



NET2 - 12

LINQ

Förra gången

- Frivillig visning av Parkeringsplats-appen
- Entity framework
 - Database First
 - Code First
 - Övningar...

Idag

- Upprop
- LINQ
- Fortsätta med förra veckans övningar
- Nya övningar för LINQ

var

- För att slippa hålla reda på vilken datatyp som returneras av ett uttryck, så använder vi var istället för den korrekta datatypen.
- Håll musen över variabeln, så ser du vilken datatyp du har fått

LINQ

- Language Integrated Query
 - Ett språk för att fråga om data, som är integrerat i C#
 - SQL gör samma sak, men då saknar vi intellisense i Visual Studio
 - Nyckelord i .NET-ramverket som används för att bland annat hämta, filtrera och sortera kollektioner.
 - Kollektionerna kan vara t.ex:
 - Databas-tabeller (DBSet)
 - Listor
 - Arrayer

Två olika syntaxer

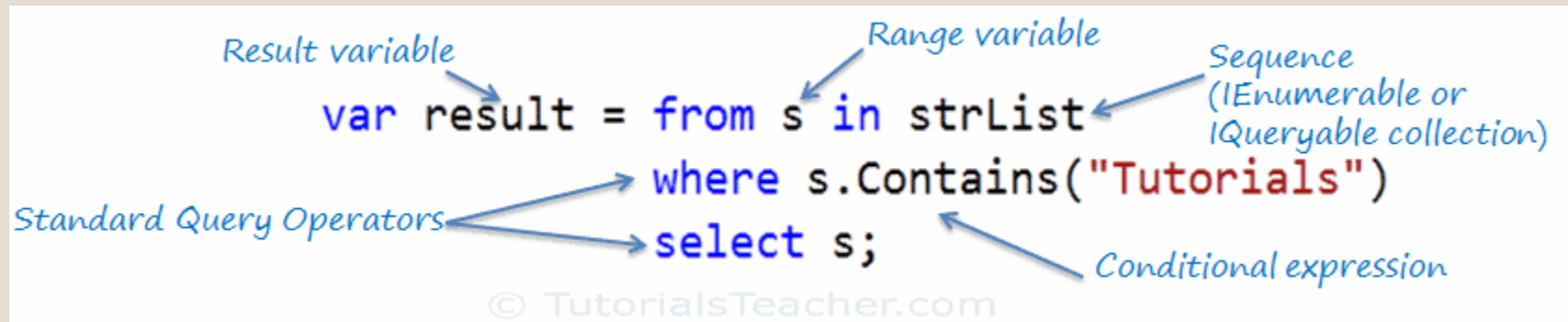
- LINQ kan skrivas på två sätt
 - Query-syntax
 - Tydligare men längre
 - Method-syntax
 - Kortare, och vanligast
- I genomgången kommer jag blanda dessa...

Query-syntax

- Det lite längre (men kanske tydligare?) sättet att skriva saker på
 - Börjar med ordet "**from**"
 - Slutar med ordet "**select**" eller en "**group by**"
- Exempel

```
var result = from person in people
where
    person.Age > 18
select
    person;
```

Query-syntax



The diagram illustrates the components of a LINQ query syntax statement. The code is: `var result = from s in strList where s.Contains("Tutorials") select s;`. Annotations with arrows point to specific parts: 'Result variable' points to 'var result'; 'Range variable' points to 's'; 'Sequence (IEnumerable or IQueryable collection)' points to 'strList'; 'Standard Query Operators' has two arrows pointing to 'where' and 'select'; 'Conditional expression' points to 's.Contains("Tutorials")'.

```
var result = from s in strList where s.Contains("Tutorials") select s;
```

Annotations:

- Result variable
- Range variable
- Sequence (IEnumerable or IQueryable collection)
- Standard Query Operators
- Conditional expression

© TutorialsTeacher.com

Metod/Lambda

- Exempel

```
var sortedDogs = dogs.OrderByDescending(x => x.Age);  
foreach (var dog in sortedDogs)  
{  
    Console.WriteLine(string.Format("Dog {0} is {1} years old.", dog.Name, dog.  
Age));  
}
```

Method(Lambda)-syntax

- Det kortare (och vanligaste) sättet att skriva LINQ queries på!

- Exempel:

- `var result = people.Where(person => person.Age > 18);`

- Jämför med query-syntax

- `var result = from person in people`

- `where person.Age > 18`

- `select person;`

Lambda-syntax

```
var result = strList.Where(s => s.Contains("Tutorials"));
```

© TutorialsTeacher.com

Extension method Lambda expression

Filtrering

- **Where** (supervanlig) och "**OfType**"
 - **Filtering** är en operation för att begränsa ett resultat där vi endast tillåter element som **uppfyller ett visst villkor**
- **Exempel**

```
string[] words = {"One", "ring", "to", "rule", "them", "all"};  
var result = words.Where(word => word.Length > 3);
```

Vilka ord kommer hamna i result?

Datatyp och var

- För att slippa hålla reda på vilken datatyp som returneras av ett uttryck, så använder vi **var** istället för den korrekta datatypen.
- Håll musen över variabeln, så ser du vilken datatyp du har fått.

Exempel

```
var result = people.Where(person => person.Name == "Bosse");
```

I "result" har vi en **IEnumerable<Person>**, en slags enkel variant av en lista
För att göra om en IEnumerable till en vanlig lista, lägg till **.ToList()**

```
var result = people.Where(person => person.Name == "Bosse").ToList();
```

Ett sträng-exempel:

```
var result = myWordArray.Where(word => word != "attans").ToArray();
```

SortBy

- Vi kan sortera våra resultat på valfritt sätt:
- `List<int> numbers = new List<int> { 1, 14, 12, 1100, 99 }`
- `var result = numbers.OrderBy(number => number);`
- Eller...

Gruppering

- Det går att gruppera precis på samma sätt som med SQL, med metoden GroupBy.
- Exempel:

```
List<int> numbers = new List<int> { 1, 14, 12, 99, 23, 44, 95, 32 };  
var evenOddGroups = numbers.GroupBy(number => number % 2);
```

- Detta används sedan såhär:

```
foreach (var group in evenOddGroups)  
{  
    Console.WriteLine("Grupp: " + group.Key);  
    foreach (var value in group)  
    {  
        Console.WriteLine(value);  
    }  
}
```

Villkoret för gruppering



Antal rader

- I SQL heter det t ex TOP 3 för att visa de tre översta värdena.
- I LINQ heter det .Take(3);

- Exempel

```
int[] array = new int[7] { 1, 3, 5, 2, 8, 6, 4 };  
int[] topThree = array.OrderByDescending(i=> i)  
    .Take(3)  
    .ToArray();
```


Join

- Exempel:

```
var result = from person in people
join pet in pets on person.FirstName equals pet.Owner
select new { OwnerName = person.FirstName, PetName = pet.Name };
```

- För att använda objektet, skriv följande

- `result.OwnerName` och `result.PetName`

Bara en post

- Om du bara förväntar dig att få EN post från ditt LINQ-query, använd då `SingleOrDefault()`;
- `Car theCar = (from c in db.Cars`
- `where c.Id == 5`
- `select c).SingleOrDefault();`

Code Along - LINQ

- Siffror
- Strängar
- Objekt
 - Person
- SortBy
- GroupBy
- Join
 - Installera: EntityFrameworkCore + SqlServer + Tools
 - Databas: Parking8 (eller annat nummer)
 - Scaffold-DbContext "Server=.\SQLEXPRESS;Database=Parking10;Trusted_Connection=True;TrustServerCertificate=true" Microsoft.EntityFrameworkCore.SqlServer - OutputDir Models

Övningar

- Övningarna från I fredags:
 - Entity Framework
 - Everloop – Products (Det går inte att ta bort produkter!)
 - Städer och parkeringshus
 - Todo-appen
- Linq:
 - Se separat dokument: LINQ-övningar.pdf
 - Lösningförslag kan fås på begäran...men prova först.

Länkar

- [LINQ overview - .NET | Microsoft Learn](#)
- [System.Linq Namespace | Microsoft Learn](#)
- [Lambda Expressions in C# \(c-sharpcorner.com\)](#)
- [Basic LINQ Query Operations \(C#\) | Microsoft Learn](#)
- [C# LINQ Joins With SQL \(dotnettricks.com\)](#)
- [Difference between IEnumerable and List - Josip Miskovic](#)