Power BI Optimization Analysis

By Mike Saul

Contents

Purpose:

To create a Financial Model in Power BI and to evaluate multiple different techniques using SQL Power Query and Power BI to deliver the most efficient, user-friendly, cost-effective dashboard for the same dashboard.

Procedure

Using the AdventureWorks2022DW dataset in SQL, a dataset which is commonly used for training purposes on SSMS, a data model was created for the purposes of building a Financial Model. These models were then evaluated to determine what is the most efficient, user-friendly and cost-effective based on Snowflake vs Star Schema as well as Direct Query vs Import and evaluated using various filtering techniques.

Models

4 main model types were created and evaluated using SQL and Power BI (each with 2 data connectivity modes for a total of 8 Power BI Reports). These models are:

1-FinancialModel-SnowflakeSchema

This method utilizes the Dimension Table and Fact Table pulls directly out of the AdventureWorks2022DW database with minimal changes made to any tables in the database. It uses a Snowflake Schema in Power BI.

- DirectQuery
- Import

2-FinancialModel-SnowflakeSchema

This method utilizes Horizontal and Vertical filtering of several of the AdventureWorks2022DW dataset through the use of views. It uses a Snowflake Schema in Power BI.

- DirectQuery
- Import

3-FinancialModel-StarSchema

This method utilizes mostly table pulls for comparability to the first model, however it also has altered Dimension tables so that the Power BI can be use Star Schema through DirectQuery. This model also shows how Power Query can be utilized to achieve the same Dimension Table Structure in Import mode.

- DirectQuery
- Import

4-FinancialModel-Fitlered-StarSchema

This is a model which uses only the filtered views created in the second model and uses only StarSchema.

- DirectQuery
- Import

Evaluations

Evaluation 1

For the purposes of this analysis, ***1-FinancialModel-SnowflakeSchema*** was chosen as the first model to build, mainly because it used the most data pulls. This will be compared to ***2-FinancialModel-Filtered-SnowflakeSchema*** to evaluate whether utilizing views is more efficient, user-friendly and cost-effective. Since they are both Snowflake Schema and the same model, these are good for comparability in determining the effects of Filtering and utilizing views in SSMS.

Evaluation 2

In this analysis, we compare ***1-FinancialModel-SnowflakeSchema*** to ***3-FinancialModel-StarSchema***. The ***3-FinancialModel-StarSchema*** was built using mostly unfiltered table pulls from SQL, except for two-dimension tables which were required to make this the comparing model Star Schema.

Evaluation 3

This analysis compares ***3-FinancialModel-StarSchema*** to ***4-FinancialModel-Filtered-StarSchema***. This serves to compare directly two Star Schema models to evaluate if a filtered version of the Star Schema model is the most optimal version.

Evaluation Criteria

We are trying to solve for how the different reports score on the following:

- Most Efficient and User-Friendly
- Cost-Effective

While User Friendly is somewhat subjective depending on the user, as well as difficult to measure. The front ends of each report are all consistent with each other. User-Friendly in this instance can be considered part of the criteria with Most-Efficient.

Efficiency will be measured through the following:

Through Power BI

- Whether a report utilizes steps in Power Query (Power BI)
    - If there are multiple steps in Power Query, this can slow down a report significantly.
  - .pbix file size (Power BI)
    - Measured by the size of each file.
  - .pbix refresh CPU Usage (Power BI)
    - Measured by the Task Manager performance CPU, the max point of CPU usage is recorded during a refresh.
  - Power BI Visuals Performance (Power BI)
    - Using the Performance Analyzer in Power BI, finding the Max Time of all visuals after a refresh.

Through SSMS

  - Cached Plan Size
    - This is an indicator of memory used in the query. If this number is small, this means it is more efficient.
  - Estimated Subtree Cost
    - This is an indication of cost every time a query is used. Since the only queries we expect to be regularly updated are the Fact tables, we will evaluate this on the Fact Tables only.
  - Estimated Number of Rows per Execution
    - This is an indication of how many rows are in the dataset, this also indicates size, when combined with Cached Plan Size, it gives a good indication of the height and width of a table.

Analysis

Evaluation 1

This analysis compares ***1-FinancialModel-SnowflakeSchema*** to ***2-FinancialModel-Filtered-SnowflakeSchema***. See Appendix I for necessary code in SSMS for this to work.

We will first compare the two models based on their evaluation criteria:

	1-FinancialModel-SnowflakeSchema		**2-FinancialModel-Filtered-SnowflakeSchema**	
Data Connectivity Mode	Direct Query	Import	Direct Query	Import
Schema	Snowflake	Snowflake	Snowflake	Snowflake
Utilizes Power Query	No	No	No	No
.pbix size	94 KB	1161 KB	103 KB	504 KB
.pbix refresh speed (seconds)	3.92	7.64	3.74	4.52
.pbix refresh CPU Usage	90%	100%	96%	92%
Power BI Visuals – Max Time	2383	1039	1508	811
Cached Plan Size	208 KB	208 KB	200 KB	200 KB
Estimated Subtree Cost – Fact Tables	0.3784337	0.3784337	0.3376802	0.3376802
Estimated Number of Rows per Execution	57599	57599	38353	38353

Findings:

- Using a filtered .pbix on Direct Query resulted in a bigger .pbix size of 103 KB vs 94 KB.
- Refresh speed increased on the Filtered .pbix in both Import and Direct Query with the biggest speed improvement on the Import model.
- CPU Usage varied greatly in a way that does not tell us anything.
- By filtering the dataset, the Power BI Visuals were noticeably quicker. Speed was 2383 vs 1508 and 1039 vs 811.
- Cached Plan Size in SSMS decreased from 208 KB to 200 KB by filtering the data.
- Estimated Subtree Cost on the Fact Table pulls decreased from 0.378 to 0.337.
- Estimated Number of Rows per Execution was down from 38353

Conclusion:

Filtering the dataset in SSMS did indicate better performance in SSMS and Power BI. Other techniques could be used to further optimize these queries so that they are even more efficient. Things like filtering based on keys or views based on reference lists could help to speed it up. It is recommended to use. More filtering based on other factors could help to reduce this more. Based on all of these factors including scaling up, between these 4 options, I would recommend 2-FinancialModel-Filtered-Snowflake Schema Direct Query as it has quicker visuals, is cheaper and only utilizes the important necessary data for the report.

Evaluation 2

This analysis compares ***1-FinancialModel-SnowflakeSchema*** to ***3-Financial Model-StarSchema***. See Appendix I and Appendix II for necessary code in SSMS for this to work.

We will first compare the two models based on their evaluation criteria:

	1-FinancialModel-SnowflakeSchema		**3-FinancialModel-StarSchema**	
Data Connectivity Mode	Direct Query	Import	Direct Query	Import
Schema	Snowflake	Snowflake	Star	Star
Utilizes Power Query	No	No	No	Yes
.pbix size	94 KB	1161 KB	88 KB	1164 KB
.pbix refresh speed (seconds)	3.92	7.64	2.96	8.43
.pbix refresh CPU Usage	90%	100%	88%	88%
Power BI Visuals – Max Time	2383	1039	2303	1669
Cached Plan Size	208 KB	208 KB	160 KB	160 KB
Estimated Subtree Cost – Fact Tables	0.3784337	0.3784337	0.3784337	0.3784337
Estimated Number of Rows per Execution	57599	57599	57495	57495

Findings:

- Using a Star Schema approach .pbix on Direct Query resulted in a .pbix size that is slightly smaller while using Direct Query but slightly bigger using Import Query.
    - The reason Import is more on the Star Schema size is likely due to the Power Query that is used. These can be performed easier using SQL.
- Refresh speed was significantly faster using Star Schema as compared to Snowflake Schema while using Direct Query, interestingly it took longer using Import.
- CPU Usage was less using Star Schema for both DirectQuery and Import. 90% to 88% in favor of Star Schema for Direct Query and 100% to 88% in favor of Star Schema for Import.
- Power BI visuals for this were slightly quicker for Star Schema using Direct Query and slower using Import.
- Cached Plan Size in SSMS decreased from 208 KB to 160 KB by filtering the data resulting in a smaller memory footprint when using Star Schema.
- Since there was no change to the Fact Tables when using Star Schema as the only change made was to Dimension tables, there is no change to the Estimated Subtree Cost – Fact Tables.
- Estimated Number of Rows per Execution was down slightly from 57599 to 57495, only slightly as the changes were done to Dimension Tables and not Fact Tables.

Other Considerations:

- Pulling data directly from the source in Power BI offers greater flexibility, especially when making edits. Filtering data at the source can limit your ability to make further changes in the report without first updating your SQL views.
- Importing data vs. Direct Querying data has major impacts on the size and speed of a report. Based on those 2 alone, it would be worth using Import mode, however some issues can be present while using Import mode.
    - One issue would be the user could accidentally mess up the logic in the report by inputting an error or accidentally changing the joins in the table. There are near infinite methods to mess up a report.
    - Scaling up is challenging in an organization when someone is using and sharing .pbix reports. This can lead to multiple sources of truth and misinformation in an organization.
    - There are some major challenges to using Direct Query including the limited functionality of using Dax, Direct Query limits the use of adding a column and may require more work to test out data.
    - Import mode is generally beneficial for proof of concept, however while doing this, it is important to understand the limitations on a Direct Query model. Some of those functions would need to be performed in SSMS while some can be performed using DAX.
    - Direct Query when scaled up will pull more on an organizations resource, as each interaction queries against the source system. Thus, the incremental cost per pull is not always a good measure of how much this will cost when there are multiple users, querying multiple times.
- Direct Query is needed for automatic refreshes in the Power BI Service, which is used to disseminate information. Import can be good depending on the users and their familiarity; however, Import can severely limit scaling up.

Conclusion:

Using Star Schema does have some advantages, particularly on the Cached plan size. If your Dimension tables are short and the columns are not too wide, it can be effectively utilized to make a simple hierarchy which is easy to follow and understand. It is noticeably quicker while using Direct Query and scale up using Star Schema as it pulls on the database less.

Evaluation 3

This analysis compares 3-FinancialModel-StarSchema to 3-Financial Model-Filtered-StarSchema. See the Power Query steps associated with 3-FinancialModel-StarSchema Import to find out how Power Query can be used to create the same table as in SQL.

We will first compare the two models based on their evaluation criteria:

	3-FinancialModel-StarSchema		**4-FinancialModel-Filtered-StarSchema**	
Data Connectivity Mode	Direct Query	Import	Direct Query	Import
Schema	Star	Star	Star	Star
Utilizes Power Query	No	Yes	No	No
.pbix size	88 KB	1164 KB	103 KB	519 KB
.pbix refresh speed (seconds)	2.96	8.43	2.84	4.1
.pbix refresh CPU Usage	88%	88%	91%	83%
Power BI Visuals – Max Time	2303	1669	3351	1447
Cached Plan Size	160 KB	160 KB	160 KB	160 KB
Estimated Subtree Cost – Fact Tables	0.3784337	0.3784337	0.3378802	0.3376802
Estimated Number of Rows per Execution	57495	57495	38300	38300

Findings:

- Using a filtered Star Schema approach .pbix on Direct Query resulted in a .pbix size that is slightly larger than using the model which pulls from Tables instead of views in SSMS.
- Refresh speed was slightly quicker using a filtered Star Schema approach as compared to non filtered while using Direct Query, interestingly it took longer using Import.
- CPU Usage was less using the non-filtered approach for both DirectQuery but was more when using the Import method. Import method for Filtered Star Schema resulted in significantly less CPU Usage.
- Power BI visuals for this were slightly quicker for the filtered approach when using Import, but were significantly longer when using Direct Query. On the Import side, this is likely because it is bringing in less rows and columns.
- Cached Plan Size was consistent at 160 KB between both methods
- Estimated Subtree Cost was lower when using the filtered approach from 0.378 to 0.337
- Estimated Number of Rows per Execution was down significantly from 57495 to 38300, only slightly as the changes were done to Dimension Tables and not Fact Tables.

Conclusion:

Using a filtered approach with Star Schema is advantageous efficiency and storage wise especially when using Import mode as it does not use as many rows and has a lower Estimated Subtree Cost on pulling information from the Fact Tables. It is also cheaper as the Direct Query option providing the best speed and efficiency. The differences in speed

on the Power BI Report are slightly more but almost negligible in terms of increasing time for the user.

General Observations:

Direct Query vs. Import

Direct Query

When to use:

This should be used when working with large data sets and when there is more than 1 user of a report. It is recommended to create it after creating a prototype using assumptions that work in DirectQuery models, for example, don't use Add Column function or Power Query steps.

This is useful when speed and price is not the biggest concern, ideally at an early stage of getting user feedback, this allows for less data pulls from the server while allowing for more flexibility to quickly create changes to a model and gather user feedback before further optimization is needed.

Advantages:

- DirectQuery provides Real-Time data access that is as current as the SQL database.
- DirectQuery is more effective when using Large Datasets as compared to Importing data as it does not store the data within the Power BI Report.
- Direct Query models ensure a single source of truth for a report resulting in consistent data and assumptions and less errors.
- Direct Query data remains in the database and can be limited to certain audiences based on their security clearance.
- Snowflake is beneficial for optimizing Queries when it is set up correctly.
- Data is normalized in most Snowflake structures, which can save storage space.

Disadvantages:

- Direct Query does not allow users in Power BI to add columns or for other DAX functionality. These kinds of functions need to be made through other means such as SSMS or wherever data is stored.
- Data is queried against the database every time a report refreshes, which depending on the number of refreshes, can become quite costly depending on the dataset.
- This version is non-filtered, meaning there are many unnecessary columns included in the report, depending on the report, some of this data may be confidential and not for the end user to see.

Import

When to use:

Import mode is beneficial when data coming from a dataset is not great. It is very useful when performing an analysis over excel data, it is great for utilizing Power Query which helps to document changes that need to be made to data before a more optimal future solution is used to store data. Import mode is great for prototyping as there is more DAX Functionality, which helps to identify what needs to be collected in the back-end system for a Direct Query model. Import mode is beneficial when the refreshed report is not needed frequently.

Advantages:

- Internet connectivity does not limit the ability to use the system.
- Data can be automatically refreshed based on a schedule.
- More DAX functionality is available when using Import mode which means more can be created in terms of visualizations. Users are able to create new tables, columns and functions that aren't available in DirectQuery.
- Power BI Compresses data which can allow for large datasets to be used, however if a dataset gets too large, it can slow down significantly.
- Visuals refresh quicker using Import mode as opposed to DirectQuery resulting in better performance.
- Can be a more cost-effective option than regularly querying the dataset in a DirectQuery mode. This method can Query the source database as much as you want it too.

Disadvantages:

- Data is only as relevant as the last refresh.
- Refresh could take a long time if the dataset is too large.
- .pbix report may not be able to handle a certain size of report.
- Refreshing data needs to be scheduled at appropriate times. If it is a large dataset, then this could greatly affect user's time.
- Imported data is duplicated in Power BI meaning that while the data reflects the source of truth, it is still not the source of truth and will be inconsistent with the source data.
- This is a .pbix file with data stored inside which has security implications, once imported into the .pbix report, it isn't subject to the security requirements of the source system.
- Since it operates on different functionality than Direct Query, it can be difficult to implement the same DAX functionality while going to DirectQuery mode.

Snowflake vs. Star Schema

Snowflake Schema

When to use:

Snowflake Schema is good to use when you have normalized data and high data quality. It is good to use when your dataset has complex relationships which can effectively and simply represent and otherwise complex hierarchy. Snowflake schemas are normalized which can result in better query performance.

Advantages:

- This method is easier to scale to a larger audience as it can handle large and complex datasets efficiently.
- This method creates easier audit trails in terms of data and the relationships between datasets which can result in lower audit costs and more simple compliance.
- Very beneficial for scenarios which require complex hierarchies within a dataset.
- Easier to maintain than many Star Schema models.
- Normalized databases will work better with certain database engines, which can save a company money.
- Normalizing data will also reduce the overall size of the data which is more economical.

Disadvantages:

- Data Model complexity is greater with Snowflake Schema than Star Schema.
- More Joins are required which can limit query performance.
- More set up is required in creating a report with Snowflake Schema as there are more tables and joins required.
- When users aren't familiar with Snowflake Schema, training may be required to teach someone about its intricacies.
- When changes are made to one table, this can trigger required changes to other tables.
- Snowflake Schema uses more storage than Star Schema.

Star Schema

When to use:

This approach can be used when there are business users and analysts who need to regularly work with and understand the model, it is simpler and easier to understand than snowflake schema. If storage and RAM isn't an issue for a user, this is a good method to use.

Advantages:

- This approach minimizes the number of joins required which can lead to faster query performance.
- This method is simpler and easier for new users to learn and to use.
- It allows for quicker ad-hoc analysis to pull information out of a system.
- This is generally smaller than Snowflake Schemas as this approach requires less tables to be uploaded.
- Consistency is ensured through one consistent join to a fact table, it becomes easier to determine if something is incorrectly loaded.

Disadvantages:

- Working with Hierarchical data is more complex, this often has to be made in wider dimension tables with joins required to itself.
- This method has scalability issues, with a large fact table, and many dimensions, this can result in large data usage and unnecessary storage and time.
- As Dimensions increase, efficiencies of queries decrease.
- Less flexible when handling changes in data structure or any business requirements.

Overall Optimal Recommendation

A recommendation should be made depending on a user's situation and familiarity with Power BI.

I would recommend for the typical user ***4-FinancialModel-Filtered-StarSchema*** as it balanced the benefit to the average user in terms of having the lowest cost, lowest storage used, and better performance. Between the 2 choices for Import or Direct Query, I would evaluate such things as frequency of the report and number of users before updating. Since this is a filtered report, I will recommend ***Direct Query*** as although there is additionally functionality that can be gained in DAX, this will be limited by the limited number of columns available. Again, this recommendation would be conditional on a user's situation and organization.

Appendix I

1-FinancialModel-SnowflakeSchema

Procedures:

1-Creating ParentOrganizationAlternateKey in DimOrganization:

```
-- Add the new column to the DimOrganization table
ALTER TABLE DimOrganization
ADD ParentOrganizationAlternateKey INT;
```

2-Populating values in ParentOrganizationAlternateKey in DimOrganization:

```
UPDATE DimOrganization
SET ParentOrganizationAlternateKey = CASE
                WHEN OrganizationKey = 10 THEN 10
                WHEN OrganizationKey = 9 THEN 9
                WHEN OrganizationKey = 2 THEN 2
                WHEN OrganizationKey = 8 THEN 8
                WHEN OrganizationKey = 14 THEN 14
                WHEN OrganizationKey = 11 THEN 11
                WHEN OrganizationKey = 12 THEN 12
                WHEN OrganizationKey = 13 THEN 13
                WHEN OrganizationKey = 3 THEN 14
                WHEN OrganizationKey = 4 THEN 14
                WHEN OrganizationKey = 5 THEN 14
                WHEN OrganizationKey = 6 THEN 14
                 WHEN OrganizationKey = 7 THEN 14
                 ELSE NULL
END;
```

3-CreateDimParentAccountOrderIS

```
-- Create the new table
CREATE TABLE DimParentAccountOrder (
   AccountKey INT PRIMARY KEY,
   AccountCodeAlternateKey INT,
   ParentAccountCodeAlternateKey INT,
   ParentAccountKey INT,
   AccountOrderNumber INT,
   ParentAccountDescription VARCHAR(255),
   ParentAccountOrderNumber INT,
   AccountDescription VARCHAR(255)
);

-- Insert data into the new table
INSERT INTO DimParentAccountOrder (
   AccountKey,
   AccountCodeAlternateKey,
   ParentAccountCodeAlternateKey,
   ParentAccountKey,
   AccountOrderNumber,
   ParentAccountDescription,
   ParentAccountOrderNumber,
   AccountDescription
)
SELECT
   da.AccountKey,
   da.AccountCodeAlternateKey,
   CASE
      WHEN da.AccountCodeAlternateKey = 4 THEN 4
      WHEN da.AccountCodeAlternateKey = 60 THEN 60
      WHEN da.AccountCodeAlternateKey = 80 THEN 80
      WHEN da.AccountCodeAlternateKey = 600 THEN 600
      WHEN da.AccountCodeAlternateKey = 620 THEN 620
      WHEN da.AccountCodeAlternateKey = 630 THEN 630
      WHEN da.AccountCodeAlternateKey = 660 THEN 660
      WHEN da.AccountCodeAlternateKey = 680 THEN 680
```

```
    WHEN da.AccountCodeAlternateKey = 4100 THEN 4100
    WHEN da.AccountCodeAlternateKey = 4110 THEN 4110
    WHEN da.AccountCodeAlternateKey = 5000 THEN 5000
  END AS ParentAccountCodeAlternateKey,
  da2.AccountKey AS ParentAccountKey,
  CASE
    WHEN da.AccountCodeAlternateKey = 4 THEN 11
    WHEN da.AccountCodeAlternateKey = 60 THEN 4
    WHEN da.AccountCodeAlternateKey = 80 THEN 10
    WHEN da.AccountCodeAlternateKey = 600 THEN 5
    WHEN da.AccountCodeAlternateKey = 620 THEN 6
    WHEN da.AccountCodeAlternateKey = 630 THEN 7
    WHEN da.AccountCodeAlternateKey = 660 THEN 8
    WHEN da.AccountCodeAlternateKey = 680 THEN 9
    WHEN da.AccountCodeAlternateKey = 4100 THEN 2
    WHEN da.AccountCodeAlternateKey = 4110 THEN 1
    WHEN da.AccountCodeAlternateKey = 5000 THEN 3
  END AS AccountOrderNumber,
  CASE
    WHEN da.AccountCodeAlternateKey = 4 THEN 'Tax'
    WHEN da.AccountCodeAlternateKey = 60 THEN 'Operating Expenses'
    WHEN da.AccountCodeAlternateKey = 80 THEN 'Other Income and Expense'
    WHEN da.AccountCodeAlternateKey = 600 THEN 'Operating Expenses'
    WHEN da.AccountCodeAlternateKey = 620 THEN 'Operating Expenses'
    WHEN da.AccountCodeAlternateKey = 630 THEN 'Operating Expenses'
    WHEN da.AccountCodeAlternateKey = 660 THEN 'Operating Expenses'
    WHEN da.AccountCodeAlternateKey = 680 THEN 'Operating Expenses'
    WHEN da.AccountCodeAlternateKey = 4100 THEN 'Total Sales'
    WHEN da.AccountCodeAlternateKey = 4110 THEN 'Total Sales'
    WHEN da.AccountCodeAlternateKey = 5000 THEN 'Total Cost of Sales'
  END AS ParentAccountDescription,
  CASE
    WHEN da.AccountCodeAlternateKey = 4 THEN 5
    WHEN da.AccountCodeAlternateKey = 60 THEN 3
    WHEN da.AccountCodeAlternateKey = 80 THEN 4
    WHEN da.AccountCodeAlternateKey = 600 THEN 3
    WHEN da.AccountCodeAlternateKey = 620 THEN 3
    WHEN da.AccountCodeAlternateKey = 630 THEN 3
    WHEN da.AccountCodeAlternateKey = 660 THEN 3
    WHEN da.AccountCodeAlternateKey = 680 THEN 3
    WHEN da.AccountCodeAlternateKey = 4100 THEN 1
    WHEN da.AccountCodeAlternateKey = 4110 THEN 1
    WHEN da.AccountCodeAlternateKey = 5000 THEN 2
  END AS ParentAccountOrderNumber,
  CASE
    WHEN da.AccountCodeAlternateKey = 4 THEN 'Tax'
    WHEN da.AccountCodeAlternateKey = 60 THEN 'Operating Expenses'
    WHEN da.AccountCodeAlternateKey = 80 THEN 'Other Income and Expense'
    WHEN da.AccountCodeAlternateKey = 600 THEN 'Labor Expenses'
    WHEN da.AccountCodeAlternateKey = 620 THEN 'Travel Expenses'
    WHEN da.AccountCodeAlternateKey = 630 THEN 'Marketing'
    WHEN da.AccountCodeAlternateKey = 660 THEN 'Telephone and Utilities'
    WHEN da.AccountCodeAlternateKey = 680 THEN 'Depreciation'
    WHEN da.AccountCodeAlternateKey = 4100 THEN 'Net Sales'
    WHEN da.AccountCodeAlternateKey = 4110 THEN 'Gross Sales'
    WHEN da.AccountCodeAlternateKey = 5000 THEN 'Total Cost of Sales'
  END AS AccountDescription
FROM DimAccount da
JOIN DimAccount da2 ON da.AccountCodeAlternateKey = da2.AccountCodeAlternateKey
WHERE da.AccountCodeAlternateKey IN (4, 60, 80, 600, 620, 630, 660, 680, 4100, 4110, 5000);
```

CreatingDimParentAccountAndOrderNumber

```
CREATE TABLE DimParentAccount (
  AccountKey INT PRIMARY KEY,
  AccountCodeAlternateKey INT,
  AccountOrderNumber INT,
  ParentAccountDescription VARCHAR(255)
);

INSERT INTO DimParentAccount (AccountKey, AccountCodeAlternateKey, AccountOrderNumber, ParentAccountDescription)
SELECT
```

```
  da.AccountKey,
  da.AccountCodeAlternateKey,
  CASE
     WHEN da.AccountCodeAlternateKey = 4 THEN 11
     WHEN da.AccountCodeAlternateKey = 60 THEN 4
     WHEN da.AccountCodeAlternateKey = 80 THEN 10
     WHEN da.AccountCodeAlternateKey = 600 THEN 5
     WHEN da.AccountCodeAlternateKey = 620 THEN 6
     WHEN da.AccountCodeAlternateKey = 630 THEN 7
     WHEN da.AccountCodeAlternateKey = 660 THEN 8
     WHEN da.AccountCodeAlternateKey = 680 THEN 9
     WHEN da.AccountCodeAlternateKey = 4100 THEN 2
     WHEN da.AccountCodeAlternateKey = 4110 THEN 1
     WHEN da.AccountCodeAlternateKey = 5000 THEN 3
  END AS AccountOrderNumber,
  CASE
     WHEN da.AccountCodeAlternateKey = 4 THEN 'Tax'
     WHEN da.AccountCodeAlternateKey = 60 THEN 'Operating Expenses'
     WHEN da.AccountCodeAlternateKey = 80 THEN 'Other Income and Expense'
     WHEN da.AccountCodeAlternateKey = 600 THEN 'Operating Expenses'
     WHEN da.AccountCodeAlternateKey = 620 THEN 'Operating Expenses'
     WHEN da.AccountCodeAlternateKey = 630 THEN 'Operating Expenses'
     WHEN da.AccountCodeAlternateKey = 660 THEN 'Operating Expenses'
     WHEN da.AccountCodeAlternateKey = 680 THEN 'Operating Expenses'
     WHEN da.AccountCodeAlternateKey = 4100 THEN 'Total Sales'
     WHEN da.AccountCodeAlternateKey = 4110 THEN 'Total Sales'
     WHEN da.AccountCodeAlternateKey = 5000 THEN 'Total Cost of Sales'
  END AS ParentAccountDescription
FROM DimAccount da
WHERE da.AccountCodeAlternateKey IN (4, 60, 80, 600, 620, 630, 660, 680, 4100, 4110, 5000);
```

Appendix II

2-FinancialModel-Filtered-SnowflakeSchema

Procedures:
1-Create dbo.vw_FactFinanceVFHF

```
CREATE VIEW dbo.vw_FactFinanceVFHF AS
SELECT
  OrganizationKey,
  DateKey,
  ScenarioKey,
  AccountKey,
  CalculatedAmountRevPlusExpMinus,
  Amount
FROM
  FactFinance
WHERE
  OperatorRevPlusExpMinus IS NOT NULL;
```

2-Create dbo.vw_DimOrganizationVF

```
CREATE VIEW dbo.vw_DimOrganizationVF AS
SELECT
  CurrencyKey,
  ParentOrganizationAlternateKey,
  ParentOrganizationKey,
  OrganizationKey,
  OrganizationName

FROM
  DimOrganization;
```

3-Create dbo.vw_FactCurrencyRateVFHF

```
CREATE VIEW dbo.vw_FactCurrencyRateVFHF AS
SELECT
  AverageRate,
  CurrencyKey,
  Date,
  DateKey
FROM
  FactCurrencyRate
WHERE
  CurrencyKey IN (6, 19, 36, 100)
```

4-Create dbo.vw_DimAccountVFHF

```
CREATE VIEW dbo.vw_DimAccountVFHF AS
SELECT
        AccountKey,
        AccountDescription,
        ParentAccountKey
FROM
         DimAccount
WHERE
        OperatorRevPlusExpMinus IS NOT NULL
```

5-Create dbo.vw_DimDateVF

```
CREATE VIEW dbo.vw_DimDateVF AS
SELECT
        DateKey,
        FiscalYear,
        EnglishMonthName,
```

```
        FiscalMonthNumber
FROM
  DimDate;
```

6-Create dbo.vw_DimCurrency

```
CREATE VIEW dbo.vw_DimCurrency AS
SELECT
        CurrencyAlternateKey,
        CurrencyKey,
        CurrencyName
FROM
  DimCurrency
```

7-Create dbo.vw_DimParentAccountOrderVF

```
CREATE VIEW dbo.vw_DimParentAccountOrderVF AS
SELECT
         AccountKey,
        AccountDescription,
        ParentAccountDescription,
        AccountOrderNumber,
        ParentAccountOrderNumber,
        ParentAccountKey
FROM
  DimParentAccountOrder;
```

8 - Create dbo.vw_DimScenario

```
CREATE VIEW dbo.vw_DimScenario AS
SELECT
  ScenarioKey,
  ScenarioName
FROM
  DimScenario
```