Harpalsinh Vaghela

Weekend Project
Data Handling with SQL Database, ADLS Gen2, SCD Type 1 Logic and Databricks

Problem Definition:

Objective:

- Build a data pipeline to extract data from an SQL database, clean it by removing duplicates, and store it in Azure Data Lake Storage (ADLS) Gen2 using Databricks.

Tasks:

Extract Data from SQL Database

- Establish a connection to the SQL database using JDBC.
- Retrieve and verify the data.

Set Up ADLS Gen2 Mount Point

- Create an ADLS Gen2 container.
- Configure a Databricks mount point for ADLS Gen2 access.

Configure Azure Key Vault and Databricks Scope

- Create an Azure Key Vault and add secrets.
- Set up a Databricks secret scope for secure credentials management.

Clean Data (Remove Duplicates)

- Identify and remove duplicate records.
- Validate the cleaned data.

Save Cleaned Data to ADLS Gen2

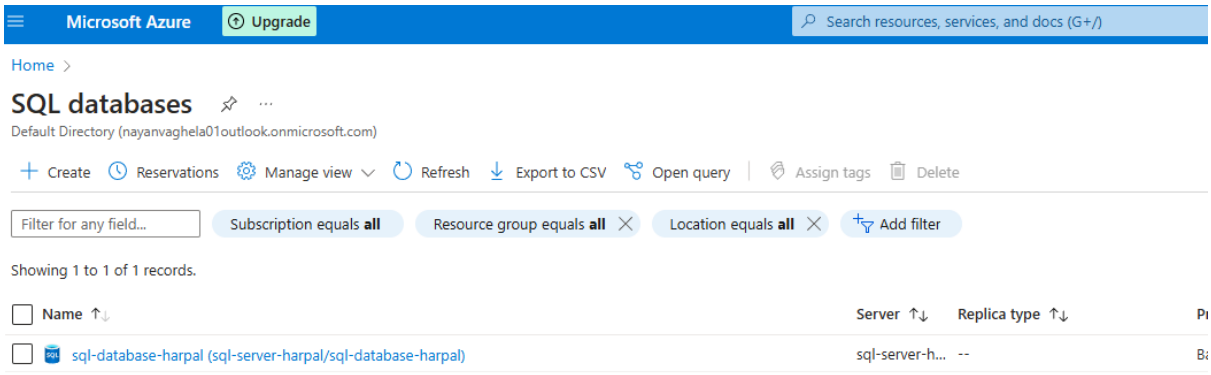
- Save processed data in the designated container.
- Verify the saved data format.

Create SCD type 1 Logic using the cleaned data.

Table of Contents

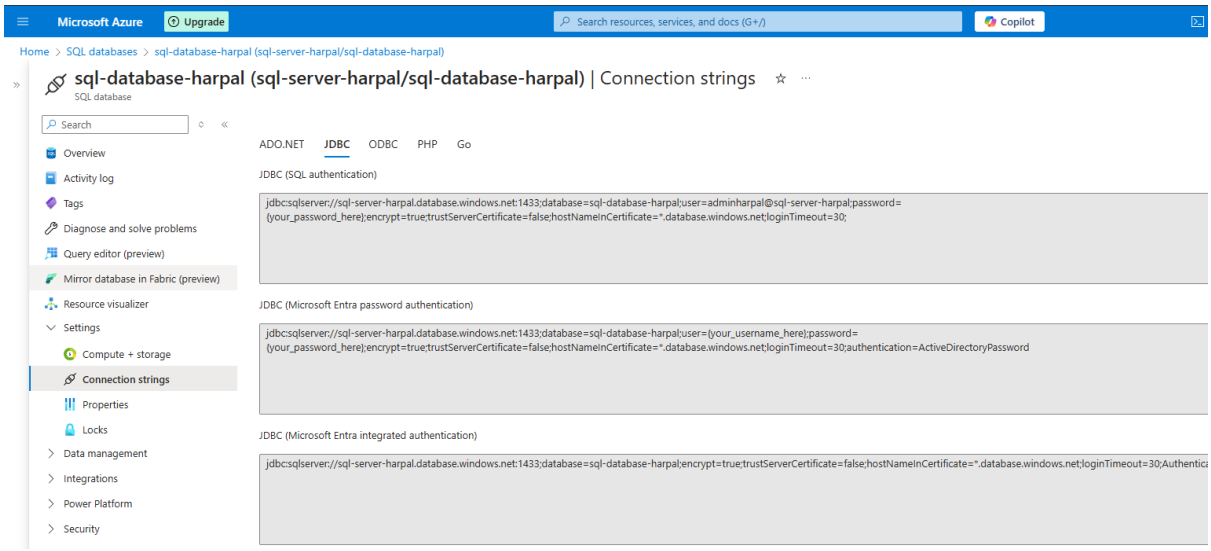
Extract Data from SQL Database

Go to Azure Home -> SQL Databases



Select the database

Go to Settings -> Connection String -> JDBC



Copy that JDBC (SQL authentication) URL

jdbc:sqlserver://sql-server-harpal.database.windows.net:1433;database=sql-database-harpal;user=adminharpal@sql-server-harpal;password={your_password_here};encrypt=true;trustServerCertificate=false;hostNameInCertificate=*.database.windows.net;loginTimeout=30;

Here, there are 3 things: URL, user, and password

URL:

jdbc:sqlserver://sql-server-harpal.database.windows.net:1433;database=sql-database-harpal;

User:

adminharpal

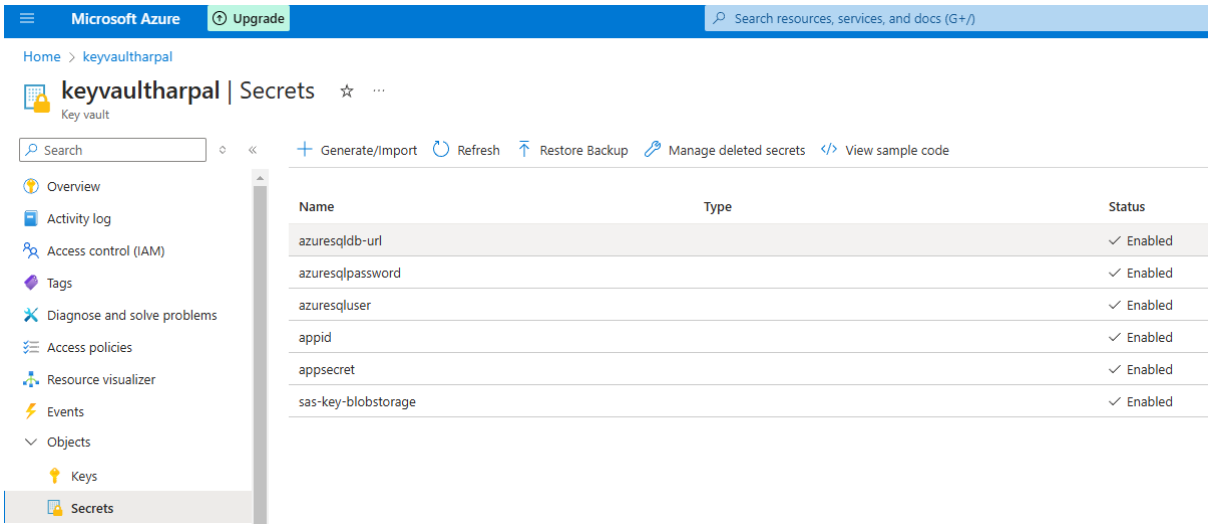
Password:

[Azure SQL Database password here]

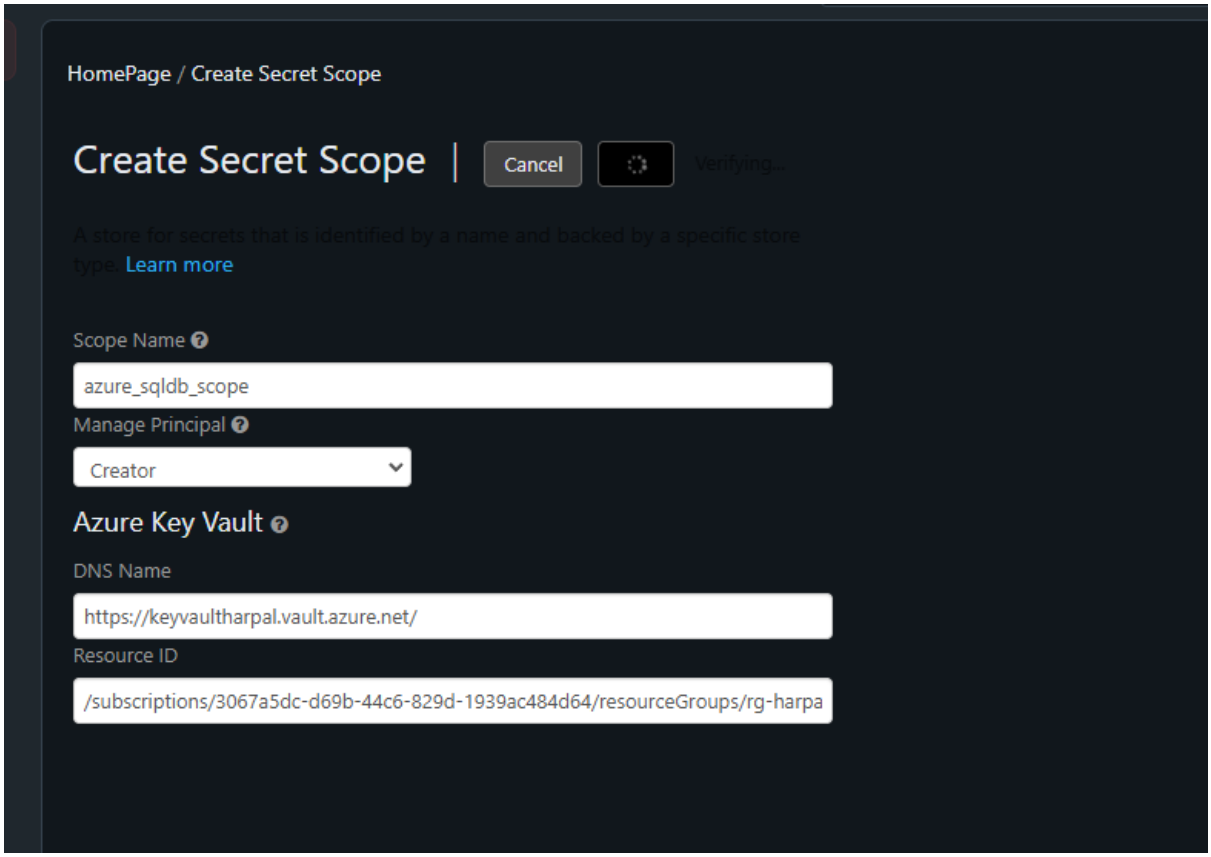
Configure Azure Key Vault and Databricks Scope

Go to **Azure Key Vault**

Object -> Secrets -> Generate/Import and write Name for Azure SQL user and paste that username in secret value, do it again for Azure SQL database URL and Azure SQL database password



Go to Azure Databricks and Open it, create scope as shown below by modifying the web browser URL after https://adb-1485266502012438.18.azuredatabricks.net**/#secret/createScope**



Set Up ADLS Gen2 Mount Point

Visit this website: https://learn.microsoft.com/en-us/azure/databricks/connect/external-systems/jdbc

Copy the code to read data with JDBC as shown below:

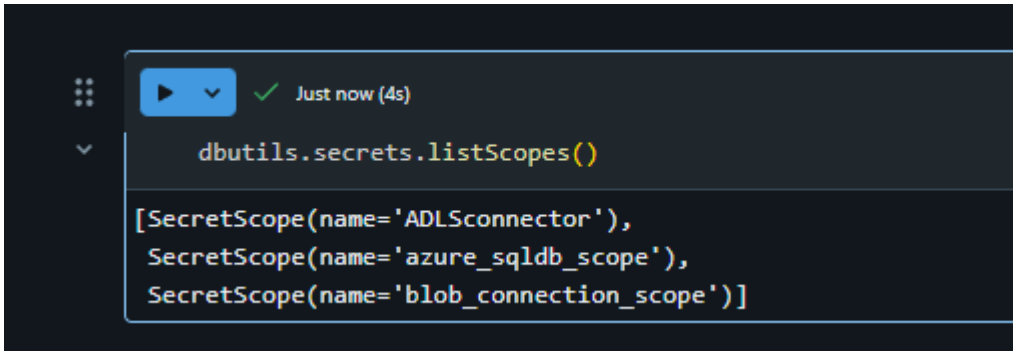
```
employees_table = (spark.read
  .format("jdbc")
  .option("url", "<jdbc-url>")
  .option("dbtable", "<table-name>")
  .option("user", "<username>")
  .option("password", "<password>")
  .load()
)
```

Start the cluster

Go to Workspace -> Open/Create the Notebook, attach the cluster

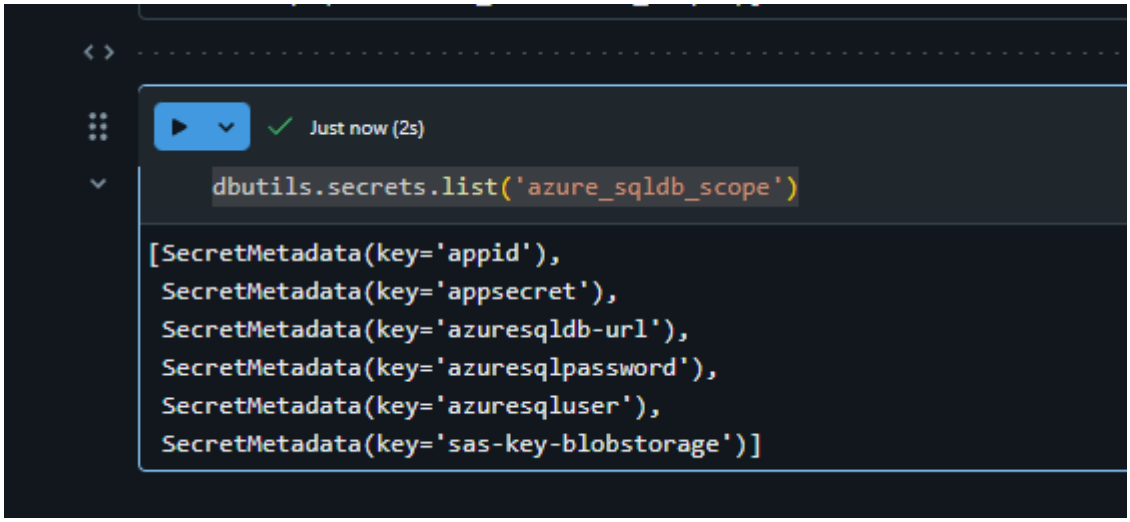
Run this command below to check all created scopes

```
dbutils.secrets.listScopes()
```



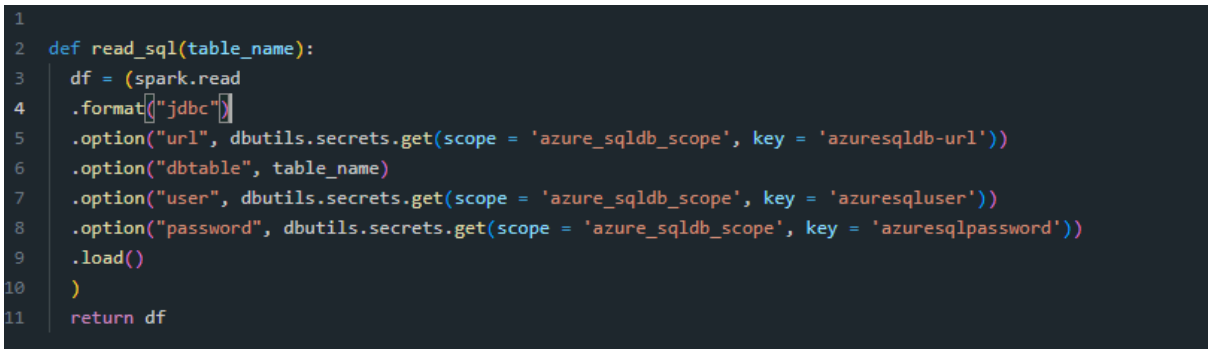
Run this

dbutils.secrets.list('azure_sqldb_scope')



Modify the code we get from the Microsoft website above and create a function to pass values dynamically

```
def read_sql(table_name):
 df = (spark.read
 .format("jdbc")
 .option("url", dbutils.secrets.get(scope = 'azure_sqldb_scope', key = 'azuresqldb-url'))
 .option("dbtable", table_name)
 .option("user", dbutils.secrets.get(scope = 'azure_sqldb_scope', key = 'azuresqluser'))
 .option("password", dbutils.secrets.get(scope = 'azure_sqldb_scope', key = 'azuresqlpassword'))
 .load()
 )
 return df
```



If we want to pass a custom query, write the code below and use the function to pass a query parameter dynamically, but for this project we will use dynamic table name parameter as explained above

```
def read_sql_query(query):
 df = (spark.read
 .format("jdbc")
 .option("url", dbutils.secrets.get(scope = 'azure_sqldb_scope', key = 'azuresqldb-url'))
 .option("query", query)
 .option("user", dbutils.secrets.get(scope = 'azure_sqldb_scope', key = 'azuresqluser'))
 .option("password", dbutils.secrets.get(scope = 'azure_sqldb_scope', key = 'azuresqlpassword'))
 .load()
 )
 return df
```

```
def read_sql_query(query):
  df = (spark.read
  .format("jdbc")
  .option("url", dbutils.secrets.get(scope = 'azure_sqldb_scope', key = 'azuresqldb-url'))
  .option("query", query)
  .option("user", dbutils.secrets.get(scope = 'azure_sqldb_scope', key = 'azuresqluser'))
  .option("password", dbutils.secrets.get(scope = 'azure_sqldb_scope', key = 'azuresqlpassword'))
  .load()
  )
  return df
```

Let's check the records of SalesLT.Product Table in Azure SQL from SSMS Tool:

```
SELECT TOP (200) [ProductID]
      ,[Name]
      ,[ProductNumber]
      ,[Color]
      ,[StandardCost]
      ,[ListPrice]
      ,[Size]
      ,[Weight]
      ,[ProductCategoryID]
      ,[ProductModelID]
```

Results Messages

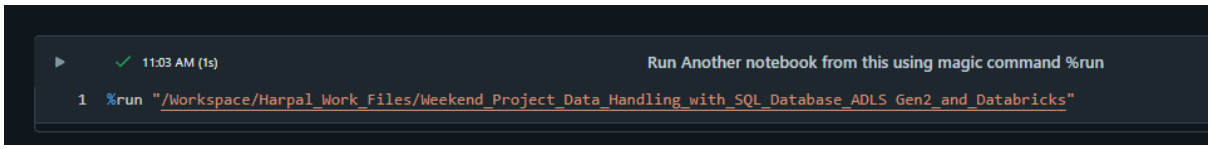
	ProductID	Name	ProductNumber	Color	StandardCost	ListPrice	Size	Weight	ProductCategoryID	ProductModelID	SellStartDate	SellEndDate	DiscontinuedDate	ThumbNailPhoto
1	680	HL Road Frame - Black, 58	FR-R92B-58	Black	1059.31	1431.50	58	1016.04	18	6	2002-06-01 00:00:00.000	NULL	NULL	0x4749463839615000
2	706	HL Road Frame - Red, 58	FR-R92R-58	Red	1059.31	1431.50	58	1016.04	18	6	2002-06-01 00:00:00.000	NULL	NULL	0x4749463839615000
3	707	Sport-100 Helmet, Red	HL-U509-R	Red	13.0863	34.99	NULL	NULL	35	33	2005-07-01 00:00:00.000	NULL	NULL	0x4749463839615000
4	708	Sport-100 Helmet, Black	HL-U509	Black	13.0863	34.99	NULL	NULL	35	33	2005-07-01 00:00:00.000	NULL	NULL	0x4749463839615000
5	709	Mountain Bike Socks, M	SO-B909-M	White	3.3963	9.50	M	NULL	27	18	2005-07-01 00:00:00.000	2006-06-30 00:00:00.000	NULL	0x4749463839615000
6	710	Mountain Bike Socks, L	SO-B909-L	White	3.3963	9.50	L	NULL	27	18	2005-07-01 00:00:00.000	2006-06-30 00:00:00.000	NULL	0x4749463839615000
7	711	Sport-100 Helmet, Blue	HL-U509-B	Blue	13.0863	34.99	NULL	NULL	35	33	2005-07-01 00:00:00.000	NULL	NULL	0x4749463839615000
8	712	AWC Logo Cap	CA-1098	Multi	6.9223	8.99	NULL	NULL	23	2	2005-07-01 00:00:00.000	NULL	NULL	0x4749463839615000
9	713	Long-Sleeve Logo Jersey, S	LJ-0192-S	Multi	38.4923	49.99	S	NULL	25	11	2005-07-01 00:00:00.000	NULL	NULL	0x4749463839615000
10	714	Long-Sleeve Logo Jersey, M	LJ-0192-M	Multi	38.4923	49.99	M	NULL	25	11	2005-07-01 00:00:00.000	NULL	NULL	0x4749463839615000
11	715	Long-Sleeve Logo Jersey, L	LJ-0192-L	Multi	38.4923	49.99	L	NULL	25	11	2005-07-01 00:00:00.000	NULL	NULL	0x4749463839615000
12	716	Long-Sleeve Logo Jersey, XL	LJ-0192-X	Multi	38.4923	49.99	XL	NULL	25	11	2005-07-01 00:00:00.000	NULL	NULL	0x4749463839615000
13	717	HL Road Frame - Red, 62	FR-R92R-62	Red	868.6342	1431.50	62	1043.26	18	6	2005-07-01 00:00:00.000	NULL	NULL	0x4749463839615000
14	718	HL Road Frame - Red, 44	FR-R92R-44	Red	868.6342	1431.50	44	961.61	18	6	2005-07-01 00:00:00.000	NULL	NULL	0x4749463839615000
15	719	HL Road Frame - Red, 48	FR-R92R-48	Red	868.6342	1431.50	48	979.75	18	6	2005-07-01 00:00:00.000	NULL	NULL	0x4749463839615000
16	720	HL Road Frame - Red, 52	FR-R92R-52	Red	868.6342	1431.50	52	997.90	18	6	2005-07-01 00:00:00.000	NULL	NULL	0x4749463839615000
17	721	HL Road Frame - Red, 56	FR-R92R-56	Red	868.6342	1431.50	56	1016.04	18	6	2005-07-01 00:00:00.000	NULL	NULL	0x4749463839615000
18	722	LL Road Frame - Black, 58	FR-R38B-58	Black	204.6251	337.22	58	1115.83	18	9	2005-07-01 00:00:00.000	NULL	NULL	0x4749463839615000
19	723	LL Road Frame - Black, 60	FR-R38B-60	Black	204.6251	337.22	60	1124.90	18	9	2005-07-01 00:00:00.000	NULL	NULL	0x4749463839615000
20	724	LL Road Frame - Black, 62	FR-R38B-62	Black	204.6251	337.22	62	1133.98	18	9	2005-07-01 00:00:00.000	NULL	NULL	0x4749463839615000
21	725	LL Road Frame - Red, 44	FR-R38R-44	Red	187.1571	337.22	44	1052.33	18	9	2005-07-01 00:00:00.000	2007-06-30 00:00:00.000	NULL	0x4749463839615000
22	726	LL Road Frame - Red, 48	FR-R38R-48	Red	187.1571	337.22	48	1070.47	18	9	2005-07-01 00:00:00.000	2007-06-30 00:00:00.000	NULL	0x4749463839615000
23	727	LL Road Frame - Red, 52	FR-R38R-52	Red	187.1571	337.22	52	1088.62	18	9	2005-07-01 00:00:00.000	2007-06-30 00:00:00.000	NULL	0x4749463839615000
24	728	LL Road Frame - Red, 58	FR-R38R-58	Red	187.1571	337.22	58	1115.83	18	9	2005-07-01 00:00:00.000	2007-06-30 00:00:00.000	NULL	0x4749463839615000
25	729	LL Road Frame - Red, 60	FR-R38R-60	Red	187.1571	337.22	60	1124.90	18	9	2005-07-01 00:00:00.000	2007-06-30 00:00:00.000	NULL	0x4749463839615000
26	730	LL Road Frame - Red, 62	FR-R38R-62	Red	187.1571	337.22	62	1133.98	18	9	2005-07-01 00:00:00.000	2007-06-30 00:00:00.000	NULL	0x4749463839615000
27	731	ML Road Frame - Red, 44	FR-R72R-44	Red	352.1394	594.83	44	1006.97	18	16	2005-07-01 00:00:00.000	2006-06-30 00:00:00.000	NULL	0x4749463839615000

Okay, so we can do some transformation on this table, such as here, we have some null values, the DiscontinuedDate column is empty with NULL values, remove duplicates if any, and rename some column names per the client's need.

Clean Data (Remove Duplicates)

First of all, after creating those 3 functions, we can create another Notebook to use those functions from the newly created notebook using a magic command called %run as shown below:

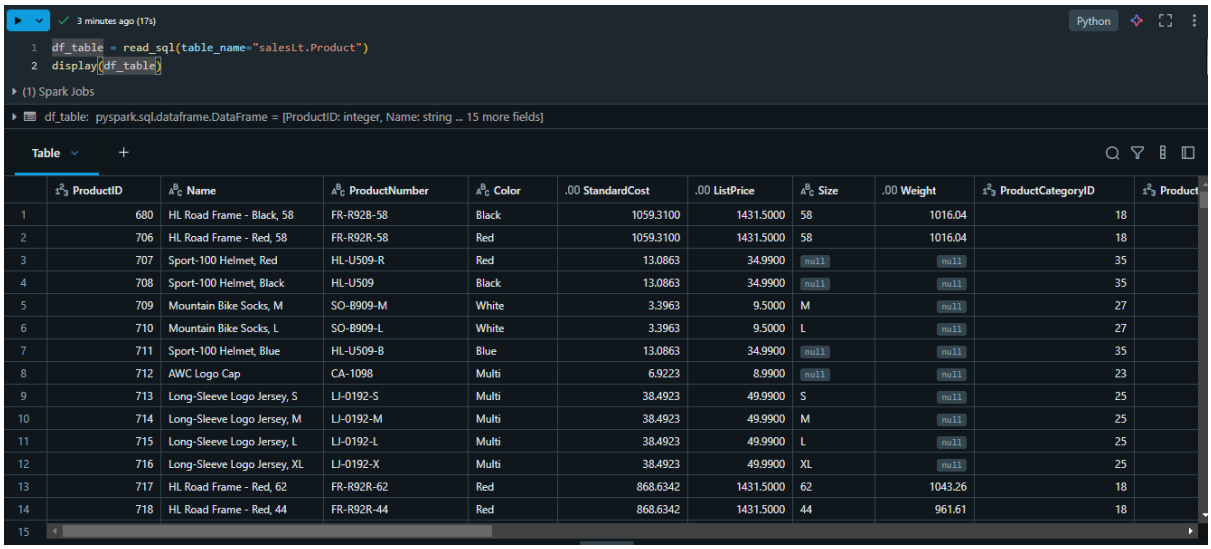
%run "/Workspace/Harpal_Work_Files/Weekend_Project_Data_Handling_with_SQL_Database_ADLS Gen2_and_Databricks"



Now, we can read the Product table

df_table = read_sql(table_name="salesLt.Product")

display(df_table)



	ProductID	Name	ProductNumber	Color	StandardCost	ListPrice	Size	Weight	ProductCategoryID	Product
1	680	HL Road Frame - Black, 58	FR-R92B-58	Black	1059.3100	1431.5000	58	1016.04	18	
2	706	HL Road Frame - Red, 58	FR-R92R-58	Red	1059.3100	1431.5000	58	1016.04	18	
3	707	Sport-100 Helmet, Red	HL-U509-R	Red	13.0863	34.9900	null	null	35	
4	708	Sport-100 Helmet, Black	HL-U509	Black	13.0863	34.9900	null	null	35	
5	709	Mountain Bike Socks, M	SO-B909-M	White	3.3963	9.5000	M	null	27	
6	710	Mountain Bike Socks, L	SO-B909-L	White	3.3963	9.5000	L	null	27	
7	711	Sport-100 Helmet, Blue	HL-U509-B	Blue	13.0863	34.9900	null	null	35	
8	712	AWC Logo Cap	CA-1098	Multi	6.9223	8.9900	null	null	23	
9	713	Long-Sleeve Logo Jersey, S	LJ-0192-S	Multi	38.4923	49.9900	S	null	25	
10	714	Long-Sleeve Logo Jersey, M	LJ-0192-M	Multi	38.4923	49.9900	M	null	25	
11	715	Long-Sleeve Logo Jersey, L	LJ-0192-L	Multi	38.4923	49.9900	L	null	25	
12	716	Long-Sleeve Logo Jersey, XL	LJ-0192-X	Multi	38.4923	49.9900	XL	null	25	
13	717	HL Road Frame - Red, 62	FR-R92R-62	Red	868.6342	1431.5000	62	1043.26	18	
14	718	HL Road Frame - Red, 44	FR-R92R-44	Red	868.6342	1431.5000	44	961.61	18	
15										

Let's first remove the duplicate values from the data frame as we have stored the table's records in the data frame

df_table_unique= df_table.dropDuplicates()

display(df_table_unique)

```
df_table_unique= df_table.dropDuplicates()
display(df_table_unique)
```

(2) Spark Jobs

df_table_unique: pyspark.sql.dataframe.DataFrame = [ProductID: integer, Name: string ... 15 more fields]

Table +

	ProductID	Name	ProductNumber	Color	StandardCost	ListPrice	Size	Weight	ProductCategoryID	Product
1	875	Racing Socks, L	SO-R809-L	White	3.3623	8.9900	L	null	27	
2	863	Full-Finger Gloves, L	GL-F110-L	Black	15.6709	37.9900	L	null	24	
3	954	Touring-1000 Yellow, 46	BK-T79Y-46	Yellow	1481.9379	2384.0700	46	11398.72	7	
4	978	Touring-3000 Blue, 44	BK-T18U-44	Blue	461.4448	742.3500	44	13049.78	7	
5	989	Mountain-500 Black, 40	BK-M188-40	Black	294.5797	539.9900	40	12405.69	5	
6	974	Road-350-W Yellow, 42	BK-R79Y-42	Yellow	1082.5100	1700.9900	42	7153.11	6	
7	804	HL Fork	FK-9939	null	101.8936	229.4900	null	null	14	
8	813	HL Road Handlebars	HB-R956	null	53.3999	120.2700	null	null	8	
9	852	Women's Tights, S	TG-W091-S	Black	30.9334	74.9900	S	null	28	
10	792	Road-250 Red, 58	BK-R89R-58	Red	1554.9479	2443.3500	58	7162.19	6	
11	816	ML Mountain Front Wheel	FW-M762	Black	92.8071	209.0250	null	null	21	
12	818	LL Road Front Wheel	FW-R623	Black	37.9909	85.5650	null	900.00	21	
13	962	Touring-3000 Yellow, 50	BK-T18Y-50	Yellow	461.4448	742.3500	50	13213.08	7	
14	748	HL Mountain Frame - Silver, ...	FR-M94S-38	Silver	747.2002	1364.5000	38	1215.62	16	
15										

#Remove Null Column

df_table = df_table.drop("Weight","SellStartDate","SellEndDate", "ThumbNailPhoto", "DiscontinuedDate", "ThumbnailPhotoFileName", "rowguid", "ModifiedDate")

display(df_table)

Table +

	ProductID	Name	ProductNumber	Color	StandardCost	ListPrice	Size	ProductCategoryID	ProductModelID
1	680	HL Road Frame - Black, 58	FR-R92B-58	Black	1059.3100	1431.5000	58	18	6
2	706	HL Road Frame - Red, 58	FR-R92R-58	Red	1059.3100	1431.5000	58	18	6
3	707	Sport-100 Helmet, Red	HL-U509-R	Red	13.0863	34.9900	null	35	33
4	708	Sport-100 Helmet, Black	HL-U509	Black	13.0863	34.9900	null	35	33
5	709	Mountain Bike Socks, M	SO-B909-M	White	3.3963	9.5000	M	27	18
6	710	Mountain Bike Socks, L	SO-B909-L	White	3.3963	9.5000	L	27	18
7	711	Sport-100 Helmet, Blue	HL-U509-B	Blue	13.0863	34.9900	null	35	33
8	712	AWC Logo Cap	CA-1098	Multi	6.9223	8.9900	null	23	2
9	713	Long-Sleeve Logo Jersey, S	LJ-0192-S	Multi	38.4923	49.9900	S	25	11
10	714	Long-Sleeve Logo Jersey, M	LJ-0192-M	Multi	38.4923	49.9900	M	25	11
11	715	Long-Sleeve Logo Jersey, L	LJ-0192-L	Multi	38.4923	49.9900	L	25	11
12	716	Long-Sleeve Logo Jersey, XL	LJ-0192-X	Multi	38.4923	49.9900	XL	25	11
13	717	HL Road Frame - Red, 62	FR-R92R-62	Red	868.6342	1431.5000	62	18	6
14	718	HL Road Frame - Red, 44	FR-R92R-44	Red	868.6342	1431.5000	44	18	6
15	719	HL Road Frame - Red, 48	FR-R92R-48	Red	868.6342	1431.5000	48	18	6

Check the schema data type of a dataframe

df_table.printSchema()

```
df_table.printSchema()

root
 |-- ProductID: integer (nullable = true)
 |-- Name: string (nullable = true)
 |-- ProductNumber: string (nullable = true)
 |-- Color: string (nullable = true)
 |-- StandardCost: decimal(19,4) (nullable = true)
 |-- ListPrice: decimal(19,4) (nullable = true)
 |-- Size: string (nullable = true)
 |-- Weight: decimal(8,2) (nullable = true)
 |-- ProductCategoryID: integer (nullable = true)
 |-- ProductModelID: integer (nullable = true)
 |-- SellStartDate: timestamp (nullable = true)
 |-- SellEndDate: timestamp (nullable = true)
 |-- DiscontinuedDate: timestamp (nullable = true)
 |-- ThumbNailPhoto: binary (nullable = true)
 |-- ThumbnailPhotoFileName: string (nullable = true)
 |-- rowguid: string (nullable = true)
 |-- ModifiedDate: timestamp (nullable = true)
```

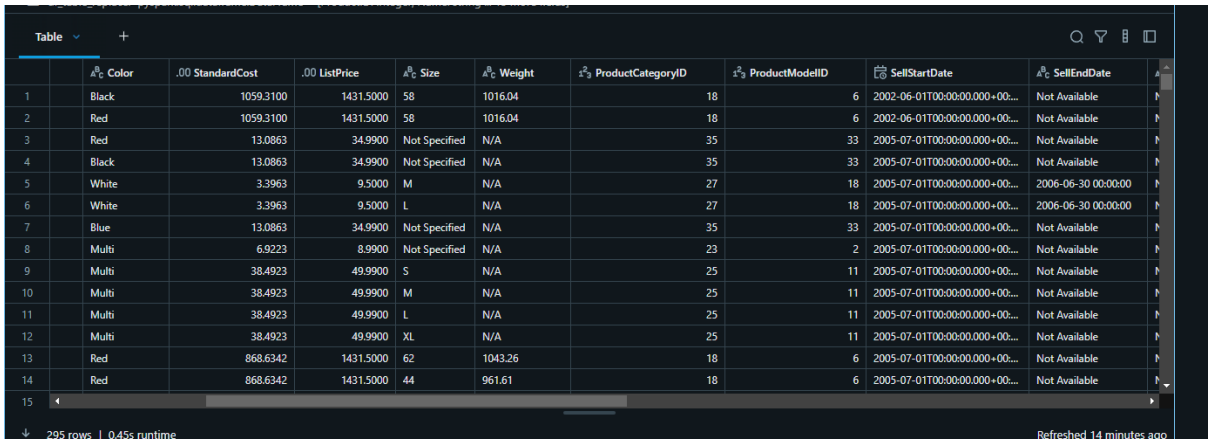
Replace NULL Values

```
from pyspark.sql.functions import col

#casting decimal and datetime into string datatype to replace with string values like (Not Specified, Not Available if they are NULL)

fill_values = {
   'Color': 'Unknown',
   'Size': 'Not Specified'
}

df_table_replace = df_table.na.fill(fill_values)
display(df_table_replace)
```



Table

	Color	StandardCost	ListPrice	Size	Weight	ProductCategoryID	ProductModelID	SellStartDate	SellEndDate
1	Black	1059.3100	1431.5000	58	1016.04	18	6	2002-06-01T00:00:00.000+00:...	Not Available
2	Red	1059.3100	1431.5000	58	1016.04	18	6	2002-06-01T00:00:00.000+00:...	Not Available
3	Red	13.0863	34.9900	Not Specified	N/A	35	33	2005-07-01T00:00:00.000+00:...	Not Available
4	Black	13.0863	34.9900	Not Specified	N/A	35	33	2005-07-01T00:00:00.000+00:...	Not Available
5	White	3.3963	9.5000	M	N/A	27	18	2005-07-01T00:00:00.000+00:...	2006-06-30 00:00:00
6	White	3.3963	9.5000	L	N/A	27	18	2005-07-01T00:00:00.000+00:...	2006-06-30 00:00:00
7	Blue	13.0863	34.9900	Not Specified	N/A	35	33	2005-07-01T00:00:00.000+00:...	Not Available
8	Multi	6.9223	8.9900	Not Specified	N/A	23	2	2005-07-01T00:00:00.000+00:...	Not Available
9	Multi	38.4923	49.9900	S	N/A	25	11	2005-07-01T00:00:00.000+00:...	Not Available
10	Multi	38.4923	49.9900	M	N/A	25	11	2005-07-01T00:00:00.000+00:...	Not Available
11	Multi	38.4923	49.9900	L	N/A	25	11	2005-07-01T00:00:00.000+00:...	Not Available
12	Multi	38.4923	49.9900	XL	N/A	25	11	2005-07-01T00:00:00.000+00:...	Not Available
13	Red	868.6342	1431.5000	62	1043.26	18	6	2005-07-01T00:00:00.000+00:...	Not Available
14	Red	868.6342	1431.5000	44	961.61	18	6	2005-07-01T00:00:00.000+00:...	Not Available
15									

295 rows | 0.45s runtime

Refreshed 14 minutes ago

Filter Data based on Filter Condition

Only take records where size should be either (M, L, SL and greater than 45 is its number.)

And Color should not be unknown

```
df_table_replace = df_table_replace.filter((col("Size") > '45') & (col("Size").isin("M", "L", "XL")))

df_table_replace = df_table_replace.filter(~col("Color").isin("Unknown"))

display(df_table_replace)
```

```
df_table_replace = df_table_replace.filter((col("Size") > '45') & (col("Size").isin("M", "L", "XL")))
df_table_replace = df_table_replace.filter(~col("Color").isin("Unknown"))
display(df_table_replace)
```

▸ (1) Spark Jobs

▸ df_table_replace: pyspark.sql.dataframe.DataFrame = [ProductID: integer, Name: string ... 15 more fields]

Table +

	ProductID	Name	ProductNumber	Color	StandardCost	ListPrice	Size	Weight	ProductCategoryID	ProductI
12	857	Men's Bib-Shorts, L	SB-M891-L	Multi	37.1209	89.9900	L	N/A	22	
13	859	Half-Finger Gloves, M	GL-H102-M	Black	9.1593	24.4900	M	N/A	24	
14	860	Half-Finger Gloves, L	GL-H102-L	Black	9.1593	24.4900	L	N/A	24	
15	862	Full-Finger Gloves, M	GL-F110-M	Black	15.6709	37.9900	M	N/A	24	
16	863	Full-Finger Gloves, L	GL-F110-L	Black	15.6709	37.9900	L	N/A	24	
17	865	Classic Vest, M	VE-C304-M	Blue	23.7490	63.5000	M	N/A	29	
18	866	Classic Vest, L	VE-C304-L	Blue	23.7490	63.5000	L	N/A	29	
19	868	Women's Mountain Shorts, M	SH-W890-M	Black	26.1763	69.9900	M	N/A	26	
20	869	Women's Mountain Shorts, L	SH-W890-L	Black	26.1763	69.9900	L	N/A	26	
21	874	Racing Socks, M	SO-R809-M	White	3.3623	8.9900	M	N/A	27	
22	875	Racing Socks, L	SO-R809-L	White	3.3623	8.9900	L	N/A	27	
23	882	Short-Sleeve Classic Jersey, M	SJ-0194-M	Yellow	41.5723	53.9900	M	N/A	25	
24	883	Short-Sleeve Classic Jersey, L	SJ-0194-L	Yellow	41.5723	53.9900	L	N/A	25	
25	884	Short-Sleeve Classic Jersey, ...	SJ-0194-X	Yellow	41.5723	53.9900	XL	N/A	25	

Rename the column Name:

df_table_col_renamed = df_table_replace.withColumnRenamed("Name","ProductName").withColumnRenamed("rowguid", "RowGUId")

display(df_table_col_renamed)

```
df_table_col_renamed = df_table_replace.withColumnRenamed("Name","ProductName").withColumnRenamed("rowguid", "RowGUId")
display(df_table_col_renamed)
```

▸ (1) Spark Jobs

▸ df_table_col_renamed: pyspark.sql.dataframe.DataFrame = [ProductID: integer, ProductName: string ... 7 more fields]

Table +

	ProductID	ProductName	ProductNumber	Color	StandardCost	ListPrice	Size	ProductCategoryID	Prod
14	860	Half-Finger Gloves, L	GL-H102-L	Black	9.1593	24.4900	L	24	
15	862	Full-Finger Gloves, M	GL-F110-M	Black	15.6709	37.9900	M	24	
16	863	Full-Finger Gloves, L	GL-F110-L	Black	15.6709	37.9900	L	24	
17	865	Classic Vest, M	VE-C304-M	Blue	23.7490	63.5000	M	29	
18	866	Classic Vest, L	VE-C304-L	Blue	23.7490	63.5000	L	29	
19	868	Women's Mountain Shorts, M	SH-W890-M	Black	26.1763	69.9900	M	26	
20	869	Women's Mountain Shorts, L	SH-W890-L	Black	26.1763	69.9900	L	26	
21	874	Racing Socks, M	SO-R809-M	White	3.3623	8.9900	M	27	
22	875	Racing Socks, L	SO-R809-L	White	3.3623	8.9900	L	27	
23	882	Short-Sleeve Classic Jersey, M	SJ-0194-M	Yellow	41.5723	53.9900	M	25	
24	883	Short-Sleeve Classic Jersey, L - Updated	SJ-0194-L	Red	43.5000	56.9900	L	25	
25	884	Short-Sleeve Classic Jersey, XL - Updated	SJ-0194-X	Orange	42.0000	55.9900	XL	25	
26	1000	Lightweight Wind Jacket, XL	WJ-NEW-XL	Grey	48.9990	99.9900	XL	30	
27	1001	All-Weather Cycling Jacket, XL - Limited Editi...	CJ-AW-1001-XL	Navy Blue	57.7500	114.9900	XL	31	

Check mount points (as we need ADLS Gen 2 storage account mount point):

dbutils.fs.mounts()

```
dbutils.fs.mounts()
```

```
[MountInfo(mountPoint='/databricks-datasets', source='databricks-datasets', encryptionType=''),
 MountInfo(mountPoint='/mnt/ctnblob', source='wasbs://ctnblob@generalpurposestharpal.blob.core.windows.net', encryptionType=''),
 MountInfo(mountPoint='/Volumes', source='UnityCatalogVolumes', encryptionType=''),
 MountInfo(mountPoint='/databricks/mlflow-tracking', source='databricks/mlflow-tracking', encryptionType=''),
 MountInfo(mountPoint='/databricks-results', source='databricks-results', encryptionType=''),
 MountInfo(mountPoint='/databricks/mlflow-registry', source='databricks/mlflow-registry', encryptionType=''),
 MountInfo(mountPoint='/mnt/csvfiles', source='abfss://csvfiles@adlsgen2harpal.dfs.core.windows.net/', encryptionType=''),
 MountInfo(mountPoint='/Volume', source='DbfsReserved', encryptionType=''),
 MountInfo(mountPoint='/volumes', source='DbfsReserved', encryptionType=''),
 MountInfo(mountPoint='/', source='DatabricksRoot', encryptionType=''),
 MountInfo(mountPoint='/volume', source='DbfsReserved', encryptionType='')]
```

So, we have access to the ADLS Gen 2 storage account.

List all content of ADLS Gen 2

dbutils.fs.ls("/mnt/csvfiles")

```
dbutils.fs.ls("/mnt/csvfiles")
```

```
[FileInfo(path='dbfs:/mnt/csvfiles/CustomersDataCsv/', name='CustomersDataCsv/', size=0, modificationTime=1742699402000),
 FileInfo(path='dbfs:/mnt/csvfiles/cities.csv', name='cities.csv', size=8373, modificationTime=1742698462000),
 FileInfo(path='dbfs:/mnt/csvfiles/day.csv', name='day.csv', size=355, modificationTime=1742692766000),
 FileInfo(path='dbfs:/mnt/csvfiles/industry.csv', name='industry.csv', size=749, modificationTime=1742692766000),
 FileInfo(path='dbfs:/mnt/csvfiles/month.csv', name='month.csv', size=244, modificationTime=1742692766000),
 FileInfo(path='dbfs:/mnt/csvfiles/sample_homes.csv', name='sample_homes.csv', size=1577, modificationTime=1742742891000),
 FileInfo(path='dbfs:/mnt/csvfiles/synapse/', name='synapse/', size=0, modificationTime=1742697838000),
 FileInfo(path='dbfs:/mnt/csvfiles/time.csv', name='time.csv', size=1575, modificationTime=1742692766000),
 FileInfo(path='dbfs:/mnt/csvfiles/username.csv', name='username.csv', size=176, modificationTime=1742692767000),
 FileInfo(path='dbfs:/mnt/csvfiles/weekday.csv', name='weekday.csv', size=162, modificationTime=1742692767000)]
```

Save Cleaned Data to ADLS Gen2

df_table_col_renamed.write.format("delta").mode("overwrite").save("/mnt/csvfiles/ProductDetails")

Now, let's read data in a Data Bricks notebook from that Delta folder

df_productData = spark.read.format("delta").option("header","true").option("inferSchema", "true").load("/mnt/csvfiles/ProductDetails")

display(df_productData)

```
df_productData = spark.read.format("delta").option("header","true").option("inferSchema", "true").load("/mnt/csvfiles/ProductDetails")
display(df_productData)
```

▸ (1) Spark Jobs

▸ df_productData: pyspark.sql.dataframe.DataFrame = [ProductID: integer, ProductName: string ... 15 more fields]

Table +

	ProductID	ProductName	ProductNumber	Color	StandardCost	ListPrice	Size	Weight	ProductCategoryID	ProductI
1	709	Mountain Bike Socks, M	SO-B909-M	White	3.3963	9.5000	M	N/A	27	
2	710	Mountain Bike Socks, L	SO-B909-L	White	3.3963	9.5000	L	N/A	27	
3	714	Long-Sleeve Logo Jersey, M	LJ-0192-M	Multi	38.4923	49.9900	M	N/A	25	
4	715	Long-Sleeve Logo Jersey, L	LJ-0192-L	Multi	38.4923	49.9900	L	N/A	25	
5	716	Long-Sleeve Logo Jersey, XL	LJ-0192-X	Multi	38.4923	49.9900	XL	N/A	25	
6	849	Men's Sports Shorts, M	SH-M897-M	Black	24.7459	59.9900	M	N/A	26	
7	850	Men's Sports Shorts, L	SH-M897-L	Black	24.7459	59.9900	L	N/A	26	
8	851	Men's Sports Shorts, XL	SH-M897-X	Black	24.7459	59.9900	XL	N/A	26	
9	853	Women's Tights, M	TG-W091-M	Black	30.9334	74.9900	M	N/A	28	
10	854	Women's Tights, L	TG-W091-L	Black	30.9334	74.9900	L	N/A	28	
11	856	Men's Bib-Shorts, M	SB-M891-M	Multi	37.1209	89.9900	M	N/A	22	
12	857	Men's Bib-Shorts, L	SB-M891-L	Multi	37.1209	89.9900	L	N/A	22	
13	859	Half-Finger Gloves, M	GL-H102-M	Black	9.1593	24.4900	M	N/A	24	
14	860	Half-Finger Gloves, L	GL-H102-L	Black	9.1593	24.4900	L	N/A	24	
15										

25 rows | 2.99s runtime Refreshed now

Create SCD type 1 Logic using the cleaned data

```
%sql
CREATE TABLE IF NOT EXISTS hive_metastore.default.ProductDetailSCD1 (
  ProductID INT,
  ProductName STRING,
  ProductNumber STRING,
  Color STRING,
  StandardCost DECIMAL(10, 4),
  ListPrice DECIMAL(10, 4),
  Size STRING,
  ProductCategoryID INT,
  ProductModelID INT,
  HashKey BIGINT,
  CreatedDate TIMESTAMP,
```

```
  UpdatedDate TIMESTAMP,
  CreatedBy STRING,
  UpdatedBy STRING
)
USING DELTA
LOCATION '/mnt/csvfiles/scdtype1/gold/ProductDetailSCD1'
```

```
%sql
CREATE TABLE IF NOT EXISTS hive_metastore.default.ProductDetailSCD1 (
    ProductID INT,
    ProductName STRING,
    ProductNumber STRING,
    Color STRING,
    StandardCost DECIMAL(10, 4),
    ListPrice DECIMAL(10, 4),
    Size STRING,
    ProductCategoryID INT,
    ProductModelID INT,
    HashKey BIGINT,
    CreatedDate TIMESTAMP,
    UpdatedDate TIMESTAMP,
    CreatedBy STRING,
    UpdatedBy STRING
)
USING DELTA
LOCATION '/mnt/csvfiles/scdtype1/gold/ProductDetailSCD1'
```

```
OK
```

Create Hashkey and store all data in to a dataframe

```
from pyspark.sql.functions import *
df_hash_derived = df_productData.withColumn("HashKey", crc32(concat(*df_productData.columns)))
display(df_hash_derived)
```

```
from pyspark.sql.functions import *
df_hash_derived = df_productData.withColumn("HashKey", crc32(concat(*df_productData.columns)))
display(df_hash_derived)
```

(1) Spark Jobs

df_hash_derived: pyspark.sql.dataframe.DataFrame = [ProductID: integer, ProductName: string ... 8 more fields]

Table

	ctName	ProductNumber	Color	.00 Standard...	.00 ListPrice	Size	ProductCategoryID	ProductModelID	HashKey
1	Bike Socks, M	SO-B909-M	White	3.3963	9.5000	M	27	18	651255083
2	Bike Socks, L	SO-B909-L	White	3.3963	9.5000	L	27	18	1603521779
3	eve Logo Jersey, M	LJ-0192-M	Multi	38.4923	49.9900	M	25	11	425618936
4	eve Logo Jersey, L	LJ-0192-L	Multi	38.4923	49.9900	L	25	11	1877216255
5	eve Logo Jersey, XL	LJ-0192-X	Multi	38.4923	49.9900	XL	25	11	26411306
6	orts Shorts, M	SH-M897-M	Black	24.7459	59.9900	M	26	13	1047720458
7	orts Shorts, L	SH-M897-L	Black	24.7459	59.9900	L	26	13	1804354953
8	orts Shorts, XL	SH-M897-X	Black	24.7459	59.9900	XL	26	13	3973345037
9	Tights, M	TG-W091-M	Black	30.9334	74.9900	M	28	38	4041466321
10	Tights, L	TG-W091-L	Black	30.9334	74.9900	L	28	38	1470224739
11	-Shorts, M	SB-M891-M	Multi	37.1209	89.9900	M	22	12	1421151035
12	-Shorts, L	SB-M891-L	Multi	37.1209	89.9900	L	22	12	2516866034
13	er Gloves, M	GL-H102-M	Black	9.1593	24.4900	M	24	4	3508159959
14	er Gloves, L	GL-H102-L	Black	9.1593	24.4900	L	24	4	1058933213
15									

Create Delta table object and give target path and then convert delta table into dataframe

```
from delta.tables import DeltaTable
deltaTable = DeltaTable.forPath(spark, "/mnt/csvfiles/scdtype1/gold/ProductDetailSCD1")
deltaTable.toDF().show() #convert delta table to dataframe and display the data
```

```
from delta.tables import DeltaTable
deltaTable = DeltaTable.forPath(spark, "/mnt/csvfiles/scdtype1/gold/ProductDetailSCD1")
deltaTable.toDF().show() #convert delta table to dataframe and display the data
```

(1) Spark Jobs

```
|859|Half-Finger Glove...|   GL-H102-M|Black|     9.1593|  24.4900|   M|            24|          4|3508159959|2025-04-03 03:30:...|2025-04-03 03:30:...|   Harpal|   Harpa
|860|Half-Finger Glove...|   GL-H102-L|Black|     9.1593|  24.4900|   L|            24|          4|1058933213|2025-04-03 03:30:...|2025-04-03 03:30:...|   Harpal|   Harpa
|862|Full-Finger Glove...|   GL-F110-M|Black|    15.6709|  37.9900|   M|            24|          3| 429149323|2025-04-03 03:30:...|2025-04-03 03:30:...|   Harpal|   Harpa
|863|Full-Finger Glove...|   GL-F110-L|Black|    15.6709|  37.9900|   L|            24|          3| 391804812|2025-04-03 03:30:...|2025-04-03 03:30:...|   Harpal|   Harpa
|865|     Classic Vest, M|   VE-C304-M| Blue|    23.7490|  63.5000|   M|            29|          1|1915151503|2025-04-03 03:30:...|2025-04-03 03:30:...|   Harpal|   Harpa
|866|     Classic Vest, L|   VE-C304-L| Blue|    23.7490|  63.5000|   L|            29|          1| 465072443|2025-04-03 03:30:...|2025-04-03 03:30:...|   Harpal|   Harpa
|868|Women's Mountain ...|   SH-W890-M|Black|    26.1763|  69.9900|   M|            26|         37|2436464081|2025-04-03 03:30:...|2025-04-03 03:30:...|   Harpal|   Harpa
|869|Women's Mountain ...|   SH-W890-L|Black|    26.1763|  69.9900|   L|            26|         37|3884129238|2025-04-03 03:30:...|2025-04-03 03:30:...|   Harpal|   Harpa
```

Check for new data for SCD Type 1 Logic:

```
df_src = (
  df_hash_derived.alias("src")
  .join(
    deltaTable.toDF().alias("tgt"),
    (col("src.ProductID") == col("tgt.ProductID")),  # Join only on id
    "left"
  )
  .filter( (col("tgt.ProductID").isNull()) | (col("src.HashKey") != col("tgt.HashKey")) )
  .select("src.*")
)
display(df_src)
```

```
df_src = (
    df_hash_derived.alias("src")
    .join(
        deltaTable.toDF().alias("tgt"),
        (col("src.ProductID") == col("tgt.ProductID")),  # Join only on id
        "left"
    )
    .filter( (col("tgt.ProductID").isNull()) | (col("src.HashKey") != col("tgt.HashKey")) )
    .select("src.*")
)
display(df_src)
```

(2) Spark Jobs

Update and Insert Condition:

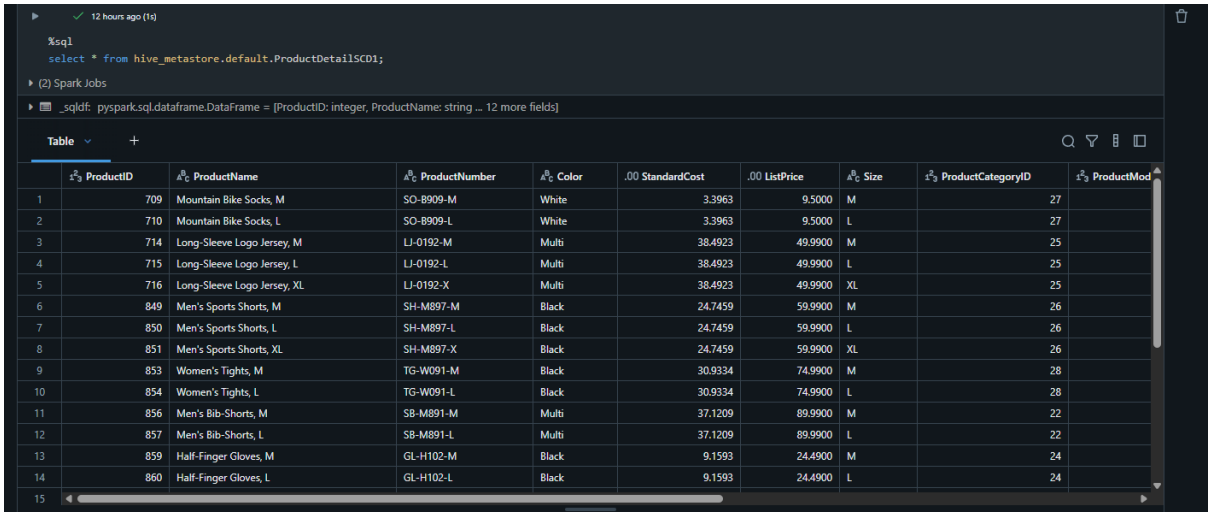
```
deltaTable.alias("tgt").merge(
  df_src.alias("src"), "tgt.ProductID = src.ProductID"
).whenMatchedUpdate(
  set={
    "tgt.ProductID": "src.ProductID",
    "tgt.ProductName": "src.ProductName",
    "tgt.ProductNumber": "src.ProductNumber",
    "tgt.Color": "src.Color",
    "tgt.StandardCost": "src.StandardCost",
    "tgt.ListPrice": "src.ListPrice",
    "tgt.Size": "src.Size",
    "tgt.ProductCategoryID": "src.ProductCategoryID",
    "tgt.ProductModelID": "src.ProductModelID",
```

```
      "tgt.HashKey": "src.HashKey",
      "tgt.UpdatedBy": lit("Harpal-Updated"),
      "tgt.UpdatedDate": current_timestamp(),
    }
).whenNotMatchedInsert(
    values={
      "tgt.ProductID": "src.ProductID",
      "tgt.ProductName": "src.ProductName",
      "tgt.ProductNumber": "src.ProductNumber",
      "tgt.Color": "src.Color",
      "tgt.StandardCost": "src.StandardCost",
      "tgt.ListPrice": "src.ListPrice",
      "tgt.Size": "src.Size",
      "tgt.ProductCategoryID": "src.ProductCategoryID",
      "tgt.ProductModelID": "src.ProductModelID",
      "tgt.HashKey": "src.HashKey",
      "tgt.CreatedBy": lit("Harpal"),
      "tgt.CreatedDate": current_timestamp(),
      "tgt.UpdatedBy": lit("Harpal"),
      "tgt.UpdatedDate": current_timestamp(),
    }
).execute()
```

```
deltaTable.alias("tgt").merge(
    df_src.alias("src"), "tgt.ProductID = src.ProductID"
).whenMatchedUpdate(
    set={
        "tgt.ProductID": "src.ProductID",
        "tgt.ProductName": "src.ProductName",
        "tgt.ProductNumber": "src.ProductNumber",
        "tgt.Color": "src.Color",
        "tgt.StandardCost": "src.StandardCost",
        "tgt.ListPrice": "src.ListPrice",
        "tgt.Size": "src.Size",
        "tgt.ProductCategoryID": "src.ProductCategoryID",
        "tgt.ProductModelID": "src.ProductModelID",
        "tgt.HashKey": "src.HashKey",
        "tgt.UpdatedBy": lit("Harpal-Updated"),
        "tgt.UpdatedDate": current_timestamp(),
    }
).whenNotMatchedInsert(
    values={
        "tgt.ProductID": "src.ProductID",
        "tgt.ProductName": "src.ProductName",
        "tgt.ProductNumber": "src.ProductNumber",
        "tgt.Color": "src.Color",
        "tgt.StandardCost": "src.StandardCost",
        "tgt.ListPrice": "src.ListPrice",
        "tgt.Size": "src.Size",
        "tgt.ProductCategoryID": "src.ProductCategoryID",
        "tgt.ProductModelID": "src.ProductModelID",
        "tgt.HashKey": "src.HashKey",
        "tgt.CreatedBy": lit("Harpal"),
        "tgt.CreatedDate": current_timestamp(),
        "tgt.UpdatedBy": lit("Harpal"),
        "tgt.UpdatedDate": current_timestamp(),
    }
).execute()
```

Check the data into a table:

```
%sql
select * from hive_metastore.default.ProductDetailSCD1;
```



```
%sql
select * from hive_metastore.default.ProductDetailSCD1;
```

(2) Spark Jobs

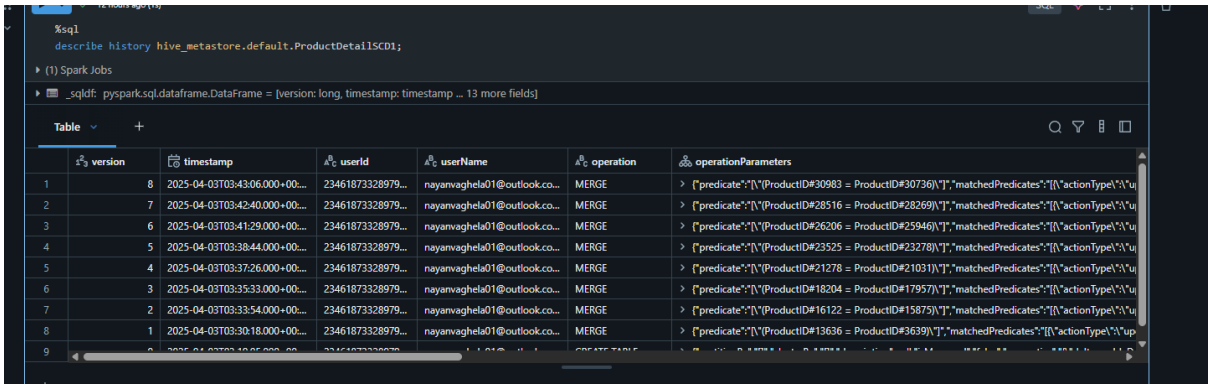
_sqldf: pyspark.sql.dataframe.DataFrame = [ProductID: integer, ProductName: string ... 12 more fields]

Table

	ProductID	ProductName	ProductNumber	Color	StandardCost	ListPrice	Size	ProductCategoryID	ProductMod
1	709	Mountain Bike Socks, M	SO-B909-M	White	3.3963	9.5000	M	27	
2	710	Mountain Bike Socks, L	SO-B909-L	White	3.3963	9.5000	L	27	
3	714	Long-Sleeve Logo Jersey, M	LJ-0192-M	Multi	38.4923	49.9900	M	25	
4	715	Long-Sleeve Logo Jersey, L	LJ-0192-L	Multi	38.4923	49.9900	L	25	
5	716	Long-Sleeve Logo Jersey, XL	LJ-0192-X	Multi	38.4923	49.9900	XL	25	
6	849	Men's Sports Shorts, M	SH-M897-M	Black	24.7459	59.9900	M	26	
7	850	Men's Sports Shorts, L	SH-M897-L	Black	24.7459	59.9900	L	26	
8	851	Men's Sports Shorts, XL	SH-M897-X	Black	24.7459	59.9900	XL	26	
9	853	Women's Tights, M	TG-W091-M	Black	30.9334	74.9900	M	28	
10	854	Women's Tights, L	TG-W091-L	Black	30.9334	74.9900	L	28	
11	856	Men's Bib-Shorts, M	SB-M891-M	Multi	37.1209	89.9900	M	22	
12	857	Men's Bib-Shorts, L	SB-M891-L	Multi	37.1209	89.9900	L	22	
13	859	Half-Finger Gloves, M	GL-H102-M	Black	9.1593	24.4900	M	24	
14	860	Half-Finger Gloves, L	GL-H102-L	Black	9.1593	24.4900	L	24	
15									

Check table history:

```
%sql
describe history hive_metastore.default.ProductDetailSCD1;
```



```
%sql
describe history hive_metastore.default.ProductDetailSCD1;
```

(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [version: long, timestamp: timestamp ... 13 more fields]

Table

	version	timestamp	userId	userName	operation	operationParameters
1	8	2025-04-03T03:43:06.000+00:...	23461873328979...	nayanvaghela01@outlook.co...	MERGE	{"predicate":"[\"(ProductID#30983 = ProductID#30736)\"]","matchedPredicates":"[{\"actionType\":\"u
2	7	2025-04-03T03:42:40.000+00:...	23461873328979...	nayanvaghela01@outlook.co...	MERGE	{"predicate":"[\"(ProductID#28516 = ProductID#28269)\"]","matchedPredicates":"[{\"actionType\":\"u
3	6	2025-04-03T03:41:29.000+00:...	23461873328979...	nayanvaghela01@outlook.co...	MERGE	{"predicate":"[\"(ProductID#26206 = ProductID#25946)\"]","matchedPredicates":"[{\"actionType\":\"u
4	5	2025-04-03T03:38:44.000+00:...	23461873328979...	nayanvaghela01@outlook.co...	MERGE	{"predicate":"[\"(ProductID#23525 = ProductID#23278)\"]","matchedPredicates":"[{\"actionType\":\"u
5	4	2025-04-03T03:37:26.000+00:...	23461873328979...	nayanvaghela01@outlook.co...	MERGE	{"predicate":"[\"(ProductID#21278 = ProductID#21031)\"]","matchedPredicates":"[{\"actionType\":\"u
6	3	2025-04-03T03:35:33.000+00:...	23461873328979...	nayanvaghela01@outlook.co...	MERGE	{"predicate":"[\"(ProductID#18204 = ProductID#17957)\"]","matchedPredicates":"[{\"actionType\":\"u
7	2	2025-04-03T03:33:54.000+00:...	23461873328979...	nayanvaghela01@outlook.co...	MERGE	{"predicate":"[\"(ProductID#16122 = ProductID#15875)\"]","matchedPredicates":"[{\"actionType\":\"u
8	1	2025-04-03T03:30:18.000+00:...	23461873328979...	nayanvaghela01@outlook.co...	MERGE	{"predicate":"[\"(ProductID#13636 = ProductID#3639)\"]","matchedPredicates":"[{\"actionType\":\"up
9						

Conclusion:

In this project, we successfully built a secure and scalable data pipeline using Azure Databricks. We extracted data from the Product table in Azure SQL Database via a secure JDBC connection, leveraging Azure Key Vault and Databricks secret scopes for credential management.

After loading the data into a Spark DataFrame, we performed several transformation and cleaning operations, including removing duplicate records, replacing null values with meaningful defaults, renaming columns to match naming conventions, and applying business logic filters based on attributes like size and color.

The cleaned dataset was then written in Delta format to an ADLS Gen2 storage account through a mounted path. To ensure the pipeline's correctness, we read the Delta data back into a new DataFrame using Spark's read.format("delta") method and successfully verified its structure and content.

Finally, we implemented Slowly Changing Dimension (SCD) Type 1 logic on the ProductDetailSCD1 table to handle data updates, ensuring that the latest product details always overwrite existing records, enabling efficient and up-to-date data tracking in the lakehouse environment.

Points to remember

- Always fetch connection parameters (url, user, password) securely using dbutils.secrets.get() to avoid hardcoding credentials.
- isnotin() doesn't exist in PySpark, use ~col().isin(...) for NOT IN Filters
- Use .option("dbtable", "<tablename>") for entire tables.
- Don’t Use BinaryType Columns with CSV. Don’t use BinaryType Columns with CSV or use Parquet or delta format. Also use .format("delta") for Full Schema + Binary Support
- Delta is perfect for structured data with rich types (like binary, timestamp, decimal)
- Use .option("query", "<SQL query>") when filtering or custom querying (not at once).
- **dbtable** and **query** options are mutually exclusive, so using both will throw an error.
- Use Parquet or Delta for ADLS Gen2 storage unless you're generating human-readable .csv files. It handles schema, data types, and performance much better.
- Use **describe history** for Built-In Table Versioning
- Make sure your Delta table exists before running merge
- Use DeltaTable.forPath() with the correct path (must match the Delta table's LOCATION)