

Microsoft Power Tools for Data Analysis #9

Power Query M Code: Values, Expressions, Functions, Parameters & Much More

Notes from Video:

Table of Contents:

Power Query does this.....	3
i. M Code.....	3
M Code Basics.....	3
1. Reference Guide for M Code Sources:.....	3
2. Queries.....	4
3. M Code is behind every Query.....	4
4. Expressions.....	5
5. let expression (let statement).....	6
6. Comments or Notes in M Code.....	8
7. Identifier.....	8
8. Keywords.....	9
9. Operators.....	9
10. Standard Library.....	9
i. Table.Sort function.....	9
11. Values: Primitive, List, Record, Table, Function and more.....	10
i. Primitive Value.....	10
ii. List Value.....	10
iv. Record Value.....	12
v. Table Value.....	12
vi. Function Value.....	12
vii. Binary Value.....	12
viii. Type Value.....	12
ix. Full List of Power Query, M Code Values.....	13
12. Primary Keys.....	14
13. Lookup or Projection and Selection.....	15
1) Define Selection & Projection.....	15
2) Lookup Operators to get Row & Column Index Numbers.....	15
ii. Positional Index Operator or Access Operator = Curly Brackets, like { and }.....	15

iii.	Lookup Operator or Field Access Operator = Square Brackets, like [and].....	15
3)	More About { } Positional Index Operator = Item Access = Curly Brackets	17
a.	Row Index.....	18
b.	Key Match	18
a.	Record Selection	19
1.	Required Record Selection.....	19
ii.	Optional Record Selection	19
4)	More About [] Lookup Operator or Field Access Operator.....	20
2.	Required Record Projections	20
3.	Optional Record Projections	20
4.	Required Table Projection.....	21
5.	Optional Table Projection	21
5)	Review of Referencing Values in Tables.....	21
6)	Drill Down & Primary Keys and Looking Up or Extracting Values.....	22
14.	Custom Functions.....	24
15.	each expressions and Underscore Character _.....	26
16.	Parmenter Query	27
4)	Getting an input value or parameter from Excel into a Power Query Solution.....	27
5)	Formula.Firewall Error: Privacy Levels for Two Queries are in Conflict.....	28
17.	Topics not in the video:.....	29

Power Query does this:

- Extract and Import From Multiple Sources.
- Clean and Transform Data.
- Load to Excel Sheet, PivotTable Cache, Excel Power Pivot Data Model, Power BI Desktop Data Model, Connection Only.
- Use Power Query to create proper data sets, data models, finished reports, parameters for other queries, custom functions and it can help replace complex Excel array formula solutions.
- Behind the Power Query solution is a “Function Based” Case Sensitive Computer Language called **M Code**.
 - i. **M Code** = Power Query’s formula language = Power Query’s Function Based Case sensitive language = M Language = Data **M**ashup language.
- Creating Power Query Solutions can be done with the User Interface or by writing M Code, or a combination of the two.

M Code Basics :

- 1) Reference Guide for M Code Sources
- 2) Queries
- 3) M Code is behind every Query
- 4) Expressions
- 5) let expression
- 6) Comments in M Code
- 7) Identifiers
- 8) Keywords
- 9) Operators
- 10) Standard Library
- 11) Values: Primitive, List, Record, Table, Function
- 12) Primary Keys
- 13) Lookup or Projection and Selection
- 14) Custom Functions
- 15) each expressions and Underscore Character _
- 16) Parmenter Queries

1. Reference Guide for M Code Sources:

- 1) Microsoft Power Query for Excel Formula Language Specifications (search Google & download)
- 2) In a blank query type:
= #shared
to get a listing of information about Power Query’s M Code.
** If you convert the =#shared information to a Table, you can filter to find the topic that you want.
- 3) Search Google and go to Microsoft site, like: <https://msdn.microsoft.com/en-us/query-bi/m/power-query-m-reference>
- 4) For a specific function information, type an equal sign and then the function name in formula bar and you will get help on that particular function

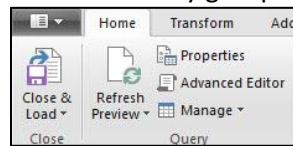
2. Queries :

- 1) A question we ask of the data.
- 2) Queries are built with M Code and return values such as numbers, text, lists, records, tables and functions.
- 3) Queries are created using a “let expression”, as we will learn about later.

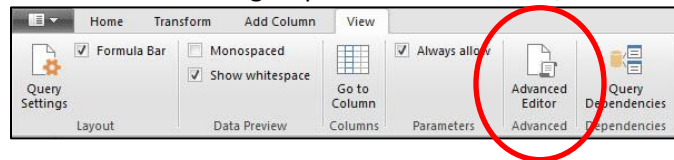
3. M Code is behind every Query :

- 1) When you use the User Interface M Code is automatically written for you and is stored in three places:
 - i. The Applied Steps list shows the name of each Step in the query.
 - ii. The full code can be seen and edited in the Advanced editor.
 - iii. The full M Code is stored behind the scenes in an M Document, either in Excel or Power BI Desktop.
- 2) You can see and edit the M Code in:
 - i. Applied Steps List allows you to rename query step, insert steps, delete steps, edit steps.
 - ii. Advanced Editor

1. Button in Query group in Home Ribbon Tab, as seen here:



2. Button in Advanced group in View Ribbon Tab, as seen here:



iii. Formula Bar

1. When you have a step selected in the Applied Steps list, you can edit the function in the Formula Bar. Enter will record the edited code.
2. Gear Icon next to step in Applied Steps list means you can edit code in dialog box, like Custom Column or Conditional Statement dialog box. However, if you change the code so that it does not conform to the dialog box, the Gear Icon will go away.
 - a. Example in video: When you add 4th argument to Table.Group function the Gear Icon goes away because there is no parallel textbox location in the Group By dialog box for the 4th argument of Table.Group.
3. You can use the Fx button in the Formula Bar to create a new step in your query.

iv. Custom Columns & Dialog Boxes

1. You can influence what code is written or write some of the code in dialog boxes like the Group By dialog box, Conditional Statement dialog box or Custom Column dialog box.

3) Blank Query

- i. Blank Query allows you to write the M Code from scratch and not use the user interface.
- ii. Ways to Open a Blank Query:
 1. Excel: Data Ribbon Tab, Get & Transform group, Get Data dropdown arrow, From Other Sources, Blank Query
 - a. Keyboard = Alt, A, P, N, O, Q
 2. Power BI Desktop: Home Ribbon Tab, External Data group, Get Data dropdown arrow, Blank Query.
 3. In the Power Query Editor, in the list of queries on the left, you can right-click, New Query, From Other Sources, Blank Query

4. Expressions :

- 1) M Code (M expression) that evaluates to a single value.
- 2) You can think of them as formulas or functions, like in Excel or DAX, which have formula inputs, arguments and deliver values.
- 3) Examples:
 - i. $43 - 43 = 0$
 - ii. `Text.Combine({43,"is","Rad!"}, " ") = "43 is Rad!"`
 - iii. `Table.Distinct = Table with Unique set of records`
 - iv. `List.Distinct = List of Distinct Values`
 - v. Let expression (full M Code from Advanced Editor that lists all steps in a query and delivers a Value). Example of a let expression, which contains a number of steps in a query that results in a Value, is shown below. Note: a let expression is an expression that contains many smaller expressions:

SalesNeedRounding

```
let
    Source = Excel.CurrentWorkbook(){[Name = "DateAndSales"]}[Content],
    #"Changed Type" = Table.TransformColumnTypes(Source,{{"Date", type date}, {"Sales", Currency.Type}}),
    #"Added Custom" = Table.AddColumn(#"Changed Type", "RoundedSales", each Number.Round([Sales],2)),
    #"Added Custom2" = Table.AddColumn(#"Changed Type", "RoundedSales", (_) => Number.Round(_[Sales],2)),
    #"Added Custom1" = Table.AddColumn(#"Added Custom2", "Record", each _)
in
    #"Added Custom1"
```

5. let expression (let statement) :

1) let expression allows you to create a query with one or more steps and delivers a Final Output Value (Primitive, List, Record, Table or Functions).

2) Example:

i. This example imports Excel Tables and then performs a number of steps to create a final Proper Data Table:

```
let
Source = Excel.CurrentWorkbook(),
FilteredRows = Table.SelectRows(Source, each not Text.Contains([Name], "Filter")),
ExtractedFieldNames = Table.AddColumn(FilteredRows, "FirstFieldName", each Table.ColumnNames([Content]){0}),
#"Filtered Rows1" = Table.SelectRows(ExtractedFieldNames, each not Text.StartsWith([FirstFieldName], "Column")),
GetSalesRepName = Table.ReplaceValue("#Filtered Rows1", "Sales", "", Replacer.ReplaceText, {"Name"}),
#"Removed Columns" = Table.RemoveColumns(GetSalesRepName, {"FirstFieldName"}),
#"Expanded Content" = Table.ExpandTableColumn("#Removed Columns", "Content", {"Date", "Product", "Sales"}, {"Date", "Product", "Sales"}),
ChangedType = Table.TransformColumnTypes("#Expanded Content",{{"Date", type date}, {"Product", type text}, {"Sales", Currency.Type}})
in
ChangedType
```

3) A “let statement” allows us to define one or more variables (variable name comes before equal sign), use the variables anywhere within the let statement and then deliver a value.

4) Any time we start a new query, a “let statement” is created.

5) Important aspects and features of a let expression:

i. Starts with the lowercase keyword: **let**

ii. Following the word **let** are lines or steps or variables, where each step/variable is an expression that delivers a Value.

1. Synonyms for step:

a. Step, Variable, Line, Transformation Step, Expression, Expression Name.

iii. Following the last step is the word **in** and then the name of the last step/variable is repeated. The Value delivered by the last step is the Value that the query delivers or the output of the query.

1. **Note:** Usually the last step / variable Name in the “let statement” follows the “in” keyword. However, you can type any variable name or query name or expression after the “in” statement that you want as the final output.

iv. Each Step has a name (Identifier) that follows this convention:

1. CharactersWithNoSpaces, when there are NO spaces in the name.

2. #”Character Space Character”, when there are spaces in the name.

v. Each step name is followed by an equal sign and then a Power Query Function or other M Code.

vi. Each step ends with a comma, except for the last line.

1. The comma indicates that this is the end of the step and a Value should be delivered.

2. The last step does not have a comma because this last step will be the Value that is delivered by the query.

vii. The word **in** (all lowercase) comes after last step.

viii. The name of the last step follows the word **in** and is the Final Output Value of the query.

6) Notes:

i. You can reference any of the previous steps in your query, including steps that were several steps back.

ii. You can reference any other query in the file at any point in a specific query.

7) Pattern of let expression is shown here:

"let statement" from Variable Point of View

```
let  
  VariableName1 = M Code,  
  #"Variable Name 2" = M Code,  
  VariableName3 = M Code  
in  
  Output
```

** Output is usually last Variable Name, but can be any Variable Name, Query Name or Expression

"let statement" from Applied Step Point of View

```
let  
  StepName1 = M Code,  
  #"Step Name 2" = M Code,  
  StepName3 = M Code  
in  
  Output
```

** Output is usually last Variable Name, but can be any Variable Name, Query Name or Expression

6. Comments or Notes in M Code :

- 1) Start the note with two forward slashes, like //, and this will allow you to write a note on a single line.
- 2) If you have multiple lines, start first line with forward slash and an asterisk, /*, followed by as many lines as you want, and then end with asterisk and forward slash, */.
- i. Example:

```
let
  // Function to determine age we will use in conversation
  Source = (Age as number , DesiredAge as number, optional Hedge as number) as number =>
  /*
  Age = Actual Age
  Desired Age = Stated Desired Age
  Hedge = Amount to add to hedge our bets
  */
  Number.Round(Age - (Age - DesiredAge)/2 + (if Hedge = null then 0 else Hedge),0)
in
  Source
```

- 3) Comments are not visible in Formula Bar unless you embed them in Function.

7. Identifier

- 1) Identifier = name used to refer to a Value (Primitive value, List, Record, Table, Function and so on).
- 2) Examples of Identifiers: Query Name, "let statement" Step / Variable Name, Field Names.
- 3) Example, as seen below:
 - i. In the below picture, the word Source is the name of the first step in the query, it is the identifier of the step and can be used in other parts of the query.
 - ii. In the below picture, the word #"Sorted Rows" is the name of the second step in the query, it is the identifier of the step and can be used in other parts of the query.

```
let
  Source = Excel.CurrentWorkbook(){[Name="dNoPrimaryKeyProduct"]}[Content],
  #"Sorted Rows" = Table.Sort(Source,{{"RetailPrice", Order.Ascending}})
in
  #"Sorted Rows"
```

- iii. Because quotes around words are reserved for actual text items in the M Code language, and because we want to distinguish an identifier, or name of part of a query, as something different than just plain text, the use of a # sign (pound sign) at the beginning of the quoted identifier instructs Power Query that this is the name of the Value that is delivered by the step in the query or the query itself.
- iv. The convention for creating identifier names is:
 1. CharactersWithNoSpaces, when there are NO spaces in the name.
 2. #"Character Space Character", when there are spaces in the name.
- 4) **Generalized Identifier** = the name of a Field in a Record Literal or a Field Access Operator. In these cases you can have spaces without using #"Character Space Character" convention.
 - i. Examples:
 1. Record Literal: = [Sales = 43, Type Item = "Quad"]
 - a. This record has the Field Names **Sales** and **Type Item**, without using quotes or pound sign.
 2. Formula in Custom Column that uses Field Access Operator, like: the formula =Number.Round([Sales Amount],2)
 - a. [**Sales Amount**] is the Field Name that contains the number we want to round.

8. Keywords

- 1) Reserved words that have an assigned meaning.
- 2) Here are some of the Keywords:
and as each else error false if in is let meta not otherwise or section shared then true try type #binary #date #datetime #datetimezone #duration #infinity #nan #sections #shared #table #time

9. Operators:

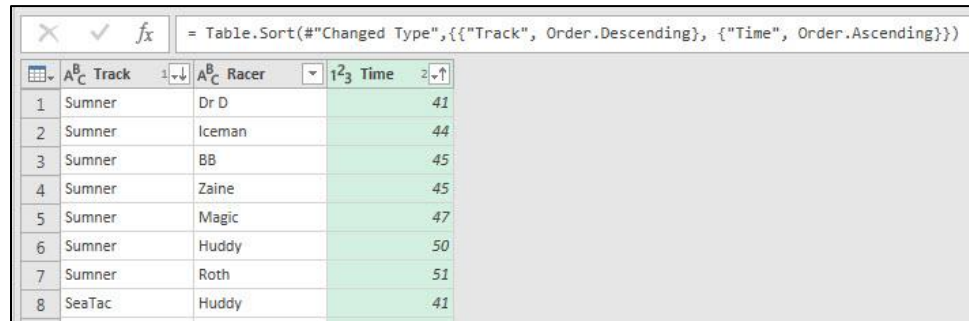
- 1) Operators and punctuators
- 2) There are several kinds of operators and punctuators. Operators are used in expressions to describe operations involving one or more operands. For example, the expression $a + b$ uses the $+$ operator to add the two operands a and b . Punctuators are for grouping and separating. operator-or-punctuator: one of: $, ; = < <= > >= <> + - * / \& () [] \{ \} @ ! ? \Rightarrow \dots$

10. Standard Library :

- 1) Standard Library is a list of built-in Functions and constants that deliver values.
- 2) Three examples of Standard Library:

i. Table.Sort function :

1. In Excel when sorting by Column Drop-downs, you click Major Sort Last
2. In Power Query when sorting by Column Drop-downs, you click Major Sort First!!!! (Similar to Excel Sort Dialog Box). Example of Table.Sort Function in Power Query:



	Track	Racer	Time
1	Sumner	Dr D	41
2	Sumner	Iceman	44
3	Sumner	BB	45
4	Sumner	Zaine	45
5	Sumner	Magic	47
6	Sumner	Huddy	50
7	Sumner	Roth	51
8	SeaTac	Huddy	41

- ii. Number.Round function is a Standard Library Function.
- iii. Number.E is a Standard Library Constant

11. Values: Primitive, List, Record, Table, Function and more :

- 1) Values = Values or Objects or Items available for use in Power Query M Code. Values are also Data Types for Columns.
- 2) Types of Values in M Code:
 - i. **Primitive Value :**
 1. Primitive Value = Single Part Value
 2. Examples:
 - a. Number, like 43
 - b. Text Value, like "Rad"
 - c. Logical Value, like true or false
 - d. null (Absence of Data), like null
 - e. Date, like: 1/1/2018 or intrinsic function: #date(Year,Month,Day)
 - f. Time, like 8 AM or intrinsic function: #time(Hour,Minute,Second)
 - g. Date-Time, like 1/1/2018 8:00 AM or intrinsic function: #datetime(Year,Month,Day, Hour,Minute,Second), like: #datetime(2018,1,1,20,15,15)
 - h. Date-Time-TimeZone, see page 37 in Formula Language Specification Manual.
 - i. #duration(Day,Hours,Minute,Second) used in math equations, like: = #datetime(2018,1,1,20,15,15)+#duration(1,1,1,1)
 - ii. **List Value :**
 1. List Value = an ordered sequence of Values
 2. Lists are similar to Arrays in Excel and other languages.
 3. Ways to create a list in Power Query:
 - a. Use a Power Query feature to create a List. Examples include:
 - i. List.Distinct function
 - ii. Convert to List button in Any Column group in Transform Ribbon Tab
 - iii. Drill Down on Column option (right-click column, Drill Down)
 - b. Create a List Literal (Hard Coded List) by typing your List, where List contains comma separated Values and is housed in the Curly Brackets { and }.
 - c. Imported List from an external data source.
 - d. Syntax like {1..43} to create a list of numbers from 1 to 43
 - e. Using M Code, like Table[Column Name], will extract a column from a table and deliver it as a list.
 4. Examples of Lists:
 - a. {43,"Rad","rad",null}
 - b. {"Yes",43},{"No",-43}}
 - c. Functions like: List.Distinct, List.Sort, List.Reverse, List.FirstN, List.Select
 - d. {1..43}
 - e. {} empty list
 - f. Functions like List.Sum consume lists.
 5. A List Value is housed in Curly Brackets, like { and }. Note: Curly Brackets are also used as the "Positional Index Operator", as we will see later.

1. List Values can contain any mix of the Power Query Values. For example, here is a Crazy Hard Coded List with many types of Values in the list. This is a { any value , any value} list example:

The screenshot shows the Power Query editor interface. On the left, a list of values is displayed in a table format:

	List
1	List
2	4
3	Rad
4	Record
5	Table
6	List
7	69
8	Error
9	11:00:00:00
10	List
11	List
12	List
13	Table
14	Table

On the right, the M code for the list is shown:

```
= {{3,4},4,"Rad",[Cat = "Rad",Value = 43],#table({"Cat","Value"},{{"Rad",43},{"dud",13}}),List.Dates(#date(2018,1,1),3,#duration(1,0,0,0)),List.Sum({43,-43,69}),List.Sum(List.Dates(#date(2018,1,1),3,#duration(1,0,0,0))),List.Sum({#duration(11,0,0,0)}),List.Sort({3,43,5}),List.Sort(List.Distinct({3,43,3}),Order.Descending),List.Select({3,5,4,7,1,99}, each _>5),#table({"Rad","Amount"},{{"Yes",43},{"No",-43}}),#table(type table [Rad = text,Amount = number],{{"Yes",43},{"No",-43}})}
```

2. One amazing thing about Lists is that some arguments in Functions will require a List, and this will mean we can specify multiple items for the argument, such as specifying two or more aggregate calculations in a Table.Group function. Here is an example (as seen in video 8.5 in this class), where the third argument of Table.Group uses a List within a List to enter two aggregate calculations into the Table.Group function:

List within a List to deliver two aggregate calculation to the third argument of Table.Group

→

```
= Table.Group("#Changed Type", {"TypeDay"},
{"Total Sales", each List.Sum([Sales]), type number},
{"Count Rows", each Table.RowCount(_),
type number}},GroupKind.Global)
```

iv. **Record Value :**

1. Record Value = an ordered sequence of Fields with Values for one record.
2. Ways to create a Record Value:
 - a. Use a Power Query feature to create a Record. Examples include:
 - i. Record.FromList function
 - ii. In a Custom Column you can use the Underscore Character to create a record (as seen in video)
 - b. Create a Record Literal (Hard Coded Record) by typing your Record, where the Record is a comma separated list of matched unique field names and Values housed in Square Brackets.
 - i. Example:
 1. The Record with the Field Names "Rad" and "Amount" that contain the values "Yes" and 43 can be written like this:
[Rad = "Yes", Amount = 43]
 - ii. Field Names must be unique. The Field Name is a Text Value but does not need to be in quotes. When we create a Field Name within a Record without quotes, this is called an Identifier.
 - c. Imported Record from an external data source.
 3. A Record is housed in Square Brackets, like [and]. Note: Squares Brackets are also used as the "Lookup Operator" or "Field Access Operator", as we will see later.

v. **Table Value :**

1. Table Value = Set of Unique Field Names with Data Types in first row, and one or more records in subsequent rows.
 - a. Occasionally you have Fields without data Types, but it is not good practice.
2. Ways to get a table:
 - a. Tables are usually imported from external data sources.
 - b. Use Power Query Functions like: Table.Sort or Table.Distinct.
 - c. Create a Table Literal (Hard Coded Table) using the #table() intrinsic function:
 - i. #table({"Rad", "Amount"}, {"Yes", 4}, {"No", -3})
 1. First Argument = Fields Names as List, Second Argument = Records as List within a List.
 - ii. #table(type table [Rad = text, Amount = number], {"Yes", 4}, {"No", -3})
 1. Where we defined the Table as a Table Data Type and Assigned Data Types to Fields.

vi. **Function Value :**

1. Later, we will learn how to create a query that results in a Custom Function which can deliver Values (Primitive, List, Record, Table).
2. These Custom Functions are Values themselves that a query can deliver as a Final Output.
3. This means that the full list of Value Types in Power Query is: Primitive, List, Record, Table, Function and a few more...
4. Built-in functions, like Splitter.SplitTextByDelimiter, is a function that delivers a function.
5. Because Functions themselves can exist as Values in a column, Custom Functions also have their own Data Type, called Function Data Type.

vii. **Binary Value :**

1. This value will hold our files, like text or Excel files.

viii. **Type Value:**

1. Value Category that sits over all other Value Types.

ix. Full List of Power Query, M Code Values (Page 31 in Formula Language Specification Manual):

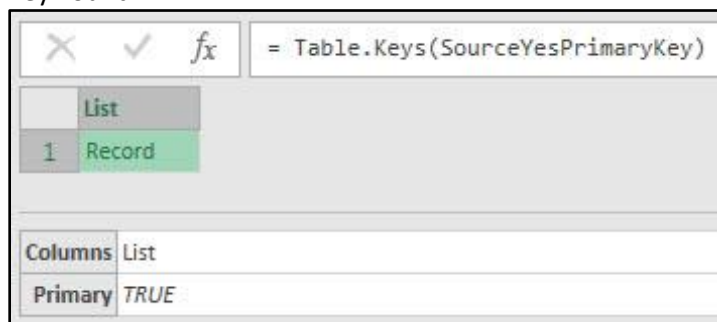
Kind	Literal
<i>Null</i>	null
<i>Logical</i>	true false
<i>Number</i>	0 1 -1 1.5 2.3e-5
<i>Time</i>	#time(09,15,00)
<i>Date</i>	#date(2013,02,26)
<i>DateTime</i>	#datetime(2013,02,26, 09,15,00)
<i>DateTimeZone</i>	#datetimezone(2013,02,26, 09,15,00, 09,00)
<i>Duration</i>	#duration(0,1,30,0)
<i>Text</i>	"hello"
<i>Binary</i>	#binary("AQID")
<i>List</i>	{1, 2, 3}
<i>Record</i>	[A = 1, B = 2]
<i>Table</i>	#table({"X", "Y"}, {{0,1},{1,0}})
<i>Function</i>	(x) => x + 1
<i>Type</i>	type { number } type table [A = any, B = text]

12. Primary Keys :

- 1) Primary Keys for Tables are nearly invisible in Power Query. And yet they play a significant role in doing Lookup in Power Query and for some calculations such as Group By Aggregate.
- 2) There is no way to see the Primary Keys in the User Interface.
- 3) Power Query will define a Primary Key in a Table in these situations:
 - i. If you use Remove Duplicates (Table.Distinct function).
 - ii. If you are connected to a database like an SQL database and a Primary Key has been defined.
 - iii. If you use Table.AddKey function.
 - iv. When you use Excel.CurrentWorkbook() Function.
- 4) You can determine if a Table has a Primary key by using the Table.Keys function, as seen here:
 - i. No Key found:



- ii. Key Found:



- 5) You can define a Primary Key in a Table using the Table.AddKey function.
 - i. The function arguments can be seen here:

Table.AddKey(table as table, columns as list, isPrimary as logical) as table

- ii. Example of Table.AddKey function:

The screenshot shows the Power Query formula bar with the function `= Table.AddKey(Source, {"Sale Date"}, true)`. Below the formula bar, the result is displayed as a table with two columns: "Sale Date" and "Units". The table has three rows of data.

	Sale Date	Units
1	7/30/2018 12:00:00 AM	34
2	7/31/2018 12:00:00 AM	54
3	8/1/2018 12:00:00 AM	71

- iii. Note: The Table.AddKey function will add a key to a column, even if there are duplicates.
- 6) There are some calculations that can take a long time to evaluate if you do not have a Primary Key.
 - i. Example: Chris Webb article (<https://blog.crossjoin.co.uk/2018/03/16/improving-the-performance-of-aggregation-after-a-merge-in-power-bi-and-excel-power-query-gettransform/>):
 1. The calculation that is illustrated is a Merge between a dimension table as Left side of merge and a large Fact table as the Right side of the merge, and then an aggregation is made on the dimension table side. This cause a very slow calculation time. But if you add a primary key to the dimension table, it may be able to dramatically decrease calculation time.

13. Lookup or Projection and Selection :

1) Define Selection & Projection:

- i. In the Official Manual for M Code from Microsoft, to select a column they use the word “Projections”, and to select a row or Value they use the word “Selection”. This terminology comes from Relational Algebra and Database Theory.
- ii. Relational Algebra & Database Theory:
 1. Projection (π , pi) = pick a column (attribute) from a table (relation)
 2. Selection (σ , sigma) = select a row (tuple) from a table (relation)
- iii. Power Query:
 1. Project = pick one or more fields or columns
 2. Select = Select a Value or a record

2) Lookup Operators to get Row & Column Index Numbers:

- i. When you want to lookup a single Primitive Value, Record, or Column we can use two operators:
- ii. **Positional Index Operator or Access Operator = Curly Brackets, like { and } :**
 1. Can be used to pull a specific item from a position in a List, Record or Table.
 2. { } can be thought of as the relative position of an item in a list, with a base zero:
 - a. 0 = 1st
 - b. 1 = 2nd
 - c. 2 = 3rd
 - d. And so on...
 3. This might remind you of what the MATCH Function in Excel does, but MATCH uses base 1, so 1 = 1st.
 4. { } will always give us the row number in a table or relative position for a list.
- iii. **Lookup Operator or Field Access Operator = Square Brackets, like [and] :**
 1. As a **Field Access Operator**, [] will allow us to lookup an entire column, or an item in a column.
 2. As a **Lookup Operator**, [] will allow us to do a Boolean operation to look an item up in a column, to then get a Row Index Number.
- iv. In Power Query, the Lookup Rule for a Table is similar to INDEX function in Excel. In Excel we might ask the INDEX Function to perform these lookup tasks:
 1. Please give me Row Index Number and Column Index Number to get the intersecting value.
 2. Please give me just the Row Index Number to get the entire Record (whole row).
 3. Please give me just the Column Index Number to get the entire column (whole column).
- v. Lookup Rule in Power Query:
 1. **TableName{RowIndex or KeyMatch}[Field Name]**
 - a. Row Index = Row Index Number, base zero = {HardCodedNumber} like {0}
 - b. Key Match = Column Index Number = {[Field Name = Condition]}
 - i. Where Key Match implies that there should be a unique list of Values in the column because if there are duplicates it will yield an error.
- vi. Examples of Lookups:
 1. **Lookup an item in a List:**
 - a. **M Code:** {43,86}{0} = 43
 - b. This gets the first item from the list (base zero).
 - c. The Curly Brackets provide the Relative Position of the item in the list, base zero.
 2. **Lookup a Record from a Table using RowIndex:**
 - a. **M Code:** ChangedType{0} = First record in the table named ChangedType.
 - b. This gets the first record in the table using a hard coded zero index number.
 - c. The Curly Brackets provided the Row Index Number, base zero, to the table, and then the record is extracted.

- d. This method will always get the first record in the table, no matter how the table is sorted or changed. This method “Hard Codes” the position into the lookup formula.
3. **Lookup a Record from a Table using KeyMatch:**
 - a. **M Code:** `Source{[Product="Sunshine"]}` = the Record for the Sunshine Product from the table named Source.
 - b. This gets the record for the product “Sunshine” in the table named “Source”.
 - c. The Square Brackets are the “Lookup Operator” that allows us to get the Row Number, base zero, for the product “Sunshine”. Inside the Square Brackets the Boolean operation allows the lookup formula to look an item up in a column, to then get a Row Index Number. Then the Curly Brackets can deliver that Row Number to the table, and thus extract the record from that row.
 - d. The Curly Brackets provide the Row Number, base zero, to the table.
 - e. This method will always get the record for the Product “Sunshine”, no matter how the table is sorted or changed. This method does NOT “Hard Code” the position into the lookup formula.
 - f. This method is intended for columns with a Unique List of Items. This method will yield an error if there are duplicates in the column.
 4. **Lookup a Value in a table (row and column intersection lookup):**
 - a. Example 1:
 - i. **M Code:** `= ChangedType{0}[Product]. = “Quad”`
 - ii. This gets the first row in the table named ChangedType and then looks up the product name from the Product Field.
 - iii. This is similar to using the INDEX Function in Excel to do a two-way lookup and give the formula a Row Index Number and a Column Index Number.
 - iv. The Curly Brackets provide the Row Index Number and the Square Brackets provide the Column Index Number to do a two-way lookup from a table.
 - b. Example 2:
 - i. **M Code:** `= Excel.CurrentWorkbook(){[Name="fSales"]}[Content]`
 - ii. `Excel.CurrentWorkbook()` returns a table with information about all the Excel Table objects in the Current Workbook, including a column named Content with the Data Type Table which contains all the Excel Tables from the Current Workbook, one for each row.
 - iii. `{[Name="fSales"]}` returns the Row Index Number for the table named fSales.
 - iv. `[Content]` returns the correct Column Index Number.
 - v. This is similar to using the INDEX Function in Excel to do a two-way lookup and give the formula a Row Index Number and a Column Index Number.
 - vii. Required or Optional Operators:
 1. If you use the Optional Operator, a Question Mark ?, after either Operator, like { }? Or []? and the position or field requested is not present, then a null value is returned. For looking up a Field, this only works if the field/s are listed at the end of the comma separated list of Fields.
 2. If you do not use the Optional Operator, a Question Mark ?, then the lookup is Required and will yield an error if there are problems like: position is not available, item not available, or duplicate values in a column.

3) **More About { } Positional Index Operator = Item Access = Curly Brackets :**

i. Zero-based index

1. 0 = 1st
2. 1 = 2nd

ii. Curly Brackets allow you to lookup or reference a specific position of an item in a List, Record or Table.

iii. **Example of Looking up (or Referencing) an item in a List:**

1. Here is the List = **{43,86}**

2. Here are lookup situations (**M Code** and Result):

- i. $\{43,86\}\{0\} = 43$
- ii. $\{43,86\}\{2\} = \text{error}$
- iii. $\{43,86\}\{2\}? = \text{null}$

1. ? is Optional Operator which avoids error and delivers a null when a match is not made, or said a different way, when the positions is not in the list.

iv. Examples of Referencing or Looking Up a Record (Row) in a Table :

1. Example Table:

Date	Sales	Product
8/8/2018	2	Quad
8/9/2018	5	Carlota
8/10/2018	71	Quad
8/11/2018	102	Sunshine
8/12/2018	54	Carlota

2. Methods for looking up record:

a. **Row Index** = Hard Code Position into Curly Brackets, like **Source{0}** to get the first record in the table names Source.

i. Row Index Example:

1. **M Code:** Source{0}. This method looks up the first record in the table. This method works whether or not there are duplicates. This method is called Row Index because it will always lookup the first record in the table. If you change the order of records, this method always gets the first record in the table – the zero value is hard coded into the formula. Example seen here:

= Source{0}	
Date	8/8/2018 12:00:00 AM
Sales	2
Product	Quad

b. **Key Match** = Uses both Lookup Operator (Square Brackets) and Positional Index Operator (Curly Brackets) in same construction. Square Bracket Logical Test is in Curly Brackets to get row position of Record with Sunshine Product, like: Source{[Product="Sunshine"]}. It is called "Key" Match because if there are duplicates in the column it returns an error.

i. Key Match Example:

1. **M Code:** Source{[Product="Sunshine"]}. This method works on columns where there is a unique list of values. It is called "Key Match" because if there are duplicates it will yield an error. This method assumes that there are no duplicates or that there is a Primary "Key" Column that contains no duplicates. Example seen here:

= Source{[Product="Sunshine"]}	
Date	8/11/2018 12:00:00 AM
Sales	102
Product	Sunshine

3. Examples of different ways to lookup a record in a table:

a. **Record Selection :**

1. **Required Record Selection**

M Code: `Source{[Product="Sunshine"]}` This searches the table for the record where the Product Column contains the value "Sunshine". No duplicates, so it works. Example seen here:

<code>= Source{[Product="Sunshine"]}</code>	
Date	8/11/2018 12:00:00 AM
Sales	102
Product	Sunshine

2. **M Code:** `Source{[Product="Quad"]}` This method only works when there are no duplicates. Duplicates cause errors. Example seen here:

<code>= Source{[Product="Quad"]}</code>	
<p>! Expression.Error: The key matched more than one row in the table.</p> <p>Details:</p> <p>Key=Record</p> <p>Table=Table</p>	

3. **M Code:** `Source{[Product="Aspen"]}` No match cause error. Example seen here:

<code>= Source{[Product="Aspen"]}</code>	
<p>! Expression.Error: The key didn't match any rows in the table.</p> <p>Details:</p> <p>Key=Record</p> <p>Table=Table</p>	

ii. **Optional Record Selection**

M Code: `Source{[Product="Aspen"]?}` Question Mark will return a null rather than an error if no match. Example seen here:

<code>= Source{[Product="Aspen"]?}</code>	
null	

iii. **Example of extracting a record that contains a Table Value:**

M Code: `=Excel.CurrentWorkbook(){[Name="fSales"]}` This returns a record with a table in it. Example seen here:

<code>= Excel.CurrentWorkbook(){[Name="fSales"]}</code>	
Content	Table
Name	fSales

iv.

4) **More About [] Lookup Operator or Field Access Operator = Square Brackets:**

i. More examples of the Field Access Operator and the Lookup Operator:

1. **Field Access Operator to lookup a specific item from a Field:**

- a. In a Custom Column we can use a formula that will be copied down a new column and needs to access the value for each row in the field named Amount, like in this formula: **M Code:** =Number.Round([Amount],2)
- b. The formula element [Amount] allows us access to the value for each row in the column Amount.

2. **Field Access Operator to lookup a Field from a Table Value:**

- a. A formula like this: **M Code:** = Source[Units], will lookup the whole Units column from the table named Source and deliver it as a list.

3. **Lookup Operator to lookup the Record for the Sunshine Product from the table named Source:**

- a. A formula like this: **M Code:** = Source{[Product="Sunshine"]}, will lookup the record for the product Sunshine.

ii. Examples of different ways to use the Lookup Operator or Field Access Operator:

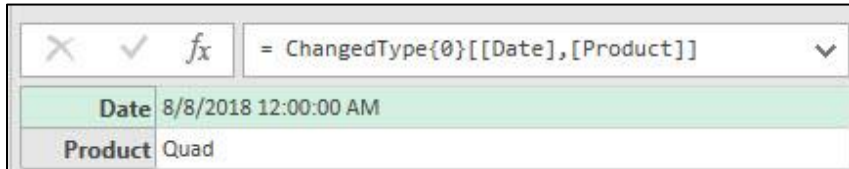
1. **Select a value from a record**

- a. = Table{RowIndex}[FieldName]
- b. = ChangedType{0}[Product]



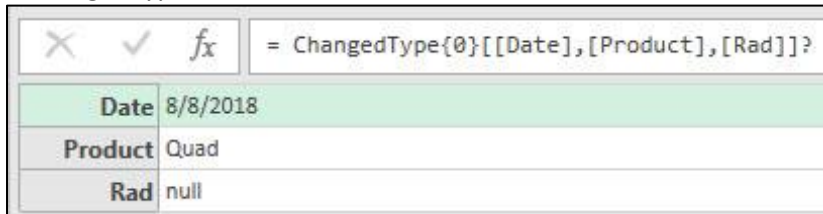
2. **Required Record Projections.** Project a record from three fields down to two fields:

- a. = Table{RowIndex}[[FieldName],[FieldName2]]
- b. = ChangedType{0}[[Date],[Product]]



3. **Optional Record Projections:** Project a record from three fields down to three fields with the last one being optional:

- a. = Table{RowIndex}[[FieldName],[FieldName2]]?
- b. = ChangedType{0}[[Date],[Product],[Rad]]?



4. **Required Table Projection** : Project a table from three fields down to two fields:

- a. = Table[[FieldName],[FieldName2]]
- b. = ChangedType[[Date],[Product]]

	Date	Product
1	8/8/2018	Quad
2	8/9/2018	Carlota
3	8/10/2018	Quad
4	8/11/2018	Sunshine
5	8/12/2018	Carlota

5. **Optional Table Projection**: Project a table from three fields down to three fields with the last one being optional:

- a. Table[[FieldName],[FieldName2],[FieldName3]]?
- b. = ChangedType[[Date],[Product],[Rad]]?

	Date	Product	Rad
1	8/8/2018	Quad	null
2	8/9/2018	Carlota	null
3	8/10/2018	Quad	null
4	8/11/2018	Sunshine	null
5	8/12/2018	Carlota	null

c.

5) **Review of Referencing Values in Tables** :

- i. Using Row Index (Hard Coded Row Number) to get the product name from the first row of a table:

	Product
1	Quad

- ii. Using Key Match to get the Table named fSales form the Content Column in the table created by the Excel.CurrentWorkbook function:

	Content
1	= Excel.CurrentWorkbook(){[Name="fSales"]}[Content]

iii.

6) **Drill Down & Primary Keys and Looking Up or Extracting Values :**

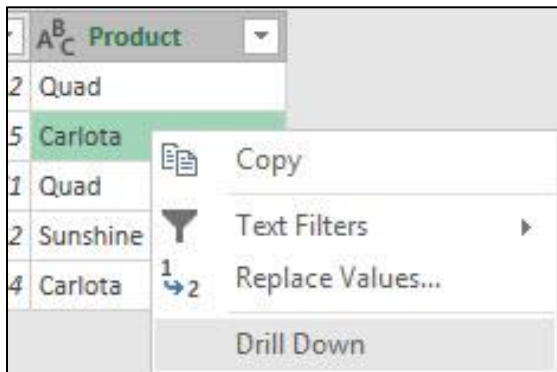
- i. We can lookup or extract a value at the intersection of a row and column in a table using the Drill Down feature in Power Query.
- ii. The Drill Down Feature will extract the Value from the intersection of a row and column in a table and deliver it as the output for the query. This means that when the query is invoked or used somewhere else, the Value will be delivered as the result of the query.
- iii. To Drill Down, simply right click any cell in a Power Query Table and then click on Drill Down.
- iv. Examples of Drill Down:

1. **Example 1:** If there is no Primary key, when you Drill Down, you get a Row Index Lookup.

a. Starting Table, as seen here:

	Date	Sales	Product
1	8/8/2018		Quad
2	8/9/2018		Carlota
3	8/10/2018	71	Quad
4	8/11/2018	102	Sunshine
5	8/12/2018	54	Carlota

b. Right-click the second row in the Product Column and click on Drill Down, as seen here:



c. The Lookup formula uses the Row Index Method for looking the value up, as seen here:



2. **Example 2:** If there is a Primary key, when you drill down, you get a Key Match Lookup.

a. This time we added a primary key to the table , as seen here:

	Date	Sales	Product
1	8/8/2018	2	Quad
2	8/9/2018	5	Carlota
3	8/10/2018	71	Quad
4	8/11/2018	102	Sunshine
5	8/12/2018	54	Carlota

b. Next, we did a Drill Down on the second product in the Product Filed, as seen here:

Product
Quad
Carlota
Quad
Sunshine
Carlota

c. This time, the Lookup formula uses the Key Match Method for looking the value up and it uses the Primary Key Field, Date, as the column to help generate the correct Row Index Number, as seen here:

Formula	Result
= Custom1{[Date=#date(2018, 8, 9)]}[Product]	Carlota

- v. Drilling Down and knowing whether or not the table has a Primary Key is important, because the Row Index Method hard coded the row number into the formula, whereas the Key Match Method will always retrieve the item in the product column that has a date of 8/9/2018.

14. Custom Functions :

- 1) Standard Library provides many built-in functions, like Text.Combine or List.Sum. But with Custom Function, we are allowed to program our own functions.
- 2) Power Query Functions allow us to program a function to perform a certain task, a certain bit of business logic. We might want to calculate the Effective Annual Rate on a loan or perform a certain data transformation that we would like to repeat over and over in the current Excel File. Power Query Functions are perfect for this.
- 3) We can create Power Query functions in three ways:
 - i. Create a query that delivers a Function Value. This is a Reusable Function. These types of functions will have the step name (identifier) of the query that defines the function as the Final Output of the query.
 - ii. Create a query or expression (like a Custom Column) that defines a Function and then use the Function to deliver a Value that is something different than a Function Value, like the “TransformFairTableFunction” example a few pages ahead.
 - iii. Create a Parameter Query that defines function that will deliver a variable or parameter to the query.
- 4) Definition of a Custom Function: **User defines the arguments and creates a mapping of the arguments for the function so it can deliver a Value (Primitive, List, Record, Table, Custom Function...)**
- 5) Custom Functions deliver a Value with a Data Type (including any Data Type).
- 6) Example of creating a query that delivers a Function Value (reusable function), named EffectiveAnnualRate, that will deliver an effective rate on a loan as a number Value:

```
EffectiveAnnualRate

let
    EffectiveAnnualRateMCode =
        (AnnualRate as number , PeriodsPerYear as number) as number =>
            Number.Power(1+AnnualRate/PeriodsPerYear,PeriodsPerYear)-1
in
    EffectiveAnnualRateMCode
```

- 7) Syntax of the reusable Query Function named EffectiveAnnualRate:
 - i. When you define a query as a function, the name of Query is name you will use when you want to invoke the function.
 - ii. To create a Query as a Function, in M Code Advanced Editor, open a blank query and use the let statement to define the Custom Function.
 - iii. Create the name of step in query followed by an Equal Sign.
 - iv. Parentheses house the arguments.
 - v. Arguments are separated by commas.
 - vi. Not pictured above: Arguments can be optional and must be listed after all required arguments. If the argument is not entered by user, it will be automatically entered as a null.
 - vii. Arguments can have Data Types.
 - viii. The Function can have a Data Type.
 - ix. The Function delivers a value.
 - x. Goes To Symbol (Equal Sign followed by Less Than Symbol, like =>) comes after the parenthetical arguments and before the mapping of the arguments
 - xi. Mapping of arguments. This mapping can be a simple math calculations, or a complex let expression with many steps for the function.
 - xii. When you create a Query as a Function, only one step will show up in the Applied Steps List, regardless of how many steps you have in your query.
 - xiii. When the name of the step that defined the function is the last step listed and the Final Output of the Query, then you have created a Query as a Function, which can be used over and over.

- 8) Example of creating a query that delivers a Function Value (reusable function) with math calculation, notes / comments and an optional argument (example not seen in video):

```
AssumedAge

let
  // Function to determine age we will use in conversation
  Source = (Age as number , DesiredAge as number, optional Hedge as number) as number =>
  /*
  Age = Actual Age
  Desired Age = Stated Desired Age
  Hedge = Amount to add to hedge our bets
  */
  Number.Round(Age - (Age - DesiredAge)/2 + (if Hedge = null then 0 else Hedge),0)
in
  Source
```

- 9) Example of creating a query that defines a Function and then use the Function to deliver a Value that is something different than a Function Value – Result is a number, not a function (example not seen in video):

The screenshot shows an Excel spreadsheet with the formula bar containing the function call `= PlusTwo(41)`. An 'Advanced Editor' window is overlaid on the spreadsheet, displaying the following query code:

```
let
  PlusTwo = (x) => x + 2,
  Result = PlusTwo(41)
in
  Result
```

- 10) You can use a query defined as a function in other areas such as:
 - i. Custom Column
 - ii. Right-click query defined as function, enter values a new query will be created
- 11) Using Power Query Functions over and over.
 - i. When we create Power Query Function, we can invoke the function in the current Excel File.
 - ii. Unlike Excel VBA functions that can be stored in a hidden Personal Workbook, and can be used in any Excel File for a particular computer; in Power Query if you want to use the Power Query function in a different Excel file, you will have to copy and paste the M Code into the Advanced Editor in the new Excel file.

15. each expressions and Underscore Character _ :

- 1) each = syntactical shorthand for declaring / defining an unnamed function taking a single untyped parameter/argument named underscore _ .
- 2) You can use each shorthand anywhere a functions can be declared.
- 3) each is often used to pass a function to an argument in another function, like Table.Group
- 4) each is often used in Custom Columns.

Example 1:

(_) => _+43
 or
 each _+43

Example 2:

= Table.Group("#Changed Type", {"Rep"}, {"Total Sales", (_) => List.Sum(_[Sales]), type number})
 or
 = Table.Group("#Changed Type", {"Rep"}, {"Total Sales", each List.Sum([Sales]), type number})

Example 3:

= Table.Group("#Changed Type2", {"Rep"}, {"All Rows", (_) => _, type table})
 or
 = Table.Group("#Changed Type2", {"Rep"}, {"All Rows", each _, type table})

Example 4:

(_) => _[a] + _[b]
 or
 Each [a] + [b]

Example 5 (Custom Column example as seen in video):

= Table.AddColumn("#Changed Type", "RoundedSales", (_) => Number.Round(_[Sales],2))
 or
 = Table.AddColumn("#Changed Type", "RoundedSales", each Number.Round([Sales],2))

16. Parmenter Query :

- 1) Parameter Query = A Query that has a variable input
- 2) Parameter synonyms:
 - i. Parameter
 - ii. Variable
 - iii. Formula Input
 - iv. Argument
- 3) Example of creating a Parameter Query that defines function that will deliver a variable or parameter to the query:
 - i. Here we defined the Custom Function with a single argument. The single argument will be tables from a column and the query will transform each table into a proper date set.
 - ii. The function is defined before the let statement, so it serves just as an input value or parameter for our query. It is not a step in the let expression (query). It does not have an identifier or step name.
 - iii. The argument in the Function is named InputTable. This argument or parameter for input value is used in the first step in the query in the Table.Transpose function.
 - iv. This is called a parameter query because the query has a variable input.

TransformFairTableFunction

```
(InputTable) =>
let
    #"Transposed Table" = Table.Transpose(InputTable),
    #"Promoted Headers" = Table.PromoteHeaders(#"Transposed Table", [PromoteAllScalars=true]),
    #"Unpivoted Other Columns" = Table.UnpivotOtherColumns(#"Promoted Headers", {"Date", "Product/Fair"}, "Attribute", "Value"),
    #"Renamed Columns" = Table.RenameColumns(#"Unpivoted Other Columns",{{"Product/Fair", "Fair"}, {"Attribute", "Product"}, {"Value", "Sales"}})
in
    #"Renamed Columns"
```

- 4) **Getting an input value or parameter from Excel into a Power Query Solution :**
 - i. You can create a single column, single row Excel Table with a query input as the record.
 - ii. Import the table.
 - iii. Drill Down on the first row in the table.
 - iv. Then use the Name of the Query in another query as an input variable.

5) **Formula.Firewall Error: Privacy Levels for Two Queries are in Conflict**

- i. This error message can occur when an external data source is used in a query that contains references to other queries.
- ii. The full error reads: “Learn how to deal with Power Query Error: Formula.Firewall: Query references other queries or steps, so it may not directly access a data source. Please rebuild this data combination. Two solutions are presented in this video”.
- iii. The Formula.Firewall Error means that the Privacy Levels for Two Queries are in Conflict.
- iv. Solution 1: Change Privacy Level to "Always Ignore Privacy Settings"
- v. Solution 2: Land First Query with External Connection as Connection Only, then use that Named Query in Second Query.
- vi. Chris Webb’s blog about this topic: <https://blog.crossjoin.co.uk/2017/06/26/data-privacy-settings-in-power-bipower-query-part-3-the-formula-firewall-error/>):
 1. “Power Query engine is not allowed to access two different data sources originating from different queries in the same step – as far as I understand it this is because it makes it too hard for the engine to work out whether a step connects to a data source or not, and so which data privacy rules should be applied.”
- vii. Ken Puls blog about this topic: <https://www.excelguru.ca/blog/2015/03/11/power-query-errors-please-rebuild-this-data-combination/>
 1. Power Query cannot combine an external data source with another query”.

17. Topics not in the video:

1. Conditional Statements – IF Function in Power Query = if then else:

1) In Power Query

- i. We don't use `IF(A1="Rad",43,-43)`
- ii. We use: `if [Column] = "Rad" then 43 else -43`

2. Error Handling :

1) Handle Errors with try otherwise

3. Data Types :

1) 1&"Text" throws an error

2) Number.ToText(1)&"Text" works

3) Is operator:

- i. `43 is number = TRUE`
- ii. `= 43 is text = FALSE`

4) Data Type for Values:

- i. Literal (1, "T", true / false, null)
- ii. List {value 1, value2}
- iii. Record [Sale = 2,Product = "Bellen"]
- iv. Table `#table(type table [Rad = text,Amount = number],{"Yes",43},{"no",-43})`
- v. Function (the column could hold a function in each row).
- vi. Binary (files like csv and Excel and others).

5) You can use these function to create certain types of data:

- i. `#date(Year,Month,Day)`
- ii. `#datetime(Year,Month,Day, Hour,Minute,Second)`, like: `#datetime(2018,1,1,20,15,15)`
- iii. `#datetimezome()`
- iv. `#duration(Day,Hours,Minute,Second)` used in match equations, like: `= #datetime(2018,1,1,20,15,15)+#duration(1,1,1,1)`

4. Query Folding :

- 1) If you connect to Relational Databases, OData data sources, Exchange and Active Directory, Power Query will try to send transformation back to data source where is usually can perform the transformations more efficiently.
- 2) There is no was to tell if it is done more efficiently from the Power Query User Interface.
- 3) You would have to go to source and measure, like with tools like SQL Server Profiler.
- 4) You can prevent Query Folding with functions like `Table.Buffer`.
- 5) Power Query Features like Custom Functions and `Table.UnPivot` will prevent Query Folding from happening.

5. Table.Pivot Function

i. Pivot & Table.Pivot :

1. Pivot = Get Unique List of Items from a specified column, then make columns out of each Unique =Item. Pivot a Column so that rows become columns.
2. Table.Pivot is not available in User Interface.
3. Example:
 - a. Start Table has unique records for each combo of racer and track:
 - b. Pivot to show times for each racer and track:
 - c. Table.Pivot Function:

Track	Racer	Time
Sumner	Zaine	45
SeaTac	Zaine	42
Mt Vernon	Zaine	48
Sumner	Roth	51
SeaTac	Roth	41
Mt Vernon	Roth	50
Sumner	Magic	47
SeaTac	Magic	42
Mt Vernon	Magic	51
Sumner	Iceman	44
SeaTac	Iceman	46
Mt Vernon	Iceman	50
Sumner	Huddy	50
SeaTac	Huddy	41
Mt Vernon	Huddy	36
Sumner	Dr D	41
Mt Vernon	Dr D	53
Sumner	BB	45
Mt Vernon	BB	36

Track	Zaine	Roth	Magic	Iceman	Huddy	Dr D	BB
Mt Vernon	48	50	51	50	36	53	36
SeaTac	42	41	42	46	41		
Sumner	45	51	47	44	50	41	45

If there are no duplicate records, the last argument is not needed, as seen here:
 = Table.Pivot("#Changed Type",List.Distinct(Table.Column("#Changed Type","Racer")),"Racer","Time")

= Table.Pivot("#Changed Type",List.Distinct(Table.Column("#Changed Type","Racer")),"Racer","Time",List.Sum)

1) Previous Step.

2) Unique List of Items from Racer Column. This argumnet is names of new columns.

3) Column in table that should have items pivoted to Column Headers.

Column in table to either aggregate if there are duplicate records, or values to use at intersection of row and column.

Aggregate Function.