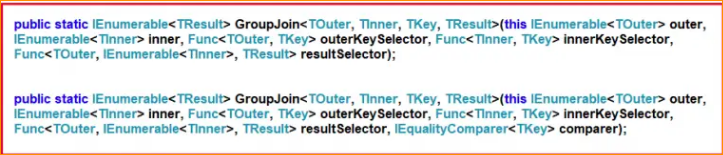
1. What is LINQ Group Join?

In LINQ, we can apply the **Group Join on two or more data sources based on a common key** (**the key must exist in both the data sources**) like the Inner Join, and then LINQ Group Join produces the result set in the form of groups. We can say that LINQ Group Join groups’ **result sets are based on a common key.**

So, the Group Join is basically used to produce hierarchical data structures. Each item from the first data source is paired with a set of correlated items from the second data source. There are two overloaded versions of this GroupJoin method available in LINQ. They are as follows.



The only difference between the two overloaded versions is that the second version takes an additional IEqualityComparer as an extra parameter. So, while working with Linq Group Join, we need to understand the following five things.

* Outer Data Source
* Inner Data Source
* Outer Key Selector
* Inner Key Selector
* Result Selector

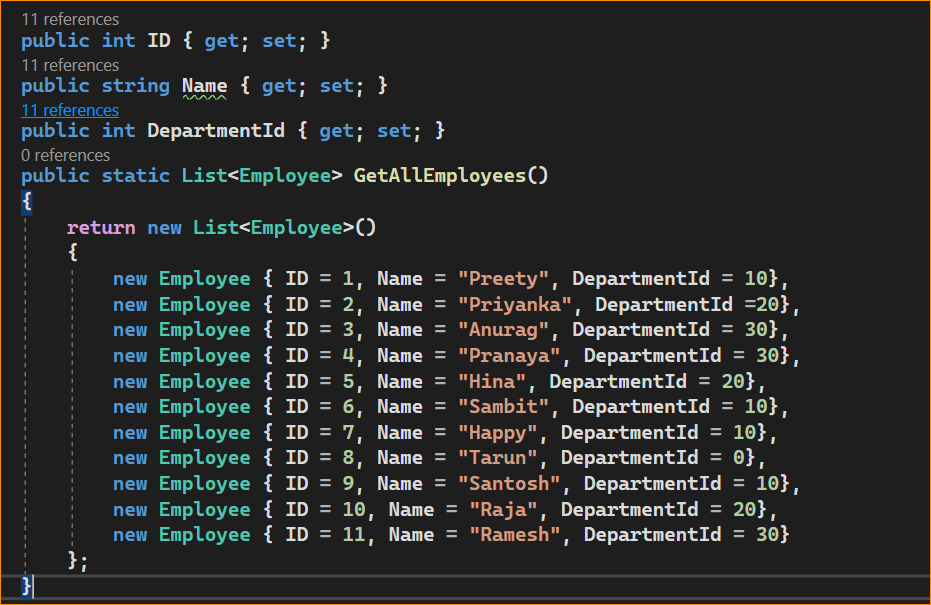
Use Cases of LINQ Group Join:

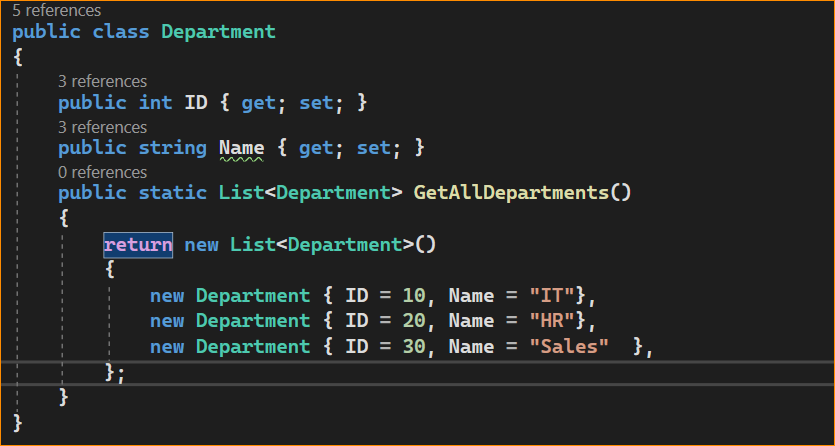
* Hierarchical Data Representation: Useful in scenarios where you need to represent hierarchical data, like categorizing products under categories.
* Data Aggregation: When aggregating data from different sources or tables based on a common key.
* Relationship Mapping: To map relationships between entities, especially in one-to-many or many-to-many relationships.

1. Examples to Understand LINQ Group Join in C#:

The LINQ GroupJoin method in C# creates a group joins between two sequences based on a common key to both sequences. This operation is akin to a left outer join in SQL, where each element from the first sequence (outer sequence) is mapped to a collection of matching elements from the second sequence (inner sequence). If an element from the outer sequence has no matches in the inner sequence, it is still included in the result with an empty collection of inner elements.

Let us understand How to Implement LINQ Group Join with examples using C# language. For this, we will use the following two model classes, i.e., Employee and Department. So, create a class file named Employee.cs and copy and paste the following code. This class has 3 properties, i.e., ID, Name, and DepartmentId. We have also created one method to return a collection of Employees, which will be our first data source for the Group Join.



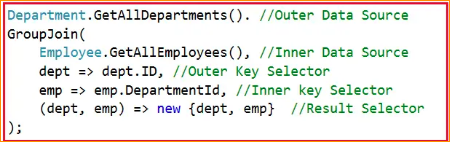


As you can see, we created the above Employee and Department classes with some simple properties. The **common property is Department ID**, i.e., the **ID property in the Department class** and **DepartmentID property in the Employee class**. Then, we create two simple methods to return the respective data sources. Further, if you notice, the employee with ID 8 does not have a department.

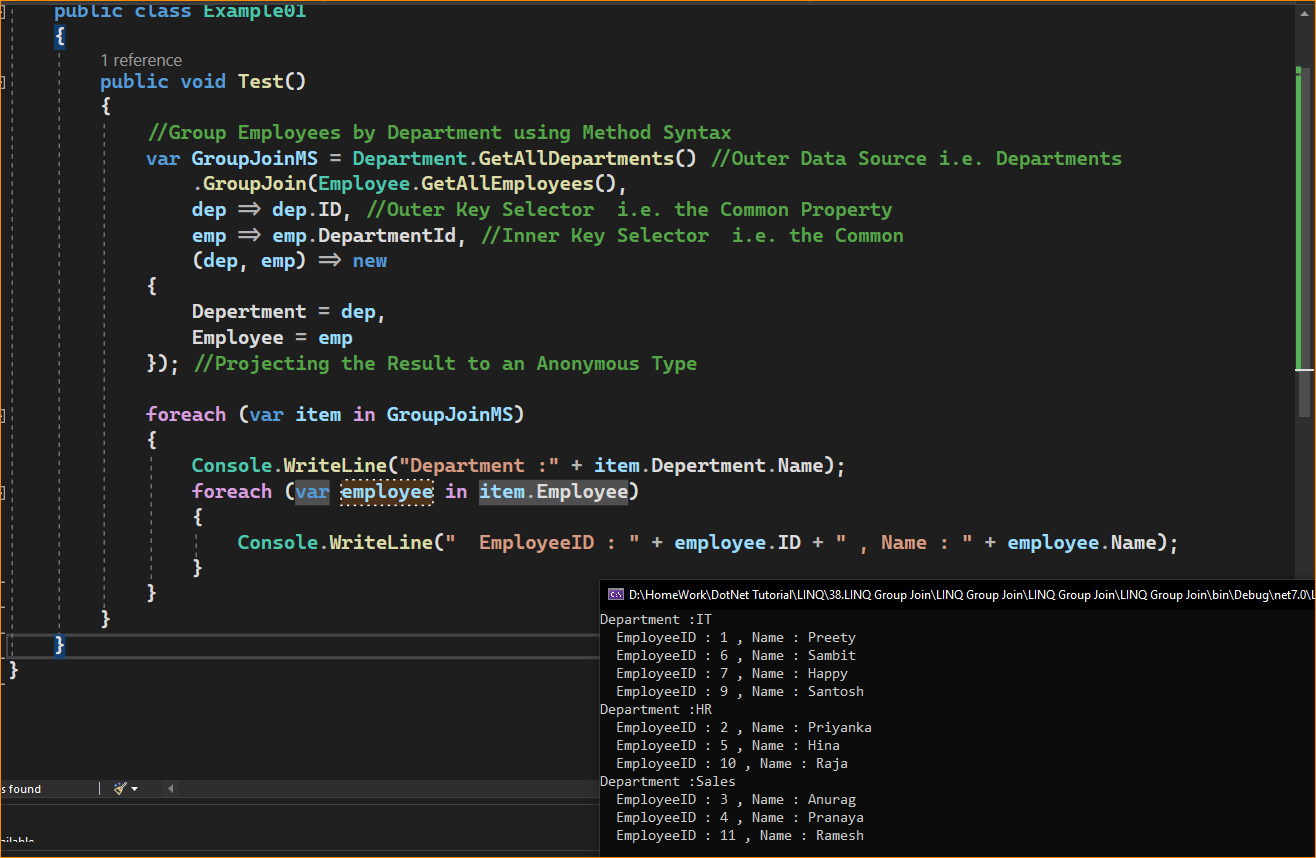
1. Example to Understand LINQ GroupJoin Method Using Method Syntax in C#

Now, our business requirement is to **group the employees by department**.

So, the outer data source will be the department data source, and the inner data source will be the employee data source. We must use the **LINQ GroupJoin Method** to group the employees by department. The following code snippet shows how to group the employees by department using the LINQ GroupJoin Method using Method Syntax.

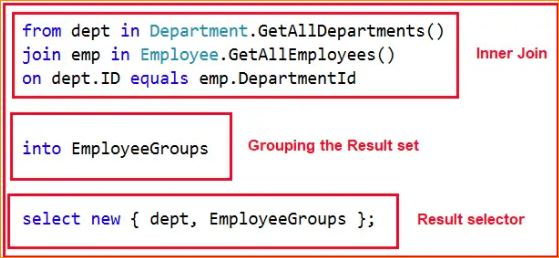


As you can see in the above code snippet, our Outer Data Source is the Department’s collection, and our Inner Data Source is the Employee’s collection. Here, we are accessing the Department collection using the dept variable and the Employees collection using the emp variable. Here, the Outer Key Selector is the ID property of the Department class, and the Inner Key Selector is the DepartmentId property of the Employee class. Finally, we are projecting the dept and emp to an Anonymous type. The complete example code is given below. The following example code is self-explained, so please go through the comment lines.

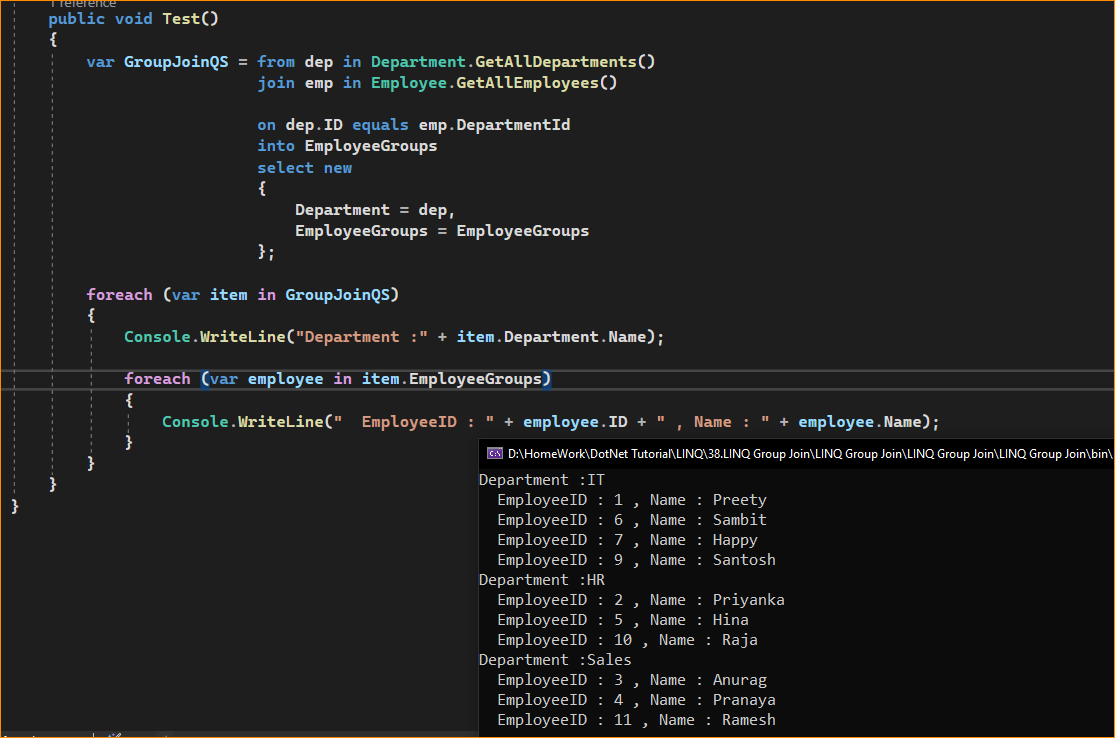


1. Example to Understand LINQ GroupJoin Using Query Syntax in C#

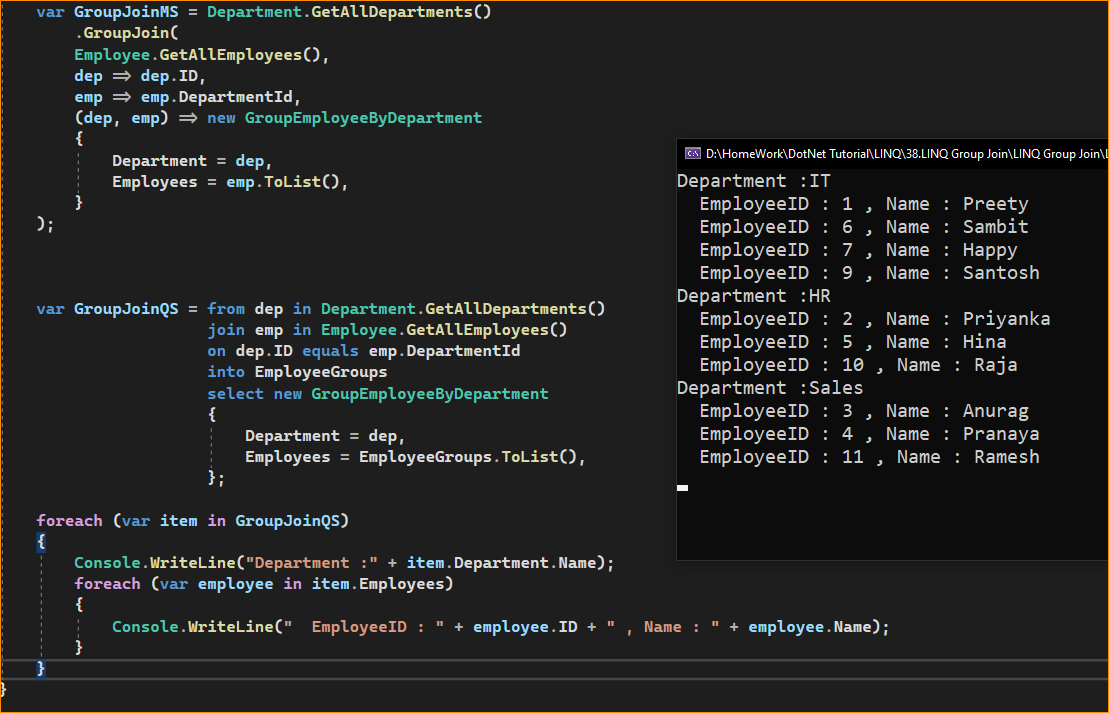
In LINQ Query Syntax, there is no such Group Join operator available. Here, we need to use the LINQ Inner Join and the “into” operator. For a better understanding, please look at the following code snippet. Here, we have divided the code snippet into three sections for a better understanding. In the first section, we perform the LINQ Inner Join Operation between the Department and Employee Data Sources. In the second section, we project the result of the Inner Join into a variable called EmployeeGroups using the “into” operator. And in the final section, we are projecting the final result set as dept and EmployeeGroups. That is, it will group employees by department.



The complete example code is given below. The following example code is self-explained, so please go through the comment lines. Here, we are using LINQ Query Syntax to implement Group Join.



So far, in the examples we have discussed, we have projected the result to an anonymous type. Can we Project the Result to a Named type? Yes, it is also possible to project the result to a named type instead of an anonymous type. Let us see how we can do this. First, create a class file named GroupEmployeeByDepartment.cs with the required properties you want in the result set. We have created the class with the following two properties per our requirement.



1. When to Use LINQ Group Join in C#?

The LINQ GroupJoin method is particularly useful in scenarios where you need to create a one-to-many relationship between two sequences. Here are some common situations where GroupJoin is beneficial:

* Hierarchical Data Representation: When you need to represent hierarchical or nested data, such as categories and their associated products, employees and their departments, or authors and their books.
* Data Aggregation: In scenarios where you want to aggregate data from related collections, like summing sales by region or calculating average scores by class.
* Combining Data from Different Sources: When dealing with data from different sources (like databases, XML, or JSON), you need to combine related items based on a common key.
* One-to-Many Relationships: Particularly useful in representing one-to-many relationships, where each element in the first collection (outer sequence) is mapped to a collection of related elements from the second collection (inner sequence).
* Reporting and Data Analysis: For generating reports or performing data analysis, where you need to group related data together, such as listing all orders for each customer.

Considerations of LINQ Group Join:

* GroupJoin is ideal when each element in the outer sequence can be related to zero or more elements in the inner sequence.
* It’s important to understand the difference between Join and GroupJoin. Use Join for one-to-one relationships and GroupJoin for one-to-many relationships.
* The resulting sequence from GroupJoin contains elements of the outer sequence, each paired with a collection of matching elements from the inner sequence.
* The GroupJoin operator is useful when performing operations involving combining elements from two collections based on a shared key and grouping the results. It is commonly used in scenarios such as database joins and data aggregation.