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.