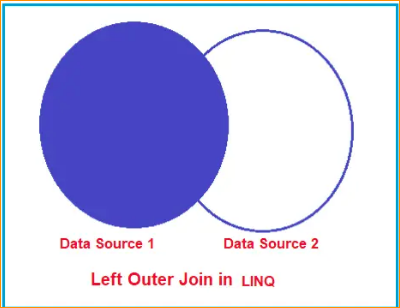
1. What is Left Join?

The Left Join or Left Outer Join is a Join in which each data from the first data source is going to be returned irrespective of whether it has any correlated data present in the second data source or not. That means the LEFT OUTER JOIN retrieves all the matching rows from both the data sources involved in the join and non-matching rows from the left side data source. In this case, the un-matching data will take a null value. For a better understanding, please look at the following diagram, which shows the graphical representation of the Left Outer Join.



So, in simple words, we can say that the Left Outer Join is going to return all the matching data from both the data sources as well as all the non-matching data from the left data source. In such cases, for the non-matching data, it will take null values for the second data source.

1. Left Outer Join in LINQ

In LINQ, you can perform a left join operation between two collections (or tables in a database) using the join keyword followed by the into keyword to create a temporary grouping of results. You can then use the DefaultIfEmpty method to specify the left join behavior. So, to implement the LINQ Left Join, it’s mandatory to use the into keyword along with the DefaultIfEmpty() method.

1. Examples to Understand Left Outer Join in LINQ:

Let us Understand Left Outer Join in LINQ with Examples using C# Language. To Understand LINQ Left Outer Join, we will use the following Employee and Address Data Sources. So, first, create a class file with the name Employee.cs and then copy and paste the following code into it. This is a very simple class having 3 properties, i.e., Id, Name, and AddressId. We have also created one method, i.e., GetAllEmployees(), which will return a collection of Employees, one of our data sources.



Next, create another class file with the name Address.cs and copy and paste the following code. This is a very simple class having 2 properties, i.e., ID and AddressLine. We have also created one method, i.e., GetAddress(), which is going to return a collection of addresses, which is going to be our second data source.



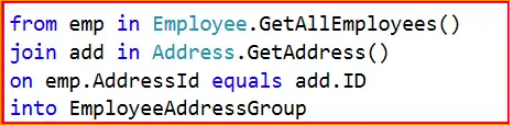
In our example, we created two methods that will be going to return the respective Employees and Addresses, which are going to be our data sources. We have hard-coded the data sources, but in real time, you will get the data from a database. If you further notice, we have two employees with address ID 0, which means these two employees do not have a matching address in the address data source.

1. Examples to Understand LINQ Left Outer Join using Query Syntax in C#:

Let us see an example to Understand LINQ Left Outer Join using Query Syntax using C#. To perform the left outer join using query syntax in LINQ, we need to call the DefaultIfEmpty() method on the results of a Group Join. In LINQ Query Syntax, there is no such Group Join operator available.

Step1:

The first step to implement the LINQ Left Outer Join is to perform Group Join and store the result into a variable using the “into” operator. In the code snippet below, as you can see, the list of Employees is inner-joined with the list of Addresses based on the AddressId Property of the Employee object that matches the ID property of the Address object. And then, we store the result in the EmployeeAddressGroup using the “into” operator. The following code exactly does the same thing.



Step2:

//Left Data Source: Employees

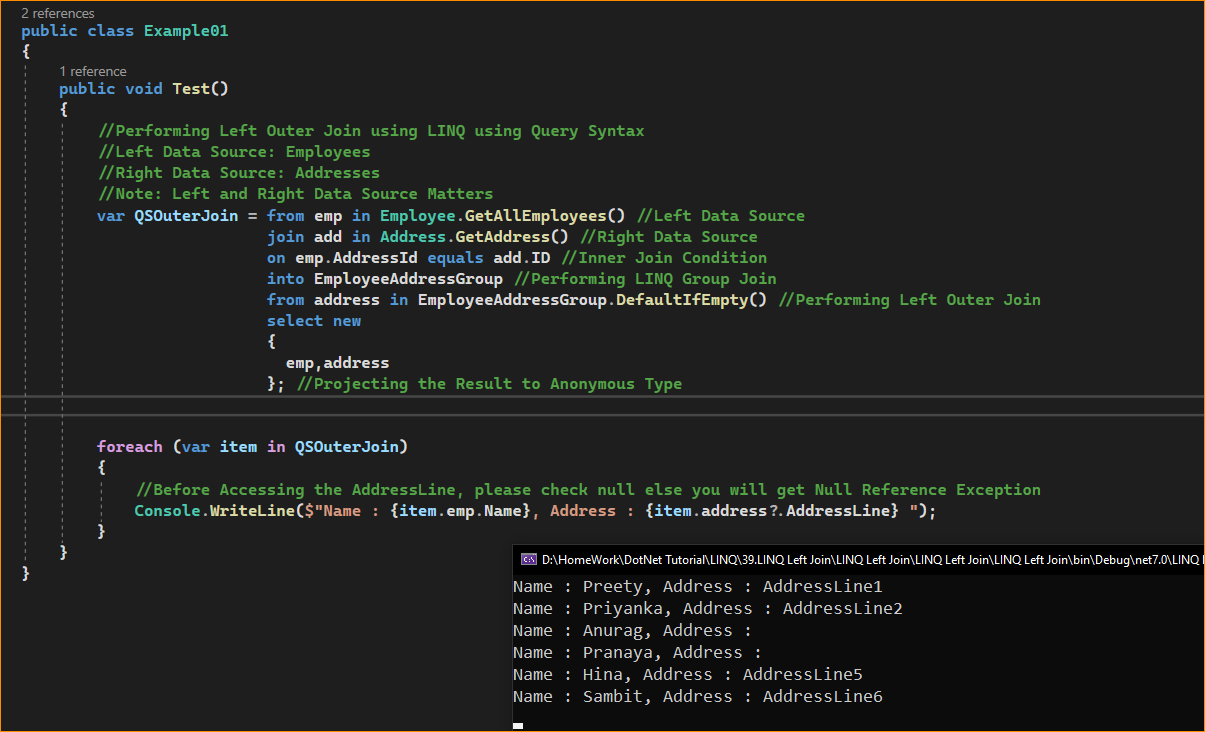
//Right Data Source: Addresses

In the second step, we need to include each element of the first (i.e., left) data source in the result set, irrespective of whether that element has any matching element found in the second (i.e., right) data source. To do this, we need to call the DefaultIfEmpty() method on each sequence of matching elements from the group join.

In our example, we must call the DefaultIfEmpty() method on each sequence of matching Address objects. The DefaultIfEmpty() method returns a collection that contains a single default value if the sequence of matching Address objects is empty for any Employee object, which will ensure that each Employee object is represented in the result collection. That means if the corresponding value is empty, it will take the default value based on the data type. The following code exactly does the same thing.



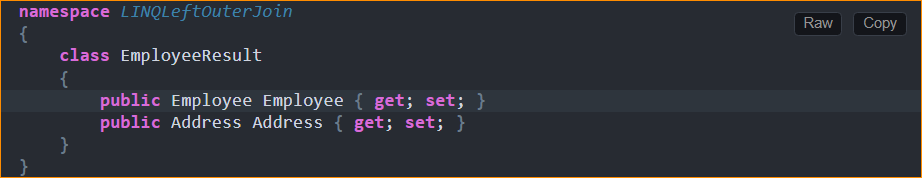
The default value for a reference type is null. So, you need to check for the null reference before accessing each element of the Address collection. The complete example code is given below. The following example code is self-explained, so please go through the comment lines for a better understanding. In the below example, we are implementing the LINQ Left Outer Join using Query Syntax. In the below example, we are projecting the result to an anonymous type.



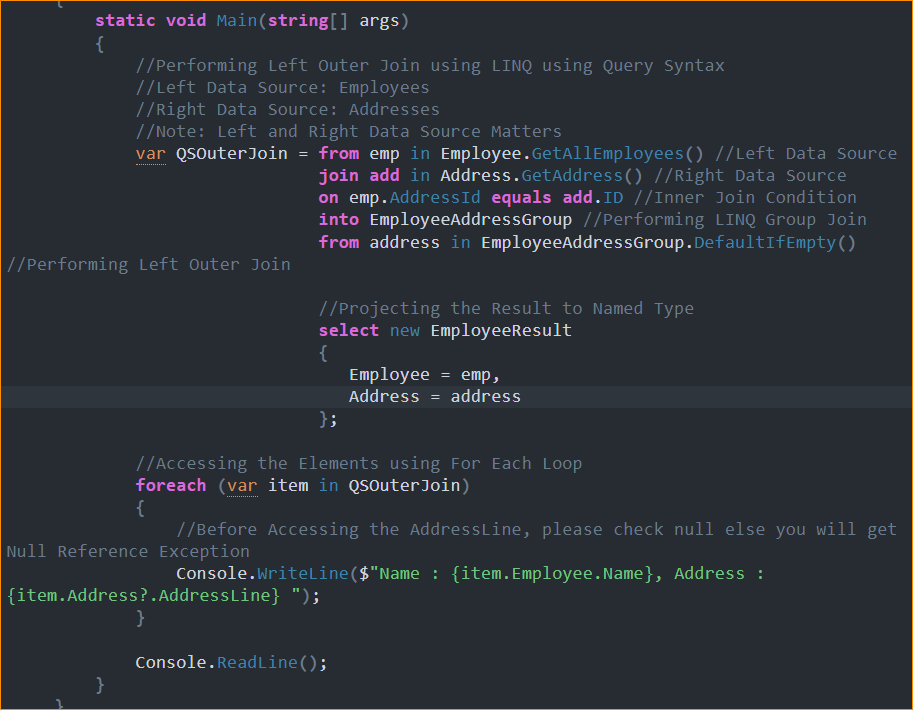
Now run the application, and you will get the following output. As you can see, even though Anurag and Pranaya do not have the corresponding address, they are still returned as part of the result set. In that case, the Address will be NULL.

1. Projecting the Result to a Named Type:

Instead of projecting the result to an anonymous type, we can also project the result to a named type. Let us see how we can do this. First, create a class file with the name EmployeeResult.cs with the required properties that you want in the result set. As per our requirement, we have created the class with two properties.



Next, modify the Main method of the Program class as follows. Here, we are projecting the result to the above-created EmployeeResult type.

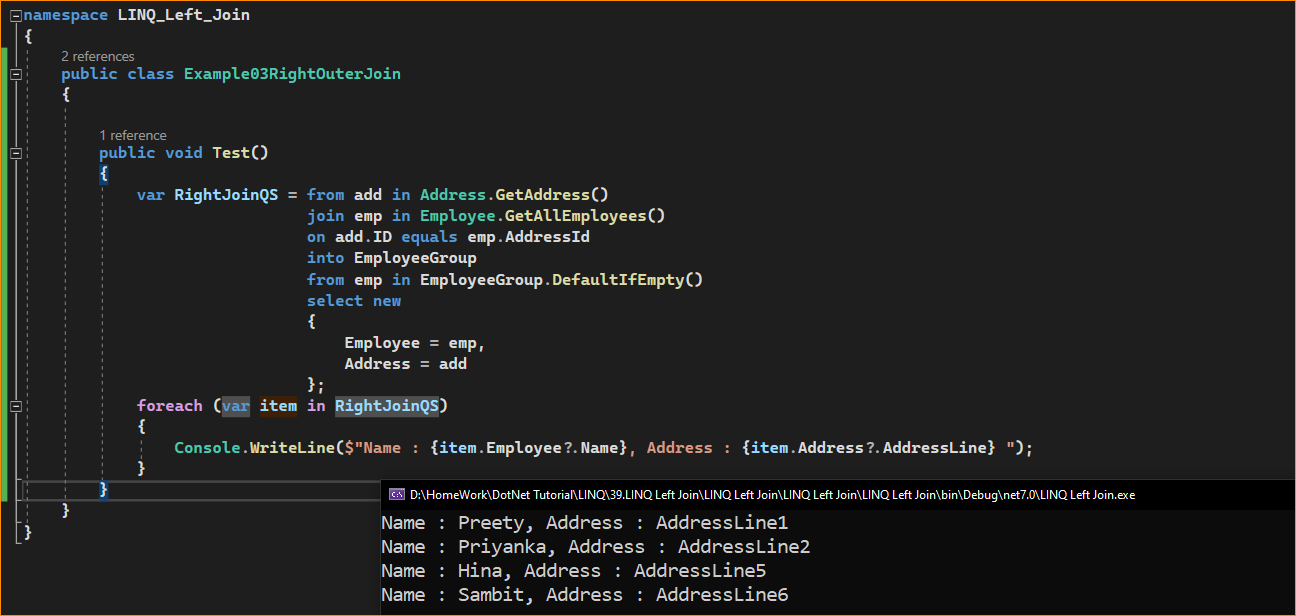


With the above changes in place, run the application, and you will also get the same output as the previous example, as shown in the below image.

1. LINQ Right Outer Join

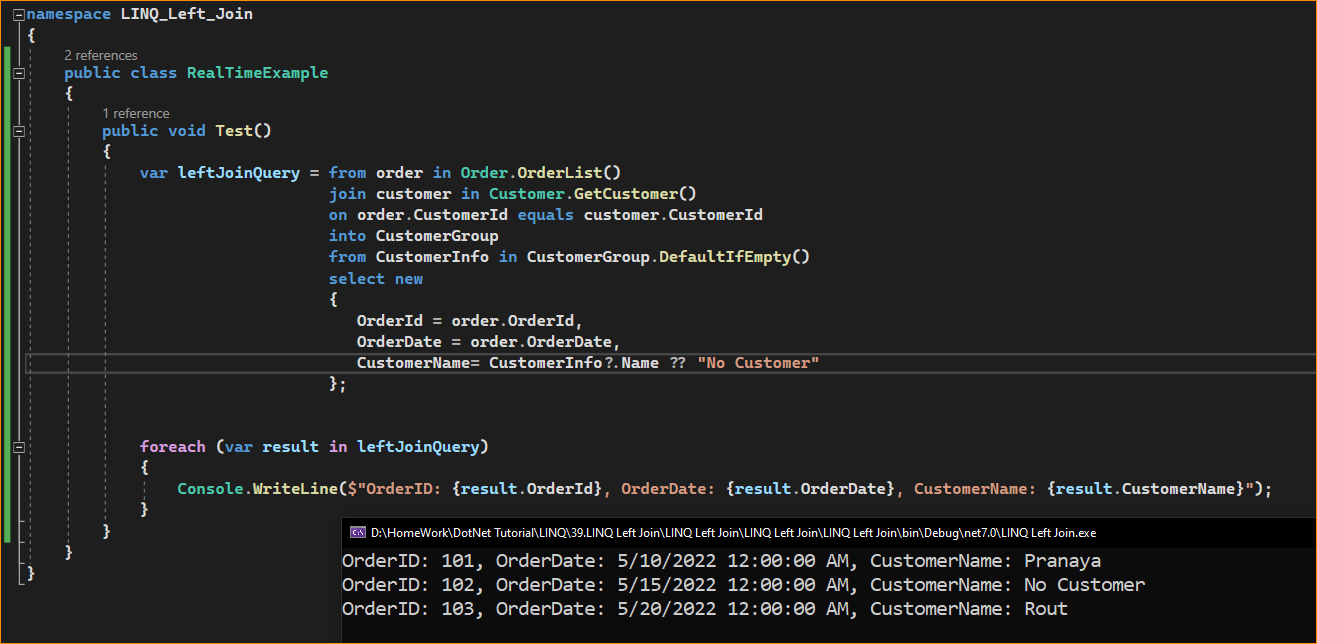
The RIGHT OUTER JOIN retrieves all the matching rows from both the data sources involved in the join and non-matching rows from the right-side data source. In this case, the un-matching data will take the default values. But, Right Outer Join is not supported with LINQ. LINQ only supports left outer joins.

To perform the Right Outer Join, exchange the data sources. In our previous examples, we have used Employees as the Left Data Source and Addresses as the Right Data Source. In the below example, we are just changing the Data Sources. Now, we are making Address as the Left Data Source and Employees the Right Data Source.



1. Real-Time Example to Understand LINQ Left Join:

Suppose you have a list of orders and a list of customers, and you want to generate a report of all orders along with the corresponding customer names (if available) or a default value if there’s no matching customer. In this case, a left join would be appropriate because you want to include all orders in the report. For a better understanding, please have a look at the below example:



In the above example, we first define two collections: orders and customers. We then use a LINQ query to perform a left join between these collections based on the CustomerId field. The into keyword is used to create a temporary grouping of results for each order, and the DefaultIfEmpty method is used to specify the left join behavior.

The result is a sequence of anonymous objects that contain order information along with customer names (if available) or “No Customer” if there is no matching customer for an order.

1. When to use LINQ Left Join in C#?

You should use a LINQ Left Join in C# when you want to combine data from two collections (or tables in a database) based on a common key, and you want to include all elements from the left (first) collection in the result, even if there are no matching elements in the right (second) collection. In other words, a left join ensures that every element from the left collection is included in the result, and for each matching element in the right collection, you combine data from both collections.

1. Here are some common scenarios where you might use a LINQ left join:

* **Combining Data with Missing Records**: When you have two collections, and you want to combine them while preserving all elements from one collection and including data from the other collection when a match is found, but not excluding any elements from the first collection.
* **Optional Relationships**: In database scenarios, if you have a parent-child relationship between two tables where some child records may not have a corresponding parent record, a left join ensures that you get all child records along with their parent data when available.
* **Reporting and Analysis**: When working with data for reporting or analysis, you may want to include all data from one source while enriching it with additional information from another source, and you don’t want to lose any records from the first source.
* **Handling Default Values**: You can use left joins to provide default values or placeholders for missing data from the second collection when combining data.