Fundamental DAX Concepts and Data Modeling Best Practice

- **Evaluation Context**: Refers to both row context and filter context. Row context is applied to calculated columns, while filter context comes from filters, slicers, or other expressions.
- **Filter Context**: The active filters on a table that affect the results of a DAX expression.
- **Filter Flow**: Filters in a model usually propagate from the "one" side to the "many" side of a relationship
  - The lookup table is generally one side and the data (facts) table is the many side
  - When we filter a table, that filter context is passed to the downstream tables (direction of arrow)
  - Filter cannot flow upstream (against the arrow direction
  - Organize lookup at the top and the data below so easy visualization
- Only include columns necessary for visual/report
- If date time column exist, split into date and time (for cardinality reduction)

DAX Function Categories

- **Math**
  - SUM, AVERAGE, MAX/MIN, DIVIDE, COUNT, COUNTA.., COUNTROWS, DISTINCTCOUNT, SUMX, AVERAGEX, RANKX, and other iterator funcs
- **Logical**
  - IF, IFERROR, AND, OR, NOT SWITCH, TRUE,FALSE
- **TEXT**
  - CONCENTENATE, FORMAT, LEFT, MID,RIGHT, PROPER,LEN,SEARCH/FIND, REPLACE, REPT, SUBSTITUTE, TRIM, UNICHAR
- **FILTER**
  - CALCULATE, FILTER, ALL, ALLEXCEPT, RELATED, RELATEDTABLE, DISTINCT, VALUES, EARLIER/EARLIEST, HASONEVALUE, HASONEFILTER, ISFILTERED, USERELATIONSHIP
- **Date and Time**
  - DATEDIFF, YEAR/MONTH/DAY. DATESYTD, DATESQTD,DATESINPERIOD…

DAX Function Categories (Table/Filter funcs)

- ALL, FILTER, DISTINCT, VALUES, ALLEXCEPT, ALLSELECTED
- **Calculated Tables**
  - **DAX expressions refer to physical tables or virtual calculated tables (particularly when we have a filter expression)**
    - FILTER is both a table function and an iterator, and is used to get a subset
    - ALL is both a table filter and a calculate modifier
    - ALL removes initial filter context, does not accept table expressions and can only reference physical tables
    - U can use DISTINCT to create lookup tables for example
    - VALUES returns a single column table of unique values when a column name is given but if a table name is supplied, it returns the entire table including duplicates
    - The biggest difference between values and distinct is distinct does not return a blank row if there is a missing value in either of the two sides of the table being joined or looked up
  - SELECTEDVALUE returns a value when there's only 1 value in a specified column otherwise returns an optional alternate result and is similar to if hasonevalue -this is very helpful when we want a scalar value from a column
  - ALLEXCEPT removes all report context filters in the table except the filters applied to the specified columns in the query, can reference included table columns or a table that is on the one-side of the relationship. This is used as a calculated modifier and not as a table function as such
  - ALLSELECTED returns all rows in a table or values in a column ignoring filters specified in the query but keeping any other existing filter context, i.e it ignores any filters within the visual like row/column but respects only the filters from the slicer

DAX Function Categories (Table/Filter funcs)

- **Add data (Add columns based on existing data in model**
  - SELECTCOLUMNS, ADDCOLUMNS, SUMMARIZE
  - SELECTCOLUMNS returns a table with selected columns from the table plus any new columns specified by the DAX expression. This function can refer to physical table or virtual table (using a filter function for example
  - ADDCOLUMNS has the same syntax like selectcolumns and are almost identical. However, selectcolumns starts with a blank table and then you select specific columns. Here though you do not start with a blank table but start with all the columns from the table but then add new columns to that
  - SUMMARIZE creates a summary of the input table grouped by specified columns. But remember it is not an aggregation group by is misleading. It is just showing distinct combinations involving the summarize column. Similar to above it can refer to physical or virtual tables (using filter for example)

DAX Function Categories (Table/Filter funcs)

- Create data (used to generate new rows, columns and tables from scratch)
  - ROW, DATATABLE, GENERATESERIES, {} Table constructor
  - ROW returns a single row table with new column(s) - it has a column name in double quotes and a Dax expression returning scalar value. Usually this function is used along with another data generation function for example generate series
  - DATATABLE returns a new table contain new static data. It has a name in double quotes, data type, and the data type. It can’t take expressions and only fixed data types
  - GENERATESERIES returns a single column table containing sequential values based on a given increment
  - {} uses curly braces to return a table containing or more columns and records. We cannot defined column headers when using the table constructor syntax. The syntax for the data can refer to measures, and have single or multiple coulmns, no strict data type enforcement here, and within curly you define tuple style column values for each row. You can have complex expressions using calculate and other expressions for column values including formatting.

CALCULATE Syntax and Context Transition

- **CALCULATE**: Modifies the filter context of an expression
  - Function with an IF condition(s). So basically, it first applies a filter expression on a table/dataset, and then it flows downstream to all all related tables, and then the measure formula evaluates against the filtered table
  - Calculated columns are good for row context, and is helpful for filtering data rather than aggregations, etc..
  - Measures are calculated values and are not visible at a table level like calculated columns, they are evaluated based on the filter context, so they recalculate when the fields or filters around them change - useful for aggregation

- **Evaluation Context - filter and row**
  - Filter context is created when Dax adds dimensions to rows, columns, slices and filters in a report; calculate is used to create or modify existing filter context, and filter context propagates from 1 to many
  - Row context iterates through the rows in a table, Dax creates a row context when we add a calculated column to our data model, iteration functions do row context, and it does not auto propagate table relationships, instead we need to use RELATED/ RELATEDTABLE.

- **Modifiers**: Examples include `ALL`, `KEEPFILTERS`, and `REMOVEFILTERS`

- **Context Transition**: Occurs when `CALCULATE` changes row context into filter context, which allows row-level calculations to propagate into aggregated measures

CALCULATE (Advanced)

- Expanded tables
  - Consists of the base table (visible to the user) along with columns from any related table connected via 1-1/1-many relationship
  - Expanded tables in models **always** got from the many side to the one side of the relationship (ex: fact to dim)
  - Once a filter is applied to a column all expanded tables containing that columns are filtered
- Context transition - the process of turning row context into filter context
  - By default calculated columns understand row context but not filter context
  - To create filter context at row-level, we can use CALCULATE - we can see how it works between CALCULATE and SUM on a report to understand
- Evaluation order (of CALCULATE)
  - First filter of all filter expressions, and then goes down the order
  - Calculate modifier like for example ALL if present will be first evaluated, and then goes to the filter expressions. If a filter expression exists, it overrides the modifiers in this case the ALL
  - Remember when multiple individual filter expressions are encountered, DAX actually combines all these arguments into 1 filter context

CALCULATE (Advanced/Modifiers)

- ALL, ALLSELECTED, ALLNOBLANKROW, ALLEXCEPT, KEEPFILTERS, REMOVEFILTERS
- REMOVEFILTER is an alias for ALL, but can only be used as a CALCULATE modifier and not as a table function
- KEEPFILTERS is like an inner join - it does not modify existing filters but adds new filter context, and it allows to control which specific filters are applied to a calculation. So for example when there are already external filters like say category = 20, and you add a keep filters of family type = 2, then it bring all 20 including family type 2, however if the external filter slicer is like family type 1, then the calculate measure referring to family type =2 is blank because there is no overlap - so the inner join similarity
- Change filter propagation using CROSSFILTER

Time Intelligence Functions and Patterns

- Performance to Date Functions: DATESYTD,QTD, MTD
- Time Period Shift: SAMEPERIODLASTYEAR, DATEADD, PARALLELPERIOD, QUARTER/MONTH/DAY, NEXTYEAR, QUARTER/MONTH/DAY
- Running Total: DATESINPERIOD
- Others: TOTALYTD, SAMEPERIODLASTYEAR, PARALLELPERIOD
- **Custom Date Tables**: Use `CALENDAR` or `CALENDARAUTO` for custom date tables.
- **Week-Based Calculations**: Challenges arise when weeks start on different days, so DAX functions like `DATEADD` can simulate custom periods. Can't use standard DAX functions, weeks can start on various days - like sundays or mondays. There can be partial weeks in a given month quarter or year, and weeks can be grouped differently based on different fiscal calendars (5-4-4, 4-5-4).
- For previous fiscal week, we can use DATEADD with an interval of -7 days. This is the only time when we can use time-intel functions months and quarters won't work. So the workaround will be to use DATEADD and days arguments to simulate different periods
- Similarly for fiscal previous period, instead of period shifting functions, we can use CALCULATE, FILTER,ALL, SELECTEDVALUE

Auto Date tables: Time Intelligence Functions and Patterns

- Automatic date tables:
- Power BI automatically creates a hidden table for any table that contains a date/datetime column on the one-side of the relationship. This auto-generated calendar include all dates through the end of the year regardless of the actual date range in the table
- Pros: auto generated, enables a few time intelligence functionality by default, simplifies data model creation and management and does not require advanced understanding of DAX
- Cons: hidden from view, can’t be modified/customized, generated for every date field across every lookup/dim table bloating the model size, can’t be enabled or disabled at table-level, date hierarchies aren’t auto created, so for example if we are doing analysis by month, it can benefit granular by year and instead be grouped by month across all years in the table. And lastly, this automatic date table can only filter the table it corresponds to and can’t traverse table relationships
- It is better to turn off the auto date/time feature in Power BI and create a new calendar functions table or import a date dimension table

Manual Date Table: Time Intelligence Functions and Patterns

- Date Table requirements for date tables we create or import:
- Must contain all the days for all years represented in the fact tables
- Has at least 1 field set as a date/datetime datatype and can’t be duplicates
- Should be marked as a date table and all time columns must be identical
- If time is present in the the date field, split the time component into a new column since it will meet relationship requirements but also decrease column cardinality
- CALENDAR returns a table with 1 column of all dates between start and end dates (Remember date fields are stored as floating point numbers)
- CALENDARAUTO returns a table with 1 column of dates based on a fiscal year end month. The range of dates is calculated automatically based on data in the model. If we do not specify a specific table/date reference here like in calendar auto, then what happens is power BI scans our entire data model for date data types. And because time columns are internally seen and stored as dates, if it comes across a time column, it looks at that but does not have a reference to the preceding date part. And it goes to 1899 as the date because the first five digits of the floating date are zeros. We can test this by using the CALENDAR function and give 0000 as the first argument, it starts with 1899. Bottomline this is a default power bi quirk

Calculated Table Join Expressions

- **CROSSJOIN**: Cartesian product of two tables.
- **UNION**: Combines rows from multiple tables with the same structure.
- **EXCEPT**: Returns rows from one table not present in the other.
- **INTERSECT**: Returns rows common to both tables

Iterator Functions and Performance

- Common Iterators: `SUMX`, `AVERAGEX`, `RANKX`, `MAXX`, `MINX`.
- **When to Use**: Useful for row-by-row calculations.
- **Performance Tips**: Higher cardinality leads to slower performance; optimize by reducing the number of rows when possible
- Iterator Cardinality
  - number of rows in the table (if only 1) being iterated, the meow unique rows the higher the cardinality
  - Physical relationships will have cardinality equal to the max number of unique rows in the largest table
  - Virtual relationships will have cardinality as the product of the number of unique rows n each table
  - When nested iterators are used, only the innermost X function is optimized by the DAX engine

Relationship Functions and Types

- Common Functions: `RELATED`, `RELATEDTABLE`, `USERELATIONSHIP`, `CROSSFILTER`.
- **Virtual Relationships**: Created using `TREATAS`, connecting tables without physical relationships.
- **Expanded Tables**: When filters are applied to one side of a relationship, all related tables are also filtered

Scalar Functions

- Aggregation Functions: `SUM`, `AVERAGE`, `COUNT`, `DISTINCTCOUNT`.
- Rounding Functions: `ROUND`, `ROUNDUP`, `ROUNDDOWN`.
- **Logical Functions**: `IF`, `SWITCH`, `AND`, `OR`, `COALESCE` (better optimized than `IF + ISBLANK`).

Table and Filter Expressions

- Filtering Functions: `FILTER`, `ALL`, `ALLEXCEPT`, `REMOVEFILTERS`.
- **Table Creation**: Use `ADDCOLUMNS`, `SUMMARIZE` to generate calculated tables.
- Row and Table Creation: `ROW`, `GENERATESERIES`, `{}` (table constructor)

DAX Engine Functionality 1 of 2

- **Columnar Data Structures**: VertiPaq stores data in a compressed columnar format.
  - Storing data as individual columns rather than rows or tables. So 1 4 column table would have 4 individual columns represented

- **Storage and Formula Engines**: The formula engine processes DAX queries, while the storage engine handles data retrieval.
  - Formula engine receives, interprets and executes all DAX request
  - Storage engine compresses and encodes raw data and only communicates with formula engine - it does not understand DAX language
  - Formula engine processes the DAX query, generates a list of logical steps called a query plan - this query plan is then passed to storage engine, where it is executed, and the data is returned as a “datacache”
  - Formula engine then works with the returned data cache to evaluate the DAX query and returns the result
  - There are 2 types of storage engines based on the connection type we use: VertiPaq is used for data stored-in-memory (connected to Power BI via Impormode); DirectQuery is used for data read directly from the source (Azure, Postgres, …)

- **Compression Methods**: Includes value encoding, hash encoding, and run-length encoding.

DAX Engine Functionality 2 of 2

- **Compression Methods**: Includes value encoding, hash encoding, and run-length encoding.

•Compression and encoding

- •Goal is to reduce the amount of memory needed to evaluate a DAX query. So a smaller footprint means fewer computer resources to be used and so more efficient
- •Based on a sample of data, the vertical compression and encoding does:
  - •Value encoding (mathematical process to reduce the number of bits needed to store integer values - the math process determines relationships between the values in a column and convert them into smaller values for storage; it works only for integers and currency. For example, 10215 will be stripped to 215 etc..
  - •Hash encoding/dictionary encoding - identifies distinct string values and creates a new table with indexes - has is a dictionary and assigns a unique index for each unique item based on the cardinality of the column. For example, if column has coffee, tea and milk, the cardinality of the column is 3 and what it does is create a unique index for the 3 distinct values and maps it to the corresponding row number. It creates a separate lookup table that has the string value for each index, and so this eliminates the need to have repeating strings, it can have just integers and so saves space
  - •Run length encoding(RLE) - reduces the size of a dataset by identifying repeated values found in adjacent rows example in calendar table, quarter and month columns have repeating quarter, and so the engine does a RLE on the quarter data. However, RLE works only with consecutive rows of duplicate values, so vertipaq basically sorts data so it gets to collate the duplicates, and then, stores the data in a separate table with 2 columns - the value and the count of occurrences
  - •Vertipaq can also blend 2 different types of encoding/compression from the above for certain columns

•vertipaq relationships

- •Manages and stores relationships between columns in the data model, allowing to evaluate more complex multi-column queries
- •This relationship is not the same as table pk fk relationships. But it is more of an optimal way of extracting integer code equivalent, row number and index number and storing them separately, so even if we filter by string for example, it maps to its corresponding integer, index and rows, does a filter it that is what is asked by the DAX, and then returns the data cache
- •It also encodes, hashes and compresses the data internal representation that it keeps
- •If the cardinality in a column is higher, i.e. more unique values then more time is needed, vertipaq also optimizes if they are frequently used columns and column values

•Remember both import and direct query create data caches

DAX Tips: Formatting, Errors, and Variables

- **Query Formatting**: Use variables (`VAR`) for readability and performance.
- **Error Handling**: Use `IFERROR`, `ISBLANK`, but avoid overuse as it can impact performance.
- **Comments**: Add comments to make code readable and maintainable.

DAX Optimization Techniques

- Optimization Workflow:
- Use tools like DAX Studio and Performance Analyzer.
- Look for long-running queries (>100ms) and optimize them.
- **Optimization Tips**: Ensure most processing is done by the storage engine (VertiPaq), avoid complex calculated columns, and minimize the use of row context where possible