



ПАРАЛЕЛЬНА ОБРОБКА ДАНИХ ЗА ДОПОМОГОЮ PARALLEL LINQ

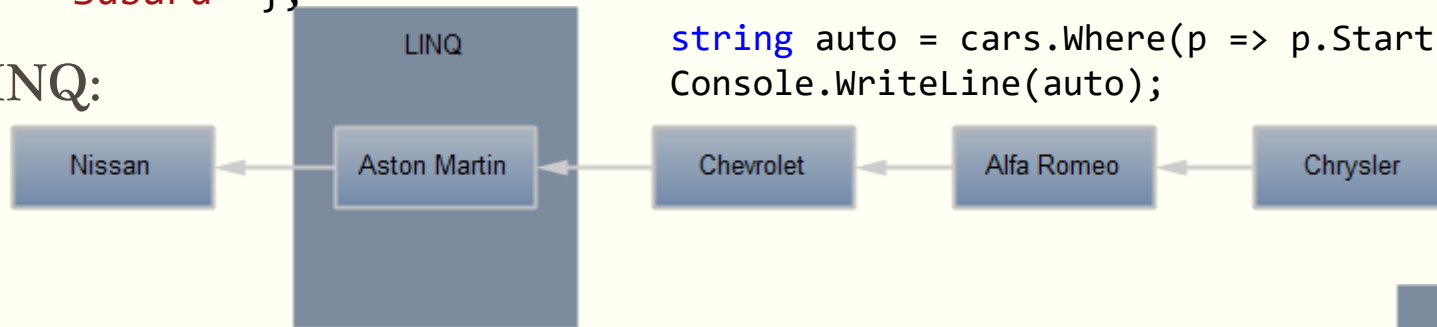
Питання 12.2.

Огляд Parallel LINQ

- Дані для прикладу:

```
string[] cars = { "Nissan", "Aston Martin", "Chevrolet", "Alfa Romeo", "Chrysler", "Dodge",  
"BMW", "Ferrari", "Audi", "Bentley", "Ford", "Lexus", "Mercedes", "Toyota", "Volvo",  
"Subaru" };
```

- LINQ:



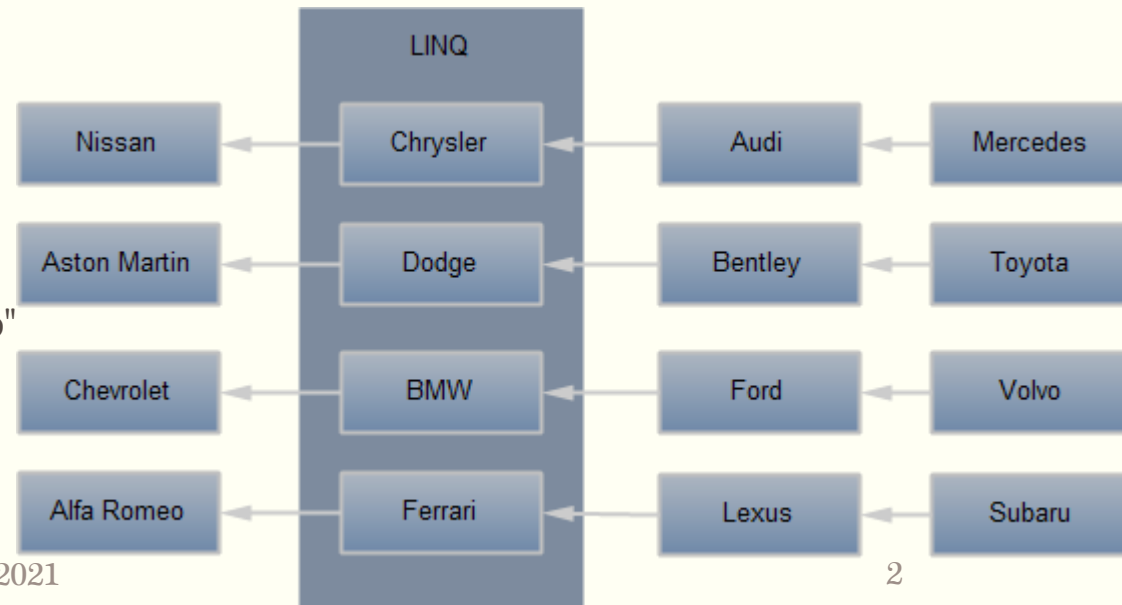
```
string auto = cars.Where(p => p.StartsWith("S")).First();  
Console.WriteLine(auto);
```

```
C:\Users\puasson\source\
Subaru
```

- PLINQ:

```
string auto = cars.AsParallel()  
    .Where(p =>  
        p.StartsWith("S")).First();  
Console.WriteLine(auto);
```

- Назви "Nissan", "Aston Martin", "Chevrolet", "Alfa Romeo" оброблюються одночасно — кожна в окремому ядрі.
- Після завершення обробки однієї назви ядро переходить до наступної незалежно від решти ядер.



Огляд Parallel LINQ

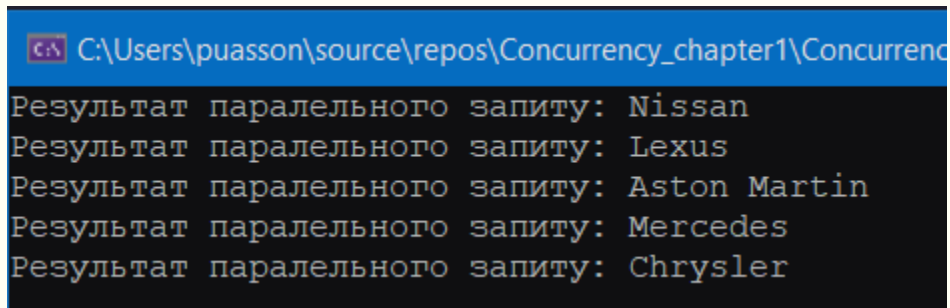
- Parallel LINQ піклується про розбиття даних, визначаючи, скільки елементів може оброблятися одночасно (зазвичай приймається рішення, що одного елемента даних на ядро достатньо), і координуючи роботу ядер так, що результат отримується, як від звичайних LINQ-запитів.
 - Parallel LINQ – паралельна реалізація API-інтерфейсу LINQ to Objects.
 - Він не реалізує паралельних засобів для інших видів LINQ, працює лише з об'єктами.
- Далеко не всі запити LINQ to Objects є хорошими кандидатами для запитів Parallel LINQ.
 - Присутні накладні витрати, пов'язані з розбиттям даних на фрагменти, встановленням та управлінням класами, які виконують паралельні задачі.
 - Якщо запит не надто довго виконується послідовно, можливо, виконувати його паралельно не має сенсу, оскільки накладні витрати можуть нівелювати весь виграш у продуктивності.

Огляд Parallel LINQ

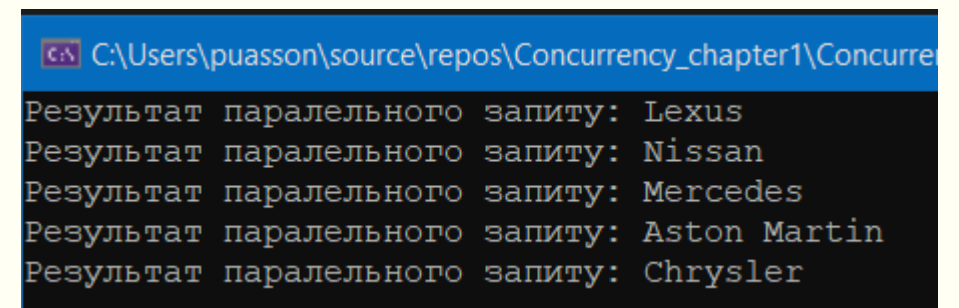
- У звичайному запиті LINQ to Query джерелом даних є IEnumerable <T>, де T – оброблюваний тип даних.
 - Механізм LINQ автоматично перемикається на використання PLINQ, коли джерелом даних є екземпляр типу ParallelQuery <T>.
 - Будь-який IEnumerable <T> може бути перетворений в ParallelQuery <T> за допомогою методу AsParallel(): джерело даних перетвориться в ParallelQuery, що автоматично має на увазі застосування Parallel LINQ.
- Дані, представлені в якості джерела для запиту PLINQ, розбиваються на частини і розділяються для паралельної обробки.
 - Кілька розділів можуть оброблятися одночасно, проте кожен з цих розділів обробляється послідовно.
 - Механізм PLINQ аналізує запит і дані, виконуючи розбиття "за лаштунками".
 - PLINQ генерує результати, які не впорядковані так, як вихідні дані.
 - Оскільки заздалегідь не відомо те, як PLINQ розіб'є дані, то не можна передбачити, яким буде порядок.
 - Ще гірше: розділи не обробляються за один крок.
 - Інші процеси на машині можуть призупиняти виконання додатку .NET на одному або більше ядер, а це означає, що в дійсності при багаторазовому запуску одного і того ж запиту будуть отримані результати, впорядковані по-різному.

Приклад кількох запусків паралельного запиту

```
string[] cars = { "Nissan", "Aston Martin", "Chevrolet", "Alfa Romeo", "Chrysler", "Dodge", "BMW",  
                  "Ferrari", "Audi", "Bentley", "Ford", "Lexus", "Mercedes", "Toyota", "Volvo", "Subaru"};  
  
// Запит Parallel LINQ  
var auto = cars.AsParallel()  
    .Where(p => p.Contains("s"));  
  
foreach (string s in auto)  
    Console.WriteLine("Результат паралельного запиту: " + s);
```



```
C:\Users\puasson\source\repos\Concurrency_chapter1\Concurrency_chapter1>dotnet run  
Результат паралельного запиту: Nissan  
Результат паралельного запиту: Lexus  
Результат паралельного запиту: Aston Martin  
Результат паралельного запиту: Mercedes  
Результат паралельного запиту: Chrysler
```



```
C:\Users\puasson\source\repos\Concurrency_chapter1\Concurrency_chapter1>dotnet run  
Результат паралельного запиту: Lexus  
Результат паралельного запиту: Nissan  
Результат паралельного запиту: Mercedes  
Результат паралельного запиту: Aston Martin  
Результат паралельного запиту: Chrysler
```

- Для збереження порядку потрібно використати метод розширення `AsOrdered()` на об'єкті `ParallelQuery`, створеному за допомогою методу `AsParallel`:

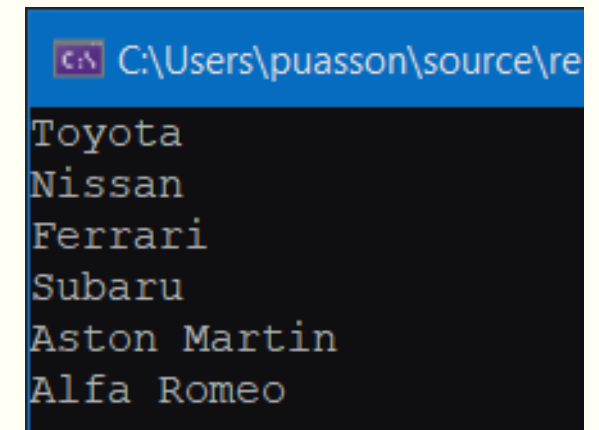
```
var auto = cars.AsParallel().AsOrdered()  
    .Where(p => p.Contains("s"));
```

Примусове паралельне виконання

- Інколи PLINQ може вирішити, що запит краще виконувати послідовно.
 - Цим можна управляти за рахунок застосування методу розширення `WithExecutionMode` до типу `ParallelQuery`.
 - Метод `WithExecutionMode` приймає одне з двох значень з перелічення `ParallelExecutionMode`:
 - `Default` (режим за умовчанням, який дозволяє PLINQ вирішувати, як виконувати обробку)
 - `ForceParallelism` (використовувати PLINQ навіть якщо накладні витрати, пов'язані з паралельним виконанням, перевершують виграш від нього).

```
// Запит Parallel LINQ
IEnumerable<string> auto = cars
    .AsParallel()
    .WithExecutionMode(ParallelExecutionMode.ForceParallelism)
    .Where(p => p.Contains("a"))
    .Select(p => p);

foreach (string s in auto)
    Console.WriteLine(s);
```



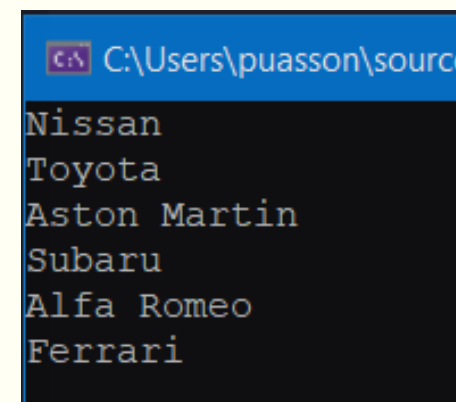
```
C:\Users\puasson\source\re
Toyota
Nissan
Ferrari
Subaru
Aston Martin
Alfa Romeo
```

Обмеження степені паралелізму

- Можна вимагати, щоб PLINQ обмежував кількість розділів, оброблюваних паралельно, за допомогою методу розширення `WithDegreeOfParallelism`, який працює з `ParallelQuery`.
 - Цей метод приймає аргумент `int`, який встановлює максимальну кількість розділів, які повинні бути оброблені одночасно; це називається *ступенем паралелізму*.
 - Установка ступеня паралелізму не примушує PLINQ використовувати саме стільки розділів, а лише вказує верхню межу.
 - PLINQ може вирішити використовувати меншу кількість розділів, ніж вказано, або ж, якщо метод `WithExecutionMode` не використовувався може взагалі виконувати запит послідовно.
 - Це може бути зручно, якщо необхідно обмежити вплив запиту на машину, яка повинна виконувати й інші завдання.

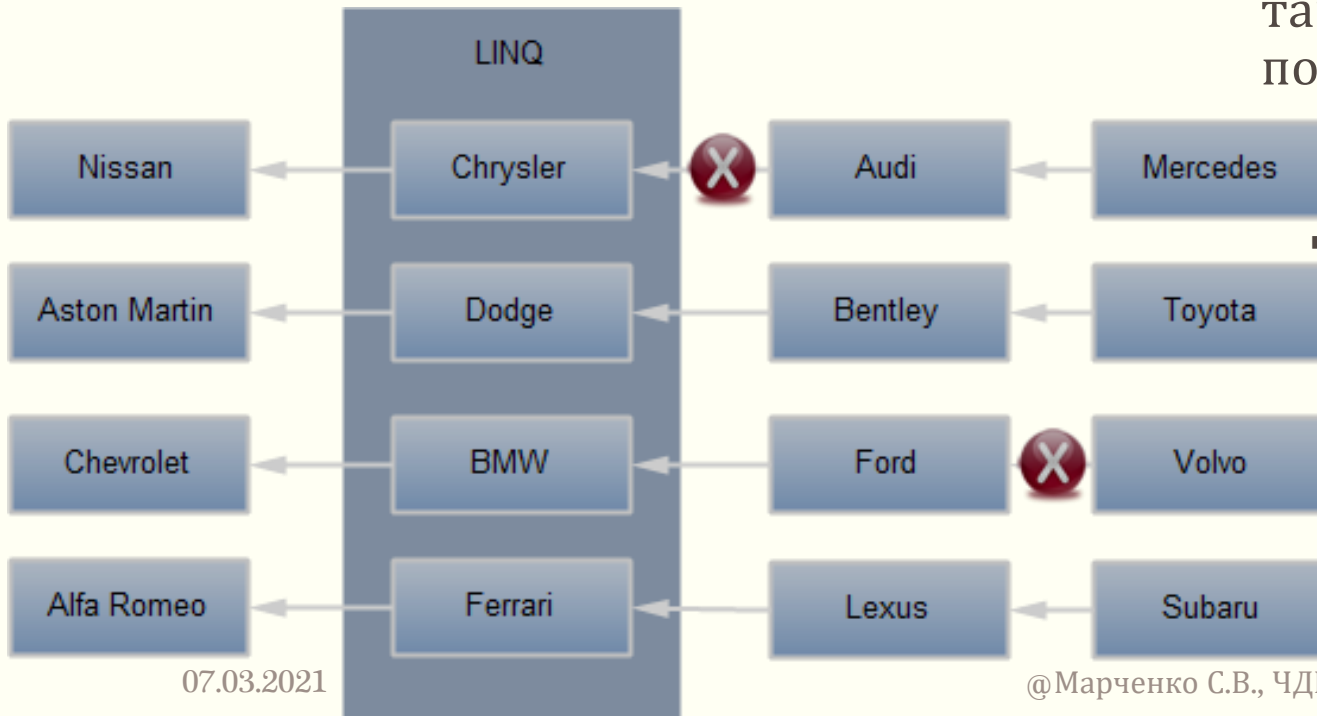
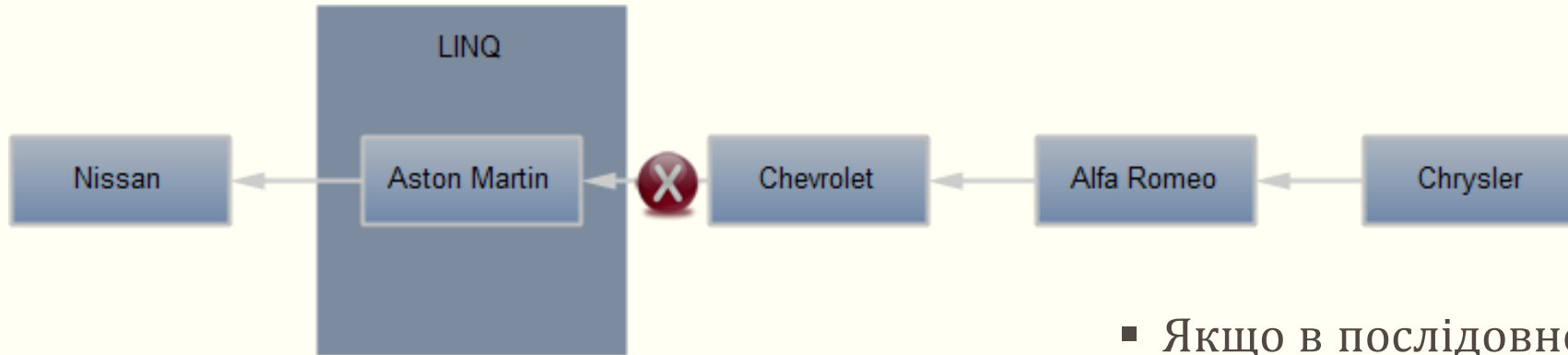
```
// Запит Parallel LINQ
IEnumerable<string> auto = cars
    .AsParallel()
    .WithExecutionMode(ParallelExecutionMode.ForceParallelism)
    .WithDegreeOfParallelism(2)
    .Where(p => p.Contains("a"))
    .Select(p => p);

foreach (string s in auto)
    Console.WriteLine(s);
```



```
C:\Users\puasson\source
Nissan
Toyota
Aston Martin
Subaru
Alfa Romeo
Ferrari
```

Обробка винятків Parallel LINQ



- Якщо в послідовному LINQ-запиті щось іде не так, згенерований виняток перериває будь-яку подальшу обробку.

Таким чином, ми повинні отримати результати, попередні Aston Martin, але не наступні за ним.

- З PLINQ все інакше: дані розбиті на розділи, які потім обробляються незалежно і паралельно.

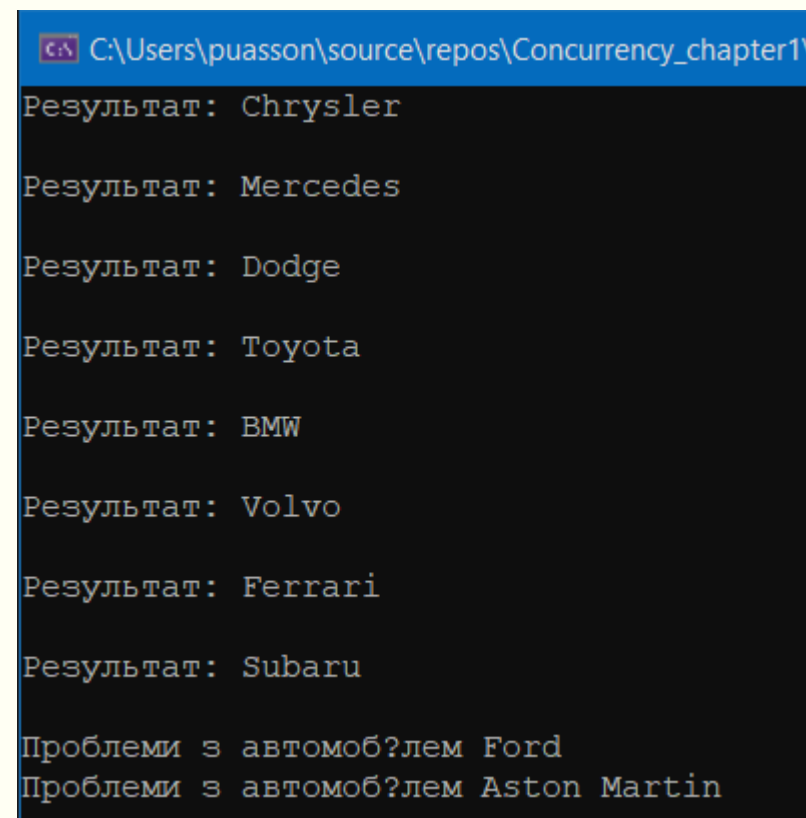
У першому розділі виникає проблема з Aston Martin, яка призводить до генерації винятку, проте це не зупиняє обробку інших розділів, а проблема з Ford викликає генерацію іншого винятку.

Вирішення проблеми з паралельними винятками

- PLINQ збирає всі винятки, які знаходить, і упаковує їх у `System.AggregateException`, який потім генерує в коді (запуск – Ctrl+F5).

```
// Запит Parallel LINQ
IEnumerable<string> auto = cars
    .AsParallel()
    .Select(p => {
        if (p == "Aston Martin" || p == "Ford")
            throw new Exception("Проблеми з автомобілем " + p);
        return p;
    });

try {
    foreach (string s in auto)
        Console.WriteLine("Результат: " + s + "\n");
}
catch (AggregateException agex) {
    agex.Handle(ex => {
        Console.WriteLine(ex.Message);
        return true;
    });
}
```



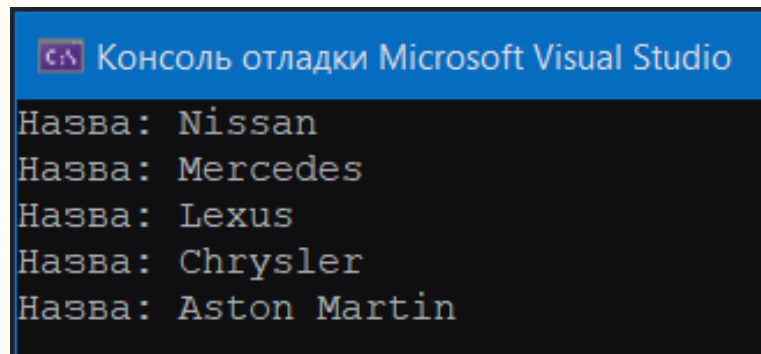
The screenshot shows the output of a console application. The title bar indicates the file path: C:\Users\puasson\source\repos\Concurrency_chapter1\... The output consists of several lines, each starting with "Результат:" followed by a car name. The names are Chrysler, Mercedes, Dodge, Toyota, BMW, Volvo, Ferrari, and Subaru. At the bottom, there are two lines of error messages: "Проблеми з автомоб?лем Ford" and "Проблеми з автомоб?лем Aston Martin". The question mark in the error messages is likely a typo for "а".

```
C:\Users\puasson\source\repos\Concurrency_chapter1\...
Результат: Chrysler
Результат: Mercedes
Результат: Dodge
Результат: Toyota
Результат: BMW
Результат: Volvo
Результат: Ferrari
Результат: Subaru
Проблеми з автомоб?лем Ford
Проблеми з автомоб?лем Aston Martin
```

Запити без результатів

- PLINQ включає корисний засіб у методі розширення ForAll.
 - Використаний з ParallelQuery-об'єктом, ForAll виконує System.Action над кожним елементом послідовності.
- В одному з раніше розглянутих прикладів шукаються всі назви машин, які містять "s".
 - Для фільтрації відповідних назв використовувалася конструкція where, а вибрані назви додавалися до результуючої колекції IEnumerable <string>.
 - Потім результати перераховуються в циклі foreach з виведенням їх на консоль за допомогою Console.WriteLine.
 - За допомогою методу ForAll можна зробити те ж саме, але набагато елегантніше: замість збору результатів назви виводяться на консоль безпосередньо, із застосуванням лямбда-виразу, переданого в метод ForAll.

```
// Запит Parallel LINQ
cars.AsParallel()
    .Where(p => p.Contains("s"))
    .ForAll(p => Console.WriteLine("Назва: " + p));
```



```
Консоль отладки Microsoft Visual Studio
Назва: Nissan
Назва: Mercedes
Назва: Lexus
Назва: Chrysler
Назва: Aston Martin
```

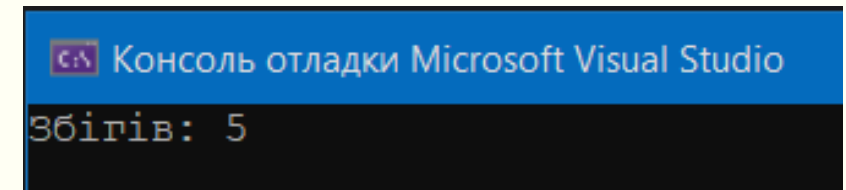
Запити без результатів

- Всередині об'єкта Action, який передається методу ForAll, можна робити досить багато всього за винятком повернення результату.

- Можна навіть фільтрувати дані без допомоги конструкції Where:

```
int count = 0;

cars.AsParallel()
    .ForAll(p =>
    {
        if (p.Contains('s'))
            System.Threading.Interlocked.Increment(ref count);
    });
Console.WriteLine("Збігів: " + count);
```



- метод ForAll є частиною PLINQ, а це значить, що визначене в об'єкті Action дію виконується в розділах послідовності даних паралельно.
 - Це дає вигреш в продуктивності за рахунок паралельного виконання, але може привести до проблем з розділяються даними, такими як значення int, що використовується в якості лічильника збігів.
 - Для забезпечення точності підрахунку застосовується клас Interlocked з простору імен System.Threading.
 - Цей прийом називається **синхронізацією**, і являє собою розширену техніку паралельного програмування.



ДЯКУЮ ЗА УВАГУ!

Наступне запитання: Бібліотека TPL Dataflow