Power Query (M):

Custom functions with function parameters

Suppose we have these data

	Alpha1	Alpha2
1	A	H
2	A	B
3	A	B
4	D	null
5	D	null
6	null	C
7	F	C
8	F	C
9	F	C
10	G	E
11	G	A
12	C	A
13	C	A
14	C	A
15	C	null

And we want to consolidate the two columns

	Alpha1	Alpha2
1	A	H
2	A	B
3	A	B
4	D	null
5	D	null
6	null	C
7	F	C
8	F	C
9	F	C
10	G	E
11	G	A
12	C	A
13	C	A
14	C	A
15	C	null

Output each unique letter the maximum number of times it appears in either column.

A appears 4 times in Alpha2 and 3 times in Alpha1, so A should appear 4 times in the output.

F appears 3 times in Alpha1 and 0 times in Alpha2, so it should appear 3 times in the output.

First solution (1/4)

```
let
    //Create a list of lists from the columns in the table
    ColumnsAsLists = Table.ToColumns(Source),

    //Combine the columns into a single list
    Combined = List.Combine(ColumnsAsLists),

    //Create a unique list of the letters in the Combined list
    Letters = {List.Min(Combined)..List.Max(Combined)},

    /*
    For each letter in the combined list:
        1: Get the count of that letter in the first column
        2: Get the count of that letter in the second column
        3: Get the max of the two counts, call it MaxCount
        4: Create a list by repeating that letter MaxCount times
    */
    Transform = List.Transform(
        Letters,
        (Letter) =>
            let L1 = List.Count(List.Select(Source[Alpha1], each _ = Letter)),
                L2 = List.Count(List.Select(Source[Alpha2], each _ = Letter)),
                MaxCount = List.Max({L1,L2})
            in
                List.Repeat({Letter},MaxCount)
    ),

    //Combine the list of lists into a single list
    Result = List.Combine(Transform)
in
    Result
```

First solution (2/4)

```
let
    //Create a list of lists from the columns in the table
    ColumnsAsLists = Table.ToColumns(Source),

    //Combine the columns into a single list
    Combined = List.Combine(ColumnsAsLists),

    //Create a unique list of the letters in the Combined list
    Letters = {List.Min(Combined)..List.Max(Combined)},

```

Creates a list of lists. Each sub-list is one column from the table

	List
1	List
2	List

List
A
A
A
D
D
null
F
F
F
G

Combines a list of lists into a single list

	List
1	A
2	A
3	A
4	D
5	D
6	*null*
7	F
8	F
9	F
10	G
11	G
12	C
13	C
14	C
15	C
16	H
17	B

Creates a list between whatever's before the dots and whatever's after the dots

	List
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H

First solution (3/4)

```
/*
For each letter in the combined list:
    1: Get the count of that letter in the first column
    2: Get the count of that letter in the second column
    3: Get the max of the two counts, call it MaxCount
    4: Create a list by repeating that letter MaxCount times
*/
Transform = List.Transform(
    Letters,
    (Letter) =>
        let L1 = List.Count(List.Select(Source[Alpha1], each _ = Letter)),
            L2 = List.Count(List.Select(Source[Alpha2], each _ = Letter)),
            MaxCount = List.Max({L1,L2})
        in
            List.Repeat({Letter},MaxCount)
```

Transforms the list in the first parameter by applying the function in the second parameter to each list element in turn

Select the items from the column that are equal to the current letter being transformed, then count the result

Calculate the maximum of the counts from both columns

Create a list by repeating the current letter MaxCount times

	List
1	List
2	List
3	List
4	List
5	List
6	List
7	List
8	List

List
A
A
A
A

The result of this List.Transform is a list of lists where each sub-list is one of the letters in the Source table, repeated the maximum count found for that letter in both columns

First solution (4/4)

```
let
    //Create a list of lists from the columns in the table
    ColumnsAsLists = Table.ToColumns(Source),

    //Combine the columns into a single list
    Combined = List.Combine(ColumnsAsLists),

    //Create a unique list of the letters in the Combined list
    Letters = {List.Min(Combined)..List.Max(Combined)},

    /*
    For each letter in the combined list:
        1: Get the count of that letter in the first column
        2: Get the count of that letter in the second column
        3: Get the max of the two counts, call it MaxCount
        4: Create a list by repeating that letter MaxCount times
    */
    Transform = List.Transform(
        Letters,
        (Letter) =>
            let L1 = List.Count(List.Select(Source[Alpha1], each _ = Letter)),
                L2 = List.Count(List.Select(Source[Alpha2], each _ = Letter)),
                MaxCount = List.Max({L1,L2})
            in
                List.Repeat({Letter},MaxCount)
    ),

    //Combine the list of lists into a single list
    Result = List.Combine(Transform)
in
    Result
```

Finally, we combine into a single list

	List
1	A
2	A
3	A
4	A
5	B
6	B
7	C
8	C
9	C
10	C
11	D
12	D
13	E
14	F
15	F
16	F
17	G
18	G
19	H

But wait!

	Alpha1	Alpha2
1	A	H
2	A	B
3	A	B
4	D	null
5	D	null
6	null	C
7	F	C
8	F	C
9	F	C
10	G	E
11	G	A
12	C	A
13	C	A
14	C	A
15	C	null

What if we wanted more flexibility?

What if we wanted the minimum count instead of the maximum count?

Or what if we wanted to repeat A by the minimum non-zero count of B from both columns? And B by the minimum non-zero count of C (and so on)?

Second solution (1/8) - full query

```
let
    /*
    Gets an optionally filtered statistic from both lists, e.g.:
        f(List.Min) returns the minimum letter from both lists
        f(List.Max) returns the maximum letter from both lists
        f(List.Max, List.Count) returns the maximum count of both lists
        f(List.Max, List.Count, each _="A") returns the maximum count of "A" in both lists
        etc.
    */
    f = ( agg1 as function, optional agg2 as function, optional filter as function ) as any
            =>  let
                    ifnull = (arg as nullable function,thenpart as function) as function
                                => if arg = null then thenpart else arg,

                    selector = (filter as function) as function => (col as list) as any
                                => ifnull(agg2,agg1)(List.Select(col,filter)),

                    s = selector(ifnull(filter,each true))
                in
                    agg1( List.Select({ s(Source[Alpha1]), s(Source[Alpha2]) }, each _ <> 0) ),

    //Transform list of unique letters by repeating each by the maximum count of that letter in both lists
    Lt = List.Transform( {f(List.Min)..f(List.Max)} , (a) => let c = f(List.Max,List.Count,each _=a) in List.Repeat({a},c) ),

    //combine to a single list
    Result = List.Combine(Lt)
in
    Result
```

Second solution (2/8) – parameters of f

f takes three parameters:

agg1 is a function and is intended to be something like List.Max or List.Min

agg2 is a function and can either be a function or it can be omitted

filter is a function intended to filter the rows in each column by some arbitrary criteria

```
10    f = ( agg1 as function, optional agg2 as function, optional filter as function ) as any
```

So, if we want to get the “Max count of each letter”, we can do this:

f(List.Max , List.Count , each _ = a)

(where “a” is the current letter being transformed)

Or if we wanted the minimum non-zero count of the *next* letter (repeat “A” by minimum non-zero count of “B”), we could do this:

f(List.Min , List.Count , each _ = Letters{List.PositionOf(Letters,a)+1}?)

Second solution (3/8) – ifnull explanation

```
f = ( agg1 as function, optional agg2 as function, optional filter as function ) as any
    => let
        ifnull = (arg as nullable function, thenpart as function) as function
            => if arg = null then thenpart else arg,
```

Since we have some optional arguments in f, we need some way of providing defaults. ifnull is a function that returns an alternative function if the first function is null (i.e. omitted).

So, if agg2 from f is omitted, we can give it a default of "the same function as agg1" by doing this:

ifnull(agg2,agg1)

Which means that if agg1 is List.Max and agg2 is null, then the above expression is *functionally equivalent* to List.Max and works and can be used in the same way.

Second solution (4/8) – ifnull demo

```
let
    f = ( agg1 as function, optional agg2 as function) as any
        => let
            ifnull = (arg as nullable function,thenpart as function) as function
                => if arg = null then thenpart else arg
        in
            ifnull(agg2,agg1)({1,2,3,4,5})
in
    Table.FromColumns(
        {
            { "agg1=List.Max, agg2=null",   "agg1=List.Max, agg2=List.Min"},
            { f(List.Max),                  f(List.Max,List.Min) }
        }
    )
```

ifnull is used where we would normally put a List.* function

	Column1	Column2
1	agg1=List.Max, agg2=null	5
2	agg1=List.Max, agg2=List.Min	1

Second solution (5/8) – selector explanation

```
10      f = ( agg1 as function, optional agg2 as function, optional filter as function ) as any
11              =>  let
12                      ifnull = (arg as nullable function, thenpart as function) as function
13                              => if arg = null then thenpart else arg,
14
15                      selector = (filter as function) as function
16                              => (col as list) as any => ifnull( agg2 , agg1 )( List.Select( col , filter ) ),
17
```

The selector function accepts a function such as each _ = Letter and returns a function with one parameter – col. This function then selects those list items from col defined by the filter function and applies ifnull(agg2,agg1) to the resulting list. So, this:

```
18                      s = selector(ifnull(filter,each true))
```

each true is the same as "no filter"

says "If the filter parameter from f is null, pass each true, otherwise pass filter, into the selector function."

The function returned by selector and assigned to s then uses that filter parameter in the second parameter of List.Select to select the elements to be aggregated by ifnull(agg2,agg1)

Second solution (6/8) – f demo

```
let
    f = ( agg1 as function, optional agg2 as function, optional filter as function ) as any
            => let
                    ifnull = (arg as nullable function,thenpart as function) as function => if arg = null then thenpart else arg,

                    selector = (filter as function) as function => (col as list) as any => ifnull(agg2,agg1)(List.Select(col,filter)),

                    s = selector(ifnull(filter,each true))
                in
                    agg1( List.Select({ s(Source[Alpha1]), s(Source[Alpha2]) }, each _ <> 0) )
in
    Table.FromColumns(
        {
            {"Maximum letter in both columns",
             "Minimum of maximum letters less than G",
             "Maximum count of letter ""A""""},
            {"agg1=List.Max, agg2=null, filter=null",
             "agg1=List.Min, agg2=List.Max, filter=each _ < ""G""",
            "agg1=List.Max, agg2=List.Count, filter=each _=""A""""},
            {f(List.Max),
             f(List.Min,List.Max, each _ < "G"),
             f(List.Max, List.Count, each _ = "A")}
        }
    )
```

	Column1	Column2	Column3
1	Maximum letter in both columns	agg1=List.Max, agg2=null, filter=null	H
2	Minimum of maximum letters less than G	agg1=List.Min, agg2=List.Max, filter=each _ < "G"	E
3	Maximum count of letter "A"	agg1=List.Max, agg2=List.Count, filter=each _="A"	4

Second solution (7/8) – List.Transform step

```
10      f = ( agg1 as function, optional agg2 as function, optional filter as function ) as any
11              =>  let
12                      ifnull = (arg as nullable function, thenpart as function) as function
13                                  => if arg = null then thenpart else arg,
14
15                      selector = (filter as function) as function
16                                  => (col as list) as any => ifnull( agg2 , agg1 )( List.Select( col , filter ) ),
17
18                      s = selector(ifnull(filter,each true))
19                  in
20                      agg1( { s( Source[Alpha1] ), s( Source[Alpha2] ) } ),
21
22      //Transform list of unique letters by repeating each by the maximum count of that letter in both lists
23      Lt = List.Transform(
24              {f(List.Min)..f(List.Max)},
25              (a) => let c = f( List.Max , List.Count , each _ = a ) in List.Repeat( {a} , c )
26          ),
27
```

For each letter between f(List.Min) and f(List.Max) (i.e. min and max letter across both columns), get the maximum count of *that* letter from the two columns using f(List.Max, List.Count, each _ = a), then create a list that repeats that letter that many times.

	List
1	List
2	List
3	List
4	List
5	List
6	List
7	List
8	List

List
A
A
A
A

Second solution (8/8) – List.Transform step

```
10      f = ( agg1 as function, optional agg2 as function, optional filter as function ) as any
11              =>  let
12                      ifnull = (arg as nullable function, thenpart as function) as function
13                                  => if arg = null then thenpart else arg,
14
15                      selector = (filter as function) as function
16                                  => (col as list) as any => ifnull( agg2 , agg1 )( List.Select( col , filter ) ),
17
18                      s = selector(ifnull(filter,each true))
19                  in
20                      agg1( { s( Source[Alpha1] ), s( Source[Alpha2] ) } ),
21
22      //Transform list of unique letters by repeating each by the maximum count of that letter in both lists
23      Lt = List.Transform(
24              {f(List.Min)..f(List.Max)},
25              (a) => let c = f( List.Max , List.Count , each _ = a ) in List.Repeat( {a} , c )
26          ),
27
28      //combine to a single list
29      Result = List.Combine(Lt)
30  in
31      Result
```

Finally combine the resulting list of lists into a single list

	List
1	A
2	A
3	A
4	A
5	B
6	B
7	C
8	C
9	C
10	C
11	D
12	D
13	E
14	F
15	F
16	F
17	G
18	G
19	H

Whew… that was WAY too complicated.

What's the point?

A walkthrough is never a definitive guide nor necessarily a realistic problem. This walkthrough is no different.

The goal of this walkthrough was to show that:

1) We can pass functions as parameters to custom functions
2) If we think in patterns, we can create flexible tools and *reduce re-work* when a spec. changes

One such pattern in this problem was *aggregate of aggregate across two optionally filtered columns*