AZURE-DE (Projects)

Project-1 (4:17:58): NYC data Project [ADLS + databricks + Delta Tables + PowerBI]

```
# Project-1 NYC Data [API + ADLS + ADF + databricks(Delta Lake) + KeyVault + AAD + PowerBI]
nyc.gov/site/tlc/about/tlc-trip-record-data.page --> rightclick copy url path Green Taxi Trip Records(Parquet) 2023
Taxi Zone Lookup Table(csv) files take from same website or github repo     # taxi_zone_lookup.csv, trip_type.csv
Phase-1: Ingestion (From API --> ADLS(bronze,parquet))
Phase-2: Transformation(databricks/pyspark(ADLS(bronze) --> ADLS(silver) --> ADLS(Gold)))
Phase-3: databricks --. partnerConnect --> PowerBi --> download connectionFile,sign-in-->select deltatables-->load-->design dashboards
    # Phase-1
Create RG, ADLS[bronze(trip-type,trip_zone folders with above 2 csv files),silver,gold]
Create ADF-->[ http_ls(http,<baseURL>), adls_ls(nnadls)] --> pipeline [ForEach(if(copy1 & copy2))]
# baseURL: https://d37xxxx.cloudfront.net/
ForEach --> Settings[Items(@range(1,12))],Activities[If(Expression(@greater(item(),9)),True(Copy1),False(Copy2))]
Copy2[Source(dataset: http,parquet,<relURL1>,parameter(p-month,string),p_month(@item())),
Sink(dataset:ADLS,parquet,bronze/trips2023data/)]            #relURL1: trip-data/green_tripdata_2023-0@{dataset().p_month}.parquet
Copy1[Source(dataset: http,parquet,<relURL2>,parameter(p-month10to12,string),p_month10to12(@item())),
Sink(dataset:ADLS,parquet,bronze/trips2023data/)]      #relURL2: trip-data/green_tripdata_2023-@{dataset().p_month10to12}.parquet
Note: for both copy activities datasets(http) are seperate
    # Phase-2
AAD --> App-registrations(+) --> nn_serviceprincipal1 --> Create Client Secret and copy secret value
ADLS(IAM) --> blob data contributor role --> nn_serviceprincipal1
Create Azure databricks workspace, compute cluster & access connector and add above role to these also in ADLS(IAM)
Create Notebooks(silver_book,gold_book) and write spark code along with ADLS-->databricks access code config (Mount/ADLS-accesskey/
UnityCatalog)      # Bronze(raw csv data) --> silver(transformed & write as parquet) --> gold(silver parquet write as delta Tables)
    # Phase-3
Connect to PowerBi and import gold layer tables and prepare dashboards
```

Notebook-1(silver):

```
spark.conf.set("fs.azure.account.auth.type.nnnyctaxisa2025.dfs.core.windows.net", "OAuth")
spark.conf.set("fs.azure.account.oauth.provider.type.nnnyctaxisa2025.dfs.core.windows.net",
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider")
spark.conf.set("fs.azure.account.oauth2.client.id.nnnyctaxisa2025.dfs.core.windows.net", "< App Id  >")
spark.conf.set("fs.azure.account.oauth2.client.secret.nnnyctaxisa2025.dfs.core.windows.net","< EntraID App Client Secret Value>")
spark.conf.set("fs.azure.account.oauth2.client.endpoint.nnnyctaxisa2025.dfs.core.windows.net",
"https://login.microsoftonline.com/< Tenant Id >/oauth2/token")

dbutils.fs.ls('abfss://bronze@nnnyctaxisa2025.dfs.core.windows.net/')
from pyspark.sql.functions import *
from pyspark.sql.types import *
df_trip_type = spark.read.format("csv").option("header",True).option("inferSchema",True)
                        .load("abfss://bronze@nnnyctaxisa2025.dfs.core.windows.net/trip_type") --> display(df_trip_type)
df_trip_zone = spark.read.format("csv").option("header",True).option("inferSchema",True)
                .load("abfss://bronze@nnnyctaxisa2025.dfs.core.windows.net/trip_zone") --> display(df_trip_zone)
my_schema = '''
                VendorID bigint,
                lpep_pickup_datetime timestamp,
                lpep_dropoff_datetime timestamp,
                store_and_fwd_flag string,
                RatecodeID bigint,
                PULocationID bigint,
                DOLocationID bigint,
                passenger_count bigint,
                trip_distance double,
                fare_amount double,
                extra double,
                mta_tax double,
                tip_amount double,
                tolls_amount double,
                ehail_fee double,
                improvement_surcharge double,
                total_amount double,
                payment_type bigint,
                trip_type bigint,
                congestion_surcharge double
'''
df_trip = spark.read.format("parquet").option("header",True).schema(my_schema).option("recursiveFileLookup",True)
        .load("abfss://bronze@nnnyctaxisa2025.dfs.core.windows.net/trips2023data") --> display(df_trip)
df_trip_type = df_trip_type.withColumnRenamed("trip_type", "trip_type_id").withColumnRenamed("description", "trip_type_description")
        display(df_trip_type)
df_trip_type.write.format('parquet').mode("append").option("path","abfss://silver@nnnyctaxisa2025.dfs.core.windows.net/trip_type")
        .save()
df_trip_zone = df_trip_zone.withColumn('zone1', split(col('Zone'), '/')[0]).withColumn('zone2', split(col('Zone'), '/')[1])
        display(df_trip_zone)
df_trip_zone.write.format('parquet').mode("append").option("path","abfss://silver@nnnyctaxisa2025.dfs.core.windows.net/trip_zone")
        .save()
df_trip1 = df_trip.withColumn('trip_date',to_date('lpep_pickup_datetime'))\
                  .withColumn('trip_year',year('lpep_pickup_datetime'))\
                  .withColumn('trip_month',month('lpep_pickup_datetime'))
df_trip2 = df_trip1.select('VendorID','trip_date', 'PULocationID', 'DOLocationID', 'fare_amount', 'total_amount') --> display(df_trip2)
df_trip2.write.format('parquet').mode("append").option("path","abfss://silver@nnnyctaxisa2025.dfs.core.windows.net/trips2025").save()
```

Notebook-2(gold):

```
spark.conf.set("fs.azure.account.auth.type.nnnyctaxisa2025.dfs.core.windows.net", "OAuth")
spark.conf.set("fs.azure.account.oauth.provider.type.nnnyctaxisa2025.dfs.core.windows.net",
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider")
spark.conf.set("fs.azure.account.oauth2.client.id.nnnyctaxisa2025.dfs.core.windows.net", "< App Id  >")
spark.conf.set("fs.azure.account.oauth2.client.secret.nnnyctaxisa2025.dfs.core.windows.net","< EntraID App Client Secret Value>")
spark.conf.set("fs.azure.account.oauth2.client.endpoint.nnnyctaxisa2025.dfs.core.windows.net",
"https://login.microsoftonline.com/< Tenant Id >/oauth2/token")

%sql create database nngold
from pyspark.sql.functions import *
from pyspark.sql.types import *
silver = 'abfss://silver@nnnyctaxisa2025.dfs.core.windows.net'
gold = 'abfss://gold@nnnyctaxisa2025.dfs.core.windows.net'
df_zone = spark.read.format("parquet").option("inferSchema",True).option("header",True).load(f'{silver}/trip_zone')--> display(df_zone)
df_zone.write.format("delta").mode("append").option('path', f'{gold}/trip_zone').saveAsTable('nngold.trip_zone')
df_type = spark.read.format("parquet").option("inferSchema",True).option("header",True).load(f'{silver}/trip_type')--> display(df_type)
df_type.write.format("delta").mode("append").option('path', f'{gold}/trip_type').saveAsTable("nngold.triptype")
df_trip = spark.read.format("parquet").option("inferSchema",True).option("header",True).load(f'{silver}/trips2023')--> display(df_trip)
df_trip.write.format("delta").mode("append").option('path', f'{gold}/trip_trip').saveAsTable("nngold.trips2023")

Queries in gold layer:
%sql select * from nngold.trip_zone where LocationID = 1
%sql update nngold.trip_zone set zone2 = 'Unknown' where zone2 is null
%sql update nngold.trip_zone set Borough = 'Narendra' where LocationID = 1;
%sql delete from nngold.trip_zone where LocationID = 1;
%sql describe history nngold.trip_zone;
%sql restore nngold.trip_zone to version as of 1
```

Project-2 (4:39:17): Adventure Salesdata Project [ADLS + databricks + Synapse + PowerBI]

```
# Project-2 Sales Data [ADLS + databricks + Synapse + PowerBi]
    # Project highlights:
keep ready github repo with needed csv files
Phase-1: Ingestion ( From github --> ADLS(bronze,parquet) )
Phase-2: Transformation(databricks/pyspark(ADLS(bronze) --> ADLS(silver) --> ADLS(Gold)))
Phase-3: Visualization(PowerBI), copy synapse serverlessSQL endpoint url, paste in powerbi(Getdata-->AzureSynapse-->Server(OK)
    # Phase-1
Create RG, ADLS[bronze,silver,gold,nn-param and upload git.json file inside nn-param container]
Create ADF-->[ http_ls(http,<baseURL>), adls_ls(nnadls)] --> pipeline [ Lookup -->ForEach(copy)]
        # baseURL: https://raw.githubusercontent.com/
Lookup --> Settings[dataset:ADLS,json,nn-param/git.json, uncheck First Row only]
ForEach --> Settings[Items(@activity('LookupGit').output.value)], Activities[Copy]
Copy --> [Source(dataset: http,csv,relURL(@dataset().p_rel_url),parameter(p_rel_url,string),p_rel_url(@item().p_rel_url)),
   Sink(dataset:ADLS,csv, bronze/@dataset().p_sink_folder/@dataset().p_sink_file,
parameters(p_sink_folder,string),p_sink_file,string)),p_sink_folder(@item().p_sink_folder),p_sink_file(@item().p_sink_file))]
    # Phase-2
AAD --> App-registrations(+) --> nn_serviceprincipal1 --> Create Client Secret and copy secret value
ADLS(IAM) --> blob data contributor role --> nn_serviceprincipal1
Create Azure databricks workspace, compute cluster & access connector and add above role to these also in ADLS(IAM)
Create Synapse and add same above role to Synapse & our user email also in ADLS(IAM)
Create Notebooks(silver_book,gold_book/synapse) and write spark code along with ADLS --> databricks access code config
(Mount/ADLS-accesskey/UnityCatalog)
    # Phase-3
Connect to PowerBi and import gold layer tables and prepare dashboards
```

Notebook-1(silver):

```
from pyspark.sql.functions import *
from pyspark.sql.types import *
df_cal = spark.read.format('csv').option("header",True).option("inferSchema",True)\
            .load('abfss://bronze@nnadventureadls2025.dfs.core.windows.net/AdventureWorks_Calendar')
df_cus = spark.read.format('csv').option("header",True).option("inferSchema",True)\
            .load('abfss://bronze@nnadventureadls2025.dfs.core.windows.net/AdventureWorks_Customers')
df_procat = spark.read.format('csv').option("header",True).option("inferSchema",True)\
            .load('abfss://bronze@nnadventureadls2025.dfs.core.windows.net/AdventureWorks_Product_Categories')
df_pro = spark.read.format('csv').option("header",True).option("inferSchema",True)\
            .load('abfss://bronze@nnadventureadls2025.dfs.core.windows.net/AdventureWorks_Products')
df_ret = spark.read.format('csv').option("header",True).option("inferSchema",True)\
            .load('abfss://bronze@nnadventureadls2025.dfs.core.windows.net/AdventureWorks_Returns')
df_sales = spark.read.format('csv').option("header",True).option("inferSchema",True)\
            .load('abfss://bronze@nnadventureadls2025.dfs.core.windows.net/AdventureWorks_Sales*')
df_ter = spark.read.format('csv').option("header",True).option("inferSchema",True)\
            .load('abfss://bronze@nnadventureadls2025.dfs.core.windows.net/AdventureWorks_Territories')
df_subcat = spark.read.format('csv').option("header",True).option("inferSchema",True)\
            .load('abfss://bronze@nnadventureadls2025.dfs.core.windows.net/Product_Subcategories')

df_cal = df.withColumn('Month',month(col('Date'))).withColumn('Year',year(col('Date')))
df_cal.write.format('parquet').mode('append')
        .option("path","abfss://silver@nnadventureadls2025.dfs.core.windows.net/AdventureWorks_Calendar").save()
df_cus.withColumn("fullName",concat(col('Prefix'),lit(' '),col('FirstName'),lit(' '),col('LastName'))).display()
df_cus = df_cus.withColumn('fullName',concat_ws(' ',col('Prefix'),col('FirstName'),col('lastName'))) --> display(df_cus)
```

```
df_cus.write.format('parquet').mode('append')
         .option("path","abfss://silver@nnadventureadls2025.dfs.core.windows.net/AdventureWorks_Customers").save()
df_subcat.write.format('parquet').mode('append')
            .option("path","abfss://silver@nnadventureadls2025.dfs.core.windows.net/AdventureWorks_SUbCategories").save()
df_pro = df_pro.withColumn('ProductSKU',split(col('ProductSKU'),'-')[0]).withColumn('ProductName',split(col('ProductName'),' ')[0])
df_pro.display()
df_pro.write.format('parquet').mode('append')
            .option("path","abfss://silver@nnadventureadls2025.dfs.core.windows.net/AdventureWorks_Products").save()
df_ret.write.format('parquet').mode('append')
            .option("path","abfss://silver@nnadventureadls2025.dfs.core.windows.net/AdventureWorks_Returns").save()
df_ter.write.format('parquet').mode('append')
            .option("path","abfss://silver@nnadventureadls2025.dfs.core.windows.net/AdventureWorks_Territories").save()
df_sales = df_sales.withColumn('StockDate',to_timestamp('StockDate'))
df_sales = df_sales.withColumn('OrderNumber',regexp_replace(col('OrderNumber'),'S','T'))
df_sales = df_sales.withColumn('multiply', col('OrderLineItem') * col('OrderQuantity'))
df_sales.display()
df_sales.write.format('parquet').mode('append')
            .option("path","abfss://silver@nnadventureadls2025.dfs.core.windows.net/AdventureWorks_Sales").save()
df_sales.groupBy('OrderDate').agg(count('OrderNumber').alias('total_order')).display()
df_procat.display() --> create some visual charts in Notebook itself
df_ter.display() --> create some visual charts in Notebook itself
```

Notebook-2 (gold):

```
Lakehouse: by using openrowset(), form an abstract layer on ADLS to extract data & creating Tableviews for querying without
loaing data
Create database(nn-db) from console manually and attach it to built-in serverless sql pool and wrtie below code in new SQL
script file
create schema gold;
create view gold.calendar as select * from OPENROWSET(
    BULK 'https://nnadventureadls2025.blob.core.windows.net/silver/AdventureWorks_Calendar/',
    FORMAT = 'parquet'
) as nnquery1
create view gold.customers as select * from OPENROWSET(
    BULK 'https://nnadventureadls2025.blob.core.windows.net/silver/AdventureWorks_Customers/',
    FORMAT = 'parquet'
) as nnquery1
create view gold.products as select * from OPENROWSET(
    BULK 'https://nnadventureadls2025.blob.core.windows.net/silver/AdventureWorks_Products/',
    FORMAT = 'parquet'
) as nnquery1
create view gold.returns as select * from OPENROWSET(
    BULK 'https://nnadventureadls2025.blob.core.windows.net/silver/AdventureWorks_Returns/',
    FORMAT = 'parquet'
) as nnquery1
create view gold.sales as select * from OPENROWSET(
    BULK 'https://nnadventureadls2025.blob.core.windows.net/silver/AdventureWorks_Sales/',
    FORMAT = 'parquet'
) as nnquery1
create view gold.subcat as select * from OPENROWSET(
    BULK 'https://nnadventureadls2025.blob.core.windows.net/silver/AdventureWorks_SUbCategories/',
    FORMAT = 'parquet'
) as nnquery1
create view gold.ter as select * from OPENROWSET(
    BULK 'https://nnadventureadls2025.blob.core.windows.net/silver/AdventureWorks_Territories/',
    FORMAT = 'parquet'
) as nnquery1
select * from gold.customers;
create master key encryption by password = 'Naren@123"
create database scoped credential cred_nn with IDENTITY = 'Managed Identity'
create external data source source_silver with
(
    location = 'https://nnadventureadls2025.blob.core.windows.net/silver',
    credential = cred_nn
)
create external data source source_gold with
(
    location = 'https://nnadventureadls2025.blob.core.windows.net/gold',
    credential = cred_nn
)
create external file format format_parquet with
(
    FORMAT_TYPE = parquet,
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
)
create external table gold.extsales with
(
    LOCATION = 'extsales',
    DATA_SOURCE = source_gold,
    FILE_FORMAT = format_parquet
) as select * from gold.sales
select * from gold.extsales        # this is query from above created External Table and its data stored in ADLS(gold)
```

Project-3 (6:42:14): Incremental Load data & SCD Project [API/Github/SQL DB + ADLS + ADF + databricks + Delta Table]

```
Github --> ADF(http dataset, Ingestion into SQL DB) --> ADLS(Bronze, Raw, Incremental Load) --> Databricks  [ ADLS(Silver, Parquet), ADLS(Gold, Delta Lake)] --> PowerBI
    # Project highlights:
keep ready github repo with needed csv files(SalesData.csv, IncrementalSales.csv)
Phase-1: Ingestion[Pipeline1(github --> SQL-DB), Pipeline2( 2 Lookups + Copy(SQL-DB-->ADLS(bronze)) + StoredProcedure)]
Phase-2: Transformation(databricks/pyspark(ADLS(bronze) --> ADLS(silver) --> ADLS(Gold)))
Phase-3: Visualization(PowerBI)
    # Phase-1  [ 2 Lookups + Copy + Stored Procedure ]
Create RG, ADLS Gen2(nncarsadls,bronze,silver,gold,unitymetastore containers)
Create Azure SQL DB [ nncarserver, nncarsdb, username(naren), Password(xxxxxx)], allow Firewall rules
Create nn-table(Empty) with Schema same as of Github csv files columns      # In SQL DB Initial Load, Incremental Load
Create ADF--> [ http_ls(http,<baseURL>), sql_ls(nncarserver, nncarsdb, username(naren), Password(xxxxxx))]  # baseURL: https://raw.githubusercontent.com/
    ADF pipeline1(initial_load) --> copy[Source(dataset: http,csv,<relURL1>,parameter(load_flag,string),load_flag(@item())), Sink(dataset:AzureSQL-DB,Table(dbo.nn-table),Import Schema Mapping)
            #relURL1: narenmrkp/Azure-DE-Project/refs/heads/main/Raw%20Data/@{dataset().load_flag}
        Create 2nd pipeline(incremental_load) drag 2 Lookups & 1 copy [SQLDB --> ADLS Bronze]
In SQL DB, create water_table by query: create table water_table (last_load varchar(2000)); --> Select * from water_table;
Insert into water_table values('DT00000'); --> select count(*) from dbo.nn_table where Date_ID > 'DT00000'
In SQL DB, create Stored Procedure as below:
create procedure nn_proc @lastload varchar(200)
as begin
begin transaction;
update water_table set last_load = @lastload
commit transaction;
end;
    ADF pipeline2(incremental_load) -->
    Lookup1[dataset:SQLDB,parameter(table_name,string),Table(dbo.@dataset().table_name),table_name(water_table),tick on Query (select * from water_table;)]
    Lookup2[dataset:SQLDB,parameter(table_name,string),Table(dbo.@dataset().table_name),table_name(nn_table),tick on Query (select max(Date_ID) as max_date from nn_table;)]
          # Connect both Lookups with Copy activity by parallel way
    Copy[Source(dataset:SQL-DB,table_name(nn_table),tick on Query(<query1>))]
        # Query1: Select * from nn_table where Date_ID > '@{activity('last_load').output.value[0].last_load}' and Date_ID <= '@{activity('current_load').output.value[0].Max_Date}'
        Sink(dataset:ADLS,parquet,bronze/raw) --> check by debug pipeline
    StoredProcedure(dataset:SQL-DB, select nn_proc), Import[Name(last_load), value(dynamic, @activity('current_load').output.value[0].Max_Date)]

     # Phase-2 [Databricks]
AAD --> App-registrations(+) --> nn_serviceprincipal1 --> Create Client Secret and copy secret value
ADLS(IAM) --> blob data contributor role --> nn_serviceprincipal1
Create Azure Databricks[RG, Workspace, Region, Premium Trial --> Review+Create
Create Access Connector for Azure Databricks(RG, Name, Region) --> Review+Create
Inside ADLS --> Access Control(IAM) --> Add(Add Role Assignment), storage blob Data contributor, Managed Identity, select our databricks access connector --> review+Assign
With databricks admin link(https://accounts.azuredatabricks.net/login) --> login with EntraID admin email id --> Catalog --> use default unity metastore or create our own Metastore(use ADLS(unitymetastore) container path) by deleting default
Inside metastore --> Catalog --> nncarscatalog --> Assign to workspace, select our workspace --> Assign, Enable
Next create compute cluster in Azure Databricks, Catalog --> External Data --> Create Storage Credential(Name, above access connector id) --> create
Catalog --> External Data --> Create External Location --> Name(bronzeExt), <bronze container abfss path>, select above Storage credential --> Create
Catalog --> External Data --> Create External Location --> Name(silverExt), <silver container abfss path>, select above Storage credential --> Create
Catalog --> External Data --> Create External Location --> Name(goldExt), <gold container abfss path>, select above Storage credential --> Create
Create Folder(nn-cars) in Workspace and Notebook(nn_db_NB) inside this nn-cars folder and connect above cluster
Write code inside this Notebook
Create silver, gold Notebooks along with code with all needed Transformations as below:
Azure databricks --> Workflows --> Create Job(Data-model) -->
    TaskName(Silver_data), Source(browse our silver notebook),select cluster --> create
 AddTask(+)-->Notebook --> TaskName(Dim_Model), Source(browse our gold-dim-model notebook),select cluster, parameters(key(incremental_flag),value(1)) --> create
 AddTask(+)-->Notebook --> TaskName(Dim_Dealer), Source(browse our gold-dim-dealer notebook),select cluster, parameters(key(incremental_flag),value(1)) --> create
 AddTask(+)-->Notebook --> TaskName(Dim_Branch), Source(browse our gold-dim-branch notebook),select cluster, parameters(key(incremental_flag),value(1)) --> create
 AddTask(+)-->Notebook --> TaskName(Dim_date), Source(browse our gold-dim-date notebook),select cluster, parameters(key(incremental_flag),value(1)) --> create
 AddTask(+)-->Notebook --> TaskName(Fact_Sales), Source(browse our gold-factsales notebook),select cluster --> create
 Note: For gold-dim-dealer, gold-dim-branch, gold-dim-date NBs depends on(Silver_data)
       For gold-factsales NB depends on(Dim_Model,Dim_Dealer,Dim_Branch,Dim_date) --> create, save & continue --> Run Now
Azure databricks --> SQL Editor --> attach cluster --> run queries as per need --> Select * from cars_catalog.gold.factsales
```

Notebook-1 (schema):

```
%sql create catalog cars_catalog;
%sql create schema cars_catalog.silver;
%sql create schema cars_catalog.gold;
```

Notebook-2 (silver):

```
df = spark.read.format("parquet").option("inferSchema",True).option("path","abfss://bronze@nnadls2025.dfs.core.windows.net/rawdata/")
        .load() --> display(df)
from pyspark.sql.functions import *
from pyspark.sql.types import *
df = df.withColumn("model_category",split(col("model_ID"),"-")[0]) --> display(df)
df = df.withColumn("Units_Sold",col("Units_Sold").cast(StringType())) --> display(df)
df = df.withColumn("RevPerUnit",col("Revenue")/col("Units_Sold")) --> display(df)
df = df.groupBy("Year","BranchName").agg(sum("Units_Sold").alias("Total_Units")).sort("Year","Total_Units",ascending=[1,0])
        display(df) --> create visualization chart
df.write.format("parquet").mode("overwrite").option("path","abfss://silver@nnadls2025.dfs.core.windows.net/carsales/").save()
%sql Select * from parquet.`abfss://silver@nnadls2025.dfs.core.windows.net/carsales/`
```

Notebook-3.1 (gold-dim-model):

```
from pyspark.sql.functions import *
from pyspark.sql.types import *
dbutils.widgets.text('incremental_flag','0') --> iflag = dbutils.widgets.get('incremental_flag') --> print(iflag)
df_src = spark.sql(“Select distinct(Model_ID) as Model_ID, model_category from
                                parquet.`abfss://silver@nnadls2025.dfs.core.windows.net/carsales`”)
if spark.catalog.tableExists('cars_catalog.gold.dim_model') :
    df_sink = spark.sql(“Select dim_model_key, Model_ID, model_category from cars_catalog.gold.dim_model”)
else :
    df_sink = spark.sql(“Select 1 as dim_model_key, Model_ID, model_category from
                                parquet.`abfss://silver@nnadls2025.dfs.core.windows.net/carsales` where 1=0”)

df_filter = df_src.join(df_sink,df_src.Model_ID == df_sink.Model_ID,'left') \
                        .select(df_src.Model_ID, df_src.model_category, df_sink.dim_model_key)
df_filter_old = df_filter.filter(col('dim_model_key').isNotNull())
df_filter_new = df_filter.filter(col('dim_model_key').isNull()).select(df_src.Model_ID, df_src.model_category) -->
display(df_filter_new)

if (incremental_flag == '0'):
    max_value = 1
else :
    max_value_df = spark.sql("Select max(dim_model_key) from cars_catalog.gold.dim_model")
    max_value = max_value_df.collect()[0][0]
df_filter_new = df_filter_new.withColumn('dim_model_key', max_value + monotonically_increasing_id())
display(df_filter_new)
df_final = df_filter_new.union(df_filter_old)
# SCD-1 (merge,upsert)
from delta.tables import DeltaTable
if spark.catalog.tableExists('cars_catalog.gold.dim_model'):
    delta_tbl = DeltaTable.forPath(spark,"abfss://gold@nnadls2025.dfs.core.windows.net/dim_model")
    delta_tbl.alias("T").merge(df_final.alias("S"), "T.dim_model_key = S.dim_model_key") \
                        .whenMatchedUpdateAll().whenNotMatchedInsertAll().execute()
else :
    df_final.write.format("delta").mode("overwrite").option("path","gold@nnadls2025.dfs.core.windows.net/dim_model/") \
                        .saveAsTable("cars_catalog.gold.dim_model")
%sql Select * from cars_catalog.gold.dim_model
```

Notebook-3.2 (gold-dim-branch):

```
from pyspark.sql.functions import *
from pyspark.sql.types import *
dbutils.widgets.text('incremental_flag','0') --> iflag = dbutils.widgets.get('incremental_flag') --> print(iflag)
df_src = spark.sql(“Select distinct(Branch_ID) as Branch_ID, BranchName from
                            parquet.`abfss://silver@nnadls2025.dfs.core.windows.net/carsales`”)
if spark.catalog.tableExists('cars_catalog.gold.dim_branch') :
    df_sink = spark.sql(“Select dim_branch_key, Branch_ID, BranchName from cars_catalog.gold.dim_branch”)
else :
    df_sink = spark.sql(“Select 1 as dim_branch_key, Branch_ID, BranchName from
                                parquet.`abfss://silver@nnadls2025.dfs.core.windows.net/carsales` where 1=0”)

df_filter = df_src.join(df_sink,df_src.Branch_ID == df_sink.Branch_ID,'left')
                    .select(df_src.Branch_ID, df_src.BranchName, df_sink.dim_branch_key)
df_filter_old = df_filter.filter(col('dim_branch_key').isNotNull())
df_filter_new = df_filter.filter(col('dim_branch_key').isNull()).select(df_src.Branch_ID, df_src.BranchName)
        --> display(df_filter_new)

if (incremental_flag == '0'):
    max_value = 1
else :
    max_value_df = spark.sql("Select max(dim_branch_key) from cars_catalog.gold.dim_branch")
    max_value = max_value_df.collect()[0][0]

df_filter_new = df_filter_new.withColumn('dim_branch_key', max_value + monotonically_increasing_id())
        display(df_filter_new)
df_final = df_filter_new.union(df_filter_old)
# SCD-1 (merge,upsert)
from delta.tables import DeltaTable
if spark.catalog.tableExists('cars_catalog.gold.dim_branch'):
    delta_tbl = DeltaTable.forPath(spark,"abfss://gold@nnadls2025.dfs.core.windows.net/dim_branch")
    delta_tbl.alias("T").merge(df_final.alias("S"), "T.dim_branch_key = S.dim_branch_key") \
```

```
                        .whenMatchedUpdateAll().whenNotMatchedInsertAll().execute()
else :
    df_final.write.format("delta").mode("overwrite").option("path","gold@nnadls2025.dfs.core.windows.net/dim_branch/") \
                    .saveAsTable("cars_catalog.gold.dim_branch")
%sql Select * from cars_catalog.gold.dim_branch
```

Notebook-3.3 (gold-dim-dealer):

```
from pyspark.sql.functions import *
from pyspark.sql.types import *
dbutils.widgets.text('incremental_flag','0') --> iflag = dbutils.widgets.get('incremental_flag') --> print(iflag)
df_src = spark.sql(“Select distinct(Dealer_ID) as Dealer_ID, DealerName from
               parquet.`abfss://silver@nnadls2025.dfs.core.windows.net/carsales`”)
if spark.catalog.tableExists('cars_catalog.gold.dim_dealer') :
    df_sink = spark.sql(“Select dim_dealer_key, Dealer_ID, DealerName from cars_catalog.gold.dim_dealer”)
else :
    df_sink = spark.sql(“Select 1 as dim_dealer_key, Dealer_ID, DealerName from
               parquet.`abfss://silver@nnadls2025.dfs.core.windows.net/carsales` where 1=0”)

df_filter = df_src.join(df_sink,df_src.Dealer_ID == df_sink.Dealer_ID,'left').select(df_src.Dealer_ID, df_src.DealerName,
df_sink.dim_dealer_key)
df_filter_old = df_filter.filter(col('dim_dealer_key').isNotNull())
df_filter_new = df_filter.filter(col('dim_dealer_key').isNull()).select(df_src.Dealer_ID, df_src.DealerName) --> display(df_filter_new)

if (incremental_flag == '0'):
    max_value = 1
else :
    max_value_df = spark.sql("Select max(dim_dealer_key) from cars_catalog.gold.dim_dealer")
    max_value = max_value_df.collect()[0][0]
df_filter_new = df_filter_new.withColumn('dim_dealer_key', max_value + monotonically_increasing_id())
display(df_filter_new)
df_final = df_filter_new.union(df_filter_old)
# SCD-1 (merge,upsert)
from delta.tables import DeltaTable
if spark.catalog.tableExists('cars_catalog.gold.dim_dealer'):
    delta_tbl = DeltaTable.forPath(spark,"abfss://gold@nnadls2025.dfs.core.windows.net/dim_dealer")
    delta_tbl.alias("T").merge(df_final.alias("S"), "T.dim_dealer_key = S.dim_dealer_key") \
                        .whenMatchedUpdateAll().whenNotMatchedInsertAll().execute()
else :
    df_final.write.format("delta").mode("overwrite").option("path","gold@nnadls2025.dfs.core.windows.net/dim_dealer/")
            .saveAsTable("cars_catalog.gold.dim_dealer")
%sql Select * from cars_catalog.gold.dim_dealer
```

Notebook-3.4 (gold-dim-date):

```
from pyspark.sql.functions import *
from pyspark.sql.types import *
dbutils.widgets.text('incremental_flag','0') --> iflag = dbutils.widgets.get('incremental_flag') --> print(iflag)
df_src = spark.sql(“Select distinct(Date_ID) as Date_ID from parquet.`abfss://silver@nnadls2025.dfs.core.windows.net/carsales`”)
if spark.catalog.tableExists('cars_catalog.gold.dim_date') :
    df_sink = spark.sql(“Select dim_date_key, Date_ID from cars_catalog.gold.dim_date”)
else :
    df_sink = spark.sql(“Select 1 as dim_date_key, Date_ID from parquet.`abfss://silver@nnadls2025.dfs.core.windows.net/carsales` where
1=0”)

df_filter = df_src.join(df_sink,df_src.Date_ID == df_sink.Date_ID,'left').select(df_src.Date_ID, df_sink.dim_date_key)
df_filter_old = df_filter.filter(col('dim_date_key').isNotNull())
df_filter_new = df_filter.filter(col('dim_date_key').isNull()).select(df_src.Date_ID) --> display(df_filter_new)
if (incremental_flag == '0'):
    max_value = 1
else :
    max_value_df = spark.sql("Select max(dim_date_key) from cars_catalog.gold.dim_date")
    max_value = max_value_df.collect()[0][0]
df_filter_new = df_filter_new.withColumn('dim_date_key', max_value + monotonically_increasing_id())
display(df_filter_new)
df_final = df_filter_new.union(df_filter_old)
# SCD-1 (merge,upsert)
from delta.tables import DeltaTable
if spark.catalog.tableExists('cars_catalog.gold.dim_date'):
    delta_tbl = DeltaTable.forPath(spark,"abfss://gold@nnadls2025.dfs.core.windows.net/dim_date")
    delta_tbl.alias("T").merge(df_final.alias("S"), "T.dim_date_key = S.dim_date_key") \
                        .whenMatchedUpdateAll().whenNotMatchedInsertAll().execute()
else :
    df_final.write.format("delta").mode("overwrite").option("path","gold@nnadls2025.dfs.core.windows.net/dim_date/")
            .saveAsTable("cars_catalog.gold.dim_date")
%sql Select * from cars_catalog.gold.dim_date
```

Notebook-3.5 (gold-factsales):

```
# %md ## Create Fact Table
df_silver = spark.sql("Select * from parquet.`abfss://silver@nnadls2025.dfs.core.windows.net/carsales`")
df_dealer = spark.sql("Select * from cars_catalog.gold.dim_dealer")
df_branch = spark.sql("Select * from cars_catalog.gold.dim_branch")
df_model = spark.sql("Select * from cars_catalog.gold.dim_model")
df_date = spark.sql("Select * from cars_catalog.gold.dim_date")
```

```
df_fact = df_silver.join(df_branch, df_silver.Branch_ID == df_branch.Branch_ID,"left")
                   .join(df_dealer, df_silver.Dealer_ID == df_dealer.Dealer_ID,"left")
                   .join(df_model, df_silver.Model_ID == df_model.Model_ID,"left")
                   .join(df_date, df_silver.Date_ID == df_date.Date_ID,"left")
                   .select(df_silver.Revenue,df_silver.Units_Sold,df_silver.RevPerUnit,df_branch.dim_branch_key,
                    df_dealer.dim_dealer.key, df_model.dim_model_key,df_date.dim_date_key)
from delta.tables import DeltaTable

if spark.catalog.tableExists('factsales'):
    deltatabl = DeltaTable.forName(spark,'cars_catalog.gold.factsales')
    deltatbl.alias('T').merge(df_fact.alias('S'),'T.dim_branch_key = S.dim_branch_key and T.dim_dealer_key = S.dim_dealer_key and
         T.dim_model_key = S.dim_model_key and T.dim_data_key = S.dim_date_key')
                       .whenMatchedUpdateAll().whenNotMatchedInsertAll().execute()
else :
    df_fact.write.format("delta").mode("Overwrite").option("path","abfss://gold@nnadls2025.dfs.core.windows.net/factsales")
            .saveAsTable("cars_catalog.gold.factsales")
%sql Select * from cars_catalog.gold.factsales
```