LINQ Standard Query Operators

Restriction Operators

Where	Enumerates the source sequence and yields those elements for which the predicate function returns **true**.

Projection Operators

Select	Enumerates the source sequence and yields the results of evaluating the selector function for each element.
SelectMany	Performs a one-to-many element projection over a sequence.

```
IEnumerable<Order> orders = customers
    .Where(c => c.Country == "Denmark")
    .SelectMany(c => c.Orders);
```

Partitioning Operators

Skip	Skips a given number of elements from a sequence and then yields the remainder of the sequence.
SkipWhile	Skips elements from a sequence while a test is **true** and then yields the remainder of the sequence. Once the predicate function returns **false** for an element, that element and the remaining elements are yielded with no further invocations of the predicate function.
Take	Yields a given number of elements from a sequence and then skips the remainder of the sequence.
TakeWhile	Yields elements from a sequence while a test is **true** and then skips the remainder of the sequence. Stops when the predicate function returns false or the end of the source sequence is reached.

```
IEnumerable<Product> MostExpensive10 =
    products.OrderByDescending(p => p.UnitPrice).Take(10);
```

Join Operators

Join	Performs an inner join of two sequences based on matching keys extracted from the elements.
GroupJoin	Performs a grouped join of two sequences based on matching keys extracted from the elements.

```
var custOrders = customers
    .Join(orders, c => c.CustomerID, o => o.CustomerID,
        (c, o) => new { c.Name, o.OrderDate, o.Total } );

var custTotalOrders = customers
    .GroupJoin(orders, c => c.CustomerID, o => o.CustomerID,
        (c, co) => new { c.Name, TotalOrders = co.Sum(o => o.Total) } );
```

Concatenation Operators

Concat	Enumerates the first sequence, yielding each element, and then it enumerates the second sequence, yielding each element.

Ordering Operators

OrderBy, OrderByDescending, ThenBy, ThenByDescending	Make up a family of operators that can be composed to order a sequence by multiple keys.
Reverse	Reverses the elements of a sequence.

```
IEnumerable<Product> orderedProducts = products
    .OrderBy(p => p.Category)
    .ThenByDescending(p => p.UnitPrice)
    .ThenBy(p => p.Name);
```

Grouping Operators

GroupBy	Groups the elements of a sequence.

```
IEnumerable<IGrouping<string, Product>> productsByCategory = products
    .GroupBy(p => p.Category);
```

Set Operators

Distinct	Eliminates duplicate elements from a sequence.
Except	Enumerates the first sequence, collecting all distinct elements; then enumerates the second sequence, removing elements contained in the first sequence.
Intersect	Enumerates the first sequence, collecting all distinct elements; then enumerates the second sequence, yielding elements that occur in both sequences.
Union	Produces the set union of two sequences.

```
IEnumerable<string> productCategories =
    products.Select(p => p.Category).Distinct();
```

Conversion Operators

AsEnumerable	Returns its argument typed as `IEnumerable<T>`.
Cast	Casts the elements of a sequence to a given type.
OfType	Filters the elements of a sequence based on a type.
ToArray	Creates an array from a sequence.
ToDictionary	Creates a `Dictionary<TKey,TElement>` from a sequence (one-to-one).

ToList	Creates a `List<T>` from a sequence.
ToLookup	Creates a `Lookup<TKey, TElement>` from a sequence (one-to-many).

```
string[] customerCountries = customers
    .Select(c => c.Country).Distinct().ToArray();

List<Customer> customersWithOrdersIn2005 = customers
    .Where(c => c.Orders.Any(o => o.OrderDate.Year == 2005)).ToList();

Dictionary<string,decimal> categoryMaxPrice = products
    .GroupBy(p => p.Category)
    .ToDictionary(g => g.Key, g => g.Max(p => p.UnitPrice));

ILookup<string,Product> productsByCategory = products
    .ToLookup(p => p.Category);
IEnumerable<Product> beverages = productsByCategory["Beverage"];

List<Person> persons = GetListOfPersons();
IEnumerable<Employee> employees = persons.OfType<Employee>();

ArrayList objects = GetOrders();
IEnumerable<Order> ordersIn2005 = objects
    .Cast<Order>()
    .Where(o => o.OrderDate.Year == 2005);
```

Equality Operators

SequenceEqual	Checks whether two sequences are equal by enumerating the two source sequences in parallel and comparing corresponding elements.

Element Operators

DefaultIfEmpty	Supplies a default element for an empty sequence. Can be combined with a grouping join to produce a left outer join.[1]
ElementAt	Returns the element at a given index in a sequence.
ElementAtOrDefault	Returns the element at a given index in a sequence, or a default value if the index is out of range.[1]
First	Returns the first element of a sequence.[2]
FirstOrDefault	Returns the first element of a sequence, or a default value if no element is found.[1]
Last	Returns the last element of a sequence.[2]
LastOrDefault	Returns the last element of a sequence, or a default value if no element is found.[1]
Single	Returns the single element of a sequence. An exception is thrown if the source sequence contains no match or more than one match.
SingleOrDefault	Returns the single element of a sequence, or a default value if no element is found.[1]

[1] *The default value for reference and nullable types is null.*

[2] *Throws an exception if no element matches the predicate or if the source sequence is empty.*

```
Customer customer = customers.First(c => c.Phone == "111-222-3333");
Customer customer = customers.Single(c => c.CustomerID == 1234);
```

Generation Operators

Empty	Returns an empty sequence of a given type.
Range	Generates a sequence of integral numbers.
Repeat	Generates a sequence by repeating a value a given number of times.

```
int[] squares = Enumerable.Range(0, 100).Select(x => x * x).ToArray();
long[]allOnes = Enumerable.Repeat(-1L, 256).ToArray();
IEnumerable<Customer> noCustomers = Enumerable.Empty<Customer>();
```

Quantifiers

Any	Checks whether any element of a sequence satisfies a condition. If no predicate function is specified, simply returns **true** if the source sequence contains any elements. Enumeration of the source sequence is terminated as soon as the result is known.
All	Checks whether all elements of a sequence satisfy a condition. Returns **true** for an empty sequence.
Contains	Checks whether a sequence contains a given element.

```
bool b = products.Any(p => p.UnitPrice >= 100 && p.UnitsInStock == 0);
IEnumerable<string> fullyStockedCategories = products
    .GroupBy(p => p.Category)
    .Where(g => g.All(p => p.UnitsInStock > 0))
    .Select(g => g.Key);
```

Aggregate Operators

Aggregate	Applies a function over a sequence.
Average	Computes the average of a sequence of numeric values.
Count LongCount	Counts the number of elements in a sequence.
Max	Finds the maximum of a sequence of numeric values.
Min	Finds the minimum of a sequence of numeric values.
Sum	Computes the sum of a sequence of numeric values.

```
int count = customers.Count(c => c.City == "London");
```