Introduction to LINQ: Part 4 - Aggregating Data

Before reading this blog post, you should read Introduction to LINQ: Part 1 - Selecting Data at https://www.pdsa.com/Resources-BlogPosts/2020-02-LINQ-Select.pdf. That post will introduce you to the data and the general approach to how LINQ works.

In this blog post you are going to learn how to use various aggregate methods of LINQ to count, sum, get a minimum and maximum value and get an average value within a collection of data.

Count All Items in the Collection

The Count() method counts all the items within a collection. Below are examples for both the query and the method syntax.

```
public void Count()
{
  int value;

  // Load all Product Data
  LoadProductsCollection();

  if (UseQuerySyntax) {
    // Query Syntax
    value = (from prod in Products
             select prod).Count();
  }
  else {
    // Method Syntax
    value = Products.Count();
  }

  ResultText = $"Total Products = {value}";
}
```

Count Only Specific Items in Collection

To only count a certain amount of records within the collection, pass in a predicate expression to only select those records you want to count.

```
public void CountFiltered()
{
  int value;

  // Load all Product Data
  LoadProductsCollection();

  if (UseQuerySyntax) {
    // Query Syntax
    value = (from prod in Products
             select prod)
            .Count(prod => prod.Color == "Red");
  }
  else {
    // Method Syntax
    value = Products.Count(prod => prod.Color == "Red");
  }

  ResultText = $"Total Products with a color of 'Red' = {value}";
}
```

Another approach to filtering data for counting is to use the where operator or the Where() method as shown in the following sample.

```
public void CountFiltered()
{
  int value;

  // Load all Product Data
  LoadProductsCollection();

  if (UseQuerySyntax) {
    // Query Syntax
    value = (from prod in Products
             where prod.Color == "Red"
             select prod).Count();
  }
  else {
    // Method Syntax
    value = Products.Where(prod => prod.Color == "Red").Count();
  }

  ResultText = $"Total Products with a color of 'Red' = {value}";
}
```

Add Up All List Prices Using Sum

You may add any numeric property within your collection using the Sum() method. Below is an example of both the query and method syntax to get the total of all *ListPrice* values in the collection of Products.

```
public void Sum()
{
  decimal? value;

  // Load all Product Data
  LoadProductsCollection();

  if (UseQuerySyntax) {
    // Query Syntax
    value = (from prod in Products
             select prod.ListPrice).Sum();
  }
  else {
    // Method Syntax
    value = Products.Sum(prod => prod.ListPrice);
  }

  if (value.HasValue) {
    ResultText = $"Total of all List Prices = {value.Value:c}";
  }
  else {
    ResultText = "No List Prices Exist.";
  }
}
```

Another approach when using the query syntax is to include the *ListPrice* property in the Sum() as you did when using the method syntax.

```
value = (from prod in Products
         select prod)
         .Sum(prod => prod.ListPrice);
```

Get the Lowest List Price Using Min

The Min() method iterates over the collection and extracts the lowest value from the property you specify.

```
public void Minimum()
{
  decimal? value;

  // Load all Product Data
  LoadProductsCollection();

  if (UseQuerySyntax) {
    // Query Syntax
    value = (from prod in Products
             select prod.ListPrice).Min();

    // Alternate Syntax
    //value = (from prod in Products
    //         select prod)
    //        .Min(prod => prod.ListPrice);
  }
  else {
    // Method Syntax
    value = Products.Min(prod => prod.ListPrice);
  }

  if (value.HasValue) {
    ResultText = $"Minimum List Price = {value.Value:c}";
  }
  else {
    ResultText = "No List Prices Exist.";
  }
}
```

Get the Highest List Price Using Max

The Max() method iterates over the collection and extracts the highest value from the property you specify.

```
public void Maximum()
{
  decimal? value;

  // Load all Product Data
  LoadProductsCollection();

  if (UseQuerySyntax) {
    // Query Syntax
    value = (from prod in Products
             select prod.ListPrice).Max();

    // Alternate Syntax
    //value = (from prod in Products
    //         select prod)
    //        .Max(prod => prod.ListPrice);
  }
  else {
    // Method Syntax
    value = Products.Max(prod => prod.ListPrice);
  }

  if (value.HasValue) {
    ResultText = $"Maximum List Price = {value.Value:c}";
  }
  else {
    ResultText = "No List Prices Exist.";
  }
}
```

Get the Average List Price

The Average() method iterates over the collection and gets the average value of all items from the property you specify.

```
public void Average()
{
  decimal? value;

  // Load all Product Data
  LoadProductsCollection();

  if (UseQuerySyntax) {
    // Query Syntax
    value = (from prod in Products
             select prod.ListPrice).Average();

    // Alternate Syntax
    //value = (from prod in Products
    //         select prod)
    //        .Average(prod => prod.ListPrice);
  }
  else {
    // Method Syntax
    value = Products.Average(prod => prod.ListPrice);
  }

  if (value.HasValue) {
    ResultText = $"Average List Price = {value.Value:c}";
  }
  else {
    ResultText = "No List Prices Exist.";
  }
}
```

Simulate Sum using Aggregate Method

The Aggregate() method allows you to iterate over a collection and summarize data based on a predicate expression. The first parameter you pass to Aggregate() is a starting value you wish to add to. The second parameter is the function to which you pass the seed value and each item in the collection as this method iterates over the collection. Within the predicate perform whatever operation you want to add to the accumulator value.

In the code below, you initialize the "sum" variable to 0M, which means you are summing a decimal value within this expression. The first pass over the collection a zero and the first product are passed to the predicate where you add the value in the *ListPrice* property to the sum. The next pass over the collection, the new sum value and the next product are passed to the predicate and so on. Once the collection has been processed, the value in the sum is returned from the Aggregate() method.

```
public void AggregateSum()
{
  decimal? value = 0;

  // Load all Product Data
  LoadProductsCollection();

  if (UseQuerySyntax) {
    // Query Syntax
    value = (from prod in Products select prod)
            .Aggregate(0M, (sum, prod) =>
              sum += prod.ListPrice.Value);
  }
  else {
    // Method Syntax
    value = Products.Aggregate(0M, (sum, prod) =>
                sum += prod.ListPrice.Value);
  }

  if (value.HasValue) {
    ResultText = $"Total of all List Prices = {value.Value:c}";
  }
  else {
    ResultText = "No List Prices Exist.";
  }
}
```

Iterate Over Collection Using Aggregate

In the following example, you use a ternary operator to only sum those products where the *Color* property is equal to 'Black'.

```
public void Aggregate()
{
  decimal? value = 0;

  // Load all Product Data
  LoadProductsCollection();

  if (UseQuerySyntax) {
    // Query Syntax
    value =
     (from prod in Products select prod)
      .Aggregate(0M, (sum, prod) =>
      prod.Color == "Black" ? sum += prod.ListPrice.Value : sum);
  }
  else {
    // Method Syntax
    value = Products.Aggregate(0M, (sum, prod) =>
      prod.Color == "Black" ? sum += prod.ListPrice.Value : sum);
  }

  if (value.HasValue) {
    ResultText = $"Total List Prices of the Color 'Black'
                  = {value.Value:c}";
  }
  else {
    ResultText = "No List Prices Exist for the Color 'Black'.";
  }
}
```

Summary

In this blog post you learned how to use various aggregate methods to count, sum, get a minimum and a maximum value. You also gathered average list prices and learned how to perform your own calculations using the Aggregate() method. As with most things in LINQ, there are multiple ways to accomplish the same result.

Sample Code

You can download the complete sample code at my https://github.com/PaulDSheriff/BlogPosts page.