```
1  // Decompiled with JetBrains decompiler
2  // Type: System.Linq.Queryable
3  // Assembly: System.Linq.Queryable, Version=4.0.3.0, Culture=neutral,
     PublicKeyToken=b03f5f7f11d50a3a
4  // MVID: 7EBF3257-2CBA-49CA-856A-CABF60BBC1AE
5  // Assembly location: C:\Program Files\dotnet\sdk\NuGetFallbackFolder
     \microsoft.netcore.app\2.0.0\ref\netcoreapp2.0\System.Linq.Queryable.dll
6
7  using System.Collections;
8  using System.Collections.Generic;
9  using System.Linq.Expressions;
10
11 namespace System.Linq
12 {
13   /// <summary>Provides a set of static (Shared in Visual Basic) methods for
       querying data structures that implement <see
       cref="T:System.Linq.IQueryable`1"></see>.</summary>
14   public static class Queryable
15   {
16     /// <summary>Applies an accumulator function over a sequence.</summary>
17     /// <param name="source">A sequence to aggregate over.</param>
18     /// <param name="func">An accumulator function to apply to each element.</
         param>
19     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
20     /// <returns>The final accumulator value.</returns>
21     /// <exception cref="T:System.ArgumentNullException"><paramref
         name="source">source</paramref> or <paramref name="func">func</paramref> is
          null.</exception>
22     /// <exception cref="T:System.InvalidOperationException"><paramref
         name="source">source</paramref> contains no elements.</exception>
23     public static TSource Aggregate<TSource>(this IQueryable<TSource> source,
         Expression<Func<TSource, TSource, TSource>> func);
24     /// <summary>Applies an accumulator function over a sequence. The specified
         seed value is used as the initial accumulator value.</summary>
25     /// <param name="source">A sequence to aggregate over.</param>
26     /// <param name="seed">The initial accumulator value.</param>
27     /// <param name="func">An accumulator function to invoke on each element.</
         param>
28     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
29     /// <typeparam name="TAccumulate">The type of the accumulator value.</
         typeparam>
30     /// <returns>The final accumulator value.</returns>
31     /// <exception cref="T:System.ArgumentNullException"><paramref
         name="source">source</paramref> or <paramref name="func">func</paramref> is
          null.</exception>
32     public static TAccumulate Aggregate<TSource, TAccumulate>(this
         IQueryable<TSource> source, TAccumulate seed, Expression<Func<TAccumulate,
         TSource, TAccumulate>> func);
33     /// <summary>Applies an accumulator function over a sequence. The specified
         seed value is used as the initial accumulator value, and the specified
         function is used to select the result value.</summary>
34     /// <param name="source">A sequence to aggregate over.</param>
```

```
35      /// <param name="seed">The initial accumulator value.</param>
36      /// <param name="func">An accumulator function to invoke on each element.</
          param>
37      /// <param name="selector">A function to transform the final accumulator
          value into the result value.</param>
38      /// <typeparam name="TSource">The type of the elements of source.</typeparam>
39      /// <typeparam name="TAccumulate">The type of the accumulator value.</
          typeparam>
40      /// <typeparam name="TResult">The type of the resulting value.</typeparam>
41      /// <returns>The transformed final accumulator value.</returns>
42      /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="func">func</paramref> or
           <paramref name="selector">selector</paramref> is null.</exception>
43      public static TResult Aggregate<TSource, TAccumulate, TResult>(this
          IQueryable<TSource> source, TAccumulate seed, Expression<Func<TAccumulate,
          TSource, TAccumulate>> func, Expression<Func<TAccumulate, TResult>>
          selector);
44      /// <summary>Determines whether all the elements of a sequence satisfy a
          condition.</summary>
45      /// <param name="source">A sequence whose elements to test for a condition.</
          param>
46      /// <param name="predicate">A function to test each element for a
          condition.</param>
47      /// <typeparam name="TSource">The type of the elements of source.</typeparam>
48      /// <returns>true if every element of the source sequence passes the test in
          the specified predicate, or if the sequence is empty; otherwise, false.</
          returns>
49      /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="predicate">predicate</
          paramref> is null.</exception>
50      public static bool All<TSource>(this IQueryable<TSource> source,
          Expression<Func<TSource, bool>> predicate);
51      /// <summary>Determines whether a sequence contains any elements.</summary>
52      /// <param name="source">A sequence to check for being empty.</param>
53      /// <typeparam name="TSource">The type of the elements of source.</typeparam>
54      /// <returns>true if the source sequence contains any elements; otherwise,
          false.</returns>
55      /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
56      public static bool Any<TSource>(this IQueryable<TSource> source);
57      /// <summary>Determines whether any element of a sequence satisfies a
          condition.</summary>
58      /// <param name="source">A sequence whose elements to test for a condition.</
          param>
59      /// <param name="predicate">A function to test each element for a
          condition.</param>
60      /// <typeparam name="TSource">The type of the elements of source.</typeparam>
61      /// <returns>true if any elements in the source sequence pass the test in the
           specified predicate; otherwise, false.</returns>
62      /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="predicate">predicate</
          paramref> is null.</exception>
```

```
63      public static bool Any<TSource>(this IQueryable<TSource> source,
          Expression<Func<TSource, bool>> predicate);
64      /// <param name="source"></param>
65      /// <param name="element"></param>
66      /// <typeparam name="TSource"></typeparam>
67      /// <returns></returns>
68      public static IQueryable<TSource> Append<TSource>(this IQueryable<TSource>
          source, TSource element);
69      /// <summary>Converts an <see cref="T:System.Collections.IEnumerable"></see>
          to an <see cref="T:System.Linq.IQueryable"></see>.</summary>
70      /// <param name="source">A sequence to convert.</param>
71      /// <returns>An <see cref="T:System.Linq.IQueryable"></see> that represents
          the input sequence.</returns>
72      /// <exception cref="T:System.ArgumentException"><paramref
          name="source">source</paramref> does not implement <see
          cref="T:System.Collections.Generic.IEnumerable`1"></see> for some <paramref
           name="T">T</paramref>.</exception>
73      /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
74      public static IQueryable AsQueryable(this IEnumerable source);
75      /// <summary>Converts a generic <see
          cref="T:System.Collections.Generic.IEnumerable`1"></see> to a generic <see
          cref="T:System.Linq.IQueryable`1"></see>.</summary>
76      /// <param name="source">A sequence to convert.</param>
77      /// <typeparam name="TElement">The type of the elements of source.</
          typeparam>
78      /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that represents
           the input sequence.</returns>
79      /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
80      public static IQueryable<TElement> AsQueryable<TElement>(this
          IEnumerable<TElement> source);
81      /// <summary>Computes the average of a sequence of <see
          cref="T:System.Decimal"></see> values.</summary>
82      /// <param name="source">A sequence of <see cref="T:System.Decimal"></see>
          values to calculate the average of.</param>
83      /// <returns>The average of the sequence of values.</returns>
84      /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
85      /// <exception cref="T:System.InvalidOperationException"><paramref
          name="source">source</paramref> contains no elements.</exception>
86      public static decimal Average(this IQueryable<decimal> source);
87      /// <summary>Computes the average of a sequence of <see
          cref="T:System.Double"></see> values.</summary>
88      /// <param name="source">A sequence of <see cref="T:System.Double"></see>
          values to calculate the average of.</param>
89      /// <returns>The average of the sequence of values.</returns>
90      /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
91      /// <exception cref="T:System.InvalidOperationException"><paramref
          name="source">source</paramref> contains no elements.</exception>
92      public static double Average(this IQueryable<double> source);
```

```
93      /// <summary>Computes the average of a sequence of <see
          cref="T:System.Int32"></see> values.</summary>
94      /// <param name="source">A sequence of <see cref="T:System.Int32"></see>
          values to calculate the average of.</param>
95      /// <returns>The average of the sequence of values.</returns>
96      /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
97      /// <exception cref="T:System.InvalidOperationException"><paramref
          name="source">source</paramref> contains no elements.</exception>
98      public static double Average(this IQueryable<int> source);
99      /// <summary>Computes the average of a sequence of <see
          cref="T:System.Int64"></see> values.</summary>
100     /// <param name="source">A sequence of <see cref="T:System.Int64"></see>
          values to calculate the average of.</param>
101     /// <returns>The average of the sequence of values.</returns>
102     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
103     /// <exception cref="T:System.InvalidOperationException"><paramref
          name="source">source</paramref> contains no elements.</exception>
104     public static double Average(this IQueryable<long> source);
105     /// <summary>Computes the average of a sequence of nullable <see
          cref="T:System.Decimal"></see> values.</summary>
106     /// <param name="source">A sequence of nullable <see
          cref="T:System.Decimal"></see> values to calculate the average of.</param>
107     /// <returns>The average of the sequence of values, or null if the source
          sequence is empty or contains only null values.</returns>
108     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
109     public static decimal? Average(this IQueryable<decimal?> source);
110     /// <summary>Computes the average of a sequence of nullable <see
          cref="T:System.Double"></see> values.</summary>
111     /// <param name="source">A sequence of nullable <see
          cref="T:System.Double"></see> values to calculate the average of.</param>
112     /// <returns>The average of the sequence of values, or null if the source
          sequence is empty or contains only null values.</returns>
113     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
114     public static double? Average(this IQueryable<double?> source);
115     /// <summary>Computes the average of a sequence of nullable <see
          cref="T:System.Int32"></see> values.</summary>
116     /// <param name="source">A sequence of nullable <see cref="T:System.Int32"></
          see> values to calculate the average of.</param>
117     /// <returns>The average of the sequence of values, or null if the source
          sequence is empty or contains only null values.</returns>
118     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
119     public static double? Average(this IQueryable<int?> source);
120     /// <summary>Computes the average of a sequence of nullable <see
          cref="T:System.Int64"></see> values.</summary>
121     /// <param name="source">A sequence of nullable <see cref="T:System.Int64"></
          see> values to calculate the average of.</param>
122     /// <returns>The average of the sequence of values, or null if the source
```

```
        sequence is empty or contains only null values.</returns>
123     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
124     public static double? Average(this IQueryable<long?> source);
125     /// <summary>Computes the average of a sequence of nullable <see
        cref="T:System.Single"></see> values.</summary>
126     /// <param name="source">A sequence of nullable <see
        cref="T:System.Single"></see> values to calculate the average of.</param>
127     /// <returns>The average of the sequence of values, or null if the source
        sequence is empty or contains only null values.</returns>
128     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
129     public static float? Average(this IQueryable<float?> source);
130     /// <summary>Computes the average of a sequence of <see
        cref="T:System.Single"></see> values.</summary>
131     /// <param name="source">A sequence of <see cref="T:System.Single"></see>
        values to calculate the average of.</param>
132     /// <returns>The average of the sequence of values.</returns>
133     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
134     /// <exception cref="T:System.InvalidOperationException"><paramref
        name="source">source</paramref> contains no elements.</exception>
135     public static float Average(this IQueryable<float> source);
136     /// <summary>Computes the average of a sequence of <see
        cref="T:System.Decimal"></see> values that is obtained by invoking a
        projection function on each element of the input sequence.</summary>
137     /// <param name="source">A sequence of values that are used to calculate an
        average.</param>
138     /// <param name="selector">A projection function to apply to each element.</
        param>
139     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
140     /// <returns>The average of the sequence of values.</returns>
141     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref name="selector">selector</
        paramref> is null.</exception>
142     /// <exception cref="T:System.InvalidOperationException"><paramref
        name="source">source</paramref> contains no elements.</exception>
143     public static decimal Average<TSource>(this IQueryable<TSource> source,
        Expression<Func<TSource, decimal>> selector);
144     /// <summary>Computes the average of a sequence of <see
        cref="T:System.Double"></see> values that is obtained by invoking a
        projection function on each element of the input sequence.</summary>
145     /// <param name="source">A sequence of values to calculate the average of.</
        param>
146     /// <param name="selector">A projection function to apply to each element.</
        param>
147     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
148     /// <returns>The average of the sequence of values.</returns>
149     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref name="selector">selector</
        paramref> is null.</exception>
150     /// <exception cref="T:System.InvalidOperationException"><paramref
```

```
          name="source">source</paramref> contains no elements.</exception>
151       public static double Average<TSource>(this IQueryable<TSource> source,
            Expression<Func<TSource, double>> selector);
152       /// <summary>Computes the average of a sequence of <see
            cref="T:System.Int32"></see> values that is obtained by invoking a
            projection function on each element of the input sequence.</summary>
153       /// <param name="source">A sequence of values to calculate the average of.</
            param>
154       /// <param name="selector">A projection function to apply to each element.</
            param>
155       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
156       /// <returns>The average of the sequence of values.</returns>
157       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> or <paramref name="selector">selector</
            paramref> is null.</exception>
158       /// <exception cref="T:System.InvalidOperationException"><paramref
            name="source">source</paramref> contains no elements.</exception>
159       public static double Average<TSource>(this IQueryable<TSource> source,
            Expression<Func<TSource, int>> selector);
160       /// <summary>Computes the average of a sequence of <see
            cref="T:System.Int64"></see> values that is obtained by invoking a
            projection function on each element of the input sequence.</summary>
161       /// <param name="source">A sequence of values to calculate the average of.</
            param>
162       /// <param name="selector">A projection function to apply to each element.</
            param>
163       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
164       /// <returns>The average of the sequence of values.</returns>
165       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> or <paramref name="selector">selector</
            paramref> is null.</exception>
166       /// <exception cref="T:System.InvalidOperationException"><paramref
            name="source">source</paramref> contains no elements.</exception>
167       public static double Average<TSource>(this IQueryable<TSource> source,
            Expression<Func<TSource, long>> selector);
168       /// <summary>Computes the average of a sequence of nullable <see
            cref="T:System.Decimal"></see> values that is obtained by invoking a
            projection function on each element of the input sequence.</summary>
169       /// <param name="source">A sequence of values to calculate the average of.</
            param>
170       /// <param name="selector">A projection function to apply to each element.</
            param>
171       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
172       /// <returns>The average of the sequence of values, or null if the <paramref
            name="source">source</paramref> sequence is empty or contains only null
            values.</returns>
173       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> or <paramref name="selector">selector</
            paramref> is null.</exception>
174       public static decimal? Average<TSource>(this IQueryable<TSource> source,
            Expression<Func<TSource, decimal?>> selector);
175       /// <summary>Computes the average of a sequence of nullable <see
```

```
          cref="T:System.Double"></see> values that is obtained by invoking a
          projection function on each element of the input sequence.</summary>
176       /// <param name="source">A sequence of values to calculate the average of.</
          param>
177       /// <param name="selector">A projection function to apply to each element.</
          param>
178       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
179       /// <returns>The average of the sequence of values, or null if the <paramref
          name="source">source</paramref> sequence is empty or contains only null
          values.</returns>
180       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
181       public static double? Average<TSource>(this IQueryable<TSource> source,
          Expression<Func<TSource, double?>> selector);
182       /// <summary>Computes the average of a sequence of nullable <see
          cref="T:System.Int32"></see> values that is obtained by invoking a
          projection function on each element of the input sequence.</summary>
183       /// <param name="source">A sequence of values to calculate the average of.</
          param>
184       /// <param name="selector">A projection function to apply to each element.</
          param>
185       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
186       /// <returns>The average of the sequence of values, or null if the <paramref
          name="source">source</paramref> sequence is empty or contains only null
          values.</returns>
187       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
188       public static double? Average<TSource>(this IQueryable<TSource> source,
          Expression<Func<TSource, int?>> selector);
189       /// <summary>Computes the average of a sequence of nullable <see
          cref="T:System.Int64"></see> values that is obtained by invoking a
          projection function on each element of the input sequence.</summary>
190       /// <param name="source">A sequence of values to calculate the average of.</
          param>
191       /// <param name="selector">A projection function to apply to each element.</
          param>
192       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
193       /// <returns>The average of the sequence of values, or null if the <paramref
          name="source">source</paramref> sequence is empty or contains only null
          values.</returns>
194       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
195       public static double? Average<TSource>(this IQueryable<TSource> source,
          Expression<Func<TSource, long?>> selector);
196       /// <summary>Computes the average of a sequence of nullable <see
          cref="T:System.Single"></see> values that is obtained by invoking a
          projection function on each element of the input sequence.</summary>
197       /// <param name="source">A sequence of values to calculate the average of.</
          param>
```

```
198     /// <param name="selector">A projection function to apply to each element.</
          param>
199     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
200     /// <returns>The average of the sequence of values, or null if the <paramref
          name="source">source</paramref> sequence is empty or contains only null
          values.</returns>
201     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
202     public static float? Average<TSource>(this IQueryable<TSource> source,
          Expression<Func<TSource, float?>> selector);
203     /// <summary>Computes the average of a sequence of <see
          cref="T:System.Single"></see> values that is obtained by invoking a
          projection function on each element of the input sequence.</summary>
204     /// <param name="source">A sequence of values to calculate the average of.</
          param>
205     /// <param name="selector">A projection function to apply to each element.</
          param>
206     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
207     /// <returns>The average of the sequence of values.</returns>
208     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
209     /// <exception cref="T:System.InvalidOperationException"><paramref
          name="source">source</paramref> contains no elements.</exception>
210     public static float Average<TSource>(this IQueryable<TSource> source,
          Expression<Func<TSource, float>> selector);
211     /// <summary>Converts the elements of an <see
          cref="T:System.Linq.IQueryable"></see> to the specified type.</summary>
212     /// <param name="source">The <see cref="T:System.Linq.IQueryable"></see> that
           contains the elements to be converted.</param>
213     /// <typeparam name="TResult">The type to convert the elements of source
          to.</typeparam>
214     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
          each element of the source sequence converted to the specified type.</
          returns>
215     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
216     /// <exception cref="T:System.InvalidCastException">An element in the
          sequence cannot be cast to type <paramref name="TResult">TResult</
          paramref>.</exception>
217     public static IQueryable<TResult> Cast<TResult>(this IQueryable source);
218     /// <summary>Concatenates two sequences.</summary>
219     /// <param name="source1">The first sequence to concatenate.</param>
220     /// <param name="source2">The sequence to concatenate to the first
          sequence.</param>
221     /// <typeparam name="TSource">The type of the elements of the input
          sequences.</typeparam>
222     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
          the concatenated elements of the two input sequences.</returns>
223     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source1">source1</paramref> or <paramref name="source2">source2</
```

```
          paramref> is null.</exception>
224       public static IQueryable<TSource> Concat<TSource>(this IQueryable<TSource>
            source1, IEnumerable<TSource> source2);
225       /// <summary>Determines whether a sequence contains a specified element by
            using the default equality comparer.</summary>
226       /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> in
            which to locate item.</param>
227       /// <param name="item">The object to locate in the sequence.</param>
228       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
229       /// <returns>true if the input sequence contains an element that has the
            specified value; otherwise, false.</returns>
230       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> is null.</exception>
231       public static bool Contains<TSource>(this IQueryable<TSource> source, TSource
             item);
232       /// <summary>Determines whether a sequence contains a specified element by
            using a specified <see
            cref="T:System.Collections.Generic.IEqualityComparer`1"></see>.</summary>
233       /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> in
            which to locate item.</param>
234       /// <param name="item">The object to locate in the sequence.</param>
235       /// <param name="comparer">An <see
            cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
            values.</param>
236       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
237       /// <returns>true if the input sequence contains an element that has the
            specified value; otherwise, false.</returns>
238       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> is null.</exception>
239       public static bool Contains<TSource>(this IQueryable<TSource> source, TSource
             item, IEqualityComparer<TSource> comparer);
240       /// <summary>Returns the number of elements in a sequence.</summary>
241       /// <param name="source">The <see cref="T:System.Linq.IQueryable`1"></see>
            that contains the elements to be counted.</param>
242       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
243       /// <returns>The number of elements in the input sequence.</returns>
244       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> is null.</exception>
245       /// <exception cref="T:System.OverflowException">The number of elements in
            <paramref name="source">source</paramref> is larger than <see
            cref="F:System.Int32.MaxValue"></see>.</exception>
246       public static int Count<TSource>(this IQueryable<TSource> source);
247       /// <summary>Returns the number of elements in the specified sequence that
            satisfies a condition.</summary>
248       /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see>
            that contains the elements to be counted.</param>
249       /// <param name="predicate">A function to test each element for a
            condition.</param>
250       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
251       /// <returns>The number of elements in the sequence that satisfies the
            condition in the predicate function.</returns>
252       /// <exception cref="T:System.ArgumentNullException"><paramref
```

```
        name="source">source</paramref> or <paramref name="predicate">predicate</
        paramref> is null.</exception>
253     /// <exception cref="T:System.OverflowException">The number of elements in
        <paramref name="source">source</paramref> is larger than <see
        cref="F:System.Int32.MaxValue"></see>.</exception>
254     public static int Count<TSource>(this IQueryable<TSource> source,
        Expression<Func<TSource, bool>> predicate);
255     /// <summary>Returns the elements of the specified sequence or the type
        parameter's default value in a singleton collection if the sequence is
        empty.</summary>
256     /// <param name="source">The <see cref="T:System.Linq.IQueryable`1"></see> to
         return a default value for if empty.</param>
257     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
258     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
        default(<paramref name="TSource">TSource</paramref>) if <paramref
        name="source">source</paramref> is empty; otherwise, <paramref
        name="source">source</paramref>.</returns>
259     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
260     public static IQueryable<TSource> DefaultIfEmpty<TSource>(this
        IQueryable<TSource> source);
261     /// <summary>Returns the elements of the specified sequence or the specified
        value in a singleton collection if the sequence is empty.</summary>
262     /// <param name="source">The <see cref="T:System.Linq.IQueryable`1"></see> to
         return the specified value for if empty.</param>
263     /// <param name="defaultValue">The value to return if the sequence is
        empty.</param>
264     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
265     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
        <paramref name="defaultValue">defaultValue</paramref> if <paramref
        name="source">source</paramref> is empty; otherwise, <paramref
        name="source">source</paramref>.</returns>
266     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
267     public static IQueryable<TSource> DefaultIfEmpty<TSource>(this
        IQueryable<TSource> source, TSource defaultValue);
268     /// <summary>Returns distinct elements from a sequence by using the default
        equality comparer to compare values.</summary>
269     /// <param name="source">The <see cref="T:System.Linq.IQueryable`1"></see> to
         remove duplicates from.</param>
270     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
271     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
        distinct elements from <paramref name="source">source</paramref>.</returns>
272     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
273     public static IQueryable<TSource> Distinct<TSource>(this IQueryable<TSource>
        source);
274     /// <summary>Returns distinct elements from a sequence by using a specified
        <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to
        compare values.</summary>
275     /// <param name="source">The <see cref="T:System.Linq.IQueryable`1"></see> to
         remove duplicates from.</param>
```

```
276     /// <param name="comparer">An <see
          cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
          values.</param>
277     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
278     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
          distinct elements from <paramref name="source">source</paramref>.</returns>
279     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="comparer">comparer</
          paramref> is null.</exception>
280     public static IQueryable<TSource> Distinct<TSource>(this IQueryable<TSource>
          source, IEqualityComparer<TSource> comparer);
281     /// <summary>Returns the element at a specified index in a sequence.</
          summary>
282     /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to
          return an element from.</param>
283     /// <param name="index">The zero-based index of the element to retrieve.</
          param>
284     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
285     /// <returns>The element at the specified position in <paramref
          name="source">source</paramref>.</returns>
286     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
287     /// <exception cref="T:System.ArgumentOutOfRangeException"><paramref
          name="index">index</paramref> is less than zero.</exception>
288     public static TSource ElementAt<TSource>(this IQueryable<TSource> source, int
           index);
289     /// <summary>Returns the element at a specified index in a sequence or a
          default value if the index is out of range.</summary>
290     /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to
          return an element from.</param>
291     /// <param name="index">The zero-based index of the element to retrieve.</
          param>
292     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
293     /// <returns>default(<paramref name="TSource">TSource</paramref>) if
          <paramref name="index">index</paramref> is outside the bounds of <paramref
          name="source">source</paramref>; otherwise, the element at the specified
          position in <paramref name="source">source</paramref>.</returns>
294     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
295     public static TSource ElementAtOrDefault<TSource>(this IQueryable<TSource>
          source, int index);
296     /// <summary>Produces the set difference of two sequences by using the
          default equality comparer to compare values.</summary>
297     /// <param name="source1">An <see cref="T:System.Linq.IQueryable`1"></see>
          whose elements that are not also in source2 will be returned.</param>
298     /// <param name="source2">An <see
          cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements
          that also occur in the first sequence will not appear in the returned
          sequence.</param>
299     /// <typeparam name="TSource">The type of the elements of the input
          sequences.</typeparam>
300     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
```

```
          the set difference of the two sequences.</returns>
301       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source1">source1</paramref> or <paramref name="source2">source2</
            paramref> is null.</exception>
302       public static IQueryable<TSource> Except<TSource>(this IQueryable<TSource>
            source1, IEnumerable<TSource> source2);
303       /// <summary>Produces the set difference of two sequences by using the
            specified <see cref="T:System.Collections.Generic.IEqualityComparer`1"></
            see> to compare values.</summary>
304       /// <param name="source1">An <see cref="T:System.Linq.IQueryable`1"></see>
            whose elements that are not also in source2 will be returned.</param>
305       /// <param name="source2">An <see
            cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements
            that also occur in the first sequence will not appear in the returned
            sequence.</param>
306       /// <param name="comparer">An <see
            cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
            values.</param>
307       /// <typeparam name="TSource">The type of the elements of the input
            sequences.</typeparam>
308       /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
            the set difference of the two sequences.</returns>
309       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source1">source1</paramref> or <paramref name="source2">source2</
            paramref> is null.</exception>
310       public static IQueryable<TSource> Except<TSource>(this IQueryable<TSource>
            source1, IEnumerable<TSource> source2, IEqualityComparer<TSource>
            comparer);
311       /// <summary>Returns the first element of a sequence.</summary>
312       /// <param name="source">The <see cref="T:System.Linq.IQueryable`1"></see> to
             return the first element of.</param>
313       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
314       /// <returns>The first element in <paramref name="source">source</
            paramref>.</returns>
315       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> is null.</exception>
316       /// <exception cref="T:System.InvalidOperationException">The source sequence
            is empty.</exception>
317       public static TSource First<TSource>(this IQueryable<TSource> source);
318       /// <summary>Returns the first element of a sequence that satisfies a
            specified condition.</summary>
319       /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to
            return an element from.</param>
320       /// <param name="predicate">A function to test each element for a
            condition.</param>
321       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
322       /// <returns>The first element in <paramref name="source">source</paramref>
            that passes the test in <paramref name="predicate">predicate</paramref>.</
            returns>
323       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> or <paramref name="predicate">predicate</
            paramref> is null.</exception>
```

```
324     /// <exception cref="T:System.InvalidOperationException">No element satisfies the condition in <paramref name="predicate">predicate</paramref>. -or- The source sequence is empty.</exception>
325     public static TSource First<TSource>(this IQueryable<TSource> source, Expression<Func<TSource, bool>> predicate);
326     /// <summary>Returns the first element of a sequence, or a default value if the sequence contains no elements.</summary>
327     /// <param name="source">The <see cref="T:System.Linq.IQueryable`1"></see> to return the first element of.</param>
328     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
329     /// <returns>default(<paramref name="TSource">TSource</paramref>) if <paramref name="source">source</paramref> is empty; otherwise, the first element in <paramref name="source">source</paramref>.</returns>
330     /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
331     public static TSource FirstOrDefault<TSource>(this IQueryable<TSource> source);
332     /// <summary>Returns the first element of a sequence that satisfies a specified condition or a default value if no such element is found.</summary>
333     /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to return an element from.</param>
334     /// <param name="predicate">A function to test each element for a condition.</param>
335     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
336     /// <returns>default(<paramref name="TSource">TSource</paramref>) if <paramref name="source">source</paramref> is empty or if no element passes the test specified by <paramref name="predicate">predicate</paramref>; otherwise, the first element in <paramref name="source">source</paramref> that passes the test specified by <paramref name="predicate">predicate</paramref>.</returns>
337     /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
338     public static TSource FirstOrDefault<TSource>(this IQueryable<TSource> source, Expression<Func<TSource, bool>> predicate);
339     /// <summary>Groups the elements of a sequence according to a specified key selector function.</summary>
340     /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> whose elements to group.</param>
341     /// <param name="keySelector">A function to extract the key for each element.</param>
342     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
343     /// <typeparam name="TKey">The type of the key returned by the function represented in keySelector.</typeparam>
344     /// <returns><p sourcefile="System.Linq.Queryable.yml" sourcestartlinenumber="1" sourceendlinenumber="1">An <code>IQueryable&lt;&gt;<_tkey2c_ tsource="">&gt;</_tkey2c_></code> in C# or <code>IQueryable(Of IGrouping(Of TKey, TSource))</code> in Visual Basic where each <xref href="System.Linq.IGrouping`2"></xref> object contains a sequence of objects and a key.</p>
345     /// </returns>
```

```
346     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref
          name="keySelector">keySelector</paramref> is null.</exception>
347     public static IQueryable<IGrouping<TKey, TSource>> GroupBy<TSource, TKey>
          (this IQueryable<TSource> source, Expression<Func<TSource, TKey>>
          keySelector);
348     /// <summary>Groups the elements of a sequence according to a specified key
          selector function and compares the keys by using a specified comparer.</
          summary>
349     /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see>
          whose elements to group.</param>
350     /// <param name="keySelector">A function to extract the key for each
          element.</param>
351     /// <param name="comparer">An <see
          cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
          keys.</param>
352     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
353     /// <typeparam name="TKey">The type of the key returned by the function
          represented in keySelector.</typeparam>
354     /// <returns><p sourcefile="System.Linq.Queryable.yml"
          sourcestartlinenumber="1" sourceendlinenumber="1">An
          <code>IQueryable&lt;&gt;<_tkey2c_ tsource="">&gt;</_tkey2c_></code> in C#
          or <code>IQueryable(Of IGrouping(Of TKey, TSource))</code> in Visual Basic
          where each <xref href="System.Linq.IGrouping`2"></xref> contains a sequence
           of objects and a key.</p>
355     /// </returns>
356     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref
          name="keySelector">keySelector</paramref> or <paramref
          name="comparer">comparer</paramref> is null.</exception>
357     public static IQueryable<IGrouping<TKey, TSource>> GroupBy<TSource, TKey>
          (this IQueryable<TSource> source, Expression<Func<TSource, TKey>>
          keySelector, IEqualityComparer<TKey> comparer);
358     /// <summary>Groups the elements of a sequence according to a specified key
          selector function and projects the elements for each group by using a
          specified function.</summary>
359     /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see>
          whose elements to group.</param>
360     /// <param name="keySelector">A function to extract the key for each
          element.</param>
361     /// <param name="elementSelector">A function to map each source element to an
           element in an <see cref="T:System.Linq.IGrouping`2"></see>.</param>
362     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
363     /// <typeparam name="TKey">The type of the key returned by the function
          represented in keySelector.</typeparam>
364     /// <typeparam name="TElement">The type of the elements in each <see
          cref="T:System.Linq.IGrouping`2"></see>.</typeparam>
365     /// <returns><p sourcefile="System.Linq.Queryable.yml"
          sourcestartlinenumber="1" sourceendlinenumber="1">An
          <code>IQueryable&lt;&gt;<_tkey2c_ telement="">&gt;</_tkey2c_></code> in C#
          or <code>IQueryable(Of IGrouping(Of TKey, TElement))</code> in Visual Basic
           where each <xref href="System.Linq.IGrouping`2"></xref> contains a
```

```
          sequence of objects of type <code data-dev-comment-type="paramref">TElement</code> and a key.</p>
366       /// </returns>
367       /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> or <paramref name="elementSelector">elementSelector</paramref> is null.</exception>
368       public static IQueryable<IGrouping<TKey, TElement>> GroupBy<TSource, TKey, TElement>(this IQueryable<TSource> source, Expression<Func<TSource, TKey>> keySelector, Expression<Func<TSource, TElement>> elementSelector);
369       /// <summary>Groups the elements of a sequence and projects the elements for each group by using a specified function. Key values are compared by using a specified comparer.</summary>
370       /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> whose elements to group.</param>
371       /// <param name="keySelector">A function to extract the key for each element.</param>
372       /// <param name="elementSelector">A function to map each source element to an element in an <see cref="T:System.Linq.IGrouping`2"></see>.</param>
373       /// <param name="comparer">An <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare keys.</param>
374       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
375       /// <typeparam name="TKey">The type of the key returned by the function represented in keySelector.</typeparam>
376       /// <typeparam name="TElement">The type of the elements in each <see cref="T:System.Linq.IGrouping`2"></see>.</typeparam>
377       /// <returns><p sourcefile="System.Linq.Queryable.yml" sourcestartlinenumber="1" sourceendlinenumber="1">An <code>IQueryable&amp;lt;&amp;gt;(_tkey2c_ telement="">&amp;gt;</_tkey2c_></code> in C# or <code>IQueryable(Of IGrouping(Of TKey, TElement))</code> in Visual Basic where each <xref href="System.Linq.IGrouping`2"></xref> contains a sequence of objects of type <code data-dev-comment-type="paramref">TElement</code> and a key.</p>
378       /// </returns>
379       /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> or <paramref name="elementSelector">elementSelector</paramref> or <paramref name="comparer">comparer</paramref> is null.</exception>
380       public static IQueryable<IGrouping<TKey, TElement>> GroupBy<TSource, TKey, TElement>(this IQueryable<TSource> source, Expression<Func<TSource, TKey>> keySelector, Expression<Func<TSource, TElement>> elementSelector, IEqualityComparer<TKey> comparer);
381       /// <summary>Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.</summary>
382       /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> whose elements to group.</param>
383       /// <param name="keySelector">A function to extract the key for each element.</param>
384       /// <param name="resultSelector">A function to create a result value from
```

```
          each group.</param>
385       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
386       /// <typeparam name="TKey">The type of the key returned by the function
            represented in keySelector.</typeparam>
387       /// <typeparam name="TResult">The type of the result value returned by
            resultSelector.</typeparam>
388       /// <returns>An T:System.Linq.IQueryable`1 that has a type argument of
            <paramref name="TResult">TResult</paramref> and where each element
            represents a projection over a group and its key.</returns>
389       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> or <paramref
            name="keySelector">keySelector</paramref> or <paramref
            name="resultSelector">resultSelector</paramref> is null.</exception>
390       public static IQueryable<TResult> GroupBy<TSource, TKey, TResult>(this
            IQueryable<TSource> source, Expression<Func<TSource, TKey>> keySelector,
            Expression<Func<TKey, IEnumerable<TSource>, TResult>> resultSelector);
391       /// <summary>Groups the elements of a sequence according to a specified key
            selector function and creates a result value from each group and its key.
            Keys are compared by using a specified comparer.</summary>
392       /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see>
            whose elements to group.</param>
393       /// <param name="keySelector">A function to extract the key for each
            element.</param>
394       /// <param name="resultSelector">A function to create a result value from
            each group.</param>
395       /// <param name="comparer">An <see
            cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
            keys.</param>
396       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
397       /// <typeparam name="TKey">The type of the key returned by the function
            represented in keySelector.</typeparam>
398       /// <typeparam name="TResult">The type of the result value returned by
            resultSelector.</typeparam>
399       /// <returns>An T:System.Linq.IQueryable`1 that has a type argument of
            <paramref name="TResult">TResult</paramref> and where each element
            represents a projection over a group and its key.</returns>
400       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> or <paramref
            name="keySelector">keySelector</paramref> or <paramref
            name="resultSelector">resultSelector</paramref> or <paramref
            name="comparer">comparer</paramref> is null.</exception>
401       public static IQueryable<TResult> GroupBy<TSource, TKey, TResult>(this
            IQueryable<TSource> source, Expression<Func<TSource, TKey>> keySelector,
            Expression<Func<TKey, IEnumerable<TSource>, TResult>> resultSelector,
            IEqualityComparer<TKey> comparer);
402       /// <summary>Groups the elements of a sequence according to a specified key
            selector function and creates a result value from each group and its key.
            The elements of each group are projected by using a specified function.</
            summary>
403       /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see>
            whose elements to group.</param>
404       /// <param name="keySelector">A function to extract the key for each
```

```
          element.</param>
405       /// <param name="elementSelector">A function to map each source element to an
             element in an <see cref="T:System.Linq.IGrouping`2"></see>.</param>
406       /// <param name="resultSelector">A function to create a result value from
          each group.</param>
407       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
408       /// <typeparam name="TKey">The type of the key returned by the function
          represented in keySelector.</typeparam>
409       /// <typeparam name="TElement">The type of the elements in each <see
          cref="T:System.Linq.IGrouping`2"></see>.</typeparam>
410       /// <typeparam name="TResult">The type of the result value returned by
          resultSelector.</typeparam>
411       /// <returns>An T:System.Linq.IQueryable`1 that has a type argument of
          <paramref name="TResult">TResult</paramref> and where each element
          represents a projection over a group and its key.</returns>
412       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref
          name="keySelector">keySelector</paramref> or <paramref
          name="elementSelector">elementSelector</paramref> or <paramref
          name="resultSelector">resultSelector</paramref> is null.</exception>
413       public static IQueryable<TResult> GroupBy<TSource, TKey, TElement, TResult>
          (this IQueryable<TSource> source, Expression<Func<TSource, TKey>>
          keySelector, Expression<Func<TSource, TElement>> elementSelector,
          Expression<Func<TKey, IEnumerable<TElement>, TResult>> resultSelector);
414       /// <summary>Groups the elements of a sequence according to a specified key
          selector function and creates a result value from each group and its key.
          Keys are compared by using a specified comparer and the elements of each
          group are projected by using a specified function.</summary>
415       /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see>
          whose elements to group.</param>
416       /// <param name="keySelector">A function to extract the key for each
          element.</param>
417       /// <param name="elementSelector">A function to map each source element to an
             element in an <see cref="T:System.Linq.IGrouping`2"></see>.</param>
418       /// <param name="resultSelector">A function to create a result value from
          each group.</param>
419       /// <param name="comparer">An <see
          cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
          keys.</param>
420       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
421       /// <typeparam name="TKey">The type of the key returned by the function
          represented in keySelector.</typeparam>
422       /// <typeparam name="TElement">The type of the elements in each <see
          cref="T:System.Linq.IGrouping`2"></see>.</typeparam>
423       /// <typeparam name="TResult">The type of the result value returned by
          resultSelector.</typeparam>
424       /// <returns>An T:System.Linq.IQueryable`1 that has a type argument of
          <paramref name="TResult">TResult</paramref> and where each element
          represents a projection over a group and its key.</returns>
425       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref
          name="keySelector">keySelector</paramref> or <paramref
```

```
          name="elementSelector">elementSelector</paramref> or <paramref
          name="resultSelector">resultSelector</paramref> or <paramref
          name="comparer">comparer</paramref> is null.</exception>
426       public static IQueryable<TResult> GroupBy<TSource, TKey, TElement, TResult>
          (this IQueryable<TSource> source, Expression<Func<TSource, TKey>>
          keySelector, Expression<Func<TSource, TElement>> elementSelector,
          Expression<Func<TKey, IEnumerable<TElement>, TResult>> resultSelector,
          IEqualityComparer<TKey> comparer);
427       /// <summary>Correlates the elements of two sequences based on key equality
          and groups the results. The default equality comparer is used to compare
          keys.</summary>
428       /// <param name="outer">The first sequence to join.</param>
429       /// <param name="inner">The sequence to join to the first sequence.</param>
430       /// <param name="outerKeySelector">A function to extract the join key from
          each element of the first sequence.</param>
431       /// <param name="innerKeySelector">A function to extract the join key from
          each element of the second sequence.</param>
432       /// <param name="resultSelector">A function to create a result element from
          an element from the first sequence and a collection of matching elements
          from the second sequence.</param>
433       /// <typeparam name="TOuter">The type of the elements of the first
          sequence.</typeparam>
434       /// <typeparam name="TInner">The type of the elements of the second
          sequence.</typeparam>
435       /// <typeparam name="TKey">The type of the keys returned by the key selector
          functions.</typeparam>
436       /// <typeparam name="TResult">The type of the result elements.</typeparam>
437       /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
          elements of type <paramref name="TResult">TResult</paramref> obtained by
          performing a grouped join on two sequences.</returns>
438       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="outer">outer</paramref> or <paramref name="inner">inner</paramref> or
           <paramref name="outerKeySelector">outerKeySelector</paramref> or <paramref
           name="innerKeySelector">innerKeySelector</paramref> or <paramref
          name="resultSelector">resultSelector</paramref> is null.</exception>
439       public static IQueryable<TResult> GroupJoin<TOuter, TInner, TKey, TResult>
          (this IQueryable<TOuter> outer, IEnumerable<TInner> inner,
          Expression<Func<TOuter, TKey>> outerKeySelector, Expression<Func<TInner,
          TKey>> innerKeySelector, Expression<Func<TOuter, IEnumerable<TInner>,
          TResult>> resultSelector);
440       /// <summary>Correlates the elements of two sequences based on key equality
          and groups the results. A specified <see
          cref="T:System.Collections.Generic.IEqualityComparer`1"></see> is used to
          compare keys.</summary>
441       /// <param name="outer">The first sequence to join.</param>
442       /// <param name="inner">The sequence to join to the first sequence.</param>
443       /// <param name="outerKeySelector">A function to extract the join key from
          each element of the first sequence.</param>
444       /// <param name="innerKeySelector">A function to extract the join key from
          each element of the second sequence.</param>
445       /// <param name="resultSelector">A function to create a result element from
          an element from the first sequence and a collection of matching elements
```

```
          from the second sequence.</param>
446       /// <param name="comparer">A comparer to hash and compare keys.</param>
447       /// <typeparam name="TOuter">The type of the elements of the first
          sequence.</typeparam>
448       /// <typeparam name="TInner">The type of the elements of the second
          sequence.</typeparam>
449       /// <typeparam name="TKey">The type of the keys returned by the key selector
          functions.</typeparam>
450       /// <typeparam name="TResult">The type of the result elements.</typeparam>
451       /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
          elements of type <paramref name="TResult">TResult</paramref> obtained by
          performing a grouped join on two sequences.</returns>
452       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="outer">outer</paramref> or <paramref name="inner">inner</paramref> or
           <paramref name="outerKeySelector">outerKeySelector</paramref> or <paramref
           name="innerKeySelector">innerKeySelector</paramref> or <paramref
          name="resultSelector">resultSelector</paramref> is null.</exception>
453       public static IQueryable<TResult> GroupJoin<TOuter, TInner, TKey, TResult>
          (this IQueryable<TOuter> outer, IEnumerable<TInner> inner,
          Expression<Func<TOuter, TKey>> outerKeySelector, Expression<Func<TInner,
          TKey>> innerKeySelector, Expression<Func<TOuter, IEnumerable<TInner>,
          TResult>> resultSelector, IEqualityComparer<TKey> comparer);
454       /// <summary>Produces the set intersection of two sequences by using the
          default equality comparer to compare values.</summary>
455       /// <param name="source1">A sequence whose distinct elements that also appear
           in source2 are returned.</param>
456       /// <param name="source2">A sequence whose distinct elements that also appear
           in the first sequence are returned.</param>
457       /// <typeparam name="TSource">The type of the elements of the input
          sequences.</typeparam>
458       /// <returns>A sequence that contains the set intersection of the two
          sequences.</returns>
459       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source1">source1</paramref> or <paramref name="source2">source2</
          paramref> is null.</exception>
460       public static IQueryable<TSource> Intersect<TSource>(this IQueryable<TSource>
           source1, IEnumerable<TSource> source2);
461       /// <summary>Produces the set intersection of two sequences by using the
          specified <see cref="T:System.Collections.Generic.IEqualityComparer`1"></
          see> to compare values.</summary>
462       /// <param name="source1">An <see cref="T:System.Linq.IQueryable`1"></see>
          whose distinct elements that also appear in source2 are returned.</param>
463       /// <param name="source2">An <see
          cref="T:System.Collections.Generic.IEnumerable`1"></see> whose distinct
          elements that also appear in the first sequence are returned.</param>
464       /// <param name="comparer">An <see
          cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
          values.</param>
465       /// <typeparam name="TSource">The type of the elements of the input
          sequences.</typeparam>
466       /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
          the set intersection of the two sequences.</returns>
```

```
467     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source1">source1</paramref> or <paramref name="source2">source2</
          paramref> is null.</exception>
468     public static IQueryable<TSource> Intersect<TSource>(this IQueryable<TSource>
           source1, IEnumerable<TSource> source2, IEqualityComparer<TSource>
          comparer);
469     /// <summary>Correlates the elements of two sequences based on matching keys.
           The default equality comparer is used to compare keys.</summary>
470     /// <param name="outer">The first sequence to join.</param>
471     /// <param name="inner">The sequence to join to the first sequence.</param>
472     /// <param name="outerKeySelector">A function to extract the join key from
          each element of the first sequence.</param>
473     /// <param name="innerKeySelector">A function to extract the join key from
          each element of the second sequence.</param>
474     /// <param name="resultSelector">A function to create a result element from
          two matching elements.</param>
475     /// <typeparam name="TOuter">The type of the elements of the first
          sequence.</typeparam>
476     /// <typeparam name="TInner">The type of the elements of the second
          sequence.</typeparam>
477     /// <typeparam name="TKey">The type of the keys returned by the key selector
          functions.</typeparam>
478     /// <typeparam name="TResult">The type of the result elements.</typeparam>
479     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that has
          elements of type <paramref name="TResult">TResult</paramref> obtained by
          performing an inner join on two sequences.</returns>
480     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="outer">outer</paramref> or <paramref name="inner">inner</paramref> or
           <paramref name="outerKeySelector">outerKeySelector</paramref> or <paramref
           name="innerKeySelector">innerKeySelector</paramref> or <paramref
          name="resultSelector">resultSelector</paramref> is null.</exception>
481     public static IQueryable<TResult> Join<TOuter, TInner, TKey, TResult>(this
          IQueryable<TOuter> outer, IEnumerable<TInner> inner,
          Expression<Func<TOuter, TKey>> outerKeySelector, Expression<Func<TInner,
          TKey>> innerKeySelector, Expression<Func<TOuter, TInner, TResult>>
          resultSelector);
482     /// <summary>Correlates the elements of two sequences based on matching keys.
           A specified <see
          cref="T:System.Collections.Generic.IEqualityComparer`1"></see> is used to
          compare keys.</summary>
483     /// <param name="outer">The first sequence to join.</param>
484     /// <param name="inner">The sequence to join to the first sequence.</param>
485     /// <param name="outerKeySelector">A function to extract the join key from
          each element of the first sequence.</param>
486     /// <param name="innerKeySelector">A function to extract the join key from
          each element of the second sequence.</param>
487     /// <param name="resultSelector">A function to create a result element from
          two matching elements.</param>
488     /// <param name="comparer">An <see
          cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to hash and
          compare keys.</param>
489     /// <typeparam name="TOuter">The type of the elements of the first
```

```
        sequence.</typeparam>
490     /// <typeparam name="TInner">The type of the elements of the second
        sequence.</typeparam>
491     /// <typeparam name="TKey">The type of the keys returned by the key selector
        functions.</typeparam>
492     /// <typeparam name="TResult">The type of the result elements.</typeparam>
493     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that has
        elements of type <paramref name="TResult">TResult</paramref> obtained by
        performing an inner join on two sequences.</returns>
494     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="outer">outer</paramref> or <paramref name="inner">inner</paramref> or
         <paramref name="outerKeySelector">outerKeySelector</paramref> or <paramref
         name="innerKeySelector">innerKeySelector</paramref> or <paramref
        name="resultSelector">resultSelector</paramref> is null.</exception>
495     public static IQueryable<TResult> Join<TOuter, TInner, TKey, TResult>(this
        IQueryable<TOuter> outer, IEnumerable<TInner> inner,
        Expression<Func<TOuter, TKey>> outerKeySelector, Expression<Func<TInner,
        TKey>> innerKeySelector, Expression<Func<TOuter, TInner, TResult>>
        resultSelector, IEqualityComparer<TKey> comparer);
496     /// <summary>Returns the last element in a sequence.</summary>
497     /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to
        return the last element of.</param>
498     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
499     /// <returns>The value at the last position in <paramref
        name="source">source</paramref>.</returns>
500     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> is null.</exception>
501     /// <exception cref="T:System.InvalidOperationException">The source sequence
        is empty.</exception>
502     public static TSource Last<TSource>(this IQueryable<TSource> source);
503     /// <summary>Returns the last element of a sequence that satisfies a
        specified condition.</summary>
504     /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to
        return an element from.</param>
505     /// <param name="predicate">A function to test each element for a
        condition.</param>
506     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
507     /// <returns>The last element in <paramref name="source">source</paramref>
        that passes the test specified by <paramref name="predicate">predicate</
        paramref>.</returns>
508     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref name="predicate">predicate</
        paramref> is null.</exception>
509     /// <exception cref="T:System.InvalidOperationException">No element satisfies
         the condition in <paramref name="predicate">predicate</paramref>.   -or-
         The source sequence is empty.</exception>
510     public static TSource Last<TSource>(this IQueryable<TSource> source,
        Expression<Func<TSource, bool>> predicate);
511     /// <summary>Returns the last element in a sequence, or a default value if
        the sequence contains no elements.</summary>
512     /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to
        return the last element of.</param>
```

```
513     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
514     /// <returns>default(<paramref name="TSource">TSource</paramref>) if
          <paramref name="source">source</paramref> is empty; otherwise, the last
          element in <paramref name="source">source</paramref>.</returns>
515     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
516     public static TSource LastOrDefault<TSource>(this IQueryable<TSource>
          source);
517     /// <summary>Returns the last element of a sequence that satisfies a
          condition or a default value if no such element is found.</summary>
518     /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to
          return an element from.</param>
519     /// <param name="predicate">A function to test each element for a
          condition.</param>
520     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
521     /// <returns>default(<paramref name="TSource">TSource</paramref>) if
          <paramref name="source">source</paramref> is empty or if no elements pass
          the test in the predicate function; otherwise, the last element of
          <paramref name="source">source</paramref> that passes the test in the
          predicate function.</returns>
522     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="predicate">predicate</
          paramref> is null.</exception>
523     public static TSource LastOrDefault<TSource>(this IQueryable<TSource> source,
           Expression<Func<TSource, bool>> predicate);
524     /// <summary>Returns an <see cref="T:System.Int64"></see> that represents the
           total number of elements in a sequence.</summary>
525     /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see>
          that contains the elements to be counted.</param>
526     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
527     /// <returns>The number of elements in <paramref name="source">source</
          paramref>.</returns>
528     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
529     /// <exception cref="T:System.OverflowException">The number of elements
          exceeds <see cref="F:System.Int64.MaxValue"></see>.</exception>
530     public static long LongCount<TSource>(this IQueryable<TSource> source);
531     /// <summary>Returns an <see cref="T:System.Int64"></see> that represents the
           number of elements in a sequence that satisfy a condition.</summary>
532     /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see>
          that contains the elements to be counted.</param>
533     /// <param name="predicate">A function to test each element for a
          condition.</param>
534     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
535     /// <returns>The number of elements in <paramref name="source">source</
          paramref> that satisfy the condition in the predicate function.</returns>
536     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="predicate">predicate</
          paramref> is null.</exception>
537     /// <exception cref="T:System.OverflowException">The number of matching
          elements exceeds <see cref="F:System.Int64.MaxValue"></see>.</exception>
538     public static long LongCount<TSource>(this IQueryable<TSource> source,
```

```
        Expression<Func<TSource, bool>> predicate);
539     /// <summary>Returns the maximum value in a generic <see
          cref="T:System.Linq.IQueryable`1"></see>.</summary>
540     /// <param name="source">A sequence of values to determine the maximum of.</
          param>
541     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
542     /// <returns>The maximum value in the sequence.</returns>
543     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
544     public static TSource Max<TSource>(this IQueryable<TSource> source);
545     /// <summary>Invokes a projection function on each element of a generic <see
          cref="T:System.Linq.IQueryable`1"></see> and returns the maximum resulting
          value.</summary>
546     /// <param name="source">A sequence of values to determine the maximum of.</
          param>
547     /// <param name="selector">A projection function to apply to each element.</
          param>
548     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
549     /// <typeparam name="TResult">The type of the value returned by the function
          represented by selector.</typeparam>
550     /// <returns>The maximum value in the sequence.</returns>
551     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
552     public static TResult Max<TSource, TResult>(this IQueryable<TSource> source,
          Expression<Func<TSource, TResult>> selector);
553     /// <summary>Returns the minimum value of a generic <see
          cref="T:System.Linq.IQueryable`1"></see>.</summary>
554     /// <param name="source">A sequence of values to determine the minimum of.</
          param>
555     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
556     /// <returns>The minimum value in the sequence.</returns>
557     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
558     public static TSource Min<TSource>(this IQueryable<TSource> source);
559     /// <summary>Invokes a projection function on each element of a generic <see
          cref="T:System.Linq.IQueryable`1"></see> and returns the minimum resulting
          value.</summary>
560     /// <param name="source">A sequence of values to determine the minimum of.</
          param>
561     /// <param name="selector">A projection function to apply to each element.</
          param>
562     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
563     /// <typeparam name="TResult">The type of the value returned by the function
          represented by selector.</typeparam>
564     /// <returns>The minimum value in the sequence.</returns>
565     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
566     public static TResult Min<TSource, TResult>(this IQueryable<TSource> source,
          Expression<Func<TSource, TResult>> selector);
567     /// <summary>Filters the elements of an <see
```

```
          cref="T:System.Linq.IQueryable"></see> based on a specified type.</summary>
568       /// <param name="source">An <see cref="T:System.Linq.IQueryable"></see> whose
           elements to filter.</param>
569       /// <typeparam name="TResult">The type to filter the elements of the sequence
           on.</typeparam>
570       /// <returns>A collection that contains the elements from <paramref
            name="source">source</paramref> that have type <paramref
            name="TResult">TResult</paramref>.</returns>
571       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> is null.</exception>
572       public static IQueryable<TResult> OfType<TResult>(this IQueryable source);
573       /// <summary>Sorts the elements of a sequence in ascending order according to
           a key.</summary>
574       /// <param name="source">A sequence of values to order.</param>
575       /// <param name="keySelector">A function to extract a key from an element.</
            param>
576       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
577       /// <typeparam name="TKey">The type of the key returned by the function that
            is represented by keySelector.</typeparam>
578       /// <returns>An <see cref="T:System.Linq.IOrderedQueryable`1"></see> whose
            elements are sorted according to a key.</returns>
579       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> or <paramref
            name="keySelector">keySelector</paramref> is null.</exception>
580       public static IOrderedQueryable<TSource> OrderBy<TSource, TKey>(this
            IQueryable<TSource> source, Expression<Func<TSource, TKey>> keySelector);
581       /// <summary>Sorts the elements of a sequence in ascending order by using a
            specified comparer.</summary>
582       /// <param name="source">A sequence of values to order.</param>
583       /// <param name="keySelector">A function to extract a key from an element.</
            param>
584       /// <param name="comparer">An <see
            cref="T:System.Collections.Generic.IComparer`1"></see> to compare keys.</
            param>
585       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
586       /// <typeparam name="TKey">The type of the key returned by the function that
            is represented by keySelector.</typeparam>
587       /// <returns>An <see cref="T:System.Linq.IOrderedQueryable`1"></see> whose
            elements are sorted according to a key.</returns>
588       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> or <paramref
            name="keySelector">keySelector</paramref> or <paramref
            name="comparer">comparer</paramref> is null.</exception>
589       public static IOrderedQueryable<TSource> OrderBy<TSource, TKey>(this
            IQueryable<TSource> source, Expression<Func<TSource, TKey>> keySelector,
            IComparer<TKey> comparer);
590       /// <summary>Sorts the elements of a sequence in descending order according
            to a key.</summary>
591       /// <param name="source">A sequence of values to order.</param>
592       /// <param name="keySelector">A function to extract a key from an element.</
            param>
593       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
```

```
594     /// <typeparam name="TKey">The type of the key returned by the function that is represented by keySelector.</typeparam>
595     /// <returns>An <see cref="T:System.Linq.IOrderedQueryable`1"></see> whose elements are sorted in descending order according to a key.</returns>
596     /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> is null.</exception>
597     public static IOrderedQueryable<TSource> OrderByDescending<TSource, TKey>(this IQueryable<TSource> source, Expression<Func<TSource, TKey>> keySelector);
598     /// <summary>Sorts the elements of a sequence in descending order by using a specified comparer.</summary>
599     /// <param name="source">A sequence of values to order.</param>
600     /// <param name="keySelector">A function to extract a key from an element.</param>
601     /// <param name="comparer">An <see cref="T:System.Collections.Generic.IComparer`1"></see> to compare keys.</param>
602     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
603     /// <typeparam name="TKey">The type of the key returned by the function that is represented by keySelector.</typeparam>
604     /// <returns>An <see cref="T:System.Linq.IOrderedQueryable`1"></see> whose elements are sorted in descending order according to a key.</returns>
605     /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="keySelector">keySelector</paramref> or <paramref name="comparer">comparer</paramref> is null.</exception>
606     public static IOrderedQueryable<TSource> OrderByDescending<TSource, TKey>(this IQueryable<TSource> source, Expression<Func<TSource, TKey>> keySelector, IComparer<TKey> comparer);
607     /// <param name="source"></param>
608     /// <param name="element"></param>
609     /// <typeparam name="TSource"></typeparam>
610     /// <returns></returns>
611     public static IQueryable<TSource> Prepend<TSource>(this IQueryable<TSource> source, TSource element);
612     /// <summary>Inverts the order of the elements in a sequence.</summary>
613     /// <param name="source">A sequence of values to reverse.</param>
614     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
615     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> whose elements correspond to those of the input sequence in reverse order.</returns>
616     /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
617     public static IQueryable<TSource> Reverse<TSource>(this IQueryable<TSource> source);
618     /// <summary>Projects each element of a sequence into a new form.</summary>
619     /// <param name="source">A sequence of values to project.</param>
620     /// <param name="selector">A projection function to apply to each element.</param>
621     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
622     /// <typeparam name="TResult">The type of the value returned by the function represented by selector.</typeparam>
```

```
623     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> whose elements
          are the result of invoking a projection function on each element of
          <paramref name="source">source</paramref>.</returns>
624     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
625     public static IQueryable<TResult> Select<TSource, TResult>(this
          IQueryable<TSource> source, Expression<Func<TSource, TResult>> selector);
626     /// <summary>Projects each element of a sequence into a new form by
          incorporating the element's index.</summary>
627     /// <param name="source">A sequence of values to project.</param>
628     /// <param name="selector">A projection function to apply to each element.</
          param>
629     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
630     /// <typeparam name="TResult">The type of the value returned by the function
          represented by selector.</typeparam>
631     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> whose elements
          are the result of invoking a projection function on each element of
          <paramref name="source">source</paramref>.</returns>
632     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
633     public static IQueryable<TResult> Select<TSource, TResult>(this
          IQueryable<TSource> source, Expression<Func<TSource, int, TResult>>
          selector);
634     /// <summary>Projects each element of a sequence to an <see
          cref="T:System.Collections.Generic.IEnumerable`1"></see> and combines the
          resulting sequences into one sequence.</summary>
635     /// <param name="source">A sequence of values to project.</param>
636     /// <param name="selector">A projection function to apply to each element.</
          param>
637     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
638     /// <typeparam name="TResult">The type of the elements of the sequence
          returned by the function represented by selector.</typeparam>
639     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> whose elements
          are the result of invoking a one-to-many projection function on each
          element of the input sequence.</returns>
640     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
641     public static IQueryable<TResult> SelectMany<TSource, TResult>(this
          IQueryable<TSource> source, Expression<Func<TSource, IEnumerable<TResult>>>
           selector);
642     /// <summary>Projects each element of a sequence to an <see
          cref="T:System.Collections.Generic.IEnumerable`1"></see> and combines the
          resulting sequences into one sequence. The index of each source element is
          used in the projected form of that element.</summary>
643     /// <param name="source">A sequence of values to project.</param>
644     /// <param name="selector">A projection function to apply to each element;
          the second parameter of this function represents the index of the source
          element.</param>
645     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
```

```
646     /// <typeparam name="TResult">The type of the elements of the sequence
          returned by the function represented by selector.</typeparam>
647     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> whose elements
          are the result of invoking a one-to-many projection function on each
          element of the input sequence.</returns>
648     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
649     public static IQueryable<TResult> SelectMany<TSource, TResult>(this
          IQueryable<TSource> source, Expression<Func<TSource, int,
          IEnumerable<TResult>>> selector);
650     /// <summary>Projects each element of a sequence to an <see
          cref="T:System.Collections.Generic.IEnumerable`1"></see> and invokes a
          result selector function on each element therein. The resulting values from
           each intermediate sequence are combined into a single, one-dimensional
          sequence and returned.</summary>
651     /// <param name="source">A sequence of values to project.</param>
652     /// <param name="collectionSelector">A projection function to apply to each
          element of the input sequence.</param>
653     /// <param name="resultSelector">A projection function to apply to each
          element of each intermediate sequence.</param>
654     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
655     /// <typeparam name="TCollection">The type of the intermediate elements
          collected by the function represented by collectionSelector.</typeparam>
656     /// <typeparam name="TResult">The type of the elements of the resulting
          sequence.</typeparam>
657     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> whose elements
          are the result of invoking the one-to-many projection function <paramref
          name="collectionSelector">collectionSelector</paramref> on each element of
          <paramref name="source">source</paramref> and then mapping each of those
          sequence elements and their corresponding <paramref name="source">source</
          paramref> element to a result element.</returns>
658     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref
          name="collectionSelector">collectionSelector</paramref> or <paramref
          name="resultSelector">resultSelector</paramref> is null.</exception>
659     public static IQueryable<TResult> SelectMany<TSource, TCollection, TResult>
          (this IQueryable<TSource> source, Expression<Func<TSource,
          IEnumerable<TCollection>>> collectionSelector, Expression<Func<TSource,
          TCollection, TResult>> resultSelector);
660     /// <summary>Projects each element of a sequence to an <see
          cref="T:System.Collections.Generic.IEnumerable`1"></see> that incorporates
          the index of the source element that produced it. A result selector
          function is invoked on each element of each intermediate sequence, and the
          resulting values are combined into a single, one-dimensional sequence and
          returned.</summary>
661     /// <param name="source">A sequence of values to project.</param>
662     /// <param name="collectionSelector">A projection function to apply to each
          element of the input sequence; the second parameter of this function
          represents the index of the source element.</param>
663     /// <param name="resultSelector">A projection function to apply to each
          element of each intermediate sequence.</param>
```

```
664     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
665     /// <typeparam name="TCollection">The type of the intermediate elements
          collected by the function represented by collectionSelector.</typeparam>
666     /// <typeparam name="TResult">The type of the elements of the resulting
          sequence.</typeparam>
667     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> whose elements
          are the result of invoking the one-to-many projection function <paramref
          name="collectionSelector">collectionSelector</paramref> on each element of
          <paramref name="source">source</paramref> and then mapping each of those
          sequence elements and their corresponding <paramref name="source">source</
          paramref> element to a result element.</returns>
668     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref
          name="collectionSelector">collectionSelector</paramref> or <paramref
          name="resultSelector">resultSelector</paramref> is null.</exception>
669     public static IQueryable<TResult> SelectMany<TSource, TCollection, TResult>
          (this IQueryable<TSource> source, Expression<Func<TSource, int,
          IEnumerable<TCollection>>> collectionSelector, Expression<Func<TSource,
          TCollection, TResult>> resultSelector);
670     /// <summary>Determines whether two sequences are equal by using the default
          equality comparer to compare elements.</summary>
671     /// <param name="source1">An <see cref="T:System.Linq.IQueryable`1"></see>
          whose elements to compare to those of source2.</param>
672     /// <param name="source2">An <see
          cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to
          compare to those of the first sequence.</param>
673     /// <typeparam name="TSource">The type of the elements of the input
          sequences.</typeparam>
674     /// <returns>true if the two source sequences are of equal length and their
          corresponding elements compare equal; otherwise, false.</returns>
675     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source1">source1</paramref> or <paramref name="source2">source2</
          paramref> is null.</exception>
676     public static bool SequenceEqual<TSource>(this IQueryable<TSource> source1,
          IEnumerable<TSource> source2);
677     /// <summary>Determines whether two sequences are equal by using a specified
          <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to
          compare elements.</summary>
678     /// <param name="source1">An <see cref="T:System.Linq.IQueryable`1"></see>
          whose elements to compare to those of source2.</param>
679     /// <param name="source2">An <see
          cref="T:System.Collections.Generic.IEnumerable`1"></see> whose elements to
          compare to those of the first sequence.</param>
680     /// <param name="comparer">An <see
          cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to use to
          compare elements.</param>
681     /// <typeparam name="TSource">The type of the elements of the input
          sequences.</typeparam>
682     /// <returns>true if the two source sequences are of equal length and their
          corresponding elements compare equal; otherwise, false.</returns>
683     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source1">source1</paramref> or <paramref name="source2">source2</
```

```
          paramref> is null.</exception>
684       public static bool SequenceEqual<TSource>(this IQueryable<TSource> source1,
            IEnumerable<TSource> source2, IEqualityComparer<TSource> comparer);
685       /// <summary>Returns the only element of a sequence, and throws an exception
            if there is not exactly one element in the sequence.</summary>
686       /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to
            return the single element of.</param>
687       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
688       /// <returns>The single element of the input sequence.</returns>
689       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> is null.</exception>
690       /// <exception cref="T:System.InvalidOperationException"><paramref
            name="source">source</paramref> has more than one element.</exception>
691       public static TSource Single<TSource>(this IQueryable<TSource> source);
692       /// <summary>Returns the only element of a sequence that satisfies a
            specified condition, and throws an exception if more than one such element
            exists.</summary>
693       /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to
            return a single element from.</param>
694       /// <param name="predicate">A function to test an element for a condition.</
            param>
695       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
696       /// <returns>The single element of the input sequence that satisfies the
            condition in <paramref name="predicate">predicate</paramref>.</returns>
697       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> or <paramref name="predicate">predicate</
            paramref> is null.</exception>
698       /// <exception cref="T:System.InvalidOperationException">No element satisfies
             the condition in <paramref name="predicate">predicate</paramref>.   -or-
             More than one element satisfies the condition in <paramref
            name="predicate">predicate</paramref>.   -or-   The source sequence is
            empty.</exception>
699       public static TSource Single<TSource>(this IQueryable<TSource> source,
            Expression<Func<TSource, bool>> predicate);
700       /// <summary>Returns the only element of a sequence, or a default value if
            the sequence is empty; this method throws an exception if there is more
            than one element in the sequence.</summary>
701       /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to
            return the single element of.</param>
702       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
703       /// <returns>The single element of the input sequence, or default(<paramref
            name="TSource">TSource</paramref>) if the sequence contains no elements.</
            returns>
704       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> is null.</exception>
705       /// <exception cref="T:System.InvalidOperationException"><paramref
            name="source">source</paramref> has more than one element.</exception>
706       public static TSource SingleOrDefault<TSource>(this IQueryable<TSource>
            source);
707       /// <summary>Returns the only element of a sequence that satisfies a
            specified condition or a default value if no such element exists; this
            method throws an exception if more than one element satisfies the
```

```
          condition.</summary>
708       /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to
            return a single element from.</param>
709       /// <param name="predicate">A function to test an element for a condition.</
            param>
710       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
711       /// <returns>The single element of the input sequence that satisfies the
            condition in <paramref name="predicate">predicate</paramref>, or default
            (<paramref name="TSource">TSource</paramref>) if no such element is
            found.</returns>
712       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> or <paramref name="predicate">predicate</
            paramref> is null.</exception>
713       /// <exception cref="T:System.InvalidOperationException">More than one
            element satisfies the condition in <paramref name="predicate">predicate</
            paramref>.</exception>
714       public static TSource SingleOrDefault<TSource>(this IQueryable<TSource>
            source, Expression<Func<TSource, bool>> predicate);
715       /// <summary>Bypasses a specified number of elements in a sequence and then
            returns the remaining elements.</summary>
716       /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to
            return elements from.</param>
717       /// <param name="count">The number of elements to skip before returning the
            remaining elements.</param>
718       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
719       /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
            elements that occur after the specified index in the input sequence.</
            returns>
720       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> is null.</exception>
721       public static IQueryable<TSource> Skip<TSource>(this IQueryable<TSource>
            source, int count);
722       /// <param name="source"></param>
723       /// <param name="count"></param>
724       /// <typeparam name="TSource"></typeparam>
725       /// <returns></returns>
726       public static IQueryable<TSource> SkipLast<TSource>(this IQueryable<TSource>
            source, int count);
727       /// <summary>Bypasses elements in a sequence as long as a specified condition
             is true and then returns the remaining elements.</summary>
728       /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to
            return elements from.</param>
729       /// <param name="predicate">A function to test each element for a
            condition.</param>
730       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
731       /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
            elements from <paramref name="source">source</paramref> starting at the
            first element in the linear series that does not pass the test specified by
             <paramref name="predicate">predicate</paramref>.</returns>
732       /// <exception cref="T:System.ArgumentNullException"><paramref
            name="source">source</paramref> or <paramref name="predicate">predicate</
            paramref> is null.</exception>
```

```
733     public static IQueryable<TSource> SkipWhile<TSource>(this IQueryable<TSource>
            source, Expression<Func<TSource, bool>> predicate);
734     /// <summary>Bypasses elements in a sequence as long as a specified condition
            is true and then returns the remaining elements. The element's index is
          used in the logic of the predicate function.</summary>
735     /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to
          return elements from.</param>
736     /// <param name="predicate">A function to test each element for a condition;
          the second parameter of this function represents the index of the source
          element.</param>
737     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
738     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
          elements from <paramref name="source">source</paramref> starting at the
          first element in the linear series that does not pass the test specified by
            <paramref name="predicate">predicate</paramref>.</returns>
739     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="predicate">predicate</
          paramref> is null.</exception>
740     public static IQueryable<TSource> SkipWhile<TSource>(this IQueryable<TSource>
            source, Expression<Func<TSource, int, bool>> predicate);
741     /// <summary>Computes the sum of a sequence of <see
          cref="T:System.Decimal"></see> values.</summary>
742     /// <param name="source">A sequence of <see cref="T:System.Decimal"></see>
          values to calculate the sum of.</param>
743     /// <returns>The sum of the values in the sequence.</returns>
744     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
745     /// <exception cref="T:System.OverflowException">The sum is larger than <see
          cref="F:System.Decimal.MaxValue"></see>.</exception>
746     public static decimal Sum(this IQueryable<decimal> source);
747     /// <summary>Computes the sum of a sequence of <see cref="T:System.Double"></
          see> values.</summary>
748     /// <param name="source">A sequence of <see cref="T:System.Double"></see>
          values to calculate the sum of.</param>
749     /// <returns>The sum of the values in the sequence.</returns>
750     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
751     public static double Sum(this IQueryable<double> source);
752     /// <summary>Computes the sum of a sequence of <see cref="T:System.Int32"></
          see> values.</summary>
753     /// <param name="source">A sequence of <see cref="T:System.Int32"></see>
          values to calculate the sum of.</param>
754     /// <returns>The sum of the values in the sequence.</returns>
755     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
756     /// <exception cref="T:System.OverflowException">The sum is larger than <see
          cref="F:System.Int32.MaxValue"></see>.</exception>
757     public static int Sum(this IQueryable<int> source);
758     /// <summary>Computes the sum of a sequence of <see cref="T:System.Int64"></
          see> values.</summary>
759     /// <param name="source">A sequence of <see cref="T:System.Int64"></see>
          values to calculate the sum of.</param>
```

```
760     /// <returns>The sum of the values in the sequence.</returns>
761     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
762     /// <exception cref="T:System.OverflowException">The sum is larger than <see
          cref="F:System.Int64.MaxValue"></see>.</exception>
763     public static long Sum(this IQueryable<long> source);
764     /// <summary>Computes the sum of a sequence of nullable <see
          cref="T:System.Decimal"></see> values.</summary>
765     /// <param name="source">A sequence of nullable <see
          cref="T:System.Decimal"></see> values to calculate the sum of.</param>
766     /// <returns>The sum of the values in the sequence.</returns>
767     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
768     /// <exception cref="T:System.OverflowException">The sum is larger than <see
          cref="F:System.Decimal.MaxValue"></see>.</exception>
769     public static decimal? Sum(this IQueryable<decimal?> source);
770     /// <summary>Computes the sum of a sequence of nullable <see
          cref="T:System.Double"></see> values.</summary>
771     /// <param name="source">A sequence of nullable <see
          cref="T:System.Double"></see> values to calculate the sum of.</param>
772     /// <returns>The sum of the values in the sequence.</returns>
773     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
774     public static double? Sum(this IQueryable<double?> source);
775     /// <summary>Computes the sum of a sequence of nullable <see
          cref="T:System.Int32"></see> values.</summary>
776     /// <param name="source">A sequence of nullable <see cref="T:System.Int32"></
          see> values to calculate the sum of.</param>
777     /// <returns>The sum of the values in the sequence.</returns>
778     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
779     /// <exception cref="T:System.OverflowException">The sum is larger than <see
          cref="F:System.Int32.MaxValue"></see>.</exception>
780     public static int? Sum(this IQueryable<int?> source);
781     /// <summary>Computes the sum of a sequence of nullable <see
          cref="T:System.Int64"></see> values.</summary>
782     /// <param name="source">A sequence of nullable <see cref="T:System.Int64"></
          see> values to calculate the sum of.</param>
783     /// <returns>The sum of the values in the sequence.</returns>
784     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
785     /// <exception cref="T:System.OverflowException">The sum is larger than <see
          cref="F:System.Int64.MaxValue"></see>.</exception>
786     public static long? Sum(this IQueryable<long?> source);
787     /// <summary>Computes the sum of a sequence of nullable <see
          cref="T:System.Single"></see> values.</summary>
788     /// <param name="source">A sequence of nullable <see
          cref="T:System.Single"></see> values to calculate the sum of.</param>
789     /// <returns>The sum of the values in the sequence.</returns>
790     /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
791     public static float? Sum(this IQueryable<float?> source);
```

```
792     /// <summary>Computes the sum of a sequence of <see cref="T:System.Single"></see> values.</summary>
793     /// <param name="source">A sequence of <see cref="T:System.Single"></see> values to calculate the sum of.</param>
794     /// <returns>The sum of the values in the sequence.</returns>
795     /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> is null.</exception>
796     public static float Sum(this IQueryable<float> source);
797     /// <summary>Computes the sum of the sequence of <see cref="T:System.Decimal"></see> values that is obtained by invoking a projection function on each element of the input sequence.</summary>
798     /// <param name="source">A sequence of values of type TSource.</param>
799     /// <param name="selector">A projection function to apply to each element.</param>
800     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
801     /// <returns>The sum of the projected values.</returns>
802     /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
803     /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Decimal.MaxValue"></see>.</exception>
804     public static decimal Sum<TSource>(this IQueryable<TSource> source, Expression<Func<TSource, decimal>> selector);
805     /// <summary>Computes the sum of the sequence of <see cref="T:System.Double"></see> values that is obtained by invoking a projection function on each element of the input sequence.</summary>
806     /// <param name="source">A sequence of values of type TSource.</param>
807     /// <param name="selector">A projection function to apply to each element.</param>
808     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
809     /// <returns>The sum of the projected values.</returns>
810     /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
811     public static double Sum<TSource>(this IQueryable<TSource> source, Expression<Func<TSource, double>> selector);
812     /// <summary>Computes the sum of the sequence of <see cref="T:System.Int32"></see> values that is obtained by invoking a projection function on each element of the input sequence.</summary>
813     /// <param name="source">A sequence of values of type TSource.</param>
814     /// <param name="selector">A projection function to apply to each element.</param>
815     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
816     /// <returns>The sum of the projected values.</returns>
817     /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="selector">selector</paramref> is null.</exception>
818     /// <exception cref="T:System.OverflowException">The sum is larger than <see cref="F:System.Int32.MaxValue"></see>.</exception>
819     public static int Sum<TSource>(this IQueryable<TSource> source, Expression<Func<TSource, int>> selector);
820     /// <summary>Computes the sum of the sequence of <see
```

```
          cref="T:System.Int64"></see> values that is obtained by invoking a
          projection function on each element of the input sequence.</summary>
821       /// <param name="source">A sequence of values of type TSource.</param>
822       /// <param name="selector">A projection function to apply to each element.</
          param>
823       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
824       /// <returns>The sum of the projected values.</returns>
825       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
826       /// <exception cref="T:System.OverflowException">The sum is larger than <see
          cref="F:System.Int64.MaxValue"></see>.</exception>
827       public static long Sum<TSource>(this IQueryable<TSource> source,
          Expression<Func<TSource, long>> selector);
828       /// <summary>Computes the sum of the sequence of nullable <see
          cref="T:System.Decimal"></see> values that is obtained by invoking a
          projection function on each element of the input sequence.</summary>
829       /// <param name="source">A sequence of values of type TSource.</param>
830       /// <param name="selector">A projection function to apply to each element.</
          param>
831       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
832       /// <returns>The sum of the projected values.</returns>
833       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
834       /// <exception cref="T:System.OverflowException">The sum is larger than <see
          cref="F:System.Decimal.MaxValue"></see>.</exception>
835       public static decimal? Sum<TSource>(this IQueryable<TSource> source,
          Expression<Func<TSource, decimal?>> selector);
836       /// <summary>Computes the sum of the sequence of nullable <see
          cref="T:System.Double"></see> values that is obtained by invoking a
          projection function on each element of the input sequence.</summary>
837       /// <param name="source">A sequence of values of type TSource.</param>
838       /// <param name="selector">A projection function to apply to each element.</
          param>
839       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
840       /// <returns>The sum of the projected values.</returns>
841       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
842       public static double? Sum<TSource>(this IQueryable<TSource> source,
          Expression<Func<TSource, double?>> selector);
843       /// <summary>Computes the sum of the sequence of nullable <see
          cref="T:System.Int32"></see> values that is obtained by invoking a
          projection function on each element of the input sequence.</summary>
844       /// <param name="source">A sequence of values of type TSource.</param>
845       /// <param name="selector">A projection function to apply to each element.</
          param>
846       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
847       /// <returns>The sum of the projected values.</returns>
848       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
```

```
          paramref> is null.</exception>
849       /// <exception cref="T:System.OverflowException">The sum is larger than <see
          cref="F:System.Int32.MaxValue"></see>.</exception>
850       public static int? Sum<TSource>(this IQueryable<TSource> source,
          Expression<Func<TSource, int?>> selector);
851       /// <summary>Computes the sum of the sequence of nullable <see
          cref="T:System.Int64"></see> values that is obtained by invoking a
          projection function on each element of the input sequence.</summary>
852       /// <param name="source">A sequence of values of type TSource.</param>
853       /// <param name="selector">A projection function to apply to each element.</
          param>
854       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
855       /// <returns>The sum of the projected values.</returns>
856       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
857       /// <exception cref="T:System.OverflowException">The sum is larger than <see
          cref="F:System.Int64.MaxValue"></see>.</exception>
858       public static long? Sum<TSource>(this IQueryable<TSource> source,
          Expression<Func<TSource, long?>> selector);
859       /// <summary>Computes the sum of the sequence of nullable <see
          cref="T:System.Single"></see> values that is obtained by invoking a
          projection function on each element of the input sequence.</summary>
860       /// <param name="source">A sequence of values of type TSource.</param>
861       /// <param name="selector">A projection function to apply to each element.</
          param>
862       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
863       /// <returns>The sum of the projected values.</returns>
864       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
865       public static float? Sum<TSource>(this IQueryable<TSource> source,
          Expression<Func<TSource, float?>> selector);
866       /// <summary>Computes the sum of the sequence of <see
          cref="T:System.Single"></see> values that is obtained by invoking a
          projection function on each element of the input sequence.</summary>
867       /// <param name="source">A sequence of values of type TSource.</param>
868       /// <param name="selector">A projection function to apply to each element.</
          param>
869       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
870       /// <returns>The sum of the projected values.</returns>
871       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="selector">selector</
          paramref> is null.</exception>
872       public static float Sum<TSource>(this IQueryable<TSource> source,
          Expression<Func<TSource, float>> selector);
873       /// <summary>Returns a specified number of contiguous elements from the start
           of a sequence.</summary>
874       /// <param name="source">The sequence to return elements from.</param>
875       /// <param name="count">The number of elements to return.</param>
876       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
877       /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
```

```
          the specified number of elements from the start of <paramref
          name="source">source</paramref>.</returns>
878       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> is null.</exception>
879       public static IQueryable<TSource> Take<TSource>(this IQueryable<TSource>
          source, int count);
880       /// <param name="source"></param>
881       /// <param name="count"></param>
882       /// <typeparam name="TSource"></typeparam>
883       /// <returns></returns>
884       public static IQueryable<TSource> TakeLast<TSource>(this IQueryable<TSource>
          source, int count);
885       /// <summary>Returns elements from a sequence as long as a specified
          condition is true.</summary>
886       /// <param name="source">The sequence to return elements from.</param>
887       /// <param name="predicate">A function to test each element for a
          condition.</param>
888       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
889       /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
          elements from the input sequence occurring before the element at which the
          test specified by <paramref name="predicate">predicate</paramref> no longer
           passes.</returns>
890       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="predicate">predicate</
          paramref> is null.</exception>
891       public static IQueryable<TSource> TakeWhile<TSource>(this IQueryable<TSource>
           source, Expression<Func<TSource, bool>> predicate);
892       /// <summary>Returns elements from a sequence as long as a specified
          condition is true. The element's index is used in the logic of the
          predicate function.</summary>
893       /// <param name="source">The sequence to return elements from.</param>
894       /// <param name="predicate">A function to test each element for a condition;
          the second parameter of the function represents the index of the element in
           the source sequence.</param>
895       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
896       /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
          elements from the input sequence occurring before the element at which the
          test specified by <paramref name="predicate">predicate</paramref> no longer
           passes.</returns>
897       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref name="predicate">predicate</
          paramref> is null.</exception>
898       public static IQueryable<TSource> TakeWhile<TSource>(this IQueryable<TSource>
           source, Expression<Func<TSource, int, bool>> predicate);
899       /// <summary>Performs a subsequent ordering of the elements in a sequence in
          ascending order according to a key.</summary>
900       /// <param name="source">An <see cref="T:System.Linq.IOrderedQueryable`1"></
          see> that contains elements to sort.</param>
901       /// <param name="keySelector">A function to extract a key from each
          element.</param>
902       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
903       /// <typeparam name="TKey">The type of the key returned by the function
```

```
          represented by keySelector.</typeparam>
904       /// <returns>An <see cref="T:System.Linq.IOrderedQueryable`1"></see> whose
          elements are sorted according to a key.</returns>
905       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref
          name="keySelector">keySelector</paramref> is null.</exception>
906       public static IOrderedQueryable<TSource> ThenBy<TSource, TKey>(this
          IOrderedQueryable<TSource> source, Expression<Func<TSource, TKey>>
          keySelector);
907       /// <summary>Performs a subsequent ordering of the elements in a sequence in
          ascending order by using a specified comparer.</summary>
908       /// <param name="source">An <see cref="T:System.Linq.IOrderedQueryable`1"></
          see> that contains elements to sort.</param>
909       /// <param name="keySelector">A function to extract a key from each
          element.</param>
910       /// <param name="comparer">An <see
          cref="T:System.Collections.Generic.IComparer`1"></see> to compare keys.</
          param>
911       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
912       /// <typeparam name="TKey">The type of the key returned by the function
          represented by keySelector.</typeparam>
913       /// <returns>An <see cref="T:System.Linq.IOrderedQueryable`1"></see> whose
          elements are sorted according to a key.</returns>
914       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref
          name="keySelector">keySelector</paramref> or <paramref
          name="comparer">comparer</paramref> is null.</exception>
915       public static IOrderedQueryable<TSource> ThenBy<TSource, TKey>(this
          IOrderedQueryable<TSource> source, Expression<Func<TSource, TKey>>
          keySelector, IComparer<TKey> comparer);
916       /// <summary>Performs a subsequent ordering of the elements in a sequence in
          descending order, according to a key.</summary>
917       /// <param name="source">An <see cref="T:System.Linq.IOrderedQueryable`1"></
          see> that contains elements to sort.</param>
918       /// <param name="keySelector">A function to extract a key from each
          element.</param>
919       /// <typeparam name="TSource">The type of the elements of source.</typeparam>
920       /// <typeparam name="TKey">The type of the key returned by the function
          represented by keySelector.</typeparam>
921       /// <returns>An <see cref="T:System.Linq.IOrderedQueryable`1"></see> whose
          elements are sorted in descending order according to a key.</returns>
922       /// <exception cref="T:System.ArgumentNullException"><paramref
          name="source">source</paramref> or <paramref
          name="keySelector">keySelector</paramref> is null.</exception>
923       public static IOrderedQueryable<TSource> ThenByDescending<TSource, TKey>(this
           IOrderedQueryable<TSource> source, Expression<Func<TSource, TKey>>
          keySelector);
924       /// <summary>Performs a subsequent ordering of the elements in a sequence in
          descending order by using a specified comparer.</summary>
925       /// <param name="source">An <see cref="T:System.Linq.IOrderedQueryable`1"></
          see> that contains elements to sort.</param>
926       /// <param name="keySelector">A function to extract a key from each
```

```
        element.</param>
927     /// <param name="comparer">An <see
        cref="T:System.Collections.Generic.IComparer`1"></see> to compare keys.</
        param>
928     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
929     /// <typeparam name="TKey">The type of the key that is returned by the
        keySelector function.</typeparam>
930     /// <returns>A collection whose elements are sorted in descending order
        according to a key.</returns>
931     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source">source</paramref> or <paramref
        name="keySelector">keySelector</paramref> or <paramref
        name="comparer">comparer</paramref> is null.</exception>
932     public static IOrderedQueryable<TSource> ThenByDescending<TSource, TKey>(this
         IOrderedQueryable<TSource> source, Expression<Func<TSource, TKey>>
        keySelector, IComparer<TKey> comparer);
933     /// <summary>Produces the set union of two sequences by using the default
        equality comparer.</summary>
934     /// <param name="source1">A sequence whose distinct elements form the first
        set for the union operation.</param>
935     /// <param name="source2">A sequence whose distinct elements form the second
        set for the union operation.</param>
936     /// <typeparam name="TSource">The type of the elements of the input
        sequences.</typeparam>
937     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
        the elements from both input sequences, excluding duplicates.</returns>
938     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source1">source1</paramref> or <paramref name="source2">source2</
        paramref> is null.</exception>
939     public static IQueryable<TSource> Union<TSource>(this IQueryable<TSource>
        source1, IEnumerable<TSource> source2);
940     /// <summary>Produces the set union of two sequences by using a specified
        <see cref="T:System.Collections.Generic.IEqualityComparer`1"></see>.</
        summary>
941     /// <param name="source1">A sequence whose distinct elements form the first
        set for the union operation.</param>
942     /// <param name="source2">A sequence whose distinct elements form the second
        set for the union operation.</param>
943     /// <param name="comparer">An <see
        cref="T:System.Collections.Generic.IEqualityComparer`1"></see> to compare
        values.</param>
944     /// <typeparam name="TSource">The type of the elements of the input
        sequences.</typeparam>
945     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains
        the elements from both input sequences, excluding duplicates.</returns>
946     /// <exception cref="T:System.ArgumentNullException"><paramref
        name="source1">source1</paramref> or <paramref name="source2">source2</
        paramref> is null.</exception>
947     public static IQueryable<TSource> Union<TSource>(this IQueryable<TSource>
        source1, IEnumerable<TSource> source2, IEqualityComparer<TSource>
        comparer);
948     /// <summary>Filters a sequence of values based on a predicate.</summary>
```

```
949     /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to filter.</param>
950     /// <param name="predicate">A function to test each element for a condition.</param>
951     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
952     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains elements from the input sequence that satisfy the condition specified by <paramref name="predicate">predicate</paramref>.</returns>
953     /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
954     public static IQueryable<TSource> Where<TSource>(this IQueryable<TSource> source, Expression<Func<TSource, bool>> predicate);
955     /// <summary>Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.</summary>
956     /// <param name="source">An <see cref="T:System.Linq.IQueryable`1"></see> to filter.</param>
957     /// <param name="predicate">A function to test each element for a condition; the second parameter of the function represents the index of the element in the source sequence.</param>
958     /// <typeparam name="TSource">The type of the elements of source.</typeparam>
959     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains elements from the input sequence that satisfy the condition specified by <paramref name="predicate">predicate</paramref>.</returns>
960     /// <exception cref="T:System.ArgumentNullException"><paramref name="source">source</paramref> or <paramref name="predicate">predicate</paramref> is null.</exception>
961     public static IQueryable<TSource> Where<TSource>(this IQueryable<TSource> source, Expression<Func<TSource, int, bool>> predicate);
962     /// <summary>Merges two sequences by using the specified predicate function.</summary>
963     /// <param name="source1">The first sequence to merge.</param>
964     /// <param name="source2">The second sequence to merge.</param>
965     /// <param name="resultSelector">A function that specifies how to merge the elements from the two sequences.</param>
966     /// <typeparam name="TFirst">The type of the elements of the first input sequence.</typeparam>
967     /// <typeparam name="TSecond">The type of the elements of the second input sequence.</typeparam>
968     /// <typeparam name="TResult">The type of the elements of the result sequence.</typeparam>
969     /// <returns>An <see cref="T:System.Linq.IQueryable`1"></see> that contains merged elements of two input sequences.</returns>
970     /// <exception cref="T:System.ArgumentNullException"><paramref name="source1">source1</paramref> or <paramref name="source2">source2</paramref> is null.</exception>
971     public static IQueryable<TResult> Zip<TFirst, TSecond, TResult>(this IQueryable<TFirst> source1, IEnumerable<TSecond> source2,
        Expression<Func<TFirst, TSecond, TResult>> resultSelector);
972   }
973 }
974
```