**Sorting**

Ordering Operators:

* OrderBy
* OrderByDescending
* Thenby
* ThenByDescending
* Reverse. Reverses the order

KeySelector – field to use as the key for sorting.

ThenBy – use for multiple sort criteria.

ThenByDescending

If you sort by a property that can have null values then the null values will be sent to the top of the list.

Int? Means its a nullable type.

You can deselect null values by using .HasValue. HasValue returns true or false depending on if it has a null value.

**Creating: Repeat and Range**

generating operators

* Range. Generates values within a range of values
* Repeat. Generates a value repeated.

The above are NOT extension methods and NOT deferred.

Range and Repeat are static method of Enumerable.

Y using a Select statement with the Range function we can manipulate the sequence to be any arithmetic range. E,g.

Enumerable.Range( 0, 50 ).Select( x => x\*5 ); //multiples of 5

Repeat can repeat any type of object.

You can generate any number of objects. Initialising a set of empty objects is good for repeat.

Comparin/Combining

Set operators:

* Intersect. Identifies the intersecting element from 2 sequences.
* Except. Defines the elements in one sequence that aren’t in another
* Concat. Produces a sequence that’s the result of combining 2 sequences
* Distinct. Identifies distinct items
* Union. As above but single command

All the above are extension methods of the IEnumerable interface.

**Transforming with Projections**

* Select
* SelectMany

Projection transforms an object into a new form. In LINQ projection is achieved using the above 2 methods. The projection is performed by using a projection function as an argument for Select. It is often a lambda expression.

No Select with a Where will return a sequence with no projection, just the the original format objects or values.

SelectMany projects multiple sequences based on a transform function and flattens them into one sequence.

Why Project?

* Limit the results to the relevant properties
* Perform an action on a property. Combine, Manipulate etc.
* Morph the elements into another type

The projection can be an anonymous type.

Anonymous type is a type without a name but is new-ed up. It’s a way of creating a type that you’ll only use locally, on the fly.

C => new{ Name = c.Name, Age = c.Age }

The above is an example of an anonymous type being created by a lambda expression.

How do we return IEnumerable of anonymous type? We can’t. Instead we can use dynamic return type.

Dynamic allows it to bypass compile time checking and will allow to return an anonymous type. It’s not a recommended technique because even though the code will compile and return the result, there’s no way to deal with the result at the calling side.

The recommended technique is to use anonymous types locally, where they are needed.

DO NOT RETURN ANONYMOUS TYPE. They’re great for using on the fly in processing and binding scenarios.

JSON?

* Join LINQ joins 2 sequences. The inside list is the calling sequence, the outside list is the argument of the LINQ method. Requires keySelectors for each list that will match the columns and a Select function.

If a lambda has 2 params, it needs parenthesis around them:

(x, y) => x.Name, y.Name

**Parent/Child or Master/Detail data**

In this type of data each parent/main object has a collection of Chile/Detail data.

**Null coalescing operator**. If the value is null it’ll return false:

bool obj = I.IsPaid ?? false;

If IsPaid is null, false is returned.

There are issues using the Select operator when working with child data.

* It doesn’t provide access to the parent.
* Dealing with IEnumerable of IENumerable requires 2 loops.

Instead use SelectMany when working with Parent/child data.

SelectMany projects each element of a sequence and then flattens the resulting sequence. So instead of an IEnumerable of IENumerable it is a single IEnumerable.

3rd overload of SelectMany allows a ResultFunction selector. It allows us to shape the returning values if we want to return the parent rather than the child sequence. The function will be a lambda with 2 params; the parent and the child.

Use SelectMany when working with parent child data.