|  |
| --- |
|  |
| LINQ Queries |
| Finding Answers |
|  |
|  |
|  |

|  |
| --- |
|  |

# 

LINQ Queries

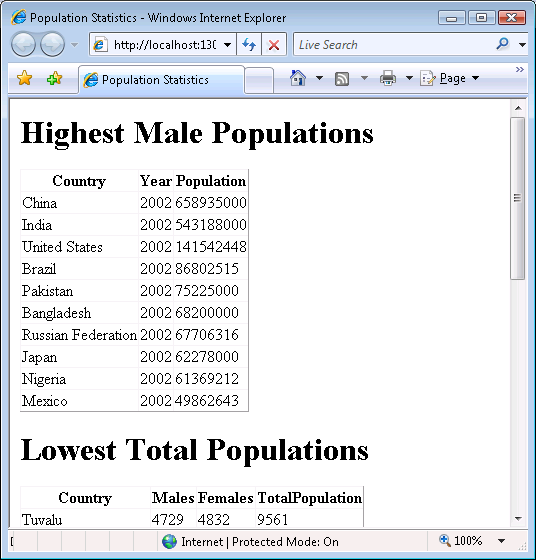
# Objectives

After completing this lab, you should understand how to do the following:

* Understand how to join, filter, sort, use subqueries, and project data using LINQ.
* Convert LINQ comprehension queries to queries using extension methods and lambda expressions, and vice versa.

# Overview

In this lab, we’ll be working with global population statistics and writing LINQ queries to render population reports from an ASP.NET web page. You’ll be working in the LINQ\_Queries\before directory. You can look at a completed version of the web page in the LINQ\_Queries\after directory.



# Part 1 – Exploring the Project

1. Open the web site in the LINQ\_Queries\before directory.
2. Take a look at the XML files in the App\_Data directory. These files represent the raw statistics we will be working with.
3. Examine the PopulationEntry.cs file in the App\_Code directory. The PopulationEntry class will be the object that represents each entry from the XML files.
4. Look at the PopulationStatistics.cs file in the App\_Code directory. This class uses LINQ to XML (which is a topic for another module) to load the raw statistics into PopulationEntry objects. This class also exposes the collection of entries through public Male and Female properties.
5. Finally, take a look at the default.aspx web page. GridView controls and headers are already in place to display the reports we need. All we have to do is fill in the LINQ queries.

# Part II –Warming Up With Filtering and Sorting

Open the Default.aspx.cs file.

1. Create a protected field named *\_stats* to hold a PopulationStatistics reference.
2. Inside of the Page\_Load method, we will databind the GridView controls when the page is not in a postback operation. Add a check for !IsPostBack. Inside the check, instantiate a new PopulationStatistics object for the \_stats field.

protected void Page\_Load(object sender, EventArgs e)

{

if (!IsPostBack)

{

\_stats = new PopulationStatistics();

}

}

1. Add a private method to the page with a name of BindHighestMalePopulations. Invoke this method from inside the postback check of Page\_Load, immediately after creating the PopulationStatistics.
2. Inside BindHighestMalePopulations, write a LINQ query (using comprehension query syntax) that will return objects from the \_stats Male property for the year 2002. Order the objects by the value of the Population property in descending order. Here is a starter for the query:

var query =

from ... in \_stats.Males

where ...

orderby ...

select ...;

1. If you need help writing the query, refer to the **Answers** section at the end of this module, or to the **after** website.
2. Once you’ve completed the query, assign the query to the DataSource property of the \_malePopulation GridView, then invoke DataBind on the grid. Since we only want the highest male populations, we can use the *Take* operator on the query, as shown below.

private void BindHighestMalePopulations()

{

var query =

...

\_malePopulation.DataSource = query.Take(10);

\_malePopulation.DataBind();

}

1. Run the project, and make sure the output is correct. You should see China at the top of the list, with Mexico at the bottom.

Notice the Population column is not formatted as a number because we are letting the GridView autogenerate our columns. We could solve this issue by specifying our columns and formatting, but in this lab we will concentrate on LINQ.

1. Rewrite your query using extension methods and lambda expressions. The query will look like the following (refer to the **Answers** section if you need the complete query).

var query = \_stats.Males

.Where(...)

.OrderByDescending(...);

Notice we don’t need to use the Select method, because we are not transforming or projecting a new sequence of objects.

# Part III – Joins

Add a new private method named BindLowestTotalPopulations your page class.

1. Invoke the new method inside the postback check in Page\_Load.
2. Inside the new method, we need to create a query that will use both Male and Female properties of \_stats to find the total population of all countries *in the year 2002*. The query should show us the following results:

Country

Male Population

Female Population

Total Population

1. Use the following points as a guide in creating the query:

You’ll need to join females and males on their Country and Year properties to create a proper record. For composite joins, you need to create a new anonymous object that includes all the properties to join.

To find the countries with the lowest populations, sort the objects in ascending order.

To achieve the correct output, you’ll need to project a new anonymous types with the properties listed in the previous step.

Use the following as a guide for your query.

var query =

from ... in \_stats.Males

join ... in \_stats.Females

on new { ... } equals

new { ... }

where ...

orderby ...

select new

{

...

};

1. Set the DataSource property of the \_lowestTotalPopulation grid to your query (with Take(10), and bind the grid.

private void BindLowestTotalPopulations()

{

var query =

...

\_lowestTotalPopulation.DataSource = query.Take(10);

\_lowestTotalPopulation.DataBind();

}

1. Run the project and verify the results. You should find that the tiny country of Tuvalu (owner of the popular .tv domain) has the smallest population.
2. Before moving on, let’s make some optimizations to our query. It would make sense to filter Males and Females by year **before** joining. Not only will there be fewer objects for LINQ to join, but then we no longer need an anonymous object for the composite join – we can simply join objects by their Country property.
3. A second optimization is to introduce a new query variable with a *let* clause. Chances are you are adding the male and female population together inside the *orderby* and *select* clauses. Do this only once and assign the total population to a new variable that you can use in the rest of the query.

var query =

from ... in \_stats.Males.Where(...)

join ... in \_stats.Females.Where(...)

on ... equals ...

let totalPopulation = ...

orderby totalPopulation ascending

select new

{

...

Total = totalPopulation

};

1. Run the project again and ensure you are seeing the expected results.
2. Finally, convert your query to an extension method / lambda query. The Join operator will require a “selector” function. It is the selector function where you can project the new anonymous type that will ultimately bind to the grid. The selector function will take two parameters (to represent the inner and outer objects).

var query =

\_stats.Males.Where(...)

.Join(\_stats.Females.Where(...),

/\* male outer key \*/,

/\* female inner key \*/,

/\* selector \*/)

.OrderBy(...);

1. Run the project one more time to ensure the query is still working.

# Part IV – Correlated Subqueries

Create a new private method with a name of BindFemalePopulationIncrease.

1. Invoke the method inside the !IsPostBack check in Page\_Load.
2. This query needs to find the countries with the largest increases in female population. To do this we will need to correlate each country’s Female population entry with the same country’s entry from the previous year. We can do this using a second *from* clause that uses a subquery to find the correct record. We need to return the following data:

Country

Year

Current female population

Change from previous year

1. The query will look like the following. Don’t forget to correlate the subquery entry with the outer query entry’s Country and Year -1 values.

var query =

from ... in \_stats.Females

from ... in (from ... in \_stats.Females

where ...

select ...)

let delta = ...

orderby ... descending

select new

{

...

Change = delta

};

1. Bind this query to the DataSource property of the \_percentIncrease grid. Make sure to take only the top 10 records, then invoke DataBind.
2. Run the project and check your results. You should find Nigeria’s 2006 entry at the top of the list. The rest of the list will include only China and India.
3. Rewrite your query using a lambda syntax. Remember that a second from clause translates into the SelectMany operator in LINQ.

var query =

\_stats.Females

.SelectMany(... => \_stats.Females

.Where(...)

.Select(...))

.OrderByDescending(...);

# Conclusion

Congratulations! You’ve written 6 tough queries that should give you a solid grounding not only in LINQ techniques like joins and subqueries, but in how the C# compiler can transform declarative queries into a series of method calls.

# Answers

# Part I - Highest Male Populations

### Comprehension Syntax

var query =

from m in \_stats.Males

where m.Year == 2002

orderby m.Population descending

select m;

### Lambda Syntax

var query = \_stats.Males

.Where(m => m.Year == 2002)

.OrderByDescending(m => m.Population);

# Part II - Lowest Total Population

### Comprehension Syntax (unoptimized)

var query =

from m in \_stats.Males

join f in \_stats.Females

on new { m.Country, m.Year } equals

new { f.Country, f.Year }

where m.Year == 2002

orderby m.Population + f.Population ascending

select new

{

Country = m.Country,

Males = m.Population,

Females = f.Population,

Total = m.Population + f.Population

};

### Comprehension Syntax (optimized)

var query =

from m in \_stats.Males.Where( m => m.Year == 2002)

join f in \_stats.Females.Where(f => f.Year == 2002)

on m.Country equals f.Country

let totalPopulation = m.Population + f.Population

orderby totalPopulation ascending

select new

{

Country = m.Country,

Males = m.Population,

Females = f.Population,

Total = totalPopulation

};

### Lambda Syntax

var query =

\_stats.Males

.Where(m => m.Year == 2002)

.Join(\_stats.Females.Where(f => f.Year == 2002),

m => m.Country,

f => f.Country,

(m, f) => new { Country = m.Country,

Males = m.Population,

Females = f.Population,

TotalPopulation =

m.Population +

f.Population})

.OrderBy(g => g.TotalPopulation);

# Part III – Female Population Increase

### Comprehension Syntax

var query =

from f in \_stats.Females

from p in (from p in \_stats.Females

where p.Country == f.Country &&

p.Year == f.Year -1

select p)

let delta = f.Population - p.Population

orderby delta descending

select new

{

f.Country,

f.Year,

f.Population,

Change = delta

};

### Lambda Syntax

var query =

\_stats.Females

.SelectMany(

f => \_stats.Females

.Where(p => p.Country ==

f.Country &&

p.Year ==

f.Year - 1)

.Select(p =>

new {

f.Country,

f.Year,

f.Population,

Change = f.Population -

p.Population

}))

.OrderByDescending(f => f.Change);