CURS 12: LINQ

Metode Avansate de Programare,
Informatică Română

realizat de

Bercea Cristian, Bubuianu Adrian (221)
Mera Mădălina, Moroz Alexandra (224)

CE ESTE LINQ?

Language Integrated Query (LINQ) - C#

Introduces Language Integrated Query (LINQ) in C#.

MicrosoftLearn /

TutorialsTeacher

Why LINQ?

This article explains why to use linq in .net. To understand why we should use LINQ, let's take some example. Suppose you want to find list of teenage...

tutorialsteacher.com

CE ESTE LINQ?

= Language-Integrated Query
= set of technologies based on the integration of query capabilities directly into the C# language.

“Traditionally, queries against data are expressed as simple strings without type checking at compile time or IntelliSense support. Furthermore, you have to learn a different query language for each type of data source: SQL databases, XML documents, various Web services, and so on. With LINQ, a query is a first-class language construct, just like classes, methods, and events.”

Preluat din documentația oficială Microsoft

LINQ API

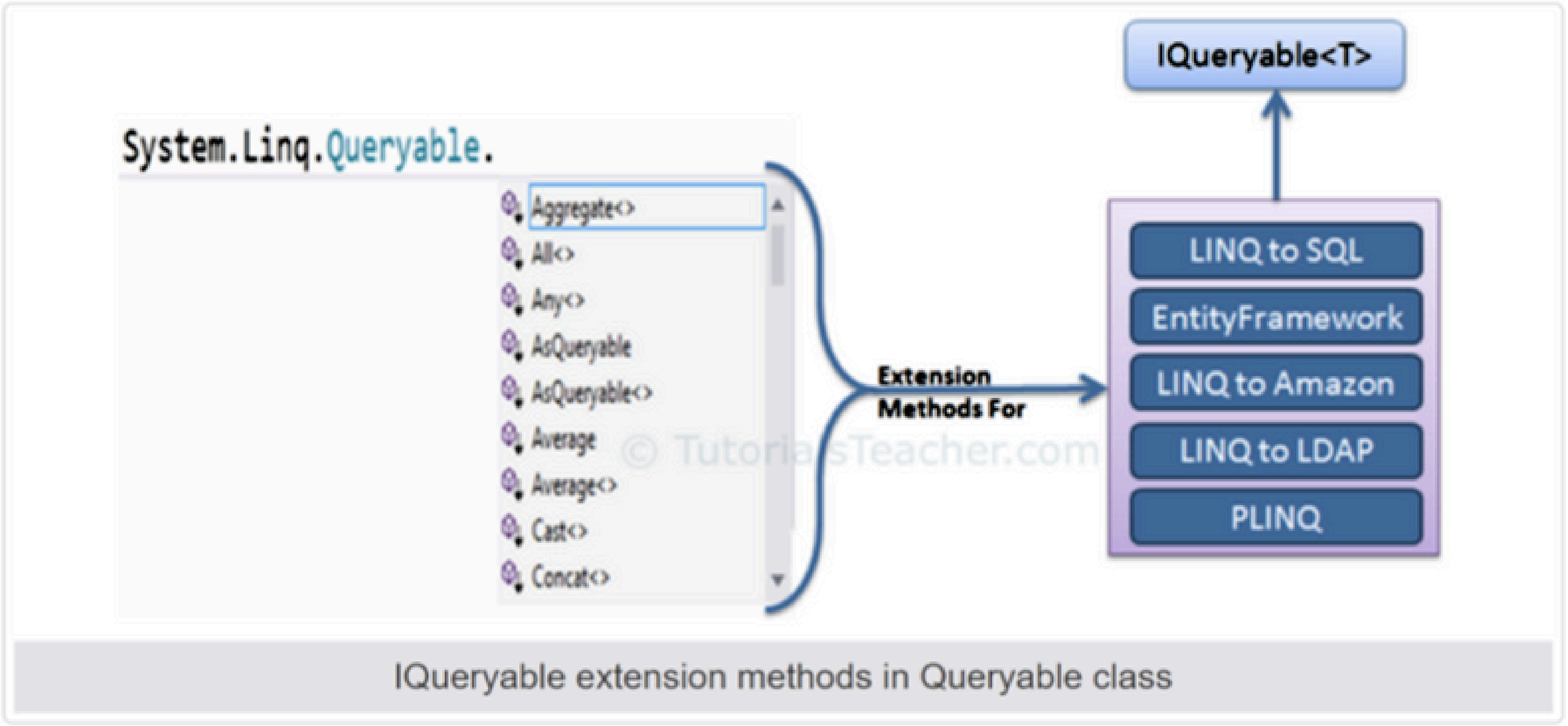
IEnumerable

pentru structuri care
îl implementează
(List, Array etc.)

IQueryable

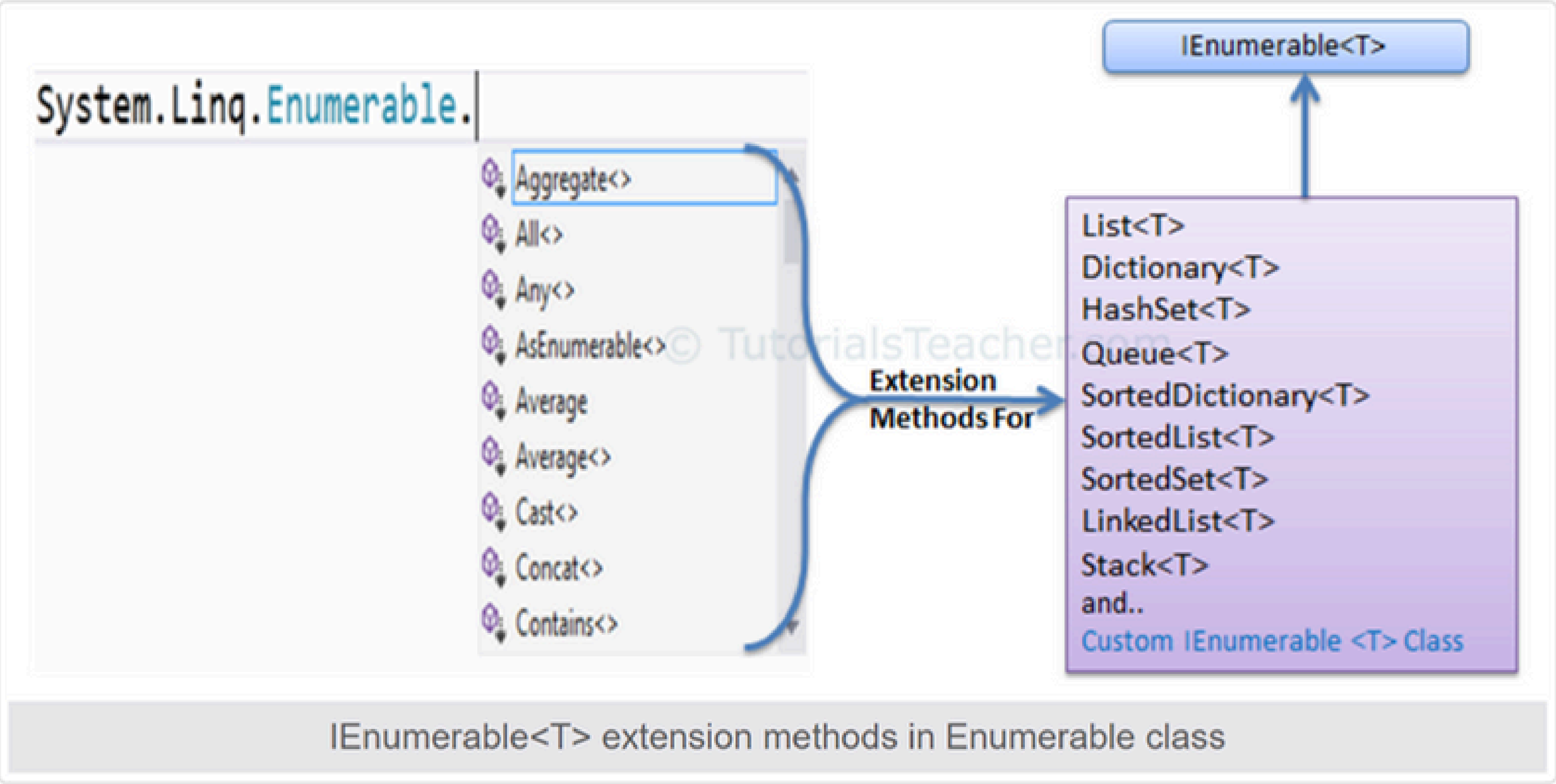
pentru baze de date

CE ESTE LINQ?



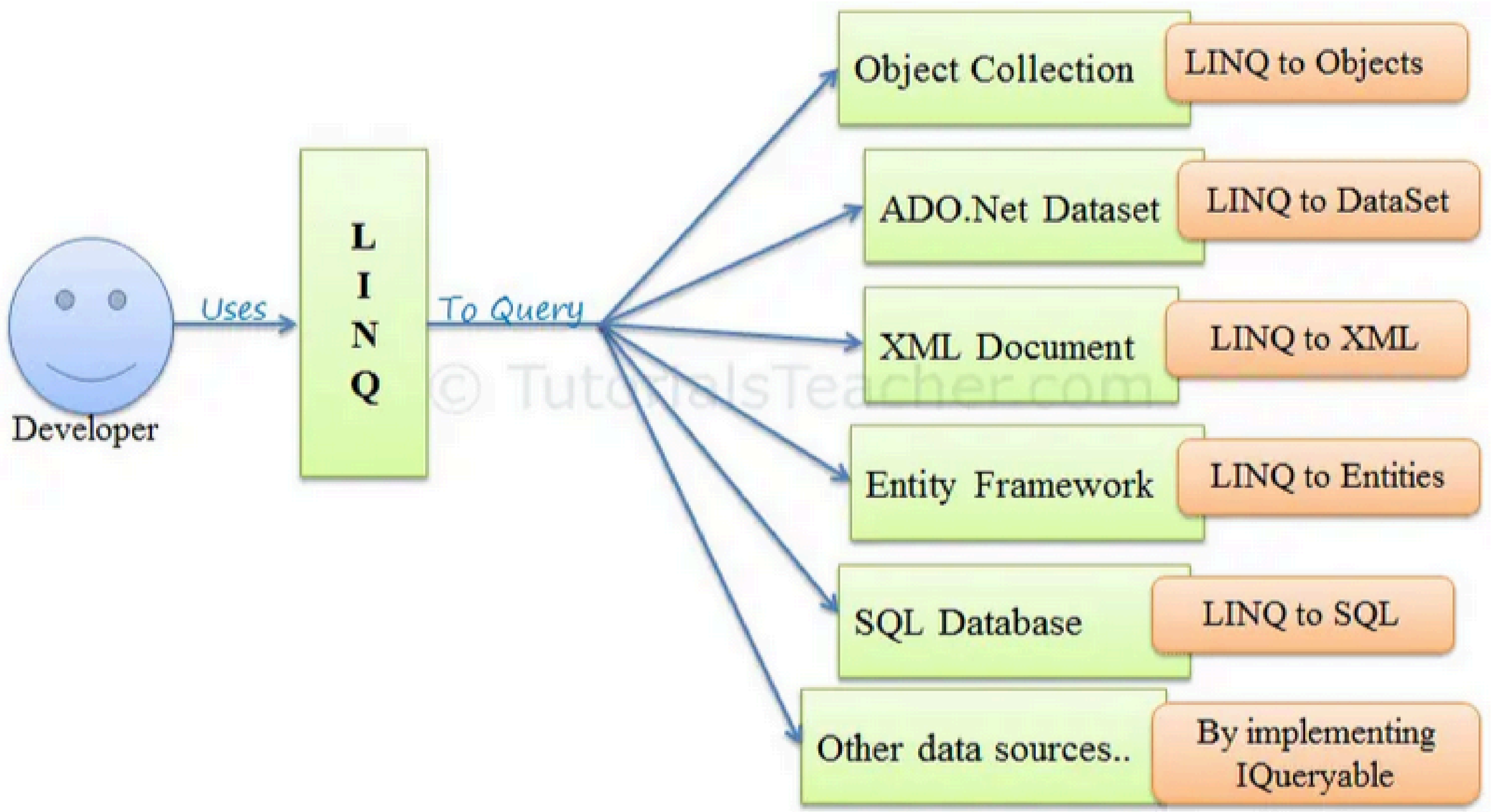
IQueryable extension methods in Queryable class

CE ESTE LINQ?



IEnumerable<T> extension methods in Enumerable class

CE ESTE LINQ?


Developer
Uses
L
I
N
Q
To Query
Object Collection
LINQ to Objects
ADO.Net Dataset
LINQ to DataSet
XML Document
LINQ to XML
Entity Framework
LINQ to Entities
SQL Database
LINQ to SQL
Other data sources..
By implementing IQueryable

```
List<Student> studentList = new List<Student>
{
    new Student { StudentID = 1, StudentName = "John", Age = 18 },
    new Student() { StudentID = 2, StudentName = "Steve", Age = 21 },
    new Student() { StudentID = 3, StudentName = "Bill", Age = 25 },
    new Student() { StudentID = 4, StudentName = "Ram", Age = 20 },
    new Student() { StudentID = 5, StudentName = "Ron", Age = 31 },
    new Student() { StudentID = 6, StudentName = "Chris", Age = 17 },
    new Student() { StudentID = 7, StudentName = "Rob", Age = 19 }
};

List<Student> teenagerStudents = studentList // List<Student>
    .Where(s :Student  => s.Age > 12 && s.Age < 20) // IEnumerable<Student>
    .ToList();

List<Student> studentsStartWithR = studentList // List<Student>
    .Where(s :Student  => s.StudentName.StartsWith("R")) // IEnumerable<Student>
    .ToList();
```

- By using query syntax, you perform filtering, ordering, and grouping operations on data sources with a minimum of code.

- You use the same query expression patterns to query and transform data from any type of data source.

- This example showcases the concise and readable nature of LINQ for querying collections of data in C#. It avoids the need for more verbose looping and conditional statements that would be required without LINQ.

AVANTAJE LINQ

- **Standardized way of querying multiple data sources**: The same LINQ syntax can be used to query multiple data sources.
- **Less coding**: It reduces the amount of code to be written as compared with a more traditional approach.
- **Readable code**: LINQ makes the code more readable so other developers can easily understand and maintain it.
- **Compile time safety of queries**: It provides type checking of objects at compile time.
- **IntelliSense Support**: LINQ provides IntelliSense for generic collections.

DEFERRED VS IMMEDIATE

Query-urile executate **deferred** nu vor genera rezultatele propriu-zise decât în momentul în care acestea sunt necesare. Acest lucru oferă flexibilitate în folosirea șirului sau expresiei pe care este aplicată interogarea LINQ.

```
List<string> vegetables = new List<string> { "Carrot", "Selleri" };
var result :IEnumerable<string>  = from v :string  in vegetables select v;
Console.WriteLine("Elements in vegetables array (before add): " + result.Count());
vegetables.Add("Broccoli");
Console.WriteLine("Elements in vegetables array (after add): " + result.Count());
```

DEFERRED VS IMMEDIATE

Query-urile executate **immediate** genereaza rezultatele pe loc:

```
string[] words = { "one", "two", "three" };
var result2 :List<{Index,Value}>  = words.Select((w :string , i :int )
        =>new { Index = i, Value = w }) // IEnumerable<{Index,Value}>
         .Where(w :{Index,Value}  => w.Value.Length == 3).ToList();
```

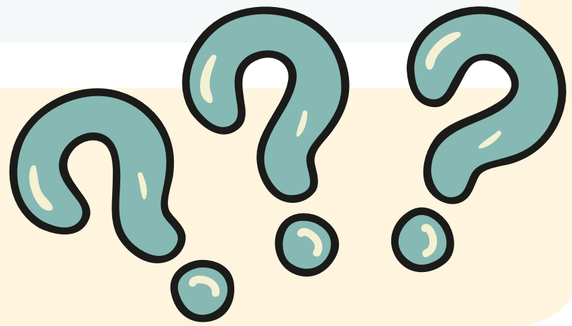
Exemplul 1

AGGREGATE

Performs a specified operation to each element in a collection, while carrying the result forward.

```
var numbers = new int[] { 1, 2, 3, 4, 5 };
var result = numbers.Aggregate((a:int, b:int) => a * b);
Console.WriteLine("Aggregated numbers by multiplication:");
Console.WriteLine(result);
```

```
Aggregated numbers by multiplication:
120
```



ANY

Checks if any elements in a collection satisfies a specified condition.

```
string[] names = { "Bob", "Ned", "Amy", "Bill" };
var result = names.Any(n :string  => n.StartsWith("B"));
Console.WriteLine("Does any of the names start " +
```

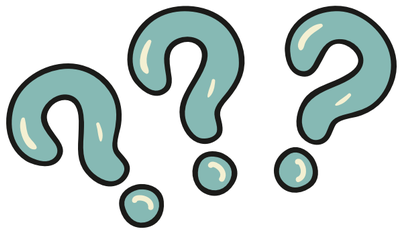
```
                  "with the letter 'B': "+result);
```

```
Does any of the names start with the letter 'B': True
```

ELEMENT AT OR DEFAULT

Retrieves element from a collection at specified (zero-based) index, but gets default value if out-of-range.

```
string[] colors = { "Red", "Green", "Blue" };
var resultIndex1 = colors.ElementAtOrDefault(1);
var resultIndex10 = colors.ElementAtOrDefault(10);
Console.WriteLine("Element at index 1 in the array " +
                  "contains: "+resultIndex1);
Console.WriteLine("Element at index 10 in the array " +
                  "does not exist: "+ (resultIndex10 == null));
```



SELECT MANY

```
string[] fruits = { "Grape", "Orange", "Apple" };
int[] amounts = { 1, 2, 3 };
var result = fruits.SelectMany(f :string  => amounts,
    (f :string , a :int ) => new { Fruit = f, Amount = a });
Console.WriteLine("Selecting all values from " +
                  "each array, and mixing them:");
foreach (var o in result)
    Console.WriteLine(o.Fruit + ", " + o.Amount);
```

SELECT MANY

Flattens collections into a single collection (similar to cross join in SQL). = cartesian product

```
Grape, 1
Grape, 2
Grape, 3
```

```
Orange, 1
Orange, 2
Orange, 3
```

```
Apple, 1
Apple, 2
Apple, 3
```

Exemplul 2

TO DICTIONARY

Converts collection into a Dictionary with Key and Value.

```
public class English2German
{
    public string EnglishSalute { get; set; }
    public string GermanSalute { get; set; }
}
```

TO DICTIONARY

```
English2German[] english2German = {
    new English2German { EnglishSalute = "Good morning",
        GermanSalute = "Guten Morgen" },
    new English2German { EnglishSalute = "Good day",
        GermanSalute = "Guten Tag" },
    new English2German { EnglishSalute = "Good evening",
        GermanSalute = "Guten Abend" },
    new English2German { EnglishSalute = "Good night",
        GermanSalute = "Gute Nacht" },
};
```

TO DICTIONARY

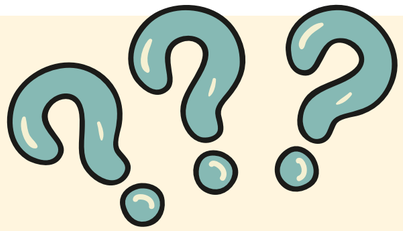
```
var result = english2German
    .ToDictionary(k:English2German => k.EnglishSalute,
        v:English2German => v.GermanSalute);
Console.WriteLine("Values inserted into dictionary:");

foreach (KeyValuePair<string, string> dic in result)
    Console.WriteLine(String.Format(
        "English salute '{0}' is '{1}' in German", dic.Key,
        dic.Value));
```

CONCAT

Concatenates (combines) two collections.

```
int[] numbers1 = { 1, 2, 3 };
int[] numbers2 = { 4, 5, 6 };
var result = numbers1.Concat(numbers2);
Console.WriteLine("The concatenation of given arrays is:");
result.ToList().ForEach(x :int =>Console.Write(x+" "));
```

```
The concatenation of given arrays is:
1 2 3 4 5 6
```

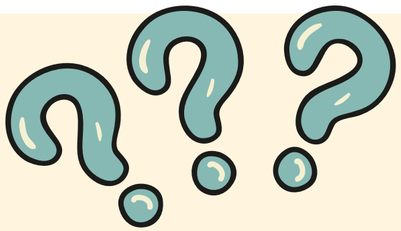


DISTINCT

Removes duplicate elements from a collection

```
int[] numbers = { 1, 2, 2, 3, 5, 6, 6, 6, 8, 9 };
var result = from n in numbers.Distinct() select n;
Console.WriteLine("Distinct elements of given array are:");
result.ToList().ForEach(x :int => Console.Write(x+" "));
```

```
Distinct elements of given array are:
1 2 3 5 6 8 9
```



EXCEPT

Removes all elements from one collection which exist in another collection.

```
int[] numbers1 = { 1, 2, 3 };
int[] numbers2 = { 3, 4, 5 };
var result = from n in numbers1.Except(numbers2) select n;
Console.WriteLine("The resulting numbers are:");
result.ToList().ForEach(x :int =>Console.Write(x+" "));
```

EXCEPT

```
int[] numbers1 = { 1, 2, 3 };
int[] numbers2 = { 3, 4, 5 };
var result = from n in numbers1.Except(numbers2) select n;
Console.WriteLine("The resulting numbers are:");
result.ToList().ForEach(x :int =>Console.Write(x+" "));
```

```
The resulting numbers are:
1 2
```

=> numbers1/numbers2 (array difference)

GROUP BY

```
int[] numbers = { 10, 15, 20, 25, 30, 35 };
var result = from n in numbers
    group n by n % 10 == 0 into groups
    select groups;
Console.WriteLine("GroupBy has created two groups:");
result.ToList().ForEach(group :IGrouping<bool,int>  => {
    Console.WriteLine(group.Key == true ?
        "\nDivisible by 10." : "\nNot Divisible by 10.");
    group.ToList().ForEach(x :int =>Console.Write(x+" ")); });
```

GROUP BY

Projects elements of a collection into groups by key.

```
Divisible by 10.
10 20 30
Not Divisible by 10.
15 25 35
```

Exemplul 3

JOIN

```
string[] warmCountries ={ "Turkey", "Italy", "Spain",
    "Saudi Arabia", "Ethiopia", "Portugal"};
string[] europeanContries={"Denmark", "Portugal",
    "Germany", "Italy", "Spain"};
```

JOIN

- Reunește două colecții/tabele folosindu-se de o valoare comună a unor coloane
- Similar cu INNER JOIN din SQL

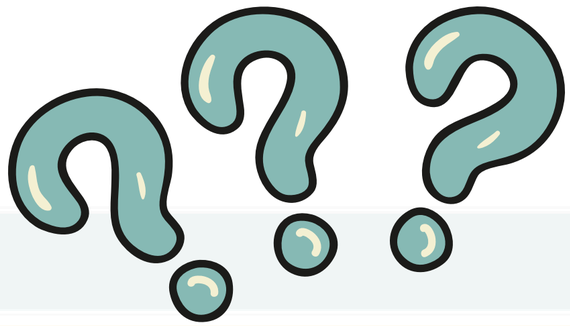
```sql
SELECT * FROM warmCountries AS w
INNER JOIN europeanCountries AS e ON w=e
```

```
Italy
Spain
Portugal
```

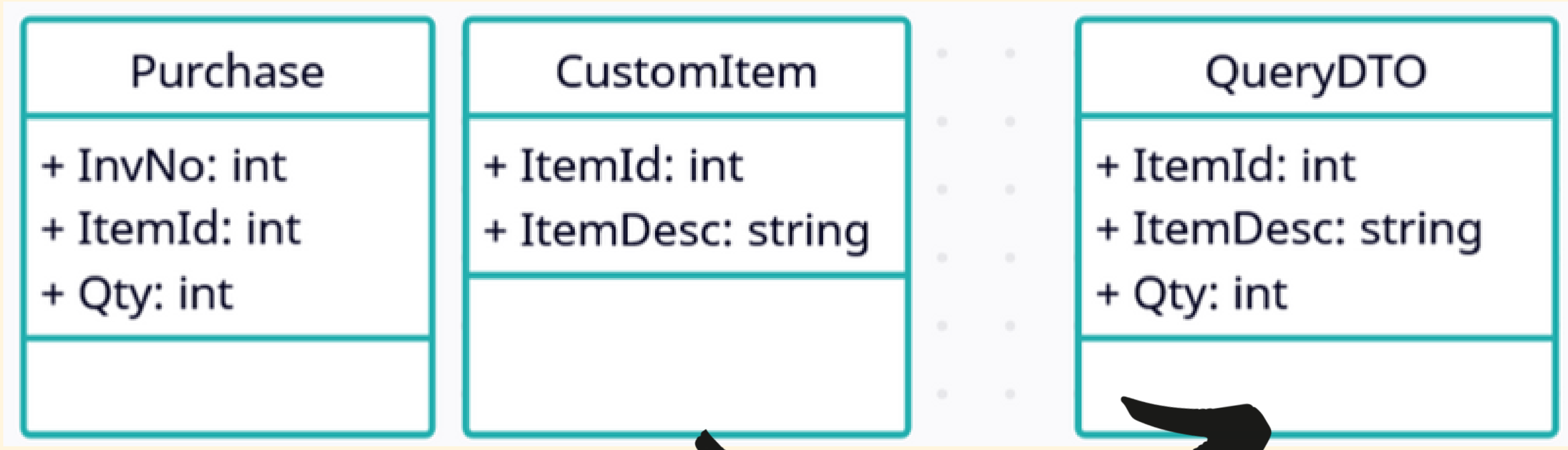
```
var result = (from w in warmCountries
        join e in europeanContries
            on w equals e select w);
```

JOIN

```
string[] warmCountries ={ "Turkey", "Italy", "Spain",
    "Saudi Arabia", "Ethiopia", "Portugal"};
string[] europeanContries={"Denmark", "Portugal",
    "Germany", "Italy", "Spain"};

var result = (from w in warmCountries
        join e in europeanContries
            on w equals e select w);
foreach (var v in result)
    Console.WriteLine(v);
```

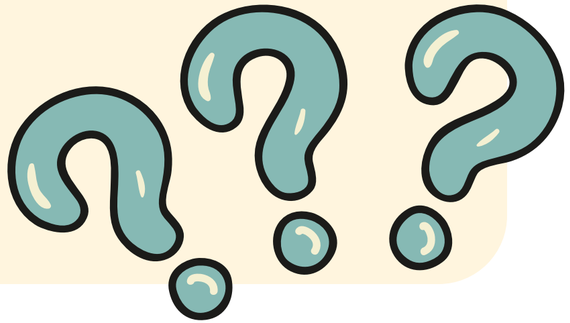


JOIN



List<Purchase> purchases

List<CustomItem> itemlist



Exemplul 4

JOIN

```
var innerJoin = from e in itemlist
    join d in purchases on e.ItemId equals d.ItemId
    select new
    {
        itid = e.ItemId,
        itdes = e.ItemDes,
        qty = d.Qty
    };
```

ORDER BY

```
Car[] cars = {
    new Car { Name = "Super Car", HorsePower = 215 },
    new Car { Name = "Economy Car", HorsePower = 75 },
    new Car { Name = "Family Car", HorsePower = 145 },
};
var result = from c in cars
    orderby c.HorsePower ascending select c;
Console.WriteLine("Ordered list of cars by " +
                  "horsepower using Query Syntax:");
result.ToList().ForEach(car=>
    Console.WriteLine(String.Format("{0}: {1} horses",
        car.Name, car.HorsePower)));
```

ORDER BY

Sorts a collection in ascending order.

```
Ordered list of cars by horsepower using Query Syntax:
Economy Car: 75 horses
Family Car: 145 horses
Super Car: 215 horses
```

THEN BY

Use it after earlier sorting, to further sort a collection in ascending order.

```
string[] capitals = { "Berlin", "Paris", "Madrid", "Tokyo", "London", "Athens",
    "Beijing", "Seoul" };
var result :IOrderedEnumerable<string> = (from c :string in capitals
        orderby c.Length
        select c)
    .ThenBy(c :string => c);
foreach (string capital in result)
{
    Console.WriteLine(capital);
}
```

```
Paris
Seoul
Tokyo
Athens
Berlin
London
Madrid
Beijing
```

XML QUERY

- XML = eXtensible Markup Language
- similar cu HTML

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- Used for storing data (Someone must write a piece of software to send, receive or display it)
- Custom tags

- Used for displaying data
- Predefined tags

Configurare XMLWriter

1. **XmlWriter xmlWriter = XmlWriter.Create("D:\\test.xml")**

<?xml version="1.0" encoding="utf-8"?><messageTasks><messageTask id="3"><desc>Sarbatori fericite!</desc><from>Jane</from>
<to>Michelle</to><message>Craciun Fericit!</message></messageTask></messageTasks>

2.

```
XmlWriterSettings settings = new
XmlWriterSettings
{
    Indent = true // Adaugă indentare pentru
lizibilitate
};
XmlWriter xmlWriter =
XmlWriter.Create("D:\\test.xml", settings);
```


```
<?xml version="1.0" encoding="utf-8"?>
<messageTasks>
<messageTask id="3">
<desc>Sarbatori fericite!</desc>
<from>Jane</from>
<to>Michelle</to>
<message>Craciun Fericit!</message>
</messageTask>
</messageTasks>
```

XML QUERY - WRITE

```
▼<messageTasks>
  ▼<messageTask id="3">
      <desc>shopping</desc>
      <from>Jane</from>
      <to>Michael</to>
      <message>Buy some bread!</message>
    </messageTask>
</messageTasks>
```

XML QUERY - WRITE

```
XmlWriter xmlWriter = XmlWriter.Create("filepath");
xmlWriter.WriteStartDocument();

xmlWriter.WriteStartElement("?");
xmlWriter.WriteStartElement("?"); <?>

xmlWriter.WriteAttributeString("attribute", "value");
xmlWriter.WriteEndElement();  </?>
xmlWriter.WriteEndDocument();
xmlWriter.Close();
```

XML QUERY - WRITE

Cod	Explicații	Format XML
xmlWriter.WriteStartDocument();	Add the XML declaration to the beginning of the file	**<?xml version="1.0"?>**
xmlWriter.WriteStartElement("messageTasks");	Write a root element called messageTasks. In XML, this is the first element that contains all other elements.	**<messageTasks>**
xmlWriter.WriteStartElement("messageTask");	Write a child element called messageTask inside the root element	**<messageTasks> <messageTask>**
xmlWriter.WriteAttributeString ("id", "3");	Add an attribute named id with the value 3 to the messageTask element	**<messageTasks> <messageTask id="3">**

XML QUERY - WRITE

Cod	Explicații	Format XML
xmlWriter.WriteStartElement("desc"); xmlWriter.WriteString("Sarbatori fericite!"); xmlWriter.WriteEndElement();	Creates an element called desc that contains the text "Sarbatori fericite!" and closes it	**<desc>Sarbatori fericite!</desc>**
xmlWriter.WriteEndElement();	Closes the <messageTask> element	**</messageTask>**
xmlWriter.WriteEndElement();	Closes the root <messageTasks> element	**</messageTasks>**
xmlWriter.WriteEndDocument(); xmlWriter.Close();	Marks the end of the XML document. Closes the write stream, freeing up the resources used.	

Full code

```
XmlWriterSettings settings = new XmlWriterSettings
{
    Indent = true // Adaugă indentare
                  // pentru lizibilitate
};

XmlWriter xmlWriter = XmlWriter.Create
    ( outputFileName: "D:\\test.xml", settings);

xmlWriter.WriteStartDocument();

xmlWriter.WriteStartElement("messageTasks");

xmlWriter.WriteStartElement("messageTask");
xmlWriter.WriteAttributeString("id", "3");

xmlWriter.WriteStartElement("desc");
xmlWriter.WriteString("Sarbatori fericite!");
xmlWriter.WriteEndElement();

xmlWriter.WriteStartElement("from");
xmlWriter.WriteString("Jane");
xmlWriter.WriteEndElement();

xmlWriter.WriteStartElement("to");
xmlWriter.WriteString("Michelle");
xmlWriter.WriteEndElement();

xmlWriter.WriteStartElement("message");
xmlWriter.WriteString("Craciun Fericit!");
xmlWriter.WriteEndElement();

xmlWriter.WriteEndElement();

xmlWriter.WriteEndDocument();
xmlWriter.Close();
```

XML QUERY - LOAD

```
XDocument document = XDocument.Load("filepath");
var result = from r in
    document.Descendants( "?") //...

        r.Attribute("?").Value
        r.Element("?").Value
```

XML QUERY - LOAD

Cod	Explicații
XDocument document = XDocument.Load("D:\\Messages.xml");	XDocument.Load loads the XML file Messages.xml from the specified path (D:\\Messages.xml) into an XDocument object.
var messages = from r in document.Descendants("messageTask") .Where (x=>x.Attribute("id").Value.CompareTo("1")==0)	Finds all messageTask elements in the XML file, regardless of their level in the document hierarchy, and returns a collection of type IEnumerable<XElement> Selects only instances that respect a specific property

XML QUERY - LOAD

Cod	Explicații
select new { Id = r.Attribute("id").Value, Desc = r.Element("desc").Value, From = r.Element("from").Value, To = r.Element("to").Value, Message = r.Element("message").Value };	For each messageTask element, the code creates an anonymous object that contains various properties
messages.ToList().ForEach(x => Console.WriteLine(x.Id + " " + x.Desc + " " + x.From + " " + x.Message));	Displays the values of the Id, Desc, From, and Message fields in one line

Full code

```
XDocument document = XDocument.Load("D:\\test.xml");
var messages2 :IEnumerable<{Id,Desc,...}> =
    from r :XElement in document.Descendants( "messageTask")
        .Where(x=>x.Attribute( "id").Value.CompareTo("3")==0) // IEnumerable<XElement>
    select new
    {
        Id = r.Attribute( "id").Value,
        Desc = r.Element( "desc").Value,
        From = r.Element( "from").Value,
        To = r.Element( "to").Value,
        Message = r.Element( "message").Value
    };
messages2.ToList().ForEach(x :{Id,Desc,...} =>
    Console.WriteLine(x.Id + " " + x.Desc + " " + x.From + " " + x.Message));
```

Exercise

🎅 Santa needs your help! 🎄

The elves are very busy preparing gifts, but they have a problem: they can't continue without listening to carols originating from England. Unfortunately, Santa has a huge list of carols and he can't check them all by himself. He needs your help! As a reward, he will give everyone a sweet surprise 🍬

Here's what you need to do:

Help Santa Claus select all the carols that come from England from his big list of carols. If you succeed, the elves will be able to continue preparing gifts for all the children in the world!

Exercise

Attention! (constraints)

Santa Claus offers you his list of all the carols in XML format and expects to receive the ones the elves want also in XML format to be saved for a loooong time, in case the elves continue to have this preference in the years to come. You don't want to do the same thing every year, do you? 🙃

So... get your coding skills out there and help Santa save Christmas! 🎁

```
<colind id="1">
  <nume>O, ce veste minunată</nume>
  <original>O, ce veste minunată</original>
  <tara>România</tara>
</colind>
```

Kahoot

XML QUERY

```
▼<messageTasks>
  ▼<messageTask id="3">
      <desc>Urare</desc>
      <from>Echipa MAP</from>
      <to>Informatica Romana</to>
      <message>Sarbatori Fericite!</message>
    </messageTask>
  </messageTasks>
```

MULȚUMIM PENTRU ATENȚIE!