**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.

```csharp
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.

```csharp
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.

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

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

### Grouping Operators

**GroupBy**
Groups the elements of a sequence.

```csharp
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.

```csharp
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.

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

**ToLookup**

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

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

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

  - `ToList()`
  - `ToDictionary()`
  - `ToLookup()`

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

- **ILookup<string,Product> productsByCategory = products**
  
  ```csharp
  ToLookup(p => p.Category);
  ```

  - `ToLookup()`
  - `ToDictionary()`

- **List<Person> persons = GetListOfPersons();**
  
  ```csharp
  IEnumerable<Employee> employees = persons.OfType<Employee>();
  ```

- **ArrayList objects = GetOrders();**
  
  ```csharp
  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.

  ```csharp
  bool b = products.Any(p => p.UnitPrice >= 100 && p.UnitsInStock == 0);
  ```

- **ElementAt**
  Returns the element at a given index in a sequence.

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

- **ElementAtOrDefault**
  Returns the element at a given index in a sequence, or a default value if the index is out of range.

- **First**
  Returns the first element of a sequence.

- **FirstOrDefault**
  Returns the first element of a sequence, or a default value if no element is found.

- **Last**
  Returns the last element of a sequence.

- **LastOrDefault**
  Returns the last element of a sequence, or a default value if no element is found.

- **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.

**Aggregate Operators**

- **Aggregate**
  Applies a function over a sequence.

- **Average**
  Computes the average of a sequence of numeric values.

- **Count**
  Counts the number of elements in a sequence.

- **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.