Power Query (M): Introduction to Comparer functions

There are four standard comparer functions

Name	Description
Comparer.Equals	Returns a logical value based on the equality check over the two given values.
Comparer.FromCulture	Returns a comparer function given the culture and a logical value for case sensitivity for the comparison. The default value for ignoreCase is false. The value for culture are well known text representations of locales used in the .NET framework.
Comparer.Ordinal	Returns a comparer function which uses Ordinal rules to compare values.
Comparer.OrdinalIgnoreCase	Returns a case-insensitive comparer function which uses Ordinal rules to compare the provided values x and y.

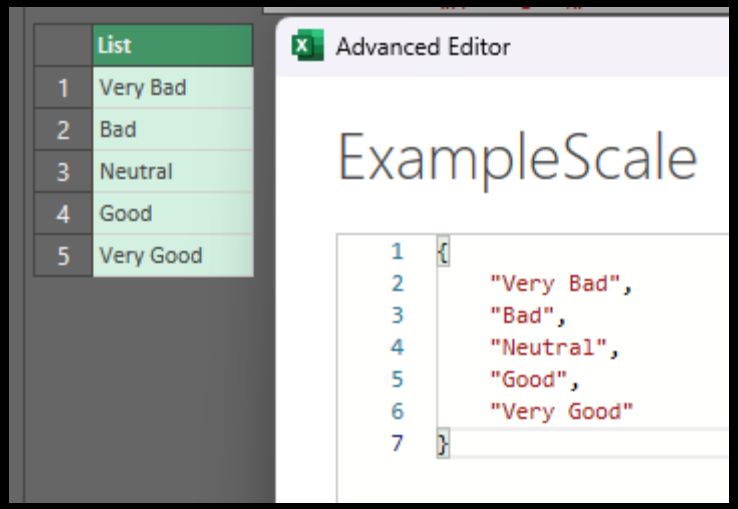
https://learn.microsoft.com/en-us/powerquery-m/comparer-functions

These slides will look at two of them

Name	Description
Comparer.Equals	Returns a logical value based on the equality check over the two given values.
Comparer.FromCulture	Returns a comparer function given the culture and a logical value for case sensitivity for the comparison. The default value for ignoreCase is false. The value for culture are well known text representations of locales used in the .NET framework.
Comparer.Ordinal	Returns a comparer function which uses Ordinal rules to compare values.
Comparer.OrdinalIgnoreCase	Returns a case-insensitive comparer function which uses Ordinal rules to compare the provided values x and y.

https://learn.microsoft.com/en-us/powerquery-m/comparer-functions

Suppose we have a likert scale of 5 values



And we want to compare each value with each other value

	Item1	Item2
1	Very Bad	Very Bad
2	Very Bad	Bad
3	Very Bad	Neutral
4	Very Bad	Good
5	Very Bad	Very Good
6	Bad	Very Bad
7	Bad	Bad
8	Bad	Neutral
9	Bad	Good
10	Bad	Very Good
11	Neutral	Very Bad
12	Neutral	Bad
13	Neutral	Neutral
14	Neutral	Good
15	Neutral	Very Good
16	Good	Very Bad
17	Good	Bad
18	Good	Neutral
19	Good	Good
20	Good	Very Good
21	Very Good	Very Bad
22	Very Good	Bad
23	Very Good	Neutral
24	Very Good	Good
25	Very Good	Very Good

Advanced Editor

CrossJoinData

Display

```
let
    AsTable = Table.FromColumns({ExampleScale},{"Item1"}),
    CrossJoinScale = Table.AddColumn(AsTable, "Subtable", each AsTable),
    Expanded = Table.ExpandTableColumn(CrossJoinScale, "Subtable", {"Item1"}, {"Item2"})
in
    Expanded
```

Comparer.Ordinal

```
= Table.AddColumn(
        CrossJoinData,
        "Comparer.Ordinal",
        each Comparer.Ordinal( [Item1] , [Item2] )
    )
```

	Item1	Item2	Comparer.Ordinal
1	Very Bad	Very Bad	0
2	Very Bad	Bad	1
3	Very Bad	Neutral	1
4	Very Bad	Good	1
5	Very Bad	Very Good	-1
6	Bad	Very Bad	-1
7	Bad	Bad	0
8	Bad	Neutral	-1
9	Bad	Good	-1
10	Bad	Very Good	-1
11	Neutral	Very Bad	-1
12	Neutral	Bad	1
13	Neutral	Neutral	0
14	Neutral	Good	1
15	Neutral	Very Good	-1
16	Good	Very Bad	-1
17	Good	Bad	1
18	Good	Neutral	-1
19	Good	Good	0
20	Good	Very Good	-1
21	Very Good	Very Bad	1
22	Very Good	Bad	1
23	Very Good	Neutral	1
24	Very Good	Good	1
25	Very Good	Very Good	0

We pass two values to the function, and it compares them using “ordinal rules”

If the two values are the same, the function returns 0

If the first value comes after the second value, the function returns 1

If the first value comes before the second value, the function returns -1

Comparer.Ordinal

If we try to compare text and number, we get an error

```
= Table.AddColumn(
        Source,
        "Comparer.Ordinal",
        each Comparer.Ordinal( [Item1] , 1 )
    )
```

	Item1	Comparer.Ordinal
1	Very Bad	Error
2	Bad	Error
3	Neutral	Error
4	Good	Error
5	Very Good	Error

Expression.Error: We cannot apply operator < to types Text and Number.
Details:
Operator=<
Left=Very Bad
Right=1

Comparer.Ordinal – compare UC with “num”

```
= Table.AddColumn(
            Source,
            "Comparer.Ordinal",
            each Comparer.Ordinal( [Item1] , "1" )
        )
```

	Item1	Comparer.Ordinal
1	Very Bad	1
2	Bad	1
3	Neutral	1
4	Good	1
5	Very Good	1

If we compare any of the scale items with “1” (one as text), then Item1 comes *after* the text representation of 1 – the function returns 1

Comparer.Ordinal – compare UC with LC

```
= Table.AddColumn(
            Source,
            "Comparer.Ordinal",
            each Comparer.Ordinal( [Item1] , "a" )
        )
```

	Item1	Comparer.Ordinal
1	Very Bad	-1
2	Bad	-1
3	Neutral	-1
4	Good	-1
5	Very Good	-1

If we compare any of the scale items with “a” (lower case a), then Item1 comes *before* that letter – the function returns -1

Comparer.Ordinal – ordinal rules

When comparing case-sensitive text using Comparer.Ordinal

- Upper case comes before lower case
- Numbers come before letters

{“0”..”9”} < {“A”..”Z”} < {“a”..”z”}

Comparer.OrdinalIgnoreCase

	Item1	Item2	Comparer.Ordinal	Comparer.OrdinalIgnoreCase
1	Very Bad	very bad	-1	0
2	Very Bad	bad	-1	1
3	Very Bad	neutral	-1	1
4	Very Bad	good	-1	1
5	Very Bad	very good	-1	-1
6	Bad	very bad	-1	-1
7	Bad	bad	-1	0
8	Bad	neutral	-1	-1
9	Bad	good	-1	-1
10	Bad	very good	-1	-1
11	Neutral	very bad	-1	-1
12	Neutral	bad	-1	1
13	Neutral	neutral	-1	0
14	Neutral	good	-1	1
15	Neutral	very good	-1	-1
16	Good	very bad	-1	-1
17	Good	bad	-1	1
18	Good	neutral	-1	-1
19	Good	good	-1	0
20	Good	very good	-1	-1
21	Very Good	very bad	-1	1
22	Very Good	bad	-1	1
23	Very Good	neutral	-1	1
24	Very Good	good	-1	1
25	Very Good	very good	-1	0

If item2 is lower case and we use OrdinalIgnoreCase, we get the same results as previously because it's only considering alphabetical order

If the two values are the same, the function returns 0

If the first value comes after the second value, the function returns 1

If the first value comes before the second value, the function returns -1

Comparer uses

These comparers can be used as the ‘equationCriteria’ parameter to control comparison behavior in functions like

- List.Distinct
- List.Contains
- List.ContainsAny

...and others

Custom Comparer - code

```
(scale as list) as function =>

    (x as any,y as any) as number

        =>  Comparer.OrdinalIgnoreCase(
                List.PositionOf(scale,x),
                List.PositionOf(scale,y)
            )

/*
x as any, y as any
if x > y then 1
if x < y then -1
if x = y then 0
*/
```

We can also create custom comparers

Here, we pass an ordered list as the scale parameter

This creates a function of x and y

The function returns the comparison of the positions of x and y in the ordered list

Custom Comparer - demo

	Item1	Item2	ComparerScaleList
1	Very Bad	Very Bad	0
2	Very Bad	Bad	-1
3	Very Bad	Neutral	-1
4	Very Bad	Good	-1
5	Very Bad	Very Good	-1
6	Bad	Very Bad	1
7	Bad	Bad	0
8	Bad	Neutral	-1
9	Bad	Good	-1
10	Bad	Very Good	-1
11	Neutral	Very Bad	1
12	Neutral	Bad	1
13	Neutral	Neutral	0
14	Neutral	Good	-1
15	Neutral	Very Good	-1
16	Good	Very Bad	1
17	Good	Bad	1
18	Good	Neutral	1
19	Good	Good	0
20	Good	Very Good	-1
21	Very Good	Very Bad	1
22	Very Good	Bad	1
23	Very Good	Neutral	1
24	Very Good	Good	1
25	Very Good	Very Good	0

Using this custom comparer recognizes that “Very bad” comes after everything else and “Very Good” comes before everything else

ExampleScale

	List
1	Very Bad
2	Bad
3	Neutral
4	Good
5	Very Good

```
let
    Source = CrossJoinData,
    MyComparer = ComparerScaleList(ExampleScale),
    Result = Table.AddColumn(
                Source,
                "ComparerScaleList",
                each MyComparer([Item1],[Item2])
            )
in
    Result
```

Custom Comparer - demo

```
let
    MyComparer = ComparerScaleList(ExampleScale),

    Converter
        = (i as number) as text
            => let middleText = if i = -1 then "worse than"
                                else if i = 0 then "the same as"
                                else if i = 1 then "better than"
                                else "not compared with"
                in "Q1 was " & middleText & " Q2",


    Result = Table.AddColumn(Source,"Change",each Converter(MyComparer([PerformanceQ1],[PerformanceQ2])))
in
    Result
```

By using the custom comparer, we've simplified the comparison of scaled items

	RespID	PerformanceQ1	PerformanceQ2	Change
1	1	Very Bad	Good	Q1 was worse than Q2
2	2	Bad	Bad	Q1 was the same as Q2
3	3	Bad	Good	Q1 was worse than Q2
4	4	Bad	Bad	Q1 was the same as Q2
5	5	Bad	Neutral	Q1 was worse than Q2
6	6	Neutral	Very Bad	Q1 was better than Q2
7	7	Very Good	Bad	Q1 was better than Q2
8	8	Neutral	Good	Q1 was worse than Q2
9	9	Very Bad	Very Good	Q1 was worse than Q2
10	10	Very Bad	Bad	Q1 was worse than Q2