1.2.0 – LINQ Basic

Introduction

LINQ stands for **L**anguage **In**tegrated **Q**uery. This is a powerful feature of the C# language that allows SQL-Like querying of data, including data stored in a database. LINQ statements can be written in two ways: (1) Query syntax, or (2) Method syntax. These ways are not in conflict either; you can combine Query syntax and Method syntax in a single statement.

Query Syntax

Query syntax appears, at first, to look like SQL. LINQ queries use keywords that closely resemble SQL keywords, but LINQ's keywords act as operators in that they are used to form an expression. LINQ query operators perform specific processing acts on the objects as their "operands". They flow together in a kind of "sequence" and end with the select.

Query syntax includes seven major clauses (five of which are the more commonly used):

- **from** clause: A query expression ***must*** begin with a **from** clause. Additionally, a query expression can contain sub-queries, which also ***must*** begin with a **from** clause
- **where** clause: The **where** clause is used in a query expression to specify which elements from the data source will be returned in the query expression
- **select** clause: In a query expression, the **select** clause specifies the type of values that will be produced when the query is executed. The result is based on the evaluation of all previous clauses and on any expressions in the **select** clause itself. A query expression must end with either a **select** clause or a **group** clause.
- **group** clause: The **group** clause returns a sequence of `IGrouping<TKey, TElement>` objects that contain zero or more items that match the key value for the group.
- **orderby** clause: In a query expression, the **orderby** clause causes the returned sequence or subsequence (**group**) to be sorted in either ascending or descending order. Multiple keys can be specified to perform one or more secondary sort operations.
- **join** clause: The **join** clause is useful for associating elements from different source sequences that have no direct relationship in the object model.
- **let** clause: In a query expression, it is sometimes useful to store the result of a sub-expression to use it in subsequent clauses. You do this with the **let** keyword.

Example

An example of the flow seen in a typical LINQ query can be illustrated in the following (over-simplified) grammar:

Listing 1: Simplified LINQ Query Structure

```
from [type-name] identified in enumerable-expression
[orderby expression], ..n [{ascending / descending}]
[where boolean-expression]
[group group-object by group-obj-property [into group-identifier]]
select expression
```

LINQPad – A Scratch-pad for LINQ

LINQPad (http://LINQPad.net) is a stand-alone application that you cause in place of creating a sample console application to test and/or explore LINQ. The LINQPad editor window allows you to select from three main contexts:

- **Expressions**: In this context, you simply have to generate an expression in the editor and run the expression. LINQPad will show the output in the Results window.
- **Statements**: Imagine lines of code in a method, and you have the Statements context. Here you write the complete program statements - declaring variables, making calculations, performing queries, writing if or flow-control statements, etc. - and LINQPad executes these like it was the body of some method call. If you want to output anything to the results window, simply call the `.Dump()` extension method that LINQPad includes for all objects.
- **Program**: This context most closely represents a console application. You are given a Main() method and you can add additional methods and/or classes to build a complete console-like application. In this context, however, instead of calling `Console.WriteLine(…)`, you continue to use the `.Dump()` method to output content to the Results window.

LINQPad allows you to connect to a database. Clicking "Add Connection…" opens the "Choose Data Context" dialog. Click "Next" (leaving the default) move you into the "LINQPad Connection" dialog where you specify your server name and your existing database. From there, you can see the database and its tables and columns in the explorer pane on the left.

Setup LINQPad Database Connection

LINQPad is already installed on the Lab computers. The only setup that is needed is to create a connection to a database. When starting LINQPad for the first time no database connections are listed.

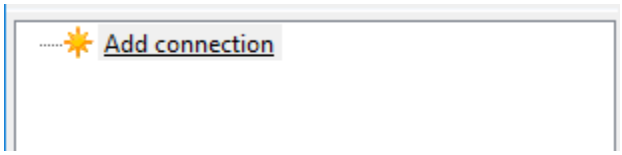


Figure 1: LINQPad – No Database Connections

To create a database connection, click on the Add connection link. The first thing that happens is shown in the figure below:

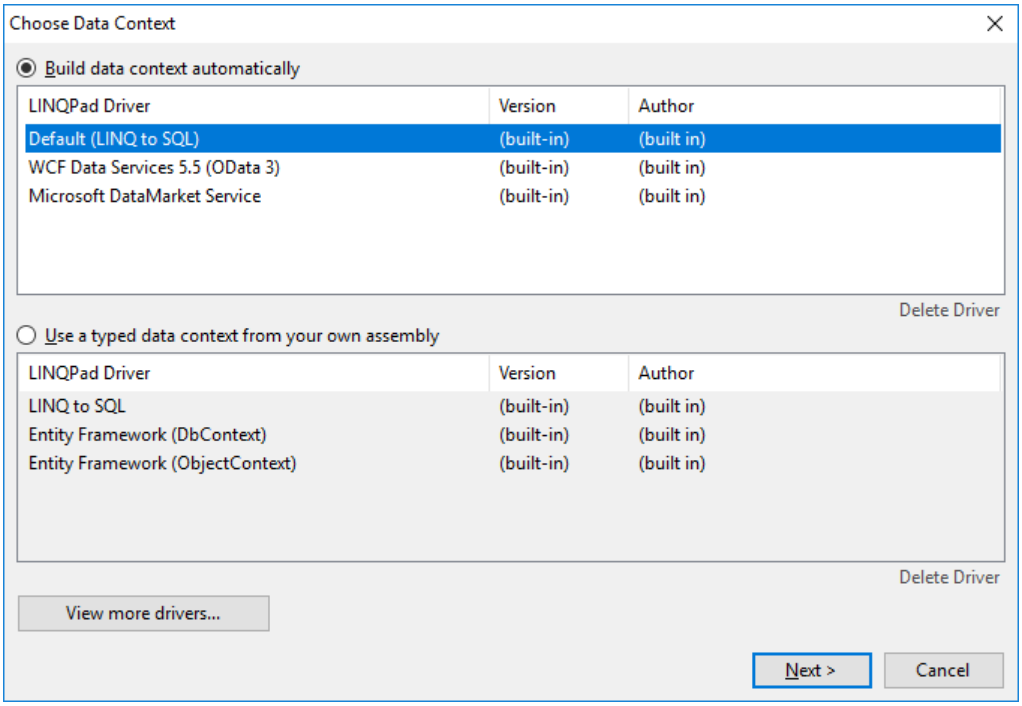


Figure 2: LINQPad - Add Database Connection (1)

Press **Next >** as you will be using the default LINQ to SQL context.

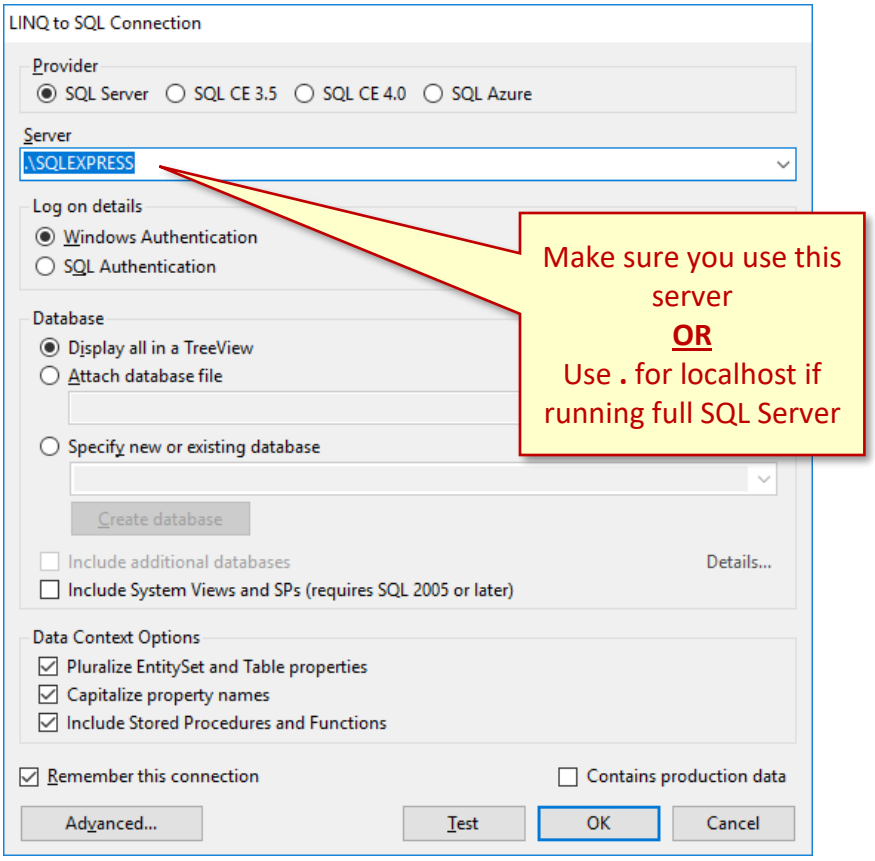


Figure 3: LINQPad - Add Database Connection (2)

The next step is to select the database you want to connect to. Use the **Specify new or existing database** radio button. It will take a few seconds for the list of databases to populate with the names of the databases. Once the list is populated, select the database you want to connect to.

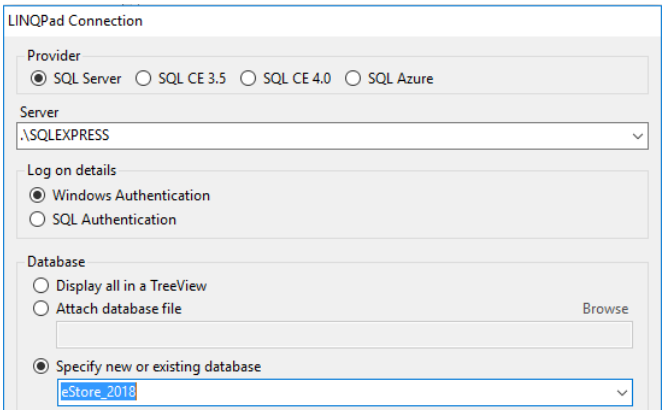


Figure 4: LINQPad - Add Database Connection (3)

Once the database is selected, press OK. The add connection wizard will close and LINQPad will show the new database connection, and should display the tables of the database.

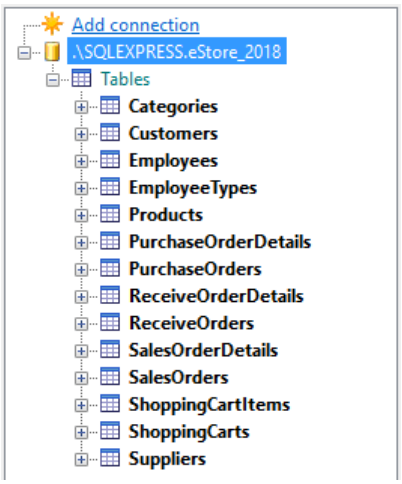


Figure 5: LINQPad - Add Database Connection (4)

Testing the Database

Before we go into creating LINQ Queries, it is best to explore the database.

Categories

Expand the Categories to see:

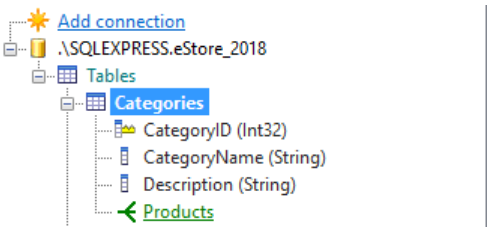
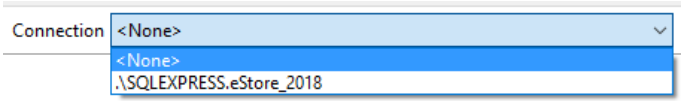


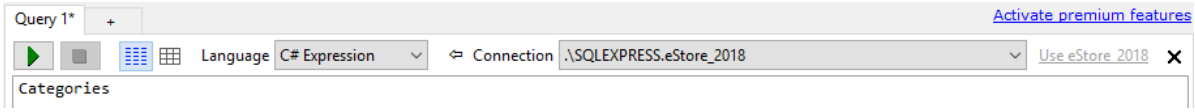
Figure 6: Categories Entity

Here we can see the fields of the **Category** table, and can identify the **Primary Key** (CategoryID). Next, we can examine the data in this Entity by following these steps:

1. In the Query 1 window type **Categories**
2. We need a database connection for this to work, thus you select from the Connection dropdown list:



   OR select the Use eStore 2018 link.
3. When this is done, you should see:



4. Press the green triangle,  , to execute your basic query.

5. The results should look like:

Results λ SQL IL

Table<Category> (11 items)

CategoryID	CategoryName	Description	Products
1	Fruit	Various fruits in season	Products
2	Vegetables	Various vegetables in season	Products
3	Drinks	Non-alcoholic drinks	Products
4	Alcohol	Selected beer and wines	Products
5	Meat	Selection of meat and meat products	Products
6	Bread	Various bread and grain products	Products
7	Candy	Assorted sweet treats	Products
8	Cooking	Spices, sauces, and cooking needs	Products
9	Cleaners	Various cleaning products	Products
10	Snacks	Seelction of non-sweet snacks	Products
11	Housewares	Pots, pans, dishes, knives and eating utensils	Products

Figure 7: Categories Query Results

You should notice the Products link in the right-hand column. If you select, click, on one of the links, you should see something like:

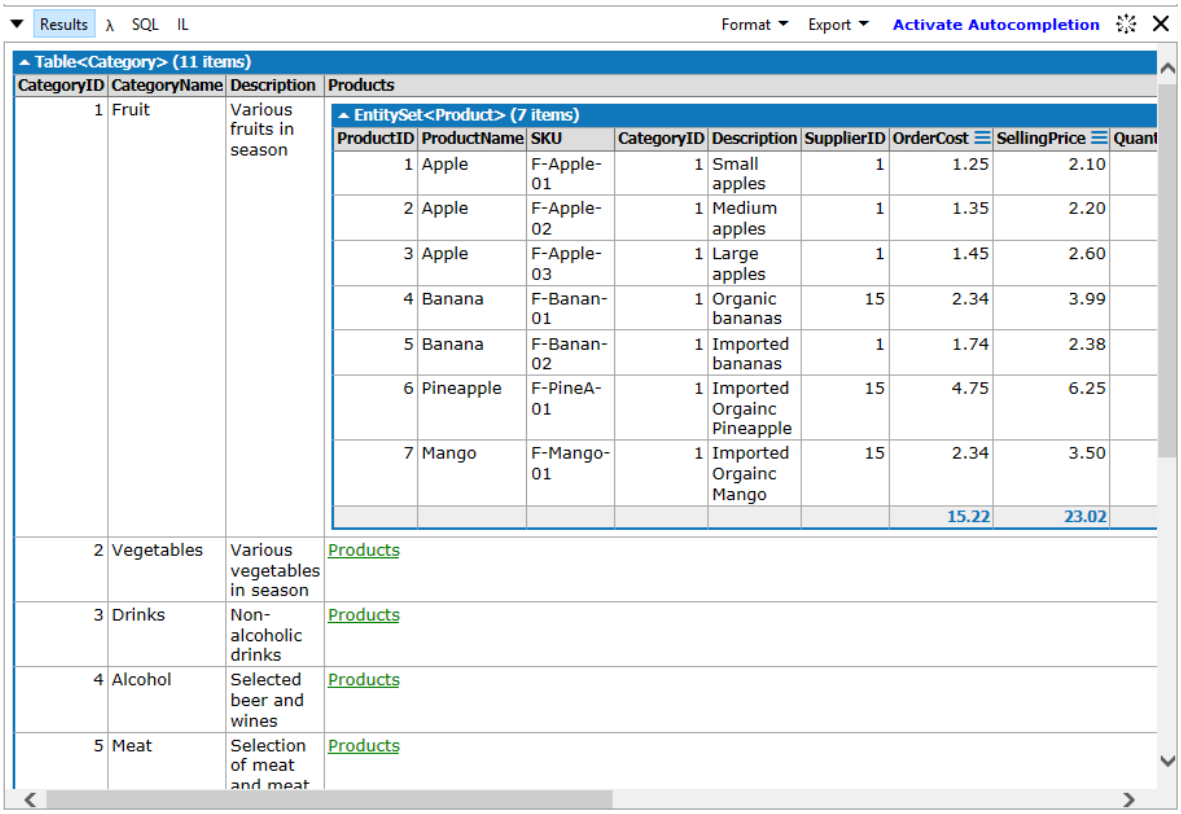


Figure 8: Products Link in Categories

This is a navigation property, lists all the **Products** in the selected **Category**, which we will learn about next.

This technique is useful to see the data, and how the table entities are related.

Navigation Properties

When a database connection is made in LINQPad, LINQPad automatically adds navigation properties for the Primary Key to Foreign Key relationships. If we examine the **Products** entity in LINQPad we see the following:

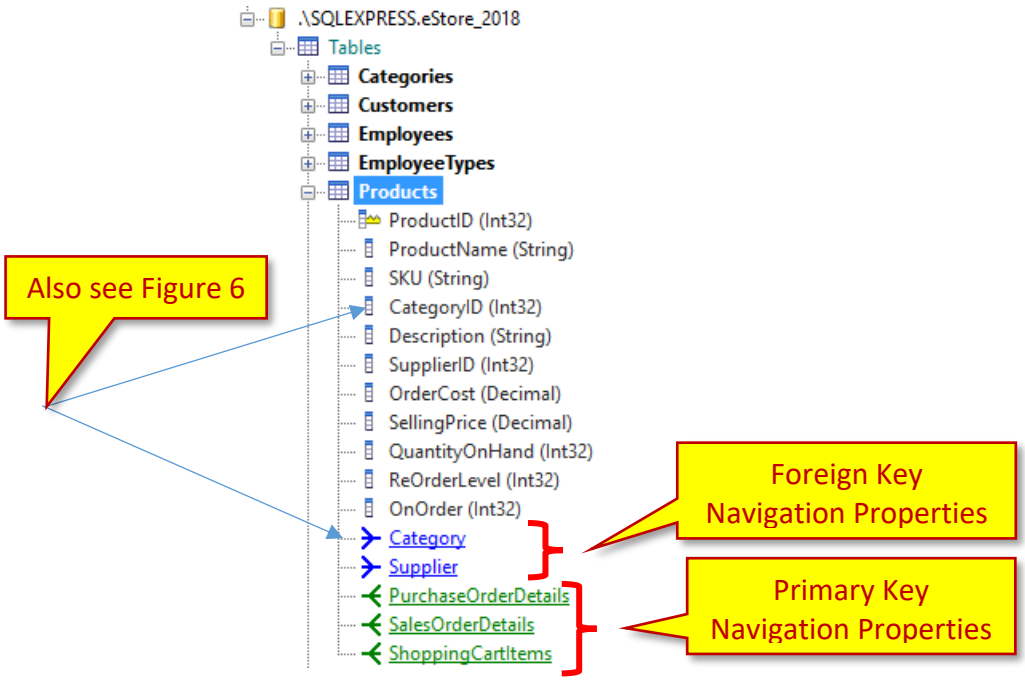


Figure 9: Products Entity in LINQPad

Foreign Key Navigation Property

These properties represent that the Entity belongs to only 1 parent entity:

- A Product belongs to only 1 Category
- A Product belongs to only 1 Supplier

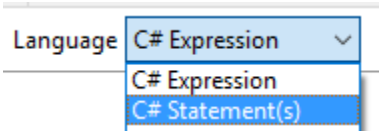
Primary Key Navigation Property

These properties represent that the Entity has many child entities:

- A Product can be on many PurchaseOrderDetail
- A Product can be on many SalesOrderDetail
- A Product can be on many ShoppingCartItem

LINQ Examples

The following examples use the eStore database (the connection was created above). The query below uses the C# Statements from the Language dropdown:



List all the Products sorted by Name of a Category:

Listing 2: All the Products of a Selected Category

```
var results = from x in Products
              where x.CategoryID == 1
              orderby x.ProductName
              select new
              {
                  ID = x.ProductID,
                  Name = x.ProductName,
                  SKU = x.SKU,
                  Description = x.Description,
                  Cost = x.OrderCost,
                  Price = x.SellingPrice,
                  QOH = x.QuantityOnHand,
                  ROL = x.ReOrderLevel,
                  Ordered = x.OnOrder
              };
results.Dump();
```

Select a known CategoryID (Foreign Key) of the Products Entity

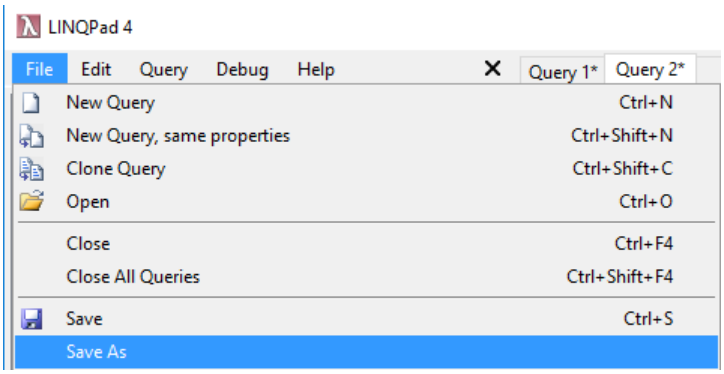
Results λ SQL IL Format Ex

IOrderedQueryable<> (7 items)

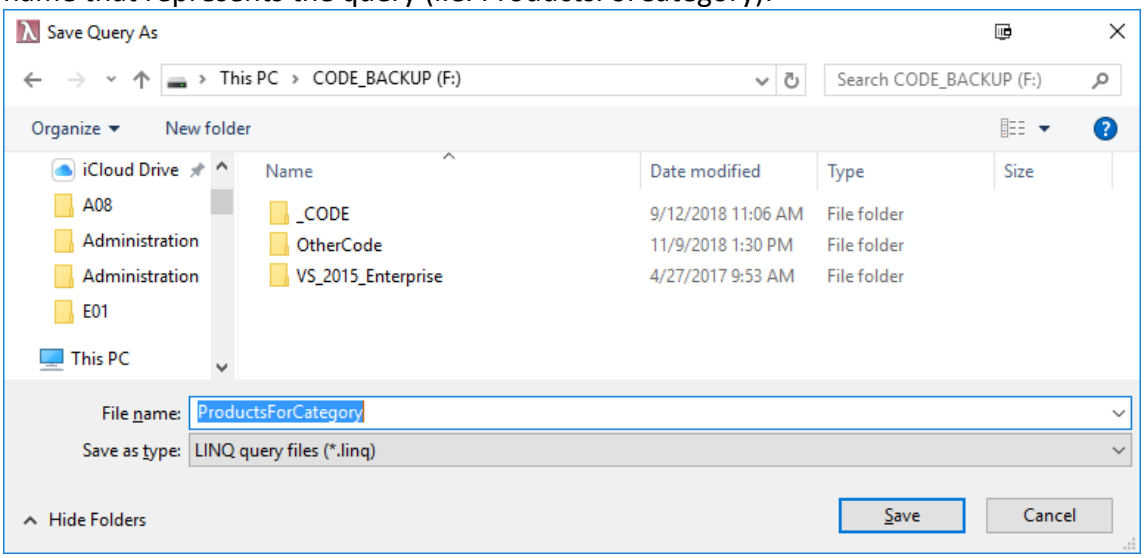
ID	Name	SKU	Description	Cost	Price	QOH	ROL	Ordered
1	Apple	F-Apple-01	Small apples	1.25	2.10	150	55	0
2	Apple	F-Apple-02	Medium apples	1.35	2.20	140	45	0
3	Apple	F-Apple-03	Large apples	1.45	2.60	130	40	0
4	Banana	F-Banan-01	Organic bananas	2.34	3.99	40	20	0
5	Banana	F-Banan-02	Imported bananas	1.74	2.38	67	70	0
7	Mango	F-Mango-01	Imported Orgainc Mango	2.34	3.50	40	35	0
6	Pineapple	F-PineA-01	Imported Orgainc Pineapple	4.75	6.25	50	60	20
				15.22	23.02	617	325	20

Figure 10: Listing 2 Results

Once you have a query that displays the correct/desired results, you should save it so it can be used later.

1. 

2. Pick a location on your computer, preferably to a USB storage device, and give the file a name that represents the query (i.e. ProductsForCategory):



The next query uses a Navigation Property, to look at the *parent* entity to read data from that entity:

List all the Products sorted by Name:

Listing 3: All Products Sorted by Name

```
var results = from x in Products
              orderby x.ProductName
              select new
              {
                  ID = x.ProductID,
                  Name = x.ProductName,
                  Category = x.Category.CategoryName,
                  Price = x.SellingPrice
              };
results.Dump();
```

Uses a navigation property

IOrderedQueryable<> (77 items)

ID	Name	Category	Price
1	Apple	Fruit	2.10
2	Apple	Fruit	2.20
3	Apple	Fruit	2.60
20	Apple Juice	Drinks	3.25
4	Banana	Fruit	3.99
5	Banana	Fruit	2.38
64	BBQ Chips	Snacks	2.45
29	Beef Sausage	Meat	6.25
30	Beef Sausage	Meat	4.65
62	Bleach	Cleaners	2.45
38	Brown Bread	Bread	2.25
39	Brown Bread	Bread	3.10
27	Budweiser	Alcohol	4.95
13	Cabbage	Vegetables	3.25

Figure 11: Listing 3 Results

This next query lists all the *child* entities of all the *parent* entities.

Products for Category

Listing 4: Products for Category

```
var results = from x in Categories
              where x.CategoryID == 1
              select new
              {
                  Name = x.CategoryName,
                  Description = x.Description,
                  Products = from y in x.Products
                             orderby y.ProductName
                             select new
                             {
                                 Name = y.ProductName,
                                 SKU = y.SKU,
                                 Supplier = y.Supplier.SupplierName,
                                 Cost = y.OrderCost,
                                 Price = y.SellingPrice,
                                 QOH = y.QuantityOnHand
                             }
               };
results.Dump();
```

Uses a navigation property

Uses a navigation property

IOrderedQueryable<> (1 item)

Name	Description	Products
Fruit	Various fruits in season	(7 items)

Name	SKU	Supplier	Cost	Price	QOH
Apple	F-Apple-01	Best Western Fruit	1.25	2.10	150
Apple	F-Apple-02	Best Western Fruit	1.35	2.20	140
Apple	F-Apple-03	Best Western Fruit	1.45	2.60	130
Banana	F-Banan-01	OK Growers	2.34	3.99	40
Banana	F-Banan-02	Best Western Fruit	1.74	2.38	67
Mango	F-Mango-01	OK Growers	2.34	3.50	40
Pineapple	F-PineA-01	OK Growers	4.75	6.25	50
			15.22	23.02	617

Figure 12: Products for Category

The query below is an example of how to join string fields to display a single item of data, and shows how to count rows.

Using the Count Function

Listing 5: Count Shopping Cart Items

```
var results = from x in ShoppingCarts
              select new
              {
                  Customer = x.Customer.FirstName + " " + x.Customer.LastName,
                  Created = x.CreatedOn,
                  Updated = x.UpdatedOn,
                  Items = x.ShoppingCartItems.Count
              };
results.Dump();
```

Joining Strings

Explicit Count()

▲ IOrderedQueryable<> (4 items)

Customer	Created	Updated	Items
Trista Gandy	4/1/2017 2:30:23 PM	4/1/2017 2:35:55 PM	3
Anthony Hazlett	4/1/2017 2:43:30 PM	4/1/2017 2:45:21 PM	3
Wei Miao	4/1/2017 6:30:23 PM	4/1/2017 6:15:25 PM	3
Paige Bain	4/2/2017 9:34:23 AM	4/2/2017 9:44:50 AM	4
			13

Figure 13: Listing 5 Results

The query below shows how to determine if you can query the data, and an alternate sorting method.

Listing 6: Ordered Products

```
var results = from x in Products
              where x.OnOrder > 0
              orderby x.OnOrder descending
              select new
              {
                   ID = x.ProductID,
                   SKU = x.SKU,
                   Name = x.ProductName,
                   QOH = x.QuantityOnHand,
                   ROL = x.ReOrderLevel,
                   Ordered = x.OnOrder
               };
results.Dump();
```

Implicit Count()

Alternate sorting

IOrderedQueryable<> (15 items)

ID	SKU	Name	QOH	ROL	Ordered
35	M-Wings-01	Chicken Wings	40	75	100
43	S-JlyBn-01	Jelly Beans	50	75	100
51	K-CnOil-02	Canola Oil	45	60	100
70	J-Cheez-01	Cheezies	60	50	100
55	K-MnSdG-01	MSG	35	50	75
41	B-DBuns-01	Hot Dog Buns	15	30	50
66	J-RitzC-01	Ritz Crackers	40	50	25
73	H-ChpSt-01	Chopsticks	50	50	25
52	K-TSalt-01	Salt	60	50	25
50	K-CnOil-01	Canola Oil	50	50	25
6	F-PineA-01	Pineapple	50	60	20
12	V-Carrt-03	Carrot	50	60	20
34	M-CkBgr-02	Chicken Burger	40	50	20
60	C-WindX-01	Windex	20	20	20
39	B-Brown-02	Brown Bread	20	30	10
			625	760	715

Figure 14: Listing 6 Results

Exercise

Complete Exercise **1.1.1 LINQ Queries Basic** and upload your queries to Moodle.