using( var ne = new NorthwindEntities())

{

var customers =

ne.Customers.Where(c =>

c.CompanyName.StartsWith("a"));

foreach( var c in customers )

{

Console.WriteLine(c.CompanyName);

}

}

**Parallel LINQ**

const int count = 50;

var items = Enumerable.Range(1, count);

var results = new int[count];

items.AsParallel().ForAll(x =>

{

int newValue = x \* x \* x;

Console.WriteLine("{0} {1}", newValue, Task.CurrentId);

results[x - 1] = newValue;

});

--------------------------------------

Task.WaitAll();

foreach( var i in results )

{

Console.WriteLine(i);

}

-------------------------------------

var cubes = items.AsParallel().AsOrdered().Select(x => x\*x\*x);

foreach( var i in cubes )

{

Console.WriteLine(i);

}

**Exceptions and Cancellation**

var items2 = ParallelEnumerable.Range(1, 20);

var results2 = items.Select(i =>

{

double result = Math.Log10(i);

Console.WriteLine(result);

return result;

});

try

{

foreach( var c in results )

{

Console.WriteLine("result=" + c);

}

}

catch(AggregateException ex)

{

ex.Handle(e =>

{

Console.WriteLine(e.Message.ToString());

return true;

});

}

--------------------------------

**Add**

CancellationTokenSource cts = new CancellationTokenSource();

**Change**

var results2 = items2.WithCancellation(cts.Token).Select(i =>

**Can now use**

Cts.Cancel();

**Merge Options**

var numbers = Enumerable.Range(1, 20).ToArray();

var results = numbers.AsParallel().Select(x =>

{

var result = Math.Log10(x);

Console.WriteLine("Produced:"+result);

return result;

});

foreach( var v in results )

{

Console.WriteLine("Consumed:"+v);

}

**Can add WithMergeOptions**

|  |  |  |
| --- | --- | --- |
|  | **Member name** | **Description** |
|  | AutoBuffered | Use a merge with output buffers of a size chosen by the system. Results will accumulate into an output buffer before they are available to the consumer of the query. |
|  | Default | Use the default merge type, which is AutoBuffered. |
|  | FullyBuffered | Use a merge with full output buffers. The system will accumulate all of the results before making any of them available to the consumer of the query. |
|  | NotBuffered | Use a merge without output buffers. As soon as result elements have been computed, make that element available to the consumer of the query. |

**Aggregation**

var sum = Enumerable.Range(1, 1000).Sum();

Console.WriteLine(sum);

var sum2 = Enumerable.Range(1, 1000).Aggregate(0, (i, j) => i + j);

Console.WriteLine(sum2);

var sum3 = ParallelEnumerable.Range(1, 1000).Aggregate(0, (partialSum, q) => partialSum += q, i => i);

Console.WriteLine(sum3);