Power Query (M): Transform column types using a control table

Consider this table of Employees

	A	B	C	D	E
1	table name: Employees				
2	EmployeeKey	FirstName	LastName	HireDate	BirthDate
3	1	Guy	Gilbert	38745	29902
4	2	Kevin	Brown	38955	31747
5	3	Roberto	Tamburello	39244	27192
6	4	Rob	Walters	39268	27233
7	5	Rob	Walters	39268	27233
8	6	Thierry	D'Hers	39274	21607
9	7	David	Bradley	39283	27319
10	8	David	Bradley	39283	27319
11	9	JoLynn	Dobney	39289	20317
12	10	Ruth	Ellerbrock	39300	20457
13	11	Gail	Erickson	39300	19111
14	12	Barry	Johnson	39301	20387
15	13	Jossef	Goldberg	39318	21467
16	14	Terri	Duffy	39325	25993

When we load the table into Power Query, the types are “any” by default

The ABC 123 icon indicates the “any” type. This is the default type for Excel columns that have not had type transformations applied

```
= Excel.CurrentWorkbook(){[Name="Employees"]}[Content]
```

	EmployeeKey	FirstName	LastName	HireDate	BirthDate
1	1	Guy	Gilbert	38745	29902
2	2	Kevin	Brown	38955	31747
3	3	Roberto	Tamburello	39244	27192
4	4	Rob	Walters	39268	27233

```
= Table.Schema(data)
```

	Name	Position	TypeName	Kind
1	EmployeeKey	0	Any.Type	any
2	FirstName	1	Any.Type	any
3	LastName	2	Any.Type	any
4	HireDate	3	Any.Type	any
5	BirthDate	4	Any.Type	any

We can confirm this with the Table.Schema function

If we use Get Data>From/Table Range, PQ will try to guess the data types

The type transformations are hard-coded as an automatically added query step

```
= Table.TransformColumnTypes(Source,{{"EmployeeKey", Int64.Type}, {"FirstName", type text}, {"LastName", type text},
    {"HireDate", Int64.Type}, {"BirthDate", Int64.Type}})
```

	EmployeeKey	FirstName	LastName	HireDate	BirthDate
1	1	Guy	Gilbert	38745	29902
2	2	Kevin	Brown	38955	31747
3	3	Roberto	Tamburello	39244	27192
4	4	Rob	Walters	39268	27233
5	5	Rob	Walters	39268	27233

Unfortunately, they might not always be what we want. This should be a date type

One option we have is to use a control table to force the data types without hard-coding them in the query

table name: EmployeesTypes

ColumnName	Type
EmployeeKey	whole number
FirstName	text
LastName	text
HireDate	date
BirthDate	date

We can list the columns whose types we want to change next to the type we want to use for that column

Then use a custom function to build a list of type transformations from the contents of that table

We pass the control table, and we're given a list of lists, where each sub-list is of the form {"column name", type}

```
(typeTable as table) as list
    =>
    let
        checkColumnNames = Table.ColumnNames(typeTable) = {"ColumnName","Type"},

        primitiveNames = {"true/false","decimal number","time","date","date/time","text"},
        primitives = {type logical, type number, type time, type date, type datetime, type text},

        subtypeNames = {"whole number","currency","percentage"},
        subtypes = {Int64.Type,Currency.Type,Percentage.Type},

        typeNames = primitiveNames & subtypeNames,
        types = primitives & subtypes,

        columnTransforms =
                List.Transform(
                    typeTable[Type],
                    each try types{List.PositionOf(typeNames,Text.Lower(_))} otherwise type any
                ),

        Result
            = List.Zip(
                {
                    typeTable[ColumnName] ,
                    columnTransforms
                }
            )
    in
        if checkColumnNames then Result else {}
```

The function checks the structure of the control table and builds some type lookups

We can directly compare two lists using =. If the table contains these columns only, this check returns true

```
4    checkColumnNames = Table.ColumnNames(typeTable) = {"ColumnName","Type"},
5
6    primitiveNames = {"true/false","decimal number","time","date","date/time","text"},
7    primitives = {type logical, type number, type time, type date, type datetime, type text},
8
9    subtypeNames = {"whole number","currency","percentage"},
10   subtypes = {Int64.Type,Currency.Type,Percentage.Type},
11
12   typeNames = primitiveNames & subtypeNames,
13   types = primitives & subtypes,
14
```

The & operator creates a new list that's the union of two lists

Here we create two lists:

1. typeNames – text representations of common column types, and
2. types - the type values that would be used in Table.TransformColumns

We retrieve the appropriate type value for each column to transform from the types list

We can build the list of column types we want to use by applying List.Transform to the text description of the type in the Type column of the control table

```
14
15          columnTransforms =
16                      List.Transform(
17                          typeTable[Type],
18                          each try types{List.PositionOf(typeNames,Text.Lower(_))} otherwise type any
19                      ),
```

We find the position in the typeNames list of the text in the Type column of the control table *(shown here as the current list item _)*, then use that position to retrieve the type value from the types list. If the type name in the control table is not found, the type used reverts to any

Then return the list of lists or an empty list

```
21        Result
22            = List.Zip(
23                {
24                    typeTable[ColumnName] ,
25                    columnTransforms
26                }
27            )
28    in
29        if checkColumnNames then Result else {}
```

We then use List.Zip to combine the column names with the type values for each column. This is the step that creates the list of lists

Finally, we either return the created list of lists if the typeTable had the correct structure, or an empty list if it didn't, in which case no transformations would be applied

Transforming types is then driven by the control table in the workbook

```
let
    data = Excel.CurrentWorkbook(){[Name="Employees"]}[Content],
    typeControl = Excel.CurrentWorkbook(){[Name="EmployeesTypes"]}[Content],

    Transformed = Table.TransformColumnTypes(
        data,
        GetTypeTransforms(typeControl)
    )
in
    Transformed
```

The new function is used in the second parameter of Table.TransformColumnTypes

```
= Table.TransformColumnTypes(
        data,
        GetTypeTransforms(typeControl)
    )
```

	EmployeeKey	FirstName	LastName	HireDate	BirthDate
1	1	Guy	Gilbert	1/28/2006	11/12/1981
2	2	Kevin	Brown	8/26/2006	12/1/1986
3	3	Roberto	Tamburello	6/11/2007	6/12/1974
4	4	Rob	Walters	7/5/2007	7/23/1974

The function can be used similarly for as many tables or in as many projects as we want

table name: EmployeesTypes

ColumnName	Type
EmployeeKey	whole number
FirstName	text
LastName	text
HireDate	date/time
BirthDate	date/time

Adjusting the types is as simple as changing the data in the control table and refreshing the query

```
= Table.TransformColumnTypes(
        data,
        GetTypeTransforms(typeControl)
    )
```

	EmployeeKey	FirstName	LastName	HireDate	BirthDate
1	1	Guy	Gilbert	1/28/2006 12:00:00 AM	11/12/1981 12:00:00 AM
2	2	Kevin	Brown	8/26/2006 12:00:00 AM	12/1/1986 12:00:00 AM
3	3	Roberto	Tamburello	6/11/2007 12:00:00 AM	6/12/1974 12:00:00 AM
4	4	Rob	Walters	7/5/2007 12:00:00 AM	7/23/1974 12:00:00 AM

“Um… Owen… isn’t this overkill?”

Well, maybe.

But even if you don’t apply this specific function, maybe it will spark some ideas about creative ways you can build the arguments that are passed to standard Power Query functions, whether Table.TransformColumnTypes or anything else…