LINQ

Language Integrated Query in Entity Framework Core

SoftUni Team

Technical Trainers

Software University

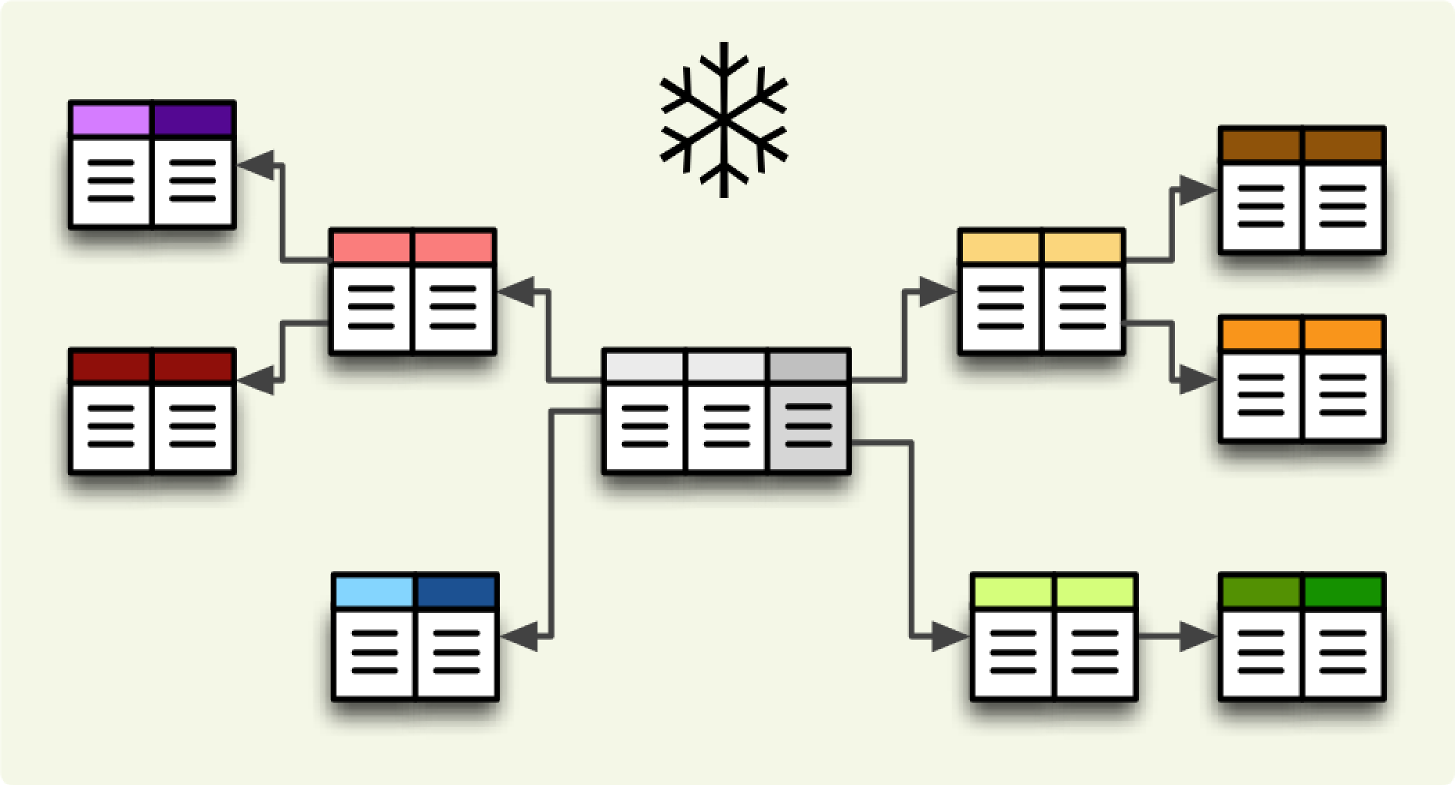
SoftUni

Software University

https://softuni.bg

Table of Contents

1. LINQ
   - Filtering
   - Select() / Projection
   - Aggregation
   - Joining
   - SelectMany()
2. IEnumerable vs IQueryable
3. Result Models



Filtering and Aggregating Tables

Select, Join and Group Data Using LINQ

Filtering

- **Where()**
  - **Selects** values that are based on a **predicate function**
  - Syntax:

```
string[] words = { "the", "quick", "brown", "fox", "jumps" };

IEnumerable<string> query =
     words.Where(word => word.Length == 3);
```

```
IEnumerable<string> query = from word in words
                            where word.Length == 3
                            select word;
```

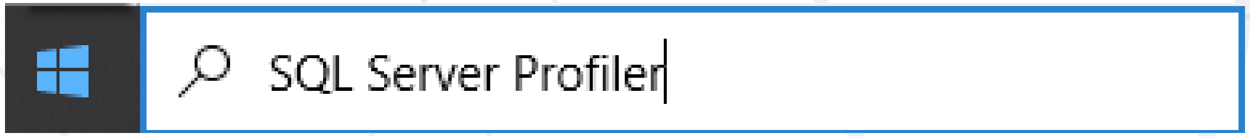
Good Reasons to Use Select

- **Select()** – Limits the network traffic by reducing the queried columns
- Syntax:

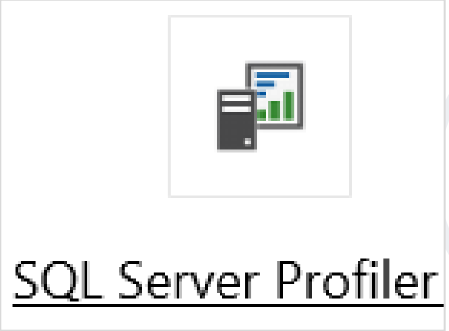
```
var employeesWithTown = context
  .Employees
  .Select(employee => new
  {
    EmployeeName = employee.FirstName,
    TownName = employee.Address.Town.Name
  });
```

Good Reasons to Use Select (2)

- SQL Server Profiler – graphical tool used to monitor an instance of Microsoft SQL Server



SQL Server Profiler

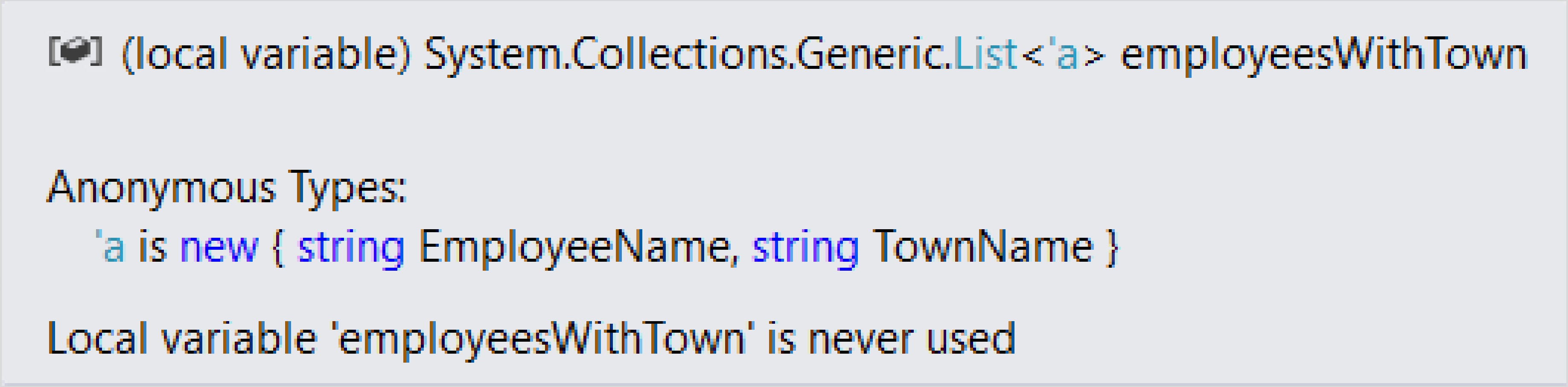


SQL Server Profiler

```
   SELECT [employee].[FirstName] AS [EmployeeName],
[employee.Address.Town].[Name] AS [TownName]
     FROM [Employees] AS [employee]
LEFT JOIN [Addresses] AS [employee.Address] ON
[employee].[AddressID] = [employee.Address].[AddressID]
LEFT JOIN [Towns] AS [employee.Address.Town] ON
[employee.Address].[TownID] =
[employee.Address.Town].[TownID]
```

Good Reasons Not to Use Select

- Data that is selected is **not** of the **initial entity type**
  - **Anonymous type**, generated at runtime



```
(local variable) System.Collections.Generic.List<'a> employeesWithTown

Anonymous Types:
    'a is new { string EmployeeName, string TownName }

Local variable 'employeesWithTown' is never used
```

- **Data cannot be modified** (updated, deleted)
  - Entity is of a **different type**
  - Not associated with the **context** anymore

Aggregation

- Aggregate functions perform **calculations on a set** of input values and return a value
  - **Average** - Calculates the average value of a collection of values
  - **Count** - Counts the elements in a collection, optionally only those elements that satisfy a predicate function
  - **Max** and **Min** - Determine the maximum and the minimum value in a collection
  - **Sum** - Calculates the sum of the values in a collection

Joining Tables in EF: Using Join()

- Join tables in EF with **LINQ** / **extension methods** on **IEnumerable<T>** (like when joining collections)

```
var employees = softUniEntities.Employees
    .Join(softUniEntities.Departments,
    (e => e.DepartmentID),
    (d => d.DepartmentID),
    (e, d) => new {
      Employee = e.FirstName,
      JobTitle = e.JobTitle,
      Department = d.Name
    }
  );
```

Grouping Tables in EF

- Grouping also can be done by LINQ
  - The same way as with collections in LINQ
- Grouping with LINQ:

```
var groupedEmployees =
   from employee in softUniEntities.Employees
   group employee by employee.JobTitle;
```

- Grouping with extension methods:

```
var groupedCustomers = softUniEntities.Employees
   .GroupBy(employee => employee.JobTitle);
```

SelectMany – Example (1)

```
public class PhoneNumber
{
  public string Number {get;set;}
}
```

```
public class Person
{
  public IEnumerable<PhoneNumber> PhoneNumbers {get;set;}

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

SelectMany – Example (2)

```
IEnumerable<Person> people = new List<Person>();

// "Select" gets a list of lists of phone numbers
IEnumerable<IEnumerable<PhoneNumber>> phoneLists =
              people.Select(p => p.PhoneNumbers);

// SelectMany flattens it to just a list of phone numbers
IEnumerable<PhoneNumber> phoneNumbers =
              people.SelectMany(p => p.PhoneNumbers);
```

SelectMany – Example (3)

```
// To include data from the parent in the result pass an
expression
// to the second parameter (resultSelector) in the
overload:
var directory = people.SelectMany(p => p.PhoneNumbers,
(parent, child) => new { parent.Name, child.Number });
```

IEnumerable vs IQueryable

Differences Between IEnumerable and IQueryable

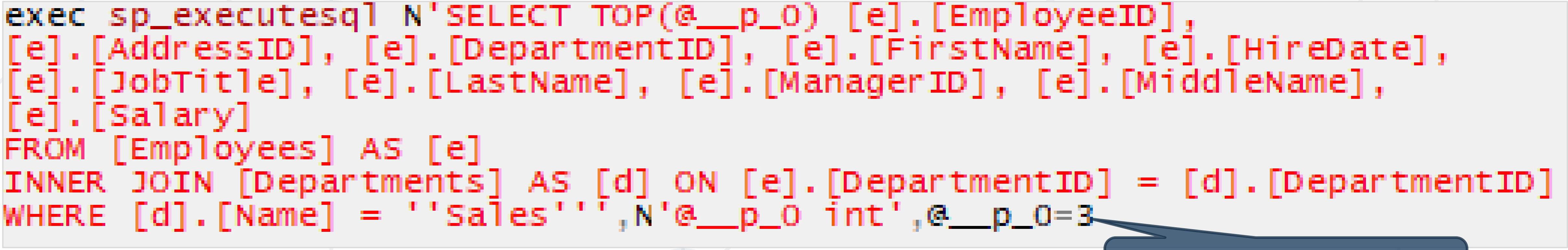
- **IEnumerable<T>**
  - System.Collections.Generic namespace
  - Base type for almost all .NET collections
  - LINQ methods works with **Func<>**
  - Good for **in-memory** collections like List, Array

- **IQueryable<T>**
  - System.Linq namespace
  - Derives the base interface from IEnumerable<T>
  - LINQ methods works with **Expression<Func<>>**
  - Good for queries over **data stores** such as databases

IEnumerable vs IQueryable Example

- Accessing the data from the Employee table and then **taking only 3 rows** from that data

```
var context = new SoftUniContext();
IQueryable<Employee> employees = context
  .Employees.Where(e => e.Department.Name == "Sales");
employees = employees.Take(3);
```



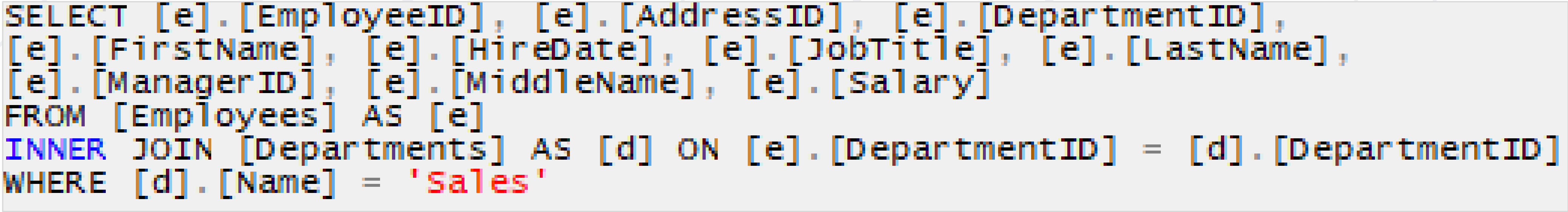
```
exec sp_executesql N'SELECT TOP(@__p_0) [e].[EmployeeID],
[e].[AddressID], [e].[DepartmentID], [e].[FirstName], [e].[HireDate],
[e].[JobTitle], [e].[LastName], [e].[ManagerID], [e].[MiddleName],
[e].[Salary]
FROM [Employees] AS [e]
INNER JOIN [Departments] AS [d] ON [e].[DepartmentID] = [d].[DepartmentID]
WHERE [d].[Name] = ''Sales''',N'@__p_0 int',@__p_0=3
```

SELECT TOP 3

IEnumerable vs IQueryable Example (2)

- IEnumerable executes SELECT query on the server-side, **loads data in-memory** on the client-side and **then filters** the data

```
var context = new SoftUniContext();
IEnumerable<Employee> employees = context
  .Employees.Where(e => e.Department.Name == "Sales");
employees = employees.Take(3);
```



```
SELECT [e].[EmployeeID], [e].[AddressID], [e].[DepartmentID],
[e].[FirstName], [e].[HireDate], [e].[JobTitle], [e].[LastName],
[e].[ManagerID], [e].[MiddleName], [e].[Salary]
FROM [Employees] AS [e]
INNER JOIN [Departments] AS [d] ON [e].[DepartmentID] = [d].[DepartmentID]
WHERE [d].[Name] = 'Sales'
```

Result Models

Simplifying Models

Result Models

- **Select()**, **GroupBy()** can work with **custom classes**
  - Allow you to **pass them** to methods and use them as a return type
  - Require some **extra code** (class definition)
- Sample Result Model:

```
public class UserResultModel
{
  public string FullName { get; set; }
  public string Age { get; set; }
}
```

Result Models (2)

- Assign the fields as you would with an anonymous object:

```
var currentUser = context.Users
  .Where(u => u.Id == 8)
  .Select(u => new UserResultModel
  {
    FullName = u.FirstName + " " + u.LastName,
    Age = u.Age
  })
  .SingleOrDefault();
```

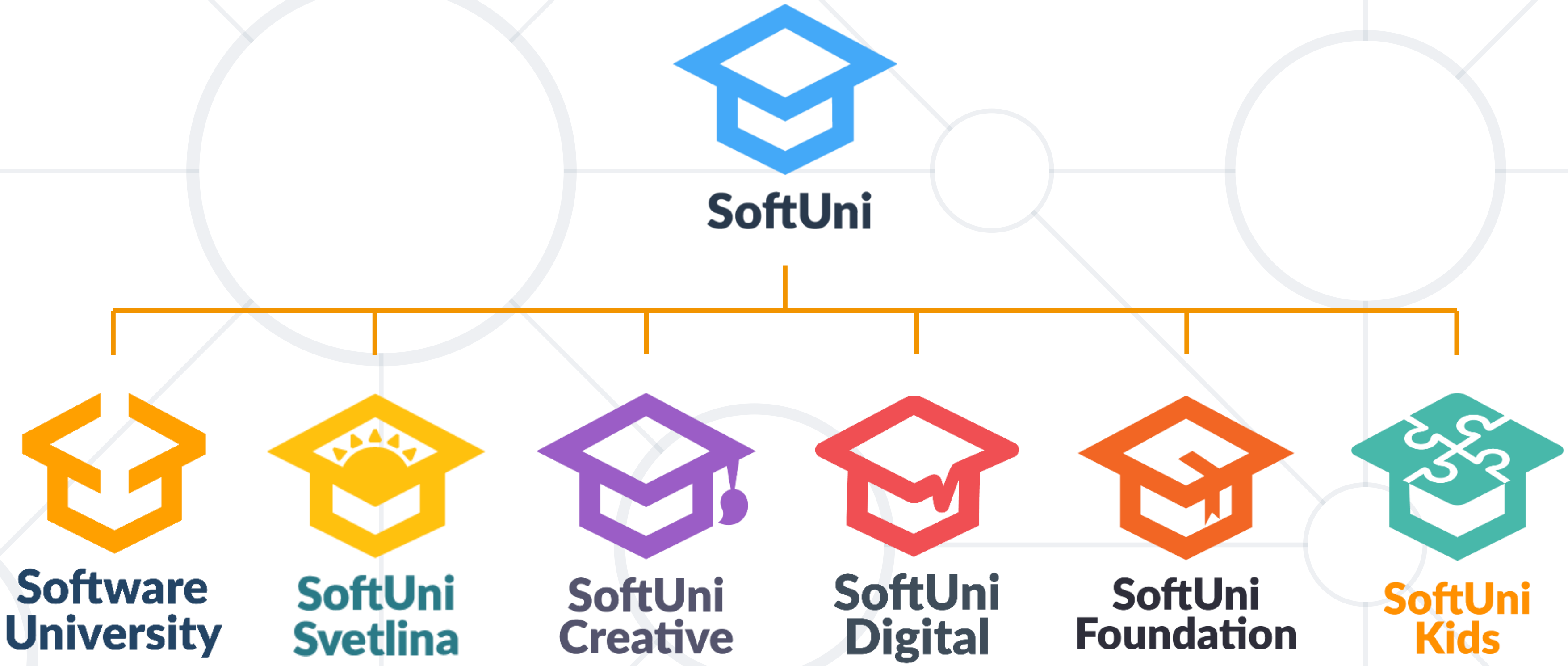
- The new type can be used in a method signature:

```
public UserResultModel GetUserInfo(int Id) { … }
```

Summary

- LINQ
  - Filtering – Where(), Select()
  - Aggregation – Average(), Count(), Sum()
  - SelectMany() – flattens to just a list
  - Join() – like when joining collections
- **IEnumerable** – Good for **in-memory collections**
  - Loads **all the data** in-memory
- **IQueryable** – Good for **queries over data** stores
  - Takes **only the needed data**
- Select(), GroupBy() can work with **custom classes**

Questions?



SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://softuni.org
- © Software University – https://softuni.bg