

LANGUAGE INTEGRATED QUERY IN .NET

Ali Khawaja

CS340 | SBASSE | LUMS

Anonymous Types

<https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/types/anonymous-types>

Anonymous types provide a convenient way to encapsulate a set of read-only properties into a single object without having to explicitly define a type first. The type name is generated by the compiler and isn't available at the source code level. The type of each property is inferred by the compiler.

You create anonymous types by using the `new` operator together with an object initializer.

The following example shows an anonymous type that is initialized with two properties named `Amount` and `Message`.

```
var v = new { Amount = 108, Message = "Hello" };

// Rest the mouse pointer over v.Amount and v.Message in the following
// statement to verify that their inferred types are int and string.
Console.WriteLine(v.Amount + v.Message);
```

Join Operations in LINQ

<https://learn.microsoft.com/en-us/dotnet/csharp/linq/standard-query-operators/join-operations>

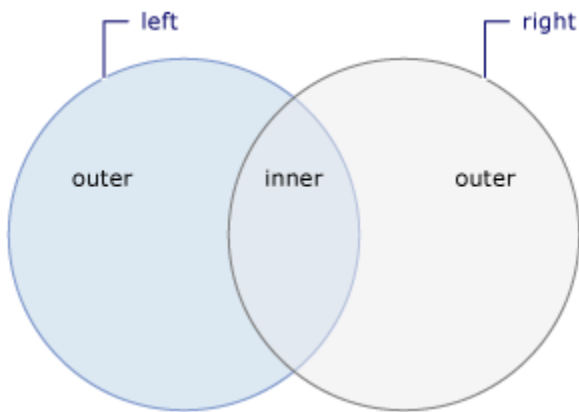
A join of two data sources is the association of objects in one data source with objects that share a common attribute in another data source.

The join methods provided in the LINQ framework are `Join` and `GroupJoin`. These methods perform equijoins, or joins that match two data sources based on equality of their keys. (For comparison, Transact-SQL supports join operators other than equals, for example the less than operator.)

In relational database terms, `Join` implements an inner join, a type of join in which only those objects that have a match in the other data set are returned.

The `GroupJoin` method has no direct equivalent in relational database terms, but it implements a superset of inner joins and left outer joins.

A left outer join is a join that returns each element of the first (left) data source, even if it has no correlated elements in the other data source.



Data & Code Repository:

<https://github.com/dotnet/docs/blob/main/docs/csharp/linq/standard-query-operators/snippets/standard-query-operators>

Join Overview – Code sample

<https://github.com/dotnet/docs/blob/main/docs/csharp/linq/standard-query-operators/snippets/standard-query-operators/JoinOverviewExamples.cs>

Inner Join

An Inner Join returns only the elements that have matching keys in both collections.

```
var result = from c in customers
              join o in orders on c.Id equals o.CustomerId
              select new { c.Name, o.OrderId, o.Amount };
```

<https://github.com/dotnet/docs/blob/main/docs/csharp/linq/standard-query-operators/snippets/standard-query-operators/InnerJoins.cs>

Group Join

A Group Join produces a hierarchical result set where each element from the first collection is paired with a group of matching elements from the second collection.

```
var result = from c in customers
              join o in orders on c.Id equals o.CustomerId into orderGroup
              select new { c.Name, Orders = orderGroup };
```

Left Outer Join

A Left Outer Join returns all elements from the first collection and the matched elements from the second collection. If no match exists, default values are used.

```
var result = from c in customers
              join o in orders on c.Id equals o.CustomerId into orderGroup
```

```
from o in orderGroup.DefaultIfEmpty()  
select new { c.Name, OrderId = o?.OrderId, Amount =  
o?.Amount };
```

Join with Multiple Keys

You can join on multiple keys by using anonymous types.

```
var result = from c in customers  
             join o in orders  
             on new { c.Id } equals new { Id = o.CustomerId }  
             select new { c.Name, o.OrderId };
```

Set Operations

<https://learn.microsoft.com/en-us/dotnet/csharp/linq/standard-query-operators/set-operations>

Set operations in LINQ refer to query operations that produce a result set based on the presence or absence of equivalent elements within the same or separate collections.

Method names	Description
Distinct or DistinctBy	Removes duplicate values from a collection.
Except or ExceptBy	Returns the set difference, which means the elements of one collection that don't appear in a second collection.
Intersect or IntersectBy	Returns the set intersection, which means elements that appear in each of two collections.
Union or UnionBy	Returns the set union, which means unique elements that appear in either of two collections.

Code Sample:

<https://github.com/dotnet/docs/blob/main/docs/csharp/linq/standard-query-operators/snippets/standard-query-operators/SetOperations.cs>