEFCore\Northwind.db'.  
Level: Debug, Event ID: 20001, State: Opened connection to database 'main' on server 'E:\cs8dotnetcore3-master\Chapter11\WorkingWithEFCore\Northwind.db'.  
Level: Debug, Event ID: 20100, State: Executing DbCommand [Parameters=[@__price_0=?], CommandType='Text', CommandTimeout='30']  
SELECT "p"."ProductID", "p"."CategoryID", "p"."UnitPrice", "p"."Discontinued", "p"."ProductName", "p"."UnitsInStock"  
FROM "Products" AS "p"  
WHERE NOT ("p"."Discontinued") AND ("p"."UnitPrice" > @__price_0)  
ORDER BY "p"."UnitPrice" DESC  
Level: Debug, Event ID: 10806, State: Context 'Northwind' started tracking 'Product' entity. Consider using 'DbContextOptionsBuilder.EnableSensitiveDataLog'
```

Логування EF Core

```
public void Log<TState>(LogLevel logLevel,
    EventId eventId, TState state, Exception exception,
    Func<TState, Exception, string> formatter)
{
    if (eventId.Id == 20100)
    {
        // log the level and event identifier
        Write($"Level: {logLevel}, Event ID: {eventId.Id}");
        .....
    }
}
```

- Значення ID події та їх зміст будуть специфічними для постачальника даних (.NET data provider).
 - Якщо бажаємо знати, як LINQ-запит транслюється в SQL-інструкції та виконується, тоді продемонструємо вивід Event ID із значенням 20100.
 - Огорнемо тіло методу Log у класі ConsoleLogger умовою на виведення тільки подій з Id = 20100



```
Консоль отладки Microsoft Visual Studio
Products that cost more than a price, highest at top.
Enter a product price: 50
Level: Debug, Event ID: 20100, State: Executing DbCommand [Parameters=[@__price_0='?'], CommandType='Text', CommandTimeout='30']
SELECT "p"."ProductID", "p"."CategoryID", "p"."UnitPrice", "p"."Discontinued", "p"."ProductName", "p"."UnitsInStock"
FROM "Products" AS "p"
WHERE NOT ("p"."Discontinued") AND ("p"."UnitPrice" > @__price_0)
ORDER BY "p"."UnitPrice" DESC
38: Cote de Blaye costs $263,50 and has 17 in stock.
20: Sir Rodney's Marmalade costs $81,00 and has 40 in stock.
18: Carnarvon Tigers costs $62,50 and has 42 in stock.
59: Raclette Courdavault costs $55,00 and has 79 in stock.
51: Manjimup Dried Apples costs $53,00 and has 20 in stock.
```

Логування з тегами запитів (query tags)

- При логуванні LINQ-запитів може бути складно узгодити повідомлення в лозі у складних сценаріях виконання.
 - В EF Core 2.2 представлено теги запитів, які допомагають, дозволяючи додати SQL-коментарі в лог.
 - Можемо анотувати запит LINQ, застосовуючи метод TagWith():

```
IQueryable<Product> prods = db.Products  
    .TagWith("Products filtered by price and sorted.")  
    .Where(product => product.Cost > price)  
    .OrderByDescending(product => product.Cost);
```

- У результаті виведеться: -- Products filtered by price and sorted.
 - Детальніше [тут](#).

Співвіднесення шаблонів (Pattern matching) за допомогою Like

```
static void QueryingWithLike() {
    using (var db = new Northwind()) {
        var loggerFactory = db.GetService<ILoggerFactory>();
        loggerFactory.AddProvider(new ConsoleLoggerProvider());

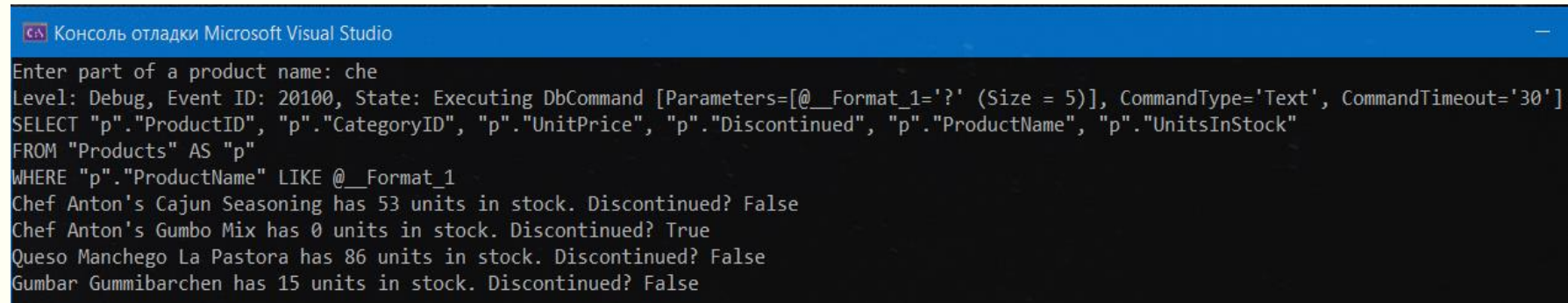
        Write("Enter part of a product name: ");
        string input = ReadLine();

        IQueryable<Product> prods = db.Products
            .Where(p => EF.Functions.Like(p.ProductName, $"%{input}%"));

        foreach (Product item in prods) {
            WriteLine("{0} has {1} units in stock. Discontinued? {2}",
                item.ProductName, item.Stock, item.Discontinued);
        }
    }
}

static void Main(string[] args) {
    // QueryingCategories();
    // QueryingProducts();
    QueryingWithLike();
}
```

- EF Core підтримує поширені команди SQL, включаючи Like.
- У Program додайте метод QueryingWithLike()
 - Було задіяно логування.
 - Пропонуємо користувачу ввести частину назви товару, а потім використовуємо метод EF.Functions.Like() для пошуку всередині властивості ProductName.
 - Для кожного співвіднесеного товару виводимо його назву, кількість на складі та стан продажу.



```
Консоль отладки Microsoft Visual Studio
Enter part of a product name: che
Level: Debug, Event ID: 20100, State: Executing DbCommand [Parameters=[@__Format_1='?' (Size = 5)], CommandType='Text', CommandTimeout='30']
SELECT "p"."ProductID", "p"."CategoryID", "p"."UnitPrice", "p"."Discontinued", "p"."ProductName", "p"."UnitsInStock"
FROM "Products" AS "p"
WHERE "p"."ProductName" LIKE @__Format_1
Chef Anton's Cajun Seasoning has 53 units in stock. Discontinued? False
Chef Anton's Gumbo Mix has 0 units in stock. Discontinued? True
Queso Manchego La Pastora has 86 units in stock. Discontinued? False
Gumbar Gummibarchen has 15 units in stock. Discontinued? False
```

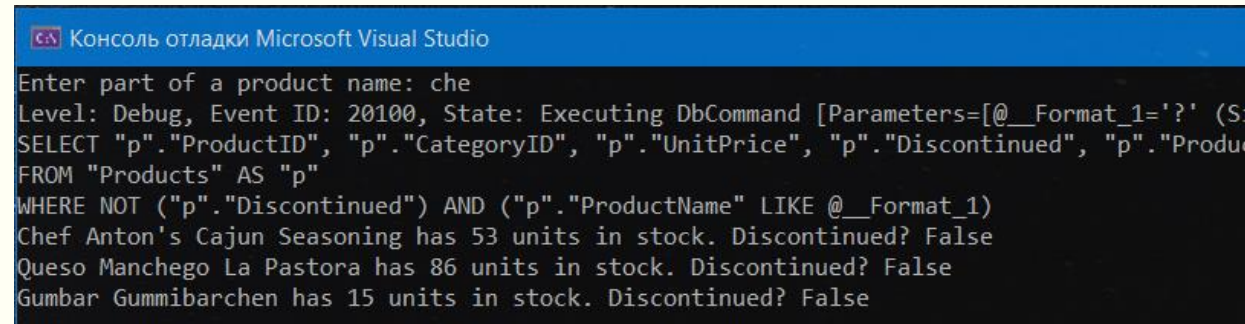

Визначення глобальних фільтрів

```
protected override void OnModelCreating(
    modelBuilder modelBuilder) {
    // приклад застосування Fluent API для обмеження
    // назви категорії 15ма символами
    modelBuilder.Entity<Category>()
        .Property(category => category.CategoryName)
        .IsRequired() // NOT NULL
        .HasMaxLength(15);

    // додано для «виправлення» підтримки decimal в SQLite
    modelBuilder.Entity<Product>()
        .Property(product => product.Cost)
        .HasConversion<double>();

    // глобальний фільтр для видалення discontinued товарів
    modelBuilder.Entity<Product>()
        .HasQueryFilter(p => !p.Discontinued);
}
```

- Товари в Northwind можуть зніматись з продажу (be discontinued), тому корисно забезпечити неповернення знятих товарів у результати пошуку, навіть якщо програміст забув додати фільтр у метод розширення Where.
- Внесемо зміни в метод OnModelCreating() з класу Northwind, додавши глобальний фільтр на видалення знятих з продажу товарів.
- У виводі зник **Chef Anton's Gumbo Mix**



Консоль отладки Microsoft Visual Studio

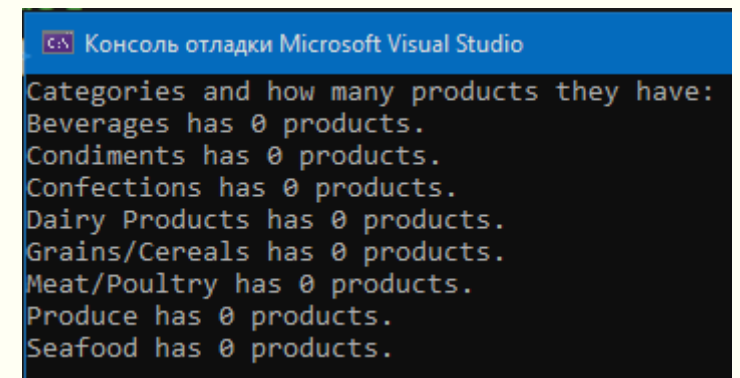
```
Enter part of a product name: che
Level: Debug, Event ID: 20100, State: Executing DbCommand [Parameters=[@__Format_1='?' (S
SELECT "p"."ProductID", "p"."CategoryID", "p"."UnitPrice", "p"."Discontinued", "p"."Produ
FROM "Products" AS "p"
WHERE NOT ("p"."Discontinued") AND ("p"."ProductName" LIKE @__Format_1)
Chef Anton's Cajun Seasoning has 53 units in stock. Discontinued? False
Queso Manchego La Pastora has 86 units in stock. Discontinued? False
Gumbar Gummibarchen has 15 units in stock. Discontinued? False
```

Схеми завантаження даних (Loading patterns) в EF Core

- Entity Framework пропонує три найбільш популярні схеми завантаження даних:
 - Ліниве завантаження (lazy loading)
 - Жадібне завантаження (eager loading, early loading)
 - Явне завантаження (explicit loading)
- У методі `QueryingCategories()` код зараз використовує властивість `Categories`, щоб обійти циклом кожну категорію, виводячи назву категорії та кількість товарів у ній.
 - Працює, тому що написаний запит містить метод `Include()`, який застосовує жадібне завантаження для відповідних товарів.
 - Закоментуємо його та викличемо метод `QueryingCategories()` у `Main()`.

```
IQueryable<Category> cats = db.Categories; //.Include(c => c.Products);
```

- Кожний елемент `foreach` є екземпляром класу `Category`, що має властивість `Products` – перелік товарів у категорії.
- Оскільки первинний запит вибирав лише в таблиці `Categories`, властивість буде порожньою для кожної категорії.



```
Консоль отладки Microsoft Visual Studio
Categories and how many products they have:
Beverages has 0 products.
Condiments has 0 products.
Confections has 0 products.
Dairy Products has 0 products.
Grains/Cereals has 0 products.
Meat/Poultry has 0 products.
Produce has 0 products.
Seafood has 0 products.
```

Ліниве завантаження даних

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Proxies;

namespace Packt.Shared
{
    // this manages the connection to the database
    public class Northwind : DbContext
    {
        // these properties map to tables in the database
        public DbSet<Category> Categories { get; set; }
        public DbSet<Product> Products { get; set; }

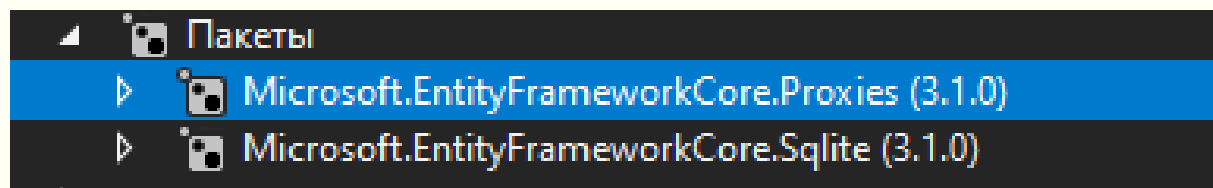
        protected override void OnConfiguring(
            DbContextOptionsBuilder optionsBuilder)
        {
            string path = System.IO.Path.Combine(
                System.Environment.CurrentDirectory, "Northwind.db");

            optionsBuilder.UseLazyLoadingProxies()
                .UseSqlite($"Filename={path}");
        }
    }
}
```

...

```
Консоль отладки Microsoft Visual Studio
Categories and how many products they have:
Beverages has 11 products.
Condiments has 11 products.
Confections has 13 products.
Dairy Products has 10 products.
Grains/Cereals has 6 products.
Meat/Poultry has 2 products.
Produce has 4 products.
Seafood has 12 products.
```

- Представлене в EF Core 2.1 та може автоматично підвантажувати відсутні пов'язані дані.
 - Для цього спочатку потрібно послатись на NuGet-пакет для завантаження з проксі
- У файлі Northwind.cs далі імпортується простір імен Microsoft.EntityFrameworkCore.Proxies та викликається метод розширення для застосування лінивого завантаження проксі.



```

using (var db = new Northwind())
{
    //var loggerFactory = db.GetService<ILoggerFactory>();
    //loggerFactory.AddProvider(new ConsoleLoggerProvider());

    WriteLine("Categories and how many products they have:");

    // a query to get all categories and their related products
    IQueryable<Category> cats; // = db.Categories; //.Include(c => c.Products);

    db.ChangeTracker.LazyLoadingEnabled = false;

    Write("Enable eager loading? (Y/N): ");
    bool eagerloading = (ReadKey().Key == ConsoleKey.Y);
    bool explicitloading = false;
    WriteLine();
    if (eagerloading) {
        cats = db.Categories.Include(c => c.Products);
    }
    else {
        cats = db.Categories;
        Write("Enable explicit loading? (Y/N): ");
        explicitloading = (ReadKey().Key == ConsoleKey.Y);
        WriteLine();
    }

    foreach (Category c in cats) {
        if (explicitloading) {
            Write($"Explicitly load products for {c.CategoryName}? (Y/N): ");
            ConsoleKeyInfo key = ReadKey();
            WriteLine();

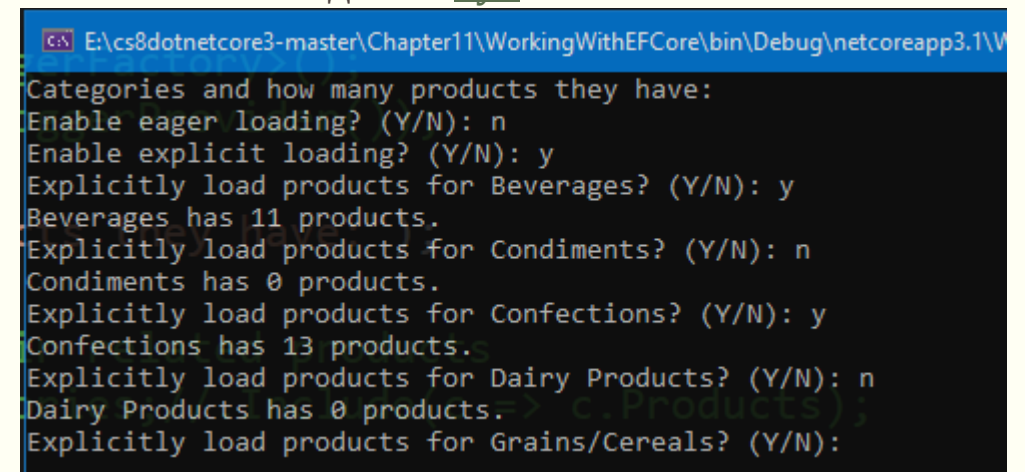
            if (key.Key == ConsoleKey.Y) {
                var products = db.Entry(c).Collection(c2 => c2.Products);
                if (!products.IsLoaded) products.Load();
            }
        }
        WriteLine($"{c.CategoryName} has {c.Products.Count} products.");
    }
}

```

Явне завантаження сутностей

Працює схоже до лінивого завантаження з тією відмінністю, що ви повністю контролюєте, які саме пов'язані дані завантажувати та коли.

- У методі `QueryingCategories()` відключимо ліниве завантаження, а потім запропонуємо користувачу включити жадібне та явне завантаження.
- Детальніше про вибір методу завантаження пов'язаних даних [тут](#).



```

E:\cs8dotnetcore3-master\Chapter11\WorkingWithEFCore\bin\Debug\netcoreapp3.1\WorkingWithEFCore.exe
Categories and how many products they have:
Enable eager loading? (Y/N): n
Enable explicit loading? (Y/N): y
Explicitly load products for Beverages? (Y/N): y
Beverages has 11 products.
Explicitly load products for Condiments? (Y/N): n
Condiments has 0 products.
Explicitly load products for Confections? (Y/N): y
Confections has 13 products.
Explicitly load products for Dairy Products? (Y/N): n
Dairy Products has 0 products.
Explicitly load products for Grains/Cereals? (Y/N):

```


Керування даними за допомогою EF Core

```
static bool AddProduct(int categoryID,
                      string productName, decimal? price)
{
    using (var db = new Northwind())
    {
        var newProduct = new Product
        {
            CategoryID = categoryID,
            ProductName = productName,
            Cost = price
        };

        // позначити товар як відстежуваний щодо змін
        db.Products.Add(newProduct);

        // зберегти відстежувані зміни в БД
        int affected = db.SaveChanges();
        return (affected == 1);
    }
}
```

- DbContext підтримує трекінг внесених у БД змін автоматично, тому локальні сутності можуть мати багато відстежуваних змін: додавання сутностей, зміни вже існуючих чи їх видалення.
 - При готовності відправити зміни в БД викликаємо метод `SaveChanges()`.
 - Буде повернено кількість успішно змінених елементів.
- Додавання елементів.
 - У клас `Program` додамо реалізацію методу `AddProduct()`.
 - Доповнимо клас методом для виводу всіх товарів – `ListProducts`.
 - А в методі `Main()` виконаємо інструкції:

```
if (AddProduct(6, "Bob's Burgers", 500M)) {
    WriteLine("Add product successful.");
}
ListProducts();
```

Виведення елементів

```
static void ListProducts()
{
    using (var db = new Northwind()) {
        WriteLine("{0,-3} {1,-35} {2,8} {3,5} {4}",
            "ID", "Product Name", "Cost", "Stock", "Disc.");

        foreach (var item in
            db.Products.OrderByDescending(p => p.Cost))
        {
            WriteLine("{0:000} {1,-35} {2,8:$#,##0.00} {3,5} {4}",
                item.ProductID, item.ProductName, item.Cost,
                item.Stock, item.Discontinued);
        }
    }
}
```

Консоль отладки Microsoft Visual Studio

Add product successful.

ID	Product Name	Cost	Stock	Disc.
078	Bob's Burgers	\$500,00		False
038	Cote de Blaye	\$263,50	17	False
020	Sir Rodney's Marmalade	\$81,00	40	False
018	Carnarvon Tigers	\$62,50	42	False
059	Raclette Courdavault	\$55,00	79	False
051	Manjimup Dried Apples	\$53,00	20	False
062	Tarte au sucre	\$49,30	17	False
043	Ipoh Coffee	\$46,00	17	False
027	Schoggi Schokolade	\$43,90	49	False
063	Vegie-spread	\$43,90	24	False
008	Northwoods Cranberry Sauce	\$40,00	6	False
012	Queso Manchego La Pastora	\$38,00	86	False
056	Gnocchi di nonna Alice	\$38,00	21	False
069	Gudbrandsdalsost	\$36,00	26	False
072	Mozzarella di Giovanni	\$34,80	14	False
060	Camembert Pierrot	\$34,00	19	False
064	Wimmers gute Semmelknodel	\$33,25	22	False
032	Mascarpone Fabioli	\$32,00	9	False
026	Gumbar Gummibarchen	\$31,23	15	False
010	Ikura	\$31,00	31	False
007	Uncle Bob's Organic Dried Pears	\$30,00	15	False

Оновлення елементів

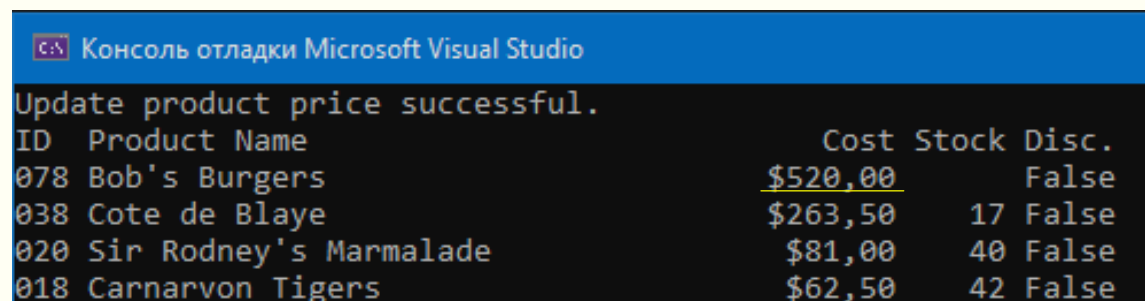
```
static bool IncreaseProductPrice(string name, decimal amount)
{
    using (var db = new Northwind())
    {
        // отримуємо 1й товар, назва якого починається з name
        Product updateProduct = db.Products.First(
            p => p.ProductName.StartsWith(name));

        updateProduct.Cost += amount;

        int affected = db.SaveChanges();
        return (affected == 1);
    }
}
```

```
static void Main(string[] args)
{
    if (IncreaseProductPrice("Bob", 20M))
    {
        WriteLine("Update product price successful.");
    }
    ListProducts();
}
```

■ Додамо в клас Program метод, щоб на 20\$ збільшити ціну першого товару, назва якого починається зі слова Bob.



ID	Product Name	Cost	Stock	Disc.
078	Bob's Burgers	<u>\$520,00</u>	17	False
038	Cote de Blaye	\$263,50	17	False
020	Sir Rodney's Marmalade	\$81,00	40	False
018	Carnarvon Tigers	\$62,50	42	False

Видалення елементів

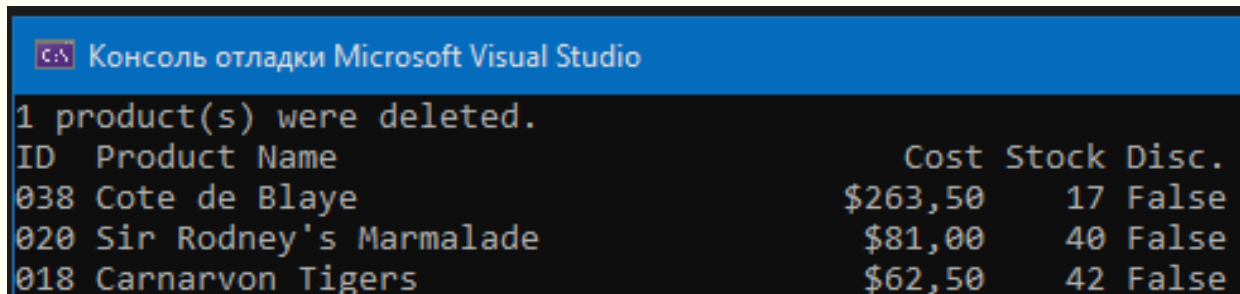
```
static int DeleteProducts(string name)
{
    using (var db = new Northwind())
    {
        var products = db.Products.Where(
            p => p.ProductName.StartsWith(name));

        db.Products.RemoveRange(products);

        int affected = db.SaveChanges();
        return affected;
    }
}
```

```
static void Main(string[] args)
{
    int deleted = DeleteProducts("Bob");
    WriteLine($"{deleted} product(s) were deleted.");
    ListProducts();
}
```

- У клас Program імпортуйте System.Collections.Generic, а потім додайте метод для видалення всіх товарів з назвою, яка починається з «Bob».
 - Можна видаляти окремі елементи за допомогою методу Remove() або кілька елементів за допомогою RemoveRange().



```
Консоль отладки Microsoft Visual Studio
1 product(s) were deleted.
ID  Product Name          Cost Stock Disc.
038 Cote de Blaye        $263,50  17 False
020 Sir Rodney's Marmalade $81,00  40 False
018 Carnarvon Tigers      $62,50  42 False
```

Групування (pooling) контекстів БД

- Клас DbContext є disposable та спроектований, слідуючи принципу unit-of-work.
 - У попередніх прикладах ми створювали всі породжені від DbContext екземпляри Northwind у блоці using.
- Можливість ASP.NET Core, пов'язана з EF Core, – підвищення ефективності коду шляхом групування контекстів БД при збиранні веб-додатків та веб-служб.
 - Це дозволяє створювати та видаляти (dispose) якзавгодно багато об'єктів типу, породженого від DbContext.
- Довідка щодо пулів контекстів.

Транзакції в базах даних

- При кожному виклику метода `SaveChanges()` система запускає неявну транзакцію, тому якщо будуть проблеми, система автоматично відкотить усі внесені зміни.
 - У випадку успішного завершення операції транзакція вважається здійсненою.
 - Транзакції дозволяють зберегти цілісність БД за допомогою блокування зчитування/запису до моменту завершення послідовності операцій.
- Для характеристики транзакцій часто використовують аббревіатуру ACID.
 - **A** (*atomic* — «**атомарний, неділимий**»): або здійснюються всі операції поточної транзакції, або жодна.
 - **C** (*consistent* — «**узгоджений**»): БД до та після здійснення транзакції не містить помилок. Це залежить від логіки програмного коду.
 - **I** (*isolated* — «**ізолирований**»): у ході виконання транзакції зміни, що вносяться, не видно іншим процесам.
 - Існує кілька рівнів ізоляваності транзакцій; чим вище рівень, тим вище цілісність даних.
 - Проте для такого встановлення потрібно ввести багато блокувань, що негативно вплине на роботу інших процесів.
 - Знімки стану (snapshot) – особливий випадок, оскільки вони є копіями рядків таблиці, дозволяючи уникнути блокувань, проте збільшуючи розмір БД при виконанні транзакцій.
 - **D** (*durable* — «**стійкий**»): якщо при виконанні транзакції виникне помилка, первинний стан БД можна відновити. Тут durable — антонім volatile (нестійкий).

Рівні ізолюваності в .NET

Уровень изолированности	Блокировка (-и)	Допустимые проблемы целостности данных
ReadUncommitted	Нет	Грязное чтение, неповторяемое чтение, фантомные данные
ReadCommitted	При редактировании применяется блокировка, предотвращающая чтение записи (-ей) другими пользователями до завершения транзакции	Неповторяемое чтение и фантомные данные
RepeatableRead	При чтении применяется блокировка, предотвращающая редактирование записи (-ей) другими пользователями до завершения транзакции	Фантомные данные
Serializable	Применяются блокировки уровня диапазона ключа, предотвращающие любые действия, способные повлиять на результат, в том числе вставку и удаление данных	Нет
Snapshot	Нет	Нет

Визначення явної транзакції

```
static int DeleteProducts(string name)
{
    using (var db = new Northwind())
    {
        using (IDbContextTransaction t =
            db.Database.BeginTransaction())
        {
            WriteLine("Transaction isolation level: {0}",
                t.GetDbTransaction().IsolationLevel);

            var products = db.Products.Where(
                p => p.ProductName.StartsWith(name));

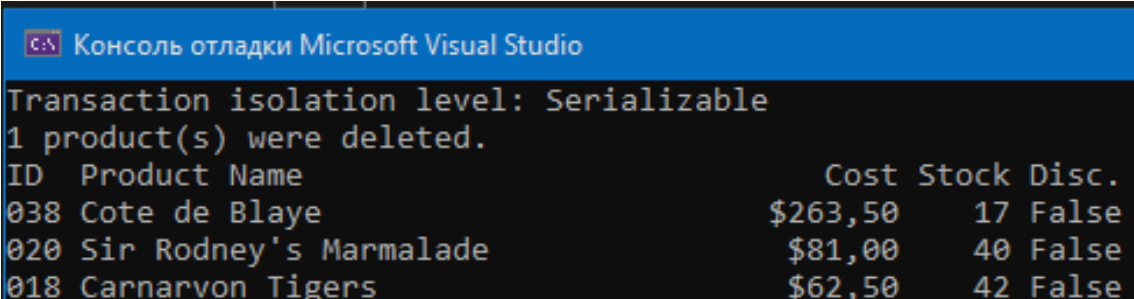
            db.Products.RemoveRange(products);

            int affected = db.SaveChanges();
            t.Commit();
            return affected;
        }
    }
}

static void Main(string[] args)
{
    int deleted = DeleteProducts("Bob");
    WriteLine($"{deleted} product(s) were deleted.");
    ListProducts();
}
```

06.09.2020

- За допомогою властивості Database контексту БД можна керувати явними транзакціями.
 - Імпортуйте в файл класу Program.cs простір імен **Microsoft.EntityFrameworkCore.Storage** для підключення інтерфейса IDbContextTransaction.
 - Після ініціалізації змінної db додайте в метод DeleteProducts() інструкції для запуску явної транзакції та виводу рівня ізолюваності цієї транзакції.
 - При використанні Microsoft SQL Server рівень ізолюваності:
 - Transaction started with this isolation level: ReadCommitted
 - При роботі з SQLite рівень ізолюваності:
 - Transaction started with this isolation level: Serializable

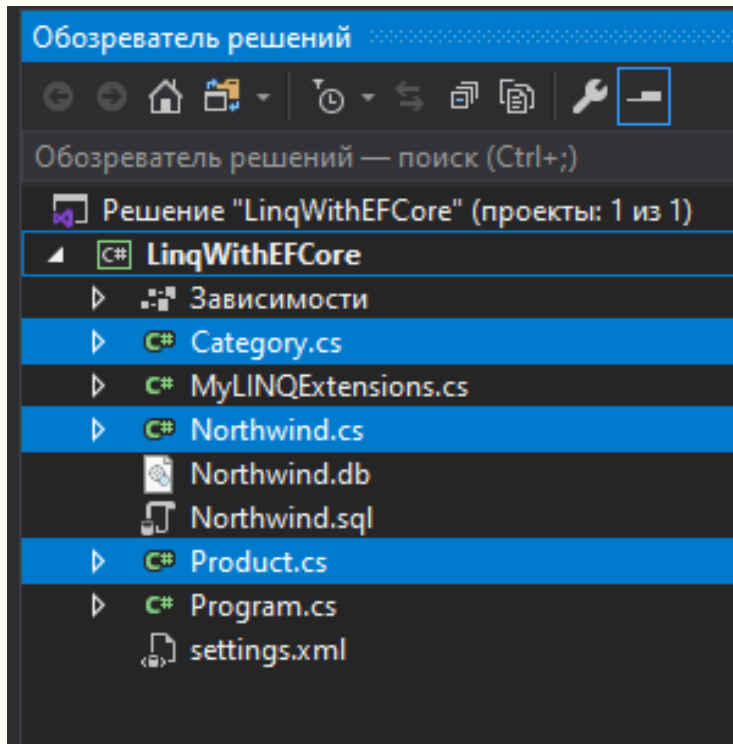


```
Консоль отладки Microsoft Visual Studio
Transaction isolation level: Serializable
1 product(s) were deleted.
ID  Product Name          Cost Stock Disc.
038 Cote de Blaye         $263,50  17 False
020 Sir Rodney's Marmalade $81,00  40 False
018 Carnarvon Tigers      $62,50  42 False
```


Застосування LINQ на платформі EF Core

- *Проекція елементів за допомогою ключового слова `select`*

- Файл Northwind.cs



```
using Microsoft.EntityFrameworkCore;

namespace Packt.Shared
{
    // this manages the connection to the database
    public class Northwind : DbContext
    {
        // these properties map to tables in the database
        public DbSet<Category> Categories { get; set; }
        public DbSet<Product> Products { get; set; }

        protected override void OnConfiguring(
            DbContextOptionsBuilder optionsBuilder)
        {
            string path = System.IO.Path.Combine(
                System.Environment.CurrentDirectory, "Northwind.db");

            optionsBuilder.UseSqlite($"Filename={path}");
        }
    }
}
```

Застосування LINQ на платформі EF Core

■ Проекція елементів за допомогою ключового слова *select*

- Файли Category.cs та Product.cs
- Ми спеціально не визначили відношення між 2 класами сутностей. Пізніше застосуємо LINQ для з'єднання 2 наборів сутностей.

```
using System.ComponentModel.DataAnnotations;
```

```
namespace Packt.Shared
{
    public class Category
    {
        public int CategoryID { get; set; }

        [Required]
        [StringLength(15)]
        public string CategoryName { get; set; }

        public string Description { get; set; }
    }
}
```

06.09.2020

```
using System.ComponentModel.DataAnnotations;
```

```
namespace Packt.Shared
{
    public class Product
    {
        public int ProductID { get; set; }

        [Required]
        [StringLength(40)]
        public string ProductName { get; set; }

        public int? SupplierID { get; set; }

        public int? CategoryID { get; set; }

        [StringLength(20)]
        public string QuantityPerUnit { get; set; }

        public decimal? UnitPrice { get; set; }

        public short? UnitsInStock { get; set; }

        public short? UnitsOnOrder { get; set; }
        public short? ReorderLevel { get; set; }
        public bool Discontinued { get; set; }
    }
}
```

@Марченко С.В., ЧДБК, 2020

42

Застосування LINQ на платформі EF Core

- **Проекція елементів за допомогою ключового слова *select***
 - Файл Program.cs
 - Наведений запит виводить необхідну інформацію, проте виконує дану дію неефективно, оскільки повертає всі стовпці таблиці Products замість трьох потрібних.
 - SQL-еквівалент: `SELECT * FROM Products;`

Консоль отладки Microsoft Visual Studio

```
Products that cost less than $10:
41: Jack's New England Clam Chowder costs $9,65
45: Rogede sild costs $9,50
47: Zaanse koeken costs $9,50
19: Teatime Chocolate Biscuits costs $9,20
23: Tunnbrod costs $9,00
75: Rhonbrau Klosterbier costs $7,75
54: Tourtiere costs $7,45
52: Filo Mix costs $7,00
13: Konbu costs $6,00
24: Guarana Fantastica costs $4,50
33: Geitost costs $2,50
```

```
static void FilterAndSort()
{
    using (var db = new Northwind())
    {
        var query = db.Products
            .Where(product => product.UnitPrice < 10M)
            // IQueryable<Product>
            .OrderByDescending(product => product.UnitPrice);

        WriteLine("Products that cost less than $10:");
        foreach (var item in query)
        {
            WriteLine("{0}: {1} costs {2:$#,##0.00}",
                item.ProductID, item.ProductName, item.UnitPrice);
        }
        WriteLine();
    }
}

static void Main(string[] args)
{
    FilterAndSort();
}
```

Проекція елементів за допомогою ключового слова select

```
static void FilterAndSort() {  
    using (var db = new Northwind())  
    {  
        var query = db.Products  
            .ProcessSequence()  
            .Where(product => product.UnitPrice < 10M)  
            // IQueryable<Product>  
            .OrderByDescending(product => product.UnitPrice)  
            // IOrderedQueryable<Product>  
            .Select(product => new // anonymous type  
            {  
                product.ProductID,  
                product.ProductName,  
                product.UnitPrice  
            });  
  
        WriteLine("Products that cost less than $10:");  
        foreach (var item in query)  
        {  
            WriteLine("{0}: {1} costs {2:$#,##0.00}",  
                item.ProductID, item.ProductName, item.UnitPrice);  
        }  
        WriteLine();  
    }  
}
```

- Додамо в запит метод Select() та повернемо три потрібних властивості у вигляді анонімного типу.
- Зауважте, що послідовності реалізують інтерфейси IQueryable<T> і IOrderedQueryable<T> замість IEnumerable<T> і IOrderedEnumerable<T>.
 - Це демонструє застосування постачальника даних LINQ, який, у свою чергу, використовує відкладене виконання та створює запит у пам'яті за допомогою дерев виразів.
 - Запит не стане виконуватись до останнього моменту, а при його настанні запит буде перетворено в іншу мову запитів, зокрема TransactSQL для Microsoft SQL Server.
 - Обхід результатів запиту за допомогою foreach або виклик методу на зразок ToArray() призведе до негайного виконання запиту.

Приєднання та групування послідовностей

```
static void JoinCategoriesAndProducts()
{
    using (var db = new Northwind())
    {
        // join every product to its category to return 77 matches
        var queryJoin = db.Categories.Join(
            inner: db.Products,
            outerKeySelector: category => category.CategoryID,
            innerKeySelector: product => product.CategoryID,
            resultSelector: (c, p) =>
                new { c.CategoryName, p.ProductName, p.ProductID });

        foreach (var item in queryJoin)
        {
            WriteLine("{0}: {1} is in {2}.",
                arg0: item.ProductID,
                arg1: item.ProductName,
                arg2: item.CategoryName);
        }
    }
}
```

- Існує 2 методи розширення для приєднання та групування.
 - Join() приймає 4 параметри:
 - послідовність, до якої потрібно приєднатись;
 - властивості *лівої* послідовності, з якими слід знайти відповідність;
 - властивості *правої* послідовності, з яким слід знайти відповідність;
 - проекцію;
 - GroupJoin приймає ті ж аргументи, але об'єднує співпадіння в груповий об'єкт, де Key використовується для зіставленого значення, а інтерфейс IEnumerable<T> — для кількох співпадінь.

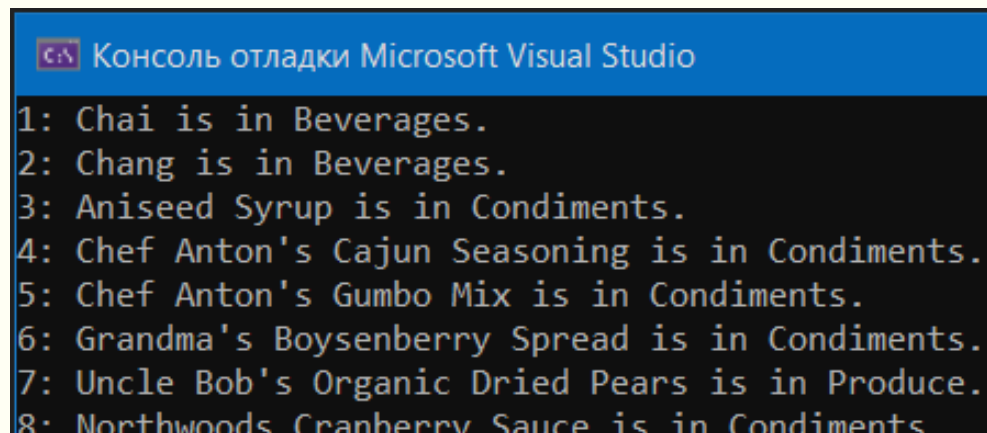
```
1: Chai is in Beverages.
2: Chang is in Beverages.
24: Guaraná Fantástica is in Beverages.
34: Sasquatch Ale is in Beverages.
35: Steeleye Stout is in Beverages.
38: Côte de Blaye is in Beverages.
39: Chartreuse verte is in Beverages.
```

Приєднання та групування послідовностей

```
static void JoinCategoriesAndProducts()
{
    using (var db = new Northwind())
    {
        // join every product to its category to return 77 matches
        var queryJoin = db.Categories.Join(
            inner: db.Products,
            outerKeySelector: category => category.CategoryID,
            innerKeySelector: product => product.CategoryID,
            resultSelector: (c, p) =>
                new { c.CategoryName, p.ProductName, p.ProductID })
            .OrderBy(cp => cp.ProductID);

        foreach (var item in queryJoin)
        {
            WriteLine("{0}: {1} is in {2}.",
                arg0: item.ProductID,
                arg1: item.ProductName,
                arg2: item.CategoryName);
        }
    }
}
```

- Змініть запит для виконання сортування по ProductID:



```
1: Chai is in Beverages.
2: Chang is in Beverages.
3: Aniseed Syrup is in Condiments.
4: Chef Anton's Cajun Seasoning is in Condiments.
5: Chef Anton's Gumbo Mix is in Condiments.
6: Grandma's Boysenberry Spread is in Condiments.
7: Uncle Bob's Organic Dried Pears is in Produce.
8: Northwoods Cranberry Sauce is in Condiments.
```

Приєднання та групування послідовностей

```
static void GroupJoinCategoriesAndProducts()
{
    using (var db = new Northwind())
    {
        // group all products by their category to return 8 matches
        var queryGroup = db.Categories.AsEnumerable().GroupJoin(
            inner: db.Products,
            outerKeySelector: category => category.CategoryID,
            innerKeySelector: product => product.CategoryID,
            resultSelector: (c, matchingProducts) => new
            {
                c.CategoryName,
                Products = matchingProducts.OrderBy(p => p.ProductName)
            });

        foreach (var item in queryGroup)
        {
            WriteLine("{0} has {1} products.",
                arg0: item.CategoryName,
                arg1: item.Products.Count());

            foreach (var product in item.Products)
            {
                WriteLine($" {product.ProductName}");
            }
        }
    }
}
```

- Продемонструємо роботу метода GroupJoin() у створеному методі GroupJoinCategoriesAndProducts().

Консоль отладки Microsoft Visual Studio

Beverages has 12 products.

- Chai
- Chang
- Chartreuse verte
- Cote de Blaye
- Guarana Fantastica
- Ipoh Coffee
- Lakkalikoori
- Laughing Lumberjack Lager
- Outback Lager
- Rhonbrau Klosterbier
- Sasquatch Ale
- Steeleye Stout

Condiments has 12 products.

- Aniseed Syrup
- Chef Anton's Cajun Seasoning
- Chef Anton's Gumbo Mix
- Genen Shouyu
- Grandma's Boysenberry Spread
- Gula Malacca


```

static void AggregateProducts()
{
    using (var db = new Northwind())
    {
        WriteLine("{0,-25} {1,10}",
            arg0: "Product count:",
            arg1: db.Products.Count());

        WriteLine("{0,-25} {1,10:$#,##0.00}",
            arg0: "Highest product price:",
            arg1: db.Products.Max(p => p.UnitPrice));

        WriteLine("{0,-25} {1,10:N0}",
            arg0: "Sum of units in stock:",
            arg1: db.Products.Sum(p => p.UnitsInStock));

        WriteLine("{0,-25} {1,10:N0}",
            arg0: "Sum of units on order:",
            arg1: db.Products.Sum(p => p.UnitsOnOrder));

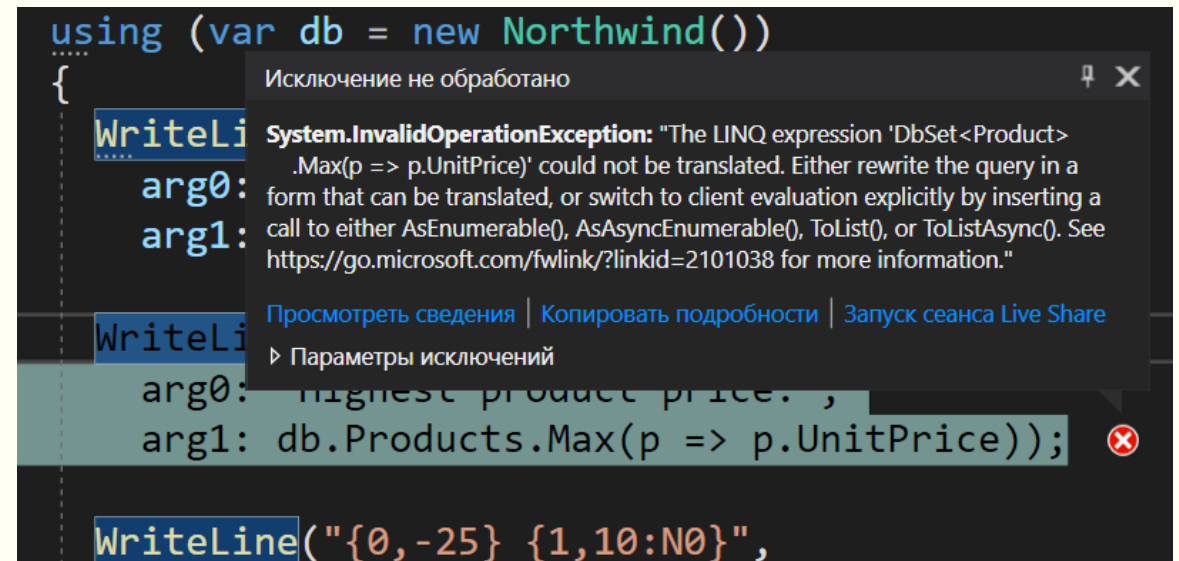
        WriteLine("{0,-25} {1,10:$#,##0.00}",
            arg0: "Average unit price:",
            arg1: db.Products.Average(p => p.UnitPrice));

        WriteLine("{0,-25} {1,10:$#,##0.00}",
            arg0: "Value of units in stock:",
            arg1: db.Products.AsEnumerable()
                .Sum(p => p.UnitPrice * p.UnitsInStock));
    }
}

```

Агрегування послідовностей

- Існують методи розширення LINQ для виконання функцій агрегування, наприклад, Average і Sum.
 - В Entity Framework Core 3.0+ операції LINQ, які не можуть транслюватись в SQL, більше автоматично не обчислюються на стороні клієнта, тому необхідно явно викликати метод AsEnumerable() для подальшої обробки запиту на клієнті.



```

static void AggregateProducts()
{
    using (var db = new Northwind())
    {
        WriteLine("{0,-25} {1,10}",
            arg0: "Product count:",
            arg1: db.Products.Count());

        WriteLine("{0,-25} {1,10:$#,##0.00}",
            arg0: "Highest product price:",
            arg1: db.Products.AsEnumerable().Max(p => p.UnitPrice));

        WriteLine("{0,-25} {1,10:N0}",
            arg0: "Sum of units in stock:",
            arg1: db.Products.Sum(p => p.UnitsInStock));

        WriteLine("{0,-25} {1,10:N0}",
            arg0: "Sum of units on order:",
            arg1: db.Products.Sum(p => p.UnitsOnOrder));

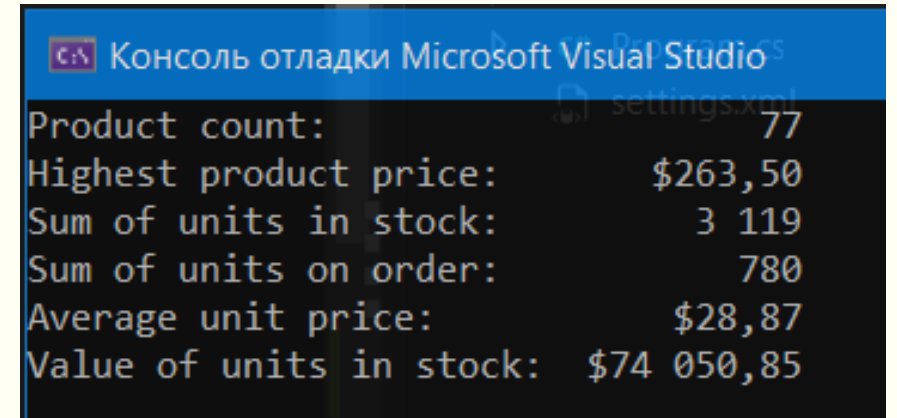
        WriteLine("{0,-25} {1,10:$#,##0.00}",
            arg0: "Average unit price:",
            arg1: db.Products.AsEnumerable().Average(p => p.UnitPrice));

        WriteLine("{0,-25} {1,10:$#,##0.00}",
            arg0: "Value of units in stock:",
            arg1: db.Products.AsEnumerable()
                .Sum(p => p.UnitPrice * p.UnitsInStock));
    }
}

```

Агрегування послідовностей

- Виправляємо виняток:



```

Консоль отладки Microsoft Visual Studio
Product count: 77
Highest product price: $263,50
Sum of units in stock: 3 119
Sum of units on order: 780
Average unit price: $28,87
Value of units in stock: $74 050,85

```

Підсолоджування синтаксису (query comprehension syntax)

- В C# 3 в 2008 году были представлены ключевые слова, которые упростили программистам работу с SQL при написании запросов LINQ.
 - Функциональность понятного синтаксиса запросов LINQ весьма ограничена.
 - Для доступа ко всем функциям LINQ нужно использовать методы расширения.
 - Компилятор самостоятельно преобразует понятный синтаксис запросов в методы расширения и лямбда-выражения.
 - Ключове слово `select` обов'язкове в зрозумілому синтаксисі запитів LINQ, а метод розширення `Select` – необов'язковий при застосуванні методів розширення та лямбда-виразів, оскільки весь `item` неявно `selected`.
 - Запити `query` та `query2` рівноцінні:

```
var names = new string[] { "Michael", "Pam", "Jim", "Dwight",  
                           "Angela", "Kevin", "Toby", "Creed" };
```

```
var query = names.Where(name => name.Length > 4)  
                  .OrderBy(name => name.Length)  
                  .ThenBy(name => name);
```

```
var query2 = from name in names  
              where name.Length > 4  
              orderby name.Length, name  
              select name;
```

Підсолоджування синтаксису (зрозумілий синтаксис запитів)

- Не у всіх методів розширення є еквівалентне ключове слово в мові C#.
 - Наприклад, методи `Skip()` і `Take()`, які зазвичай використовуються для посторінкового перегляду великих об'ємів даних.
 - Наступний запит не може бути записаний тільки за допомогою синтаксису запитів:

```
var query = names.Where(name => name.Length > 4)
                .OrderBy(name => name.Length)
                .ThenBy(name => name).Skip(80).Take(10);
```

- Проте можна огорнути синтаксис запиту в дужки та перейти до застосування методів розширення:

```
var query = (from name in names
              where name.Length > 4
              orderby name.Length, name
              select name).Skip(80).Take(10);
```

Доповіді

- Використання декількох потоків за допомогою PLINQ
 - Price M.J., с# 8.0 (ст. 418-421)
- Створення власних методів розширення LINQ
 - Price M.J., с# 8.0 (ст. 421-424)
- Робота з LINQ to XML
 - Price M.J., с# 8.0 (ст. 425-426)



ДЯКУЮ ЗА УВАГУ!

Наступне питання: