DATA MODELING

What is **DATA MODELING**?

"Data modeling aims to capture and describe the part of reality that we want to keep information about."

Data models

› **Data models –** define how the logical structure of a database is modeled. Data models define how data is connected and processed, and stored inside the system.
› **Entity-Relationship model –** Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenarios into the database model. This model is based on Entities (and their attributes) and Relationships (among entities).
› **Entities** - have properties called attributes, where every set of values called domain defines an attribute.
› **Relationships** – are logical association among entities. The cardinality mapping within a bind establishes the number of associations between two entities.
› **Relation / Normalized model –** This model is based on first-order predicate logic and defines a table as an n-ary relation. Data stored in tables are called **relations** and can be normalized. ***Each column in a relation contains values from the same domain!***
› **Dimensional model –** This is a technique that uses **Dimensions** and **Facts** to store data efficiently. Dimensional Models have a specific structure and organize the data to **generate reports** that improve performance. Five main components are used in any of these models:
› **Attributes/Measures –** are the elements of the DT.
› **Fact Tables –** Are utilized to store measures or transactions in the business. They are related to DT with the foreign key.
› **Dimensions Tables (DT) –** contain descriptive data that is linked to the Fact Table. DT are usually optimized tables and hence have large columns and fewer rows.
› **Relationships** - Is a link between two tables, based on a primary key on one side, to the foreign key on the other side of the relationship. A regular dimension relationship represents the relationship between dimension tables and a fact table. Relationships have properties like cardinality, condition, direction and type.

Dimensions

› **Conformed –** has the same meaning to all the Facts it relates to.
› **Outrigger –** represents a connection between different Dimension Tables.
› **Shrunken –** is a perfect subset of a more general data entity.
› **Role-Playing –** has multiple valid relationships between itself and various other tables. *(Date table)*
› **Junk –** is used to combine two or more related low cardinality Facts into one Dimension.
› **Degenerate –** are standard Dimensions that are built from the attribute columns of Fact Tables. Sometimes data are stored in Fact Tables to avoid duplication.
› **Swappable –** has multiple similar versions of itself, which can get swapped at query time.
› **Step –** explains where a particular step fits into the process.
› **Slowly changing –** contains relatively static data which can change slowly but unpredictably.
› **Rapidly changing** - one or more of its attributes in the table changes very fast and in many rows
› **Static –** are not extracted from the real data source.

Dimensional schemas

› **Star –** modeling approach adopted in data warehouses and other analytical systems. Inteded for large volumes of data.
› Star schemas can be identified by having one or multiple fact tables and connected dimensions. Star schemas typically work with column based compression.
› In a star schema, the dimensions support filtering and grouping of data. When viewing the data, the data from related facts will be aggregated and viewed on the level of the dimension.
› Example*: Sales table, containing all sales transactions in the middle. This fact table is surounded by various dimensions such as a Customer, Product and Date dimensions adding context to the transactions.*
› Star schemas are key to optimized, well performing and easy usable data models.
› **Snowflake –** snowflake models follow the same patterns as star schemas. The both work based on relationships between tables, typically from fact to dimensional tables. However, in snowflake schemas, there are also relations between dimensions and dimensions.
› Example*: Product dimension, which has an active relationship to the product category dimension.*
› In case of related dimensions, you could consider to join them together in one table to create a star schema out of it. Take data duplication for multiple rows and column compression into consideration when exploring the optoins to join.

Data modeling rules to live by
- Simple DAX is a sign of a good data model.
- DAX complexity down, performance goes up.

Types of relationships

Cardinality
› **1:1 –** One record in the left table, relates to exactly one record in the right table. Typically, these tables can also be joined in a single table. Example*: a customer has one address.*
› **1:N –** One record on the left side of the relationship, relates to many records on the right side of the relationship. Example*: A customer has multiple sales transactions.*
› **M:N –** many records on one side of the relationship, relate to many records on the other side of the relationship. *Example: A student attends many courses, and a course contains many students.*

State
› **Active –** the default state of the relationship.
› **Non-Active -** only one relationship between two tables can be active at a time. All other relationships will become inactive. Relationships can be activated for calculations by adding the **USERELATIONSHIP()** expression in DAX.

Direction
› **Single –** filters applied only flow in the relationship direction.
› **Both –** filters work are applied in both directions, but can lead to ambiguous data models and performance implications.

Types
› **Regular Relationships –** a relationship where the engine can validate the one-side of the relationship and both tables are in the same source group.
› **Limited Relationships –** relationships with a M:N cardinality or cross source group. For example a relationship with an imported table on one side and direct query on the other side.

Data storage modes

Power BI supports three types of data storage **Import, DirectQuery, Dual** for data sources. These types have their own requirements on data sources, so not all sources will support even most of them.
› **Import –** Imported data is stored on a disk. To requirement of querying are **all** data loaded into the memory of Service. This in-memory querying supports receiving very fast results. There is no way to have the model loaded into memory, just partially. This model is only as current as the last refresh is, so Import models need to be refreshed, usually on a scheduled basis. Refresh model drops all refreshing data and needs to load all data again.
› **DirectQuery –** DirectQuery only consists only of metadata defining the model structure. These metadata are used for building native queries **against** the data sources. This means that shown data are actual only as data was in the data source while the query was sent to execution. Because every visual load sends these native queries this **brings near real-time experience**. M and DAX functions are limited to only using functions that can be transposed to native queries understood by the data source.
› **Dual –** A table configured as Dual storage mode is both Import and DirectQuery. This setting allows the **Power BI Service** to determine the most efficient method to use on a **query-by-query** basis.

Connectivity types
› **Composite –** The composite model is combination of Import and DirectQuery mode or more DirectQuery sources. Against alone DirectQuery **this supports DAX defined calculated tables**. These models strive to deliver the best of Import and DirectQuery modes.
› **Live Connection –** Is the connectivity type used between report and Power BI dataset, or analysis services dataset, where the report sends a query intended to render a visual, to the dataset which will process the query and return the relevant data. **(RLS & OLS)**.

Partitions

› **Partitions divide a table into logical parts.** Each partition can then be processed independently of other partitions. Partitions defined for a model during model authoring are duplicated in a deployed model. These partitions are part of the **Tabular Object Model** (TOM) and can be managed by **Tabular Model Scripting Language.** There is no hard limit on the number of partition objects in a model. On the other hand, too many small partitions can lead to a very negative impact on query speed.
› By default, each table in a model has a single partition. For models with structured data sources, partitions are defined by using a **M expression**.
› When partitions are processed, multiple partitions are evaluated simultaneously to increase performance. However, there are settings like **maxParallelism** that limit parallel processing operations.
› Within Power BI, partitions can also be produced separately via External Tools. An example of a tool that allows this is the **Tabular Editor**. It enables you to manage the Tabular Object Model, including adding new components, such as partitions.

DAX or M

› **DAX** (Data Analysis Expressions) is a library of functions and operators combined to create formulas and expressions. It is the best language to answer analytical questions which their responses will be different based on the selection criteria in the report.
› **M** is the scripting language behind the scene for Power Query. This language is great for capturing, preparing, transforming, and combining data before loading it into your model.

"Data should be transformed as far upstream as possible, and as far downstream as necessary.

› In this context "upstream" means closer to where the data is originally produced, and "downstream" means closer to where the data is consumed.
› Power Query is farther upstream. Performing data transformation in Power Query ensures that the data is loaded into the data model in the shape it needs to be in when your dataset is refreshed. Your report logic will be simplified and thus easier to maintain, and will likely perform better because the Vertipaq engine will need to do less work as users interact with the report.
› If you need data transformation logic that depends on the context of the current user interacting with the report – things like slicers and cross-filtering? This is the perfect job for a DAX measure, because Power Query doesn't have access to the report context. Implementing this logic farther downstream in DAX makes sense because it's necessary." Matthew Roche

Calculated Columns

› A calculated column is a **new column** added to the model using DAX respectively by its formulas. This new column behaves like any other column in the table to which we add it and can be used to define **relationships**.
› When creating a column, the context of the calculation is **directly dependent** on the row for which the result is currently executed. So, the references of the other columns naturally return **only** the value that comes from that row. *(Unless the context is further modified.)*
› The column is **calculated** and **stored** during the processing of the model database. This causes an increase of the time required to process the model but does not affect the resulting query time. The result of the calculated column is with model stored in **Memory**, so it can waste very needed space for other computes. This fact is actual only in **Import** mode. In **DirectQuery** mode, these columns are computed as the Tabular engine queries the data source. However, this can have a very negative impact on performance.

Measures

› Measures are **aggregators of values** where the type of aggregation is defined by **DAX expression**, and evaluation **contexts** define data. The measure must be defined in a table, so, you can't create a measure without a table. Measures can reference each other inside expression but only if the result is not a recursive reference.
› Contexts are provided into evaluation by visual element and by DAX query. We have two types, "**Filter context**" & "**Row context**," of evaluation contexts, and these contexts can be combined in many ways.
› The resulting aggregation corresponds to a combination of all **current input contexts.** It follows that if the context is given, for example, using a **slicer** visual, which filters the input table of the calculation, and the selected value is changed, then the whole measure is recalculated, and a new result is obtained.
› Measures are calculated on the fly at visual render time and is processed in CPU.

Dataset

› A Dataset in the Power BI service is a source for reporting and visualization. There are various types of datasets;
- Created from **Power BI desktop** and published to the service.
- **Excel workbook** uploaded to the Power BI Service
- **Push dataset**, which can only be created in the Power BI Service and is fed via the Power BI REST API.
- **Streaming datasets**, for real time purposes and populated with data via an Azure Eventhub, PubNub, or REST API.

› Datasets can also live outside the Power BI Service, such as an (Azure) Analysis Services model. In case of external models, reports will have a live connection to the dataset.
› Power BI Desktop created models are saved in the ***.pbix** file, and after publishing the file is split in a **dataset** and a **report**.
› Datasets created in Power BI are Tabular models, while Analysis Services could also create multi-dimensional models. Power BI works best with **Tabular models**.
› **Datasets** in the Power BI Service can be shared with others by granting **build permissions**, so others can build new reports on top of the same dataset. Others can find datasets by using the datasets hub in the Power BI Service to explore and directly create reports from scratch on top of the dataset.

Dataflows

› Dataflow is intended for (self service) data preparation inside **Power BI Service.** We can call it with a different name, "**Power Query Online**" so it's also using also language **M**. All transformed data are stored inside **CDM** compliant folders inside **Azure Data Lake Gen2** (ADLG2).
› In which ADLG2 will data be stored can be set up on **workspace level** or **tenant level**. Without any Data Lake will data be stored in the native one.
› Every dataflow is a separated artifact that can be reused in many datasets without re-calling data sources.
› Inside one dataflow can be one or more queries. There are three types of result queries that can be used.
› **Standard** – Data are fetched directly from a data source or with data from non-stored entities within the same dataflow.
› **Computed*** – This is a type of query, which is created thanks to combinations of multiple loaded queries.
› **Linked*** – Enables you to reference an existing table, defined in another dataflow, in a read-only fashion.

**available only with Premium per Capacity / User*

Composite models

› A composite model is a data model that combines two different storage modes in a single data model.
› Use cases for composite models are for example dealing with large data volumes and near **real-time data** which cannot be solved in import storage mode.
› Storage modes supported in composite models are **Import**, **Direct Query**, **Dual** and **Hybrid**. Each composite model contains a combination of two or more storage modes.
› To improve performance, consider adding **aggregations** (user defined or automated) for **direct query** fact tables and benefit from imported data on aggregated level.
› As com posite models combines various storage modes in a single model, be aware of the potential limited relationsihps that could be introduced in your model. **Consider Dual storage model to avoid limited relationships.**
› Composite models can be connecting to each type of data source, but also to existing **Power BI datasets** or **Azure Analysis Services** models.

OVERVIEW

What is **Power BI**?

"It is Microsoft's Self-Service Business Intelligence tool for processing and analyzing data."

Components

› **Power BI Desktop**—Desktop application
› **Report**—Multi-page canvas visible to end users. It serves for the placement of visuals, buttons, images, slicers, etc.
› **Data**—Preview pane for data loaded into a model.
› **Model**—Editable scheme of relationships between tables in a model. Pages can be used in a model for easier navigation.
› **Power Query**—A tool for connecting, transforming, and combining data.

"Apart from the standard version, there is also a version for Report Server."

› **Power BI Service**—A cloud service enabling access to, and sharing and administration of, output data.
› **Workspace**—There are three types of workspaces: **Personal**, **Team**, and **Develop a template app**. They serve as storage and enable controlled access to output data.
› **Dashboard**—A space consisting of **tiles** in which visuals and report pages are stored.*
› **Report**—A **report** of pages containing visuals.*
› **Worksheet**—A published Excel worksheet. Can be used as a tile on a dashboard.
› **Dataset**—A published sequence for fetching and transforming data from **Power BI Desktop**.
› **Dataflow**—Online Power Query representing a special dataset outside of Power BI Desktop.*
› **Application**—A single location combining one or more reports or dashboards.*
› **Admin portal**—Administration portal that lets you configure capacities, permissions, and capabilities for individual users and workspaces.

**Can be created and edited in the Power BI Service environment.*

› **Data Gateway**—On-premises data gateway that lets you transport data from an internal network or a custom device to the **Power BI Service**.
› **Power BI Mobile**—Mobile app for viewing reports. Mobile view is applied, if it exists, otherwise the desktop view is used.
› **Report Server**—On-premises version of Power BI Service.
› **Report Builder**—A tool for creating page reports.

Built-in and additional languages

Built-in languages

› **M/Query Language**—Lets you transform data in Power Query.
› **DAX** (Data Analysis Expressions)—Lets you define custom calculated tables, columns, and measures in Power BI Desktop.

"Both languages are natively available in Power BI, which eliminates the need to install anything."

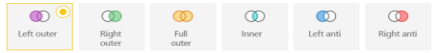
Additional languages

› **Python**—Lets you fetch data and create visuals. Requires installation of the Python language on your computer and enabling Python scripting.
› **R**—Lets you fetch and transform data and create visuals. Requires installation of the R language on your computer and enabling R scripting.

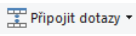
Power Query

Works with data fetched from data sources using connectors. This data is then processed at the Power BI app level and stored to an **in-memory** database in the program background. This means that data is not processed at the source level. The basic unit in Power Query is **query**, which means one sequence consisting of **steps**. A step is a data command that dictates what should happen to the data when it is loaded into Power BI. The basic definition of each step is based on its use:

› **Connecting data**—Each query begins with a function that provides data for the subsequent steps. E.g., data can be loaded from **Excel**, **SQL database**, **SharePoint** etc. Connection steps can also be used later.
› **Transforming data**—Steps that modify the structure of the data. These steps include features such as Pivot Column, converting columns to rows, grouping data, splitting columns, removing columns, etc. Transformation steps are necessary in order to clean data from not entirely clean data sources.
› **Combining data**—Data split into multiple source files needs to be combined so that it can be analyzed in bulk. Functions include merging queries and appending queries.
› **Merge queries**—This function merges queries based on the selected key. The primary query then contains a column which can be used to extract data from a secondary query. Supports typical join types:



› **Append query**—Places the resulting data from one or more selected queries under the primary query. In this case, data is placed in columns with names that are an exact match. Non-matching columns form new columns with a unique name in the primary query.



› **Custom function**—A query intended to apply a pre-defined sequence of steps so that the author does not need to create them repeatedly. The custom function can also accept input data (values, sheets, etc.) to be used in the sequence.
› **Parameter**—Values independent of datasets. These values can then be used in queries. Values enable the quick editing of a model because they can be changed in the Power BI Service environment.

Dataflow

The basic unit is a table or **Entity** consisting of columns or **Fields**. Just like Queries in Power Query, Entities in Dataflows consist of sequences of steps. The result of such steps is stored in native Azure Data Lake Gen 2.

"You can connect a custom Data Lake where the data will be stored."

There are three types of entities:
Standard entity—It only works with data fetched directly from a data source or with data from non-stored entities within the same dataflow.
Computed entity*—It uses data from another stored entity within the same dataflow.
Linked entity*—Uses data from an entity located in another dataflow. If data in the original entity is updated, the new data is directly passed to all linked entities.

**Can only be used in a dedicated Power BI Premium workspace.*

"It supports custom functions as well as parameters."

DAX

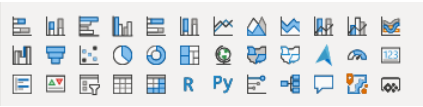
Language developed for data analysis. It enables the creation of the following objects using expressions:
› **Measures**
› **Calculated Columns**
› **Calculated Tables**

Each expression starts with the = sign, followed by links to tables/columns/functions/measures and operators. The following operators are supported:
› **Arithmetic** { + , - , / , * , ^ }
› **Comparison** { = , == , > , < , >= , <= , <> }
› **Text concatenation** { & , && , II , IN }
› **Precedence** { (,) }

Operators and functions require that all values/columns used are of the same data type or of a type that can be freely converted; such as a date or a number.

Visualization

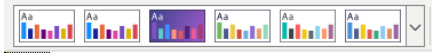
Visualizations or visuals let you present data in various graphical forms, from graphs to tables, maps, and values. Some visuals are linked to other services outside Power BI, such as Power Apps.



In addition to basic visuals, Power BI supports creating custom visuals. Custom visuals can be added using a file import or from a free Marketplace offering certified and non-certified visuals. Certification is optional, but it verifies whether, among other things, a visual accesses external services and resources.

Themes

Serves as a single location for configuring all native graphical settings for visuals and pages.



By default, you can choose from 19 predefined themes. Custom themes can be added.

A custom theme can be applied in two different ways:
› **Modification of an existing theme**—A native window that lets you modify a theme directly in the Power BI environment.
› **Importing a JSON file**—Any file you create only defines the formatting that should change. Everything else remains the same. The advantage of this approach is that you can customize any single visual.

*"The resulting theme **can be exported** in the JSON format and **used** in any report without the need to create a theme from scratch."*

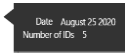
Drill Down

The Visual that supports the embedding of hierarchies enables drilling down to the embedded hierarchy's individual levels using the following symbols:

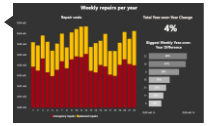
Drill up to a higher-level hierarchy
Drill down to a specific field
Drill down to the next level in the hierarchy
Expand next-level hierarchy

Tooltip/Custom Tooltip

› **Tooltip** —A default detail preview pane which appears above a visual when you hover over its values.

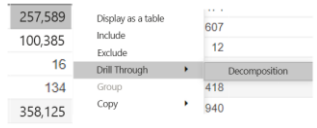


› **Custom Tooltip** —A custom tooltip is a custom-designed report page identified as descriptive. When you hover over visual, a page appears with content filtered based on criteria specified by the value in the visual.



Drill-through

Drill-through lets you pass from a data overview visual to a page with specific details. The target page is displayed with all the applied filters affecting the value from which the drill-through originated.



Bookmarks

Bookmarks capture the currently configured view or a report page visual. Later, you can go back to that state by selecting the saved bookmark. Setting options:
› **Data**—Stores filters, applied sort order in visuals and slicers. By selecting the bookmark, you can re-apply the corresponding settings.
› **Display**—Stores the state of the display for visuals and report elements (buttons, images, etc.). By selecting the bookmark, you can go back to the previously stored state of the display.
› **Current page**—Stores the currently displayed page. By selecting the bookmark, you can go back the to stored page.

License

Per-user License
› **Free**—Can be obtained for any Microsoft work or school email account. Intended for personal use. Users with this license can only use the personal workspace. They cannot share or consume shared content.

"If it is not available in Premium workspace"

› **Pro**—It is associated with a work/school account priced at €8.40 per month or it is included in the E5 license. Intended for team collaboration. Let's users access team workspaces, consume shared content, and use apps.
› **Premium per User** – Includes all Power BI Pro license capabilities, and adds features such as paginated reports, AI, greater frequency for refresh rate, XMLA endpoint and other capabilities that are only available to Premium subscribers.
Per-tenant License
› **Premium**—Premium is set up for individual workspaces. 0 to N workspaces can be used with a single version of this license. It provides dedicated server computing power based on license type: P1, P2, P3, P4*, P5*. It offers more space for datasets, extended metrics for individual workspaces, managed consumption of dedicated capacity, linking of Azure AI features with datasets, and access for users with **Free** licenses to shared content. Prices start at €4,212.30.

**Only available upon special request. Intended for models larger than 100GB.*

› **Embedded**—Supports embedding dashboards and reports in custom apps.
› **Report Server**—Included in Premium or SQL Server Enterprise licenses.

Administration

› **Use metrics**—Usage metrics let you monitor Power BI usage for your organization.
› **Users**—The Users tab provides a link to the Microsoft 365 admin center.
› **Audit logs**—The Audit logs tab provides a link to the Security & Compliance center.
› **Tenant settings**—Tenant settings enable fine-grained control over features made available to your organization. It controls which features will be enabled or disabled and for which users and groups.
› **Capacity settings**—The Power BI Premium tab enables you to manage any Power BI Premium and Embedded capacities.
› **Embed codes**—You can view the embed codes that are generated for your tenant to share reports publicly. You can also revoke or delete codes.
› **Organization visuals**—You can control which type of Power BI visuals users can access across the organization.
› **Azure connections**—You can control workspace-level storage permissions for Azure Data Lake Gen 2.
› **Workspaces**—You can view the workspaces that exist in your tenant on the Workspaces tab.
› **Custom branding**—You can customize the look of Power BI for your whole organization.
› **Protection metrics**—The report shows how sensitivity labels help protect your content.
› **Featured content**—You can manage all the content promoted in the Featured section.

External Tools

They simplify the use of Power BI and extend the capabilities offered in Power BI. These tools are mostly developed by the community. Recommended external tools:
› Tabular Editor
› DAX studio
› ALM Toolkit
› VertiPaq Analyzer

POWER QUERY

What is **Power Query**?

"An IDE for M development"

Components

› **Ribbon –** A ribbon containing settings and pre-built features by Power Query itself rewrites in M language for user convenience.
› **Queries –** simply a named M expression. Queries can be moved into groups
› **Primitive –** A primitive value is a single-part value, such as a number, logical, date, text, or null. A **null** value can be used to indicate the absence of any data.
› **List –** The list is an ordered sequence of values. M supports endless lists. Lists define the characters **"{"** and **"}"** indicate the beginning and the end of the list.
› **Record –** A record is a set of fields, where the field is a pair of which form the name and value. The name is a text value that is in the field record unique.
› **Table –** A table is a set of values arranged in named columns and rows.
Table can be operated on as if it is a list of records, or as if it is a record of lists. Table[Field]` (field reference syntax for records) returns a list of values in that field. `Table{i}` (list index access syntax) returns a record representing a row of the table.
› **Function –** A function is a value that when called using arguments creates a new value. Functions are written by listing the function argumets in parentheses, followed by the transition symbol "=>" and the expression defining the function. This expression usually refers to argumets by name. There are also functions without argumets.
› **Parameter –** The parameter stores a value that can be used for transformations. In addition to the name of the parameter and the value it stores, it also has other properties that provide metadata. The undeniable advantage of the parameter is that it can be changed from the **Power BI Service** environment without the need for direct intervention in the data set. Syntax of parameter is as regular query only thing that is special is that the metadata follows a specific format.
› **Formula Bar –** Displays the currently loaded step and allows you to edit it.To be able to see formula bar, It has to be enabled in the ribbon menu inside **View** category.
› **Query settings –** Settings that include the ability to edit the name and description of the query. It also contains an overview of all currently applied steps. Applied Steps are the variables defined in a let expression and they are represented by varaibles names.
› **Data preview –** A component that displays a preview of the data in the currently selected transformation step.
› **Status bar –** This is the bar located at the bottom of the screen. The row contains information about the approximate state of the rows, columns, and time the data was last reviewed. In addition to this information, there is profiling source information for the columns. Here it is possible to switch the profiling from 1000 rows to the entire data set.

Functions in Power Query

Knowledge of functions is your best helper when working with a functional language such as **M**. Functions are called with parentheses.
› **Shared –** Is a keyword that loads all functions (including help and example) and enumerators in result set. The call of function is made inside empty query using by **= # shared**

```
= #shared
```

Functions can be divided into two categories:
› Prefabricated – Example: Date.From()
› Custom – these are functions that the user himself prepares for the model by means of the extension of the notation by „()=> ", where the argumets that will be required for the evaluation of the function can be placed in parentheses. When using multiple argumets, it is necessary to separate them using a delimiter.

Data values

Each value type is associated with a literal syntax, a set of values of that type, a set of operators defined above that set of values, and an internal type attributed to the newly created values.
› **Null –** null
› **Logical –** true, false
› **Number –** 1, 2, 3, ...
› **Time –** #time(HH,MM,SS)
› **Date –** #date(yyyy,mm,ss)
› **DateTime –** #datetime(yyyy,mm,dd,HH,MM,SS)
› **DateTimeZone –** #datetimezone(yyyy,mm,dd,HH,MM,SS, 9,00)
› **Duration –** #duration(DD,HH,MM,SS)
› **Text –** "text"
› **Binary –** #binary("link")
› **List –** {1, 2, 3}
› **Record –** [A = 1, B = 2]
› **Table –** #table({columns},{{first row contenct},{}...})*
› **Function –** (x) => x + 1
› **Type –** type { number }, type table [A = any, B = text]
* The index of the first row of the table is the same as for the records in sheet 0

Operators

There are several operators within the M language, but not every operator can be used for all types of values.
› **Primary operators**
› **(x)** – Parenthesized expression
› **x[i]** – Field Reference. Return value from record, list of values from table.
› **x{i}** – Item access. Return value from list, record from table. "Placing the "**?**" Character after the operator returns null if the index is not in the list "
› **x(...)** – Function invocation
› **{1 .. 10}** – Automatic list creation from 1 to 10
› **...** – Not implemented
› **Mathematical operators** – **+, -, *, /**
› **Comparative operators**
› **>** , **>=** – Greater than, greater than or equal to
› **<** , **<=** – Less than, less than or equal to
› **=** , **<>** – is equal, is not equal. Equal returns true even for null = null
› **Logical operators**
› **and** – short-circuiting conjunction
› **or** – short-circuiting disjunction
› **not** – logical negation
› **Type operators**
› **as** – Is compatible nullable-primitive type or error
› **is** – Test if compatible nullable-primitive type
› **Metadata** - The word **meta** assigns metadata to a value.
Example of assigning metadata to variable **x:**
"x meta y" or **"x meta [name = x, value = 123,...]"**
Within Power Query, the priority of the operators applies, so for example "X + Y * Z" will be evaluated as "X + (Y * Z)"

Comments

M language supports **two** versions of comments:
› Single-line comments – can be created by **//** before code
› Shortcut: **CTRL + ´**
› Multi-line comments – can be created by **/*** before code and ***/** after code
› Shortcut: **ALT + SHIFT + A**

let expression

The expression let is used to capture the value from an intermediate calculation in a named variable. These named variables are local in scope to the `let` expression. The construction of the term let looks like this:

```
let
    name_of_variable = <expression>,
   returnVariable = <function>(name_of_variable)
in
returnVariable
```

When it is evaluated, the following always applies:

› Expressions in variables define a new range containing identifiers from the production of the list of variables and must be present when evaluating terms within a list variables. The expressions in the list of variables are they can refer to each other
› All variables must be evaluated before the term let is evaluated.
› If expressions in variables are not available, let will not be evaluated
› Errors that occur during query evaluation propagate as an error to other linked queries.

Conditions

Even in Power Query, there is an "**If**" expression, which, based on the inserted condition, decides whether the result will be a true-expression or a false-expression.

Syntactic form of If expression:
if <predicate> **then** < true-expression > **else** < false-expression >
"**else** is required in M's conditional expression "

Condition entry:
If x > 2 **then** 1 **else** 0
If [Month] > [Fiscal_Month] **then** true **else** false

If expression is the only conditional in M. If you have multiple predicates to test, you must chain together like:

```
if <predicate>
  then < true-expression >
  else if <predicate>
    then < false-true-expression >
    else  < false-false-expression >
```

When evaluating the conditions, the following applies:

› If the value created by evaluating the if a condition is not a logical value, then an error with the reason code "**Expression.Error**„ is raised
› A true-expression is evaluated only if the **if** condition evaluates to **true. Otherwise**, false-expression is evaluated**.**
› If expressions in variables are not available, they must not be evaluated
› The error that occurred during the evaluation of the condition will spread further either in the form of a failure of the entire query or "**Error**" value in the record.

The expression try... otherwise

Capturing errors is possible, for example, using the **try** expression. An attempt is made to evaluate the expression after the word **try**. If an error occurs during the evaluation, the expression after the word **otherwise** is applied

Syntax example:
try Date.From([textDate]) **otherwise** null

Custom function

Example of custom function entries:
(x, y) => Number.From(**x**) + Number.From(**y**)

```
(x) =>
let
   out = Number.From(x) +
   Number.From(Date.From(DateTime.LocalNow()))
in
   out
```

The input argumets to the functions are of two types:
› **Required –** All commonly written argumets in (). Without these argumets, the function cannot be called.
› **Optional –** Such a parameter may or may not be to function to enter. Mark the parameter as **optional** by placing text before the argument name "**Optional**". For example **(optional x)**. If it does not happen fulfillment of an optional argument, so be the same for for calculation purposes, but its value will be null.
Optional arguments must come after required arguments.

Arguments can be annotated with `as <type>` to indicate required type of the argument. The function will throw a type error if called with arguments of the wrong type. Functions can also have annotated return of them. This annotation is provided as:
(x as number, y as text) as logical => <expression>

The return of the functions is very different. The output can be a sheet, a table, one value but also other functions. This means that one function can produce another function. Such a function is written as follows:

let first = **(x)=> () =>** let out = {1..x} in out in first

When evaluating functions, it holds that:

› Errors caused by evaluating expressions in a list of expressions or in a function expression will propagate further either as a failure or as an "**Error**" value
› The number of arguments created from the argument list must be compatible with the formal argumets of the function, otherwise an error will occur with reason code "**Expression.Error**"

Recursive functions

For recursive functions is necessary to use the character "**@**" which refers to the function within its calculation. A typical recursive function is the factorial. The function for the factorial can be written as follows:

```
let
  Factorial = (x) =>
    if x = 0 then 1 else x * @Factorial(x - 1),
  Result = Factorial(3)
in
  Result // = 6
```

Each

Functions can be called against specific arguments. However, if the function needs to be executed for each record, an entire sheet, or an entire column in a table, it is necessary to append the word **each** to the code. As the name implies, for each context record, it applies the procedure behind it. **Each** is never required! It simply makes it easier to define a function in-line for functions which require a function as their argument.

Syntax Sugar

› Each is essentially a syntactic abbreviation for declaring non-type functions, using a single formal parameter named. Therefore, the following notations are semantically equivalent:

```
let
  Source = ...,
  addColumn = Table.AddColumn(Source, „NewName", each [field1] + 1)
in
  addColumn
```

```
let
  Source = ...,
  add1ToField1 = (_) => [field1] + 1,
  addColumn(Source,"NewName",add1ToField1)
in
```

The second piece of syntax sugar is that bare square brackets are syntax sugar for field access of a Record named `_`.

Query Folding

As the name implies, it is about composing. Specifically, the steps in Power Query are composed into a single query, which is then implemented against the data source. Data sources that supports Query folding are resources that support the concept of query languages as relational database sources. This means that, for example, a CSV or XML file as a flat file with data will definitely not be supported by Query Folding. Therefore, the transformation does not have to take place until after the data is loaded, but it is possible to get the data ready immediately. Unfortunately, not every source supports this feature.
› Valid functions
› Remove, Rename columns
› Row filtering
› Grouping, summarizing, pivot and unpivot
› Merge and extract data from queries
› Connect queries based on the same data source
› Add custom columns with simple logic
› Invalid functions
› Merge queries based on different data sources
› Adding columns with Index
› Change the data type of a column

DEMO

› Operators can be combined. For example, as follows:
› LastStep[Year]{[ID]}

*This means that you can get the value from another step based on the index of the column

› Production of a DateKey dimension goes like this:

```
#table(
   type table [Date=date, Day=Int64.Type, Month=Int64.Type,
MonthName=text, Year=Int64.Type,Quarter=Int64.Type],
   List.Transform(
      List.Dates(start_date, (start_date-endd_ate),
                  #duration(1, 0, 0 ,0)),
      each {_, Date.Day(_), Date.Month(_),
Date.MonthName(_), Date.Year(_), Date.QuarterOfYear(_)}
))
```

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