Calculation items

```
MTD = CALCULATE(SELECTEDMEASURE(), DATESMTD(DimDate[Date]))

QTD = CALCULATE(SELECTEDMEASURE(), DATESQTD(DimDate[Date]))

YTD = CALCULATE(SELECTEDMEASURE(), DATESYTD(DimDate[Date]))

PY = CALCULATE(SELECTEDMEASURE(), SAMEPERIODLASTYEAR(DimDate[Date]))

SalesAmountPYMTD = CALCULATE(SUM(FactInternetSales[SalesAmount]), SAMEPERIODLASTYEAR(DimDate[Date]),'Time
Intelligence'[Time Calculation] = "MTD")

PY MTD = CALCULATE(SELECTEDMEASURE(),SAMEPERIODLASTYEAR(DimDate[Date]),'Time Intelligence'[Time Calculation] = "MTD")

PY QTD = CALCULATE(SELECTEDMEASURE(),SAMEPERIODLASTYEAR(DimDate[Date]),'Time Intelligence'[Time Calculation] = "QTD")

PY YTD = CALCULATE(SELECTEDMEASURE(),SAMEPERIODLASTYEAR(DimDate[Date]),'Time Intelligence'[Time Calculation] = "YTD")

YOY = SELECTEDMEASURE() - CALCULATE(SELECTEDMEASURE(), 'Time Intelligence'[Time Calculation] = "PY")

YOY% = DIVIDE(CALCULATE(SELECTEDMEASURE(),'Time Intelligence'[Time Calculation]="YOY"),
CALCULATE(SELECTEDMEASURE(),'Time Intelligence'[Time Calculation]="PY"))

SalesAmountYOY% = CALCULATE([SumOfSalesAmount], 'Time Intelligence'[Time Calculation] = "YOY%")

Field chooser = {
    ("SalesAmount", NAMEOF('FactInternetSales'[SumOfSalesAmount]), 0),
    ("OrderQuantity", NAMEOF('FactInternetSales'[OrderQuantitySum]), 1),
    ("ExtendedAmount", NAMEOF('FactInternetSales'[ExtendedAmountSum]), 2)
}
```

Using DAX variables

```
SalesAmountCalculation =
VAR MonthToDate = CALCULATE(SUM(FactInternetSales[SalesAmount]),DATESMTD(FactInternetSales[DueDate]))
VAR CurrentSales = SUM(FactInternetSales[SalesAmount])
RETURN (MonthToDate - CurrentSales) / MonthToDate
```

ROWNUMBER

```
RowNumberColumn = ROWNUMBER(
    ALLSELECTED(DimProduct[EnglishProductSubcategoryName],DimProduct[EnglishProductCategoryName]),
    ORDERBY(DimProduct[EnglishProductSubcategoryName],ASC),
    PARTITIONBY(DimProduct[EnglishProductCategoryName]))

RowNumberColumn2 = ROWNUMBER(
    ALLSELECTED(DimProduct[EnglishProductSubcategoryName],DimProduct[EnglishProductCategoryName]),
    ORDERBY(DimProduct[EnglishProductCategoryName],ASC))
```

RANK

```
RankNumberColumnDense = RANK(DENSE,
    ALLSELECTED(DimProduct[EnglishProductSubcategoryName],DimProduct[EnglishProductCategoryName]),
    ORDERBY(DimProduct[EnglishProductCategoryName],ASC))

RankNumberColumnSkip = RANK(SKIP,
    ALLSELECTED(DimProduct[EnglishProductSubcategoryName],DimProduct[EnglishProductCategoryName]),
    ORDERBY(DimProduct[EnglishProductCategoryName],ASC))
```

INDEX

```
SummaryTable = SUMMARIZECOLUMNS(DimDate[CalendarYear], DimProduct[EnglishProductCategoryName], "SalesAmount",
SUM(FactInternetSales[SalesAmount]))
```

```
IndexCalculation = INDEX(2, ALL(SummaryTable[CalendarYear]), ORDERBY(SummaryTable[CalendarYear], DESC))
```

OFFSET

```
TheOtherYearSales = CALCULATE(SUM(SummaryTable[SalesAmount]), OFFSET(1, ,
ORDERBY(SummaryTable[EnglishProductCategoryName],ASC, SummaryTable[CalendarYear])))
```

WINDOW

```
RunningTotal = CALCULATE(
    sum(FactInternetSales[SalesAmount]),
    WINDOW(-1, REL, 0, REL, ORDERBY(DimDate[CalendarYear])))
```

Dynamic strings

```
if(MIN(MtoMTransactions[Currency])<>MAX(MtoMTransactions[Currency]),"Multiple",
if(MIN(MtoMTransactions[Currency])="USD","$#,##0",
if(MIN(MtoMTransactions[Currency])="EUR","€#,##0",
if(MIN(MtoMTransactions[Currency])="GBP","£#,##0",
MIN(MtoMTransactions[Currency]) & " #,##0"))))
```

Loading data from csv file into a dataframe using Spark

```
df = spark.read.option("header","true").format("csv").load("Files/MtoMActual.csv")

df2 =
spark.read.csv("abfss://FabricWorkspace@onelake.dfs.fabric.microsoft.com/DemoLakehouse.Lakehouse/Files/MtoMActual.csv")
```

```
df2 =
spark.read.load("abfss://FabricWorkspace@onelake.dfs.fabric.microsoft.com/DemoLakehouse.Lakehouse/Files/MtoMActual.csv",
format='csv', header=True)
```

Loading data from csv file into a dataframe using Pandas

```
import pandas as pd
# Load data into pandas DataFrame from "/lakehouse/default/" + "Files/MtoMActual.csv"
df = pd.read_csv("/lakehouse/default/Files/MtoMActual.csv")
display(df)
```

Saving data from a dataframe to a csv file or table

```
df.write.mode("overwrite").format("csv").save("Files/MtoMActual2.csv")

df.write.mode("overwrite").format("delta").saveAsTable("MtoMActual")
```

Loading data from a table

```
df = spark.read.table("Mtomactual")
display(df)

df2 = spark.read.format("delta").load("Tables/mtomactual")
display(df2)
```

```
%%sql
SELECT *
FROM mtomactual
```

Other ways to display data

```
df.collect()
df.schema
df.summary()
df.show()
```

Reducing the number of columns shown

```
dfreduced = df.select("Country", "Actual") # This is not case sensitive.
display(dfreduced)

dfreduced = df.select(df.Country, df.Actual) # This IS case sensitive.
display(dfreduced)

dfreduced = df.select(df.Country, df.Actual.alias("ActualSales"))
display(dfreduced)
```

```
%%sql
SELECT Country, Actual
FROM mtomactual; -- Semicolon needed if you have two statements in the one cell.

SELECT Country, Actual AS `Actual Sales` -- Need to use backticks if you are including a space. The “AS” is optional.
FROM mtomactual;
```

Filter data with a simple “where”

```
df = spark.read.table("Mtomactual")
dfreduced = df.select(df.Country, df.Actual.alias("ActualSales"))
display(dfreduced.where("ActualSales > 10000"))           # Use >, <, >=, <=, = or ==, != or <>
display(dfreduced.filter(dfreduced.Country == "England")) # Use >, <, >=, <=,      ==, !=.      Cannot use = or <>
display(dfreduced.limit(3))
display(dfreduced.tail(2))
```

```
%%sql
SELECT Country, Actual AS ActualSales
FROM mtomactual
WHERE Actual > 10000
LIMIT (2) -- cannot use TOP(2)
```

Adding additional columns

```
df = spark.read.table("Mtomactual")
display(df.withColumn("ActualDoubled", df.Actual * 2)\
          .withColumn("SecondLetter", df.Country.substr(2,1))
          )
```

```
from pyspark.sql.functions import *
df2 = df.withColumn("ColDate", add_months(lit("2028-01-01"), df.Actual/1000))
display(df2)
df2.write.mode("overwrite").format("delta").saveAsTable("MtoMActualWithDates")

df = spark.read.table("mtomactualwithdates")
display(df)
```

```
%%sql
SELECT *, Actual * 2 AS ActualDoubled, SUBSTR(Country, 2, 1) AS SecondLetter
FROM mtomactual;

SELECT *
FROM mtomactualwithdates
```

Advanced Filtering

```
df = spark.read.table("mtomactualwithdates")
display(df.where(df.Actual.between(6000, 11000)))
display(df.where(df.Country.contains("n")))
display(df.where(df.Country.like("%n_")))
display(df.where("Country = 'England' OR Country = 'France'"))
display(df.where( (df.Country == 'England') |  (df.Country == 'France')) )     # | = OR,    & = AND,    ~ = NOT
display(df.where(df.Country.isin("England", "France")))
```

```
%%sql
SELECT *
FROM mtomactualwithdates
WHERE Actual BETWEEN 6000 AND 11000
WHERE Country LIKE '%n_' -- %=0, 1, or more characters,   _ = 1 character.
WHERE Country = "England" OR Country = "France"
WHERE Country IN ("England", "France")
```

Convert data types

```
df = spark.read.table("mtomactual")
df = df.select(df.Country, df.Location, df.Actual.cast("int"))
display(df)
display(df.describe(["Country", "Actual"]).show())
```

```
%%sql
SELECT Country, Location, CAST(Actual as int)
FROM mtomactual
```

Importing data using a different data structure

```
from pyspark.sql.types import *
```

```
schemaTarget = StructType([StructField('Country', StringType(), True),
                           StructField('Location', StringType(), True),
                           StructField('Actual', IntegerType(), True)])
df = spark.read.option("header","true").format("csv").schema(schemaTarget).load("Files/MtoMActual.csv")
display(df)
df.schema
df.write.format("delta").saveAsTable("mtomactualstruct")

df = spark.read.table("mtomactualstruct")
display(df)
df.schema
```

Formatting dates as strings

```
from pyspark.sql.functions import *
df = spark.read.table("mtomactualwithdates")
df = df.select(df.Country, df.Location, df.Actual.cast("int"), \
               concat(lit("The date is: "),date_format(df.ColDate, "EEEE d MMMM yyyy")).alias("FormattedDate")
               )
display(df)
```

```
%%sql
SELECT Country, Location, CAST(Actual as int), concat("The date is: ",date_format(ColDate, "EEEE d MMMM yyyy")) AS
FormattedDate
FROM mtomactualwithdates
```

Grouping and Re-filtering data

```
df = spark.read.table("mtomactualstruct")
display(df.groupBy("Country").sum("Actual")\
          .withColumnRenamed("sum(Actual)","ActualTotal")\
          .where("ActualTotal>10000")
)
```

```
%%sql
SELECT Country, SUM(Actual) AS ActualTotal
FROM mtomactualstruct
GROUP BY Country
HAVING SUM(Actual) > 10000 -- ActualTotal
```

Sorting the results

```
df = spark.read.table("mtomactualstruct")
# display(df.orderBy("Location"))
# display(df.orderBy(df.Location))
# display(df.orderBy(asc(df.Location)))
# display(df.orderBy(desc(df.Country), df.Location))
# display(df.sort(desc("Country"), "Location"))
# display(df.sort("Country", ascending=False))
display(df.sort(df.Country.desc(), df.Location.asc()))
```

```
%%sql
SELECT *
FROM mtomactualstruct
--ORDER BY Location ASC
ORDER BY Country DESC, Location
```

Using all 6 SQL clauses

```
df = spark.read.table("mtomactualstruct")
display(df.select("Country", "Actual")\
          .where("Actual>4000")\
          .groupBy("Country").sum("Actual")\
          .withColumnRenamed("sum(Actual)","ActualTotal")\
          .where("ActualTotal>10000")
          .orderBy(desc("ActualTotal"),"Country")
          )
```

```
%%sql
SELECT Country, SUM(Actual) AS ActualTotal
FROM mtomactualstruct
WHERE Actual > 4000
GROUP BY Country
HAVING SUM(Actual) > 10000
ORDER BY ActualTotal DESC, Country
```

31a. Merging data

```
df = spark.read.table("mtomactual")
dfdates = spark.read.table("mtomactualwithdates")
dfadditional = spark.read.table("mtomactualadditional")
display(df)
display(dfadditional)
display(dfdates)

dfunion = df.union(dfadditional)
display(dfunion)
dfunion.write.mode("overwrite").format("delta").saveAsTable("MtoMActualCombined")
```

```
%%sql
SELECT *
FROM mtomactual
UNION -- OR UNION ALL
SELECT *
FROM mtomactualadditional
```

```
dfunion = df.unionByName(dfdates, allowMissingColumns=True)
display(dfunion)
```

```
%%sql
SELECT Country, Location, Actual, NULL as ColDate
FROM mtomactual
UNION ALL
SELECT Country, Location, Actual, ColDate
FROM mtomactualwithdates
```

32a. Identifying and resolving duplicate data

```
df = spark.read.table("mtomactualcombined")
display(df.distinct())
```

```
%%sql
SELECT DISTINCT Country, Location, Actual
FROM mtomactualcombined
```

```
display(df.groupBy("Country","Location","Actual").count().where("count>1"))
```

```
%%sql
SELECT Country, Location, Actual
FROM mtomactualcombined
GROUP BY Country, Location, Actual
HAVING COUNT(*)>1
```

```
display(df.dropDuplicates(["Country"]))
```

31b. Joining data

```
dfactual = spark.read.table("mtomactual")
```

```
dftarget = spark.read.format("csv").option("header","true").load("Files/MtoMTarget.csv")
dftarget.write.mode("overwrite").format("delta").saveAsTable("MtoMTarget")
display(dfactual)
display(dftarget)

dfactual = dfactual.select(dfactual.Country, dfactual.Actual.cast("int")) \
                   .groupBy("Country").sum("Actual").withColumnRenamed("sum(Actual)","ActualTotal")
display(dfactual)
dftarget = dftarget.select(dftarget.Country, dftarget.Target.cast("int")) \
                   .groupBy("Country").sum("Target").withColumnRenamed("sum(Target)","TargetTotal")
display(dftarget)

dfactual.write.mode("overwrite").format("delta").saveAsTable("MtoMActualSum")
dftarget.write.mode("overwrite").format("delta").saveAsTable("MtoMTargetSum")

dfjoin = dfactual.join(dftarget, dfactual.Country == dftarget.Country)
display(dfjoin)

display(dfactual.join(dftarget, "Country"))
```

```
%%sql
SELECT MtoMActualSum.Country, ActualTotal,
       MtoMTargetSum.Country, TargetTotal
FROM MtoMActualSum
FULL JOIN MtoMTargetSum
ON MtoMActualSum.Country = MtoMTargetSum.Country
```

```
dfactual = dfactual.withColumnRenamed("Country", "ActualCountry")
dftarget = dftarget.withColumnRenamed("Country", "TargetCountry")
dfjoin = dfactual.join(dftarget, dfactual.ActualCountry == dftarget.TargetCountry, "full")
display(dfjoin)
dfjoin.write.mode("overwrite").format("delta").saveAsTable("MtoMJoin")
```

```
%%sql
SELECT MtoMActualSum.Country AS ActualCountry, ActualTotal,
       MtoMTargetSum.Country AS TargetCountry, TargetTotal
FROM MtoMActualSum
FULL JOIN MtoMTargetSum
ON MtoMActualSum.Country = MtoMTargetSum.Country
```

Practice Activity – Spark

```
dfactual = spark.read.table("mtomactualsum")
dftarget = spark.read.table("mtomtargetsum")
display(dfactual)
display(dftarget)

dfbridge = dfactual.select("Country")
dfbridge = dfbridge.union(dftarget.select("Country")).distinct()
display(dfbridge)

display(dfbridge.join(dfactual, "Country", "left").join(dftarget, "Country", "left"))
```

Creating graphs in Matplotlib

Bar chart

```
import matplotlib.pyplot as plt
dfPanda = dfactual.toPandas()
plt.bar(dfPanda.Country, dfPanda.ActualTotal)
plt.show()
```

```
import matplotlib.pyplot as plt
dfPanda = dfactual.toPandas()
```

```
plt.figure(figsize=(4,4))
plt.bar(dfPanda.Country, dfPanda.ActualTotal, color='yellow')
plt.xlabel("Country")
plt.ylabel("Total")
plt.title("Country and Totals")
plt.show()
```

```


dfactual = spark.read.table("mtomactualsum")
import matplotlib.pyplot as plt

dfPanda = dfactual.toPandas()

plt.figure(figsize=(8,8))
plt.bar(dfPanda.Country, dfPanda.ActualTotal, label="Actual", color='yellow')

plt.xlabel("Country")
plt.ylabel("Total")
plt.title("Country and Totals")

plt.show()
```

Pie chart

```
import matplotlib.pyplot as plt

dfactual = spark.read.table("mtomactualsum")
Pandadf = dfactual.toPandas()

plt.figure(figsize=(8, 8))  # Set the figure size for better visibility
plt.pie(Pandadf.ActualTotal, labels=Pandadf.Country, autopct='%.1f%%', startangle=0, colors=plt.cm.Pastel1.colors)
plt.title("Actual Total by Country")
plt.legend(title="Legend", labelcolor = "b", loc="best", fontsize = "medium")
plt.show()
```

Line chart

```
dfactual = spark.read.table("mtomactualsum")
dftarget = spark.read.table("mtomtargetsum")
dfbridge = dfactual.select("Country")
dfbridge = dfbridge.union(dftarget.select("Country")).distinct()
display(dfbridge.join(dfactual, "Country", "left").join(dftarget, "Country", "left"))

import matplotlib.pyplot as plt

# Convert to Pandas DataFrame
Pdfjoin = dfjoin.toPandas()

# Line Chart
plt.plot(Pdfjoin.Country, Pdfjoin.ActualTotal, marker='o', label="Actual", color='b', linewidth=0)
plt.plot(Pdfjoin.Country, Pdfjoin.TargetTotal, marker='s', label="Target", color='g', linewidth=0)

plt.xlabel("Country")
plt.ylabel("Total")
plt.title("Country and Totals")
plt.legend()
plt.ylim(bottom=0)

plt.show()
```

Creating tables in a Data Warehouse

```
DROP TABLE IF EXISTS tblTarget

CREATE TABLE tblTarget
(
Country VARCHAR(20),
Type VARCHAR(20),
Target INT
)

DROP TABLE IF EXISTS tblActual

CREATE TABLE tblActual
(
Country VARCHAR(20),
Location VARCHAR(20),
Actual INT
)
```

Inserting data into tables and transforming data

```
DROP TABLE IF EXISTS tblTarget

CREATE TABLE tblTarget
(
Country VARCHAR(20),
Type VARCHAR(20),
Target INT
)

DROP TABLE IF EXISTS tblActual

CREATE TABLE tblActual
(
Country VARCHAR(20),
Location VARCHAR(20),
```

```sql
Actual INT
)

INSERT INTO tblActual (Country, Location, Actual) VALUES
('England', 'London', 5000),
('England', 'Birmingham', 7000),
('England', 'Manchester', 11000),
('France', 'Paris', 4000),
('Italy', 'Milan', 3000),
('Italy', 'Rome', 13000);

INSERT INTO tblTarget (Country, Type, Target) VALUES
('England', 'In Store', 10000),
('England', 'Internet/Post', 5000),
('France', 'In Store', 7500),
('France', 'Internet/Post', 3000),
('Germany', 'In Store', 8000),
('Germany', 'Internet/Post', 4000);

SELECT Country, SUM(Actual) AS TotalActual
INTO tblActualSum
FROM tblActual
GROUP BY Country

SELECT Country, SUM(Target) as TotalTarget
INTO tblTargetSum
FROM tblTarget
GROUP BY Country

SELECT * FROM [DemoWarehouse].[dbo].[tblActualSum]
SELECT * FROM [DemoWarehouse].[dbo].[tblTargetSum]
```

Implementing a bridge table for a warehouse

```
SELECT Country
FROM tblActualSum
UNION
SELECT Country
FROM tblTargetSum

SELECT B.Country, A.TotalActual, T.TotalTarget
FROM tblBridge AS B
LEFT JOIN tblActualSum AS A
ON B.Country = A.Country
LEFT JOIN tblTargetSum AS T
ON B.Country = T.Country
ORDER BY B.Country
```

Creating a running total

```
SELECT Country, Location, Actual, SUM(Actual) OVER(PARTITION BY Country ORDER BY Location) as RunningTotal
FROM tblActual
--ROWNUMBER  PARTITIONBY ORDERBY
```

Slowly Changing Dimensions (SCD)

Query 1 – Creating the tables

```
DROP TABLE IF EXISTS FactImport

CREATE TABLE FactImport(
ID INT,
OrderDate DATE,
ProductID INT,
Cost DECIMAL(7,2))

DROP TABLE IF EXISTS DimensionImport

CREATE TABLE DimensionImport(
ProductID INT,
Name VARCHAR(20),
UpdateDate DATE)

DROP TABLE IF EXISTS FactOverall

CREATE TABLE FactOverall(
ID INT,
OrderDate DATE,
ProductID INT,
Cost DECIMAL(7,2))

DROP TABLE IF EXISTS DimensionOverall

CREATE TABLE DimensionOverall(
ProductID INT,
Name VARCHAR(20),
UpdateDate DATE
, StartDate DATE
, EndDate DATE
, IsCurrent CHAR(1)
)
```

Query 2 – Adding data and running the stored procedure

```sql
-- Empty Import tables
DELETE FROM FactImport
DELETE FROM DimensionImport

-- Import new data
INSERT INTO FactImport(ID, OrderDate, ProductID, Cost) VALUES
(1, '2024-01-02', 1, 34),
(2, '2024-01-03', 2, 48),
(3, '2024-02-02', 1, 60),
(4, '2024-02-03', 2, 23),
(5, '2024-03-02', 1, 76),
(6, '2024-03-03', 2, 12),
(7, '2024-04-02', 1, 95),
(8, '2024-04-03', 2, 34)

INSERT INTO DimensionImport(ProductID, Name, UpdateDate) VALUES
(1, 'Product 1', '2023-12-31'),
(2, 'Product 2', '2023-12-31'),
(1, 'Product 1', '2024-01-31'),
(2, 'Product 2', '2024-01-31'),
(1, 'Product 1a', '2024-02-29'),
(2, 'Product 2', '2024-02-29'),
(1, 'Product 1a', '2024-03-31'),
(2, 'Product 2', '2024-03-31'),
(1, 'Product 1b', '2024-04-30'),
(2, 'Product 2', '2024-04-30')

DELETE FROM FactImport      WHERE MONTH(OrderDate)  <> 12 -- Change to 1, 2, 3, and 4
DELETE FROM DimensionImport WHERE MONTH(UpdateDate) <> 12 -- Change to 1, 2, 3, and 4
```

```
EXECUTE SCD0 -- Change to SCD1 and SCD2
```

Query 3 – Slowly Changing Dimensions Type 0

```
DROP PROCEDURE IF EXISTS SCD0
GO

CREATE PROC SCD0 AS
BEGIN
  INSERT INTO DimensionOverall(ProductID, Name, UpdateDate)
  SELECT ProductID, Name, UpdateDate
  FROM DimensionImport
  WHERE ProductID NOT IN (SELECT ProductID FROM DimensionOverall)

  INSERT INTO FactOverall(ID, OrderDate, ProductID, Cost)
  SELECT ID, OrderDate, ProductID, Cost
  FROM FactImport
  WHERE ID NOT IN (SELECT ID FROM FactOverall)

  SELECT * FROM FactOverall ORDER BY ID
  SELECT ProductID, Name FROM DimensionOverall ORDER BY UpdateDate, ProductID
END
```

Query 4 – Slowly Changing Dimensions Type 1

```
DROP PROCEDURE IF EXISTS SCD1
GO

CREATE PROC SCD1 AS
BEGIN
  UPDATE DimensionOverall
  SET DimensionOverall.Name = DimensionImport.Name, DimensionOverall.UpdateDate = DimensionImport.UpdateDate
  FROM DimensionImport
  LEFT JOIN DimensionOverall
  ON DimensionImport.ProductID = DimensionOverall.ProductID
  WHERE DimensionImport.ProductID IN (SELECT ProductID FROM DimensionOverall)
    AND (DimensionImport.Name <> DimensionOverall.Name)

  INSERT INTO DimensionOverall(ProductID, Name, UpdateDate)
  SELECT ProductID, Name, UpdateDate
  FROM DimensionImport
  WHERE ProductID NOT IN (SELECT ProductID FROM DimensionOverall)

  INSERT INTO FactOverall(ID, OrderDate, ProductID, Cost)
  SELECT ID, OrderDate, ProductID, Cost
  FROM FactImport
  WHERE ID NOT IN (SELECT ID FROM FactOverall);

  SELECT * FROM FactOverall ORDER BY ID
  SELECT ProductID, Name, UpdateDate FROM DimensionOverall ORDER BY UpdateDate, ProductID
END
```

Query 5 – Slowly Changing Dimensions Type 2

```
DROP PROCEDURE IF EXISTS SCD2
GO

CREATE PROC SCD2 AS
BEGIN
    UPDATE DimensionOverall
    SET EndDate = (Select Max(UpdateDate) from DimensionOverall), IsCurrent = 'X'
    FROM DimensionImport
    LEFT JOIN DimensionOverall
    ON DimensionImport.ProductID = DimensionOverall.ProductID
    WHERE DimensionImport.ProductID IN (SELECT ProductID FROM DimensionOverall)
```

```
     AND (DimensionImport.Name <> DimensionOverall.Name)
     AND EndDate IS NULL;

    INSERT INTO DimensionOverall(ProductID, Name, UpdateDate, StartDate, IsCurrent)
    SELECT DISTINCT ProductID, Name, UpdateDate, dateadd(day,1,(Select Max(UpdateDate) from DimensionOverall WHERE 
DimensionOverall.ProductID = DimensionImport.ProductID)), 'Y'
    FROM DimensionImport
    WHERE ProductID IN (SELECT ProductID FROM DimensionOverall WHERE DimensionImport.ProductID = 
DimensionOverall.ProductID AND DimensionImport.Name <> DimensionOverall.Name AND DimensionOverall.IsCurrent IN ('Y', 
'X'));

    UPDATE DimensionOverall
    SET IsCurrent = 'N'
    FROM DimensionImport
    WHERE IsCurrent = 'X'

    UPDATE DimensionOverall
    SET UpdateDate = DimensionImport.UpdateDate
    FROM DimensionImport
    LEFT JOIN DimensionOverall
    ON DimensionImport.ProductID = DimensionOverall.ProductID
    WHERE DimensionImport.ProductID IN (SELECT ProductID FROM DimensionOverall)
    AND IsCurrent = 'Y';

    INSERT INTO DimensionOverall(ProductID, Name, UpdateDate, StartDate, IsCurrent)
    SELECT ProductID, Name, UpdateDate, '2010-01-01', 'Y'
    FROM DimensionImport
    WHERE ProductID NOT IN (SELECT ProductID FROM DimensionOverall);

    INSERT INTO FactOverall(ID, OrderDate, ProductID, Cost)
    SELECT ID, OrderDate, ProductID, Cost
    FROM FactImport
    WHERE ID NOT IN (SELECT ID FROM FactOverall);

    SELECT ID, OrderDate, FactOverall.ProductID, Name as ProductName, Cost
    FROM FactOverall
    LEFT JOIN DimensionOverall
    ON FactOverall.ProductID = DimensionOverall.ProductID
    AND OrderDate >= StartDate AND OrderDate <= ISNULL(EndDate, '2099-01-01')
    ORDER BY ID;

    SELECT ProductID, Name, UpdateDate, StartDate, EndDate, IsCurrent FROM DimensionOverall ORDER BY UpdateDate, 
ProductID;
END
```

Implement file partitioning

```
df = spark.read.table("AddressData")
display(df)

df = spark.read.parquet("Tables/AddressData")
display(df)

df = spark.read.parquet("Tables/AddressData/CountryRegion=Canada")
display(df)

df = spark.read.option("recursiveFileLookup","true").parquet("Tables/AddressData/CountryRegion=Canada/*/*.parquet")
display(df)

df = spark.read.parquet("Tables/AddressData/CountryRegion=Canada/StateProvince=Alberta")
display(df)
```

Creating views, stored procedures and functions

```
CREATE VIEW dbo.view_AddressData2 AS
SELECT *
FROM [DemoLakehouse].[dbo].[AddressData]

SELECT *
FROM view_AddressData

CREATE PROC dbo.proc_AddressData @Country varchar(20) AS
BEGIN
SELECT *
FROM view_AddressData

SELECT *
FROM view_AddressData2
WHERE CountryRegion = @Country
END

EXEC dbo.proc_AddressData "Canada"

CREATE FUNCTION dbo.func_AddressData (@Country AS varchar(20))
RETURNS TABLE
AS
RETURN
SELECT *
FROM AddressData
WHERE CountryRegion = @Country

SELECT AddressID, City
FROM func_AddressData('Canada')
```

Improvement performance in notebooks

```
spark.conf.get('spark.sql.parquet.vorder.enabled')
spark.conf.set('spark.sql.parquet.vorder.enabled', 'true')
```

```
%%sql
SET spark.sql.parquet.vorder.enabled=TRUE
```

```
spark.conf.set('spark.microsoft.delta.optimizeWrite.enabled', 'true')
```

Data Loading bottlenecks in SQL queries

```
SELECT *
FROM [DemoLakehouse].[dbo].[FactInternetSales]
```

Performance improvements in SQL queries

```
SELECT mtomactualsum.Country AS ActualCountry, ActualTotal,
       mtomtargetsum.Country AS TargetCountry, TargetTotal
FROM mtomactualsum
FULL JOIN mtomtargetsum
ON mtomactualsum.Country = mtomtargetsum.Country

SELECT Country, Location, Actual
FROM mtomactual
WHERE Country = 'England'

SELECT Country, SUM(Actual) AS TotalActual
FROM mtomactualstruct
```

```sql
GROUP BY Country

SELECT Country, Location, Actual
FROM mtomactual
ORDER BY Country, Location

SELECT *
FROM [DemoLakehouse].[dbo].[FactInternetSales]
WHERE Year(OrderDate) = 2007

SELECT *
FROM [DemoLakehouse].[dbo].[FactInternetSales]
WHERE SUBSTRING(SalesOrderNumber, 1, 3) = 'SO5'
```