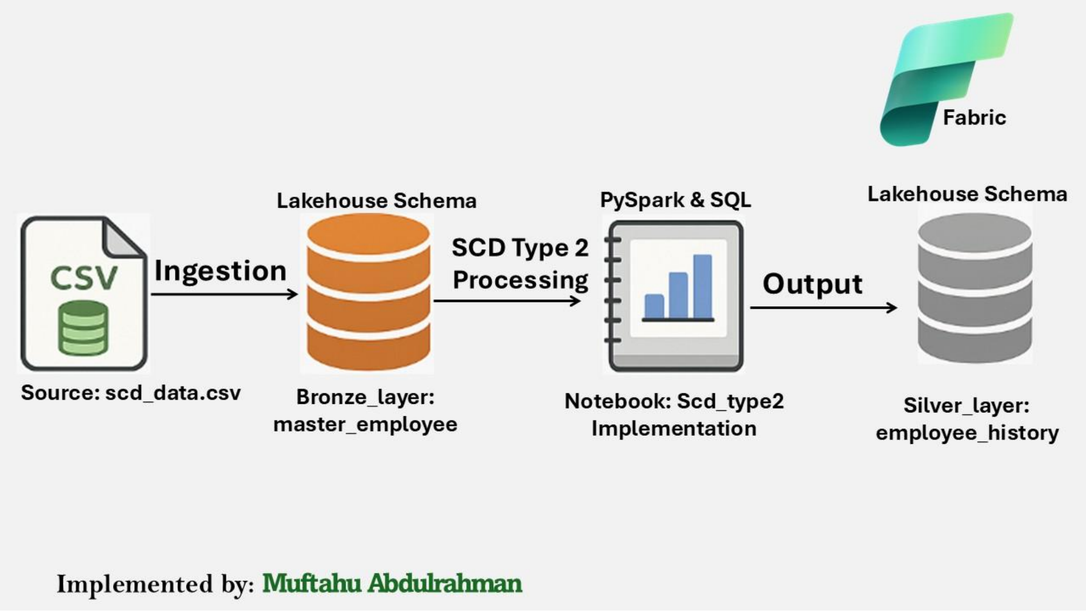
Project Title: Implement SCD Type 2 in Lakehouse for Employee History

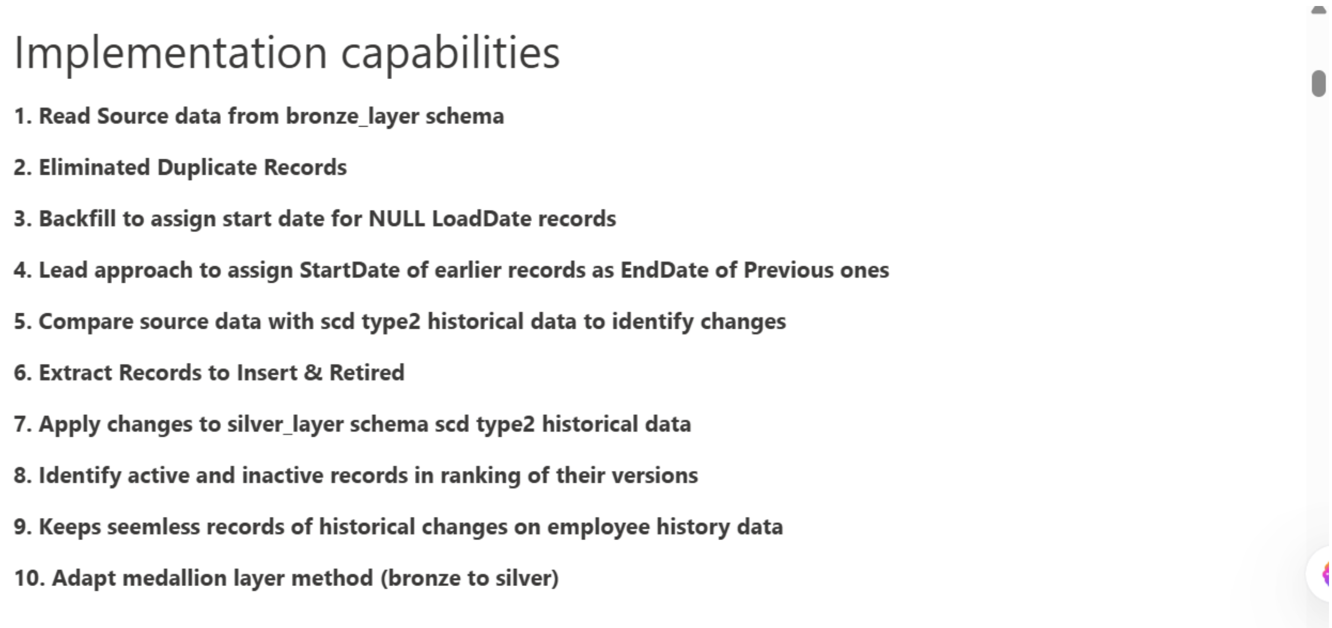
🎯 Objective

Use PySpark in a Microsoft Fabric Lakehouse to implement Slowly Changing Dimension Type 2 (SCD Type 2) to track changes in employee records over time.

USE CASE: Human Resources want to monitor and track the history of employees record changes. This is difficult in current database employee_master table given that the table can accept duplicates, can't keep track of history (what change, when and which record is active for an employee)

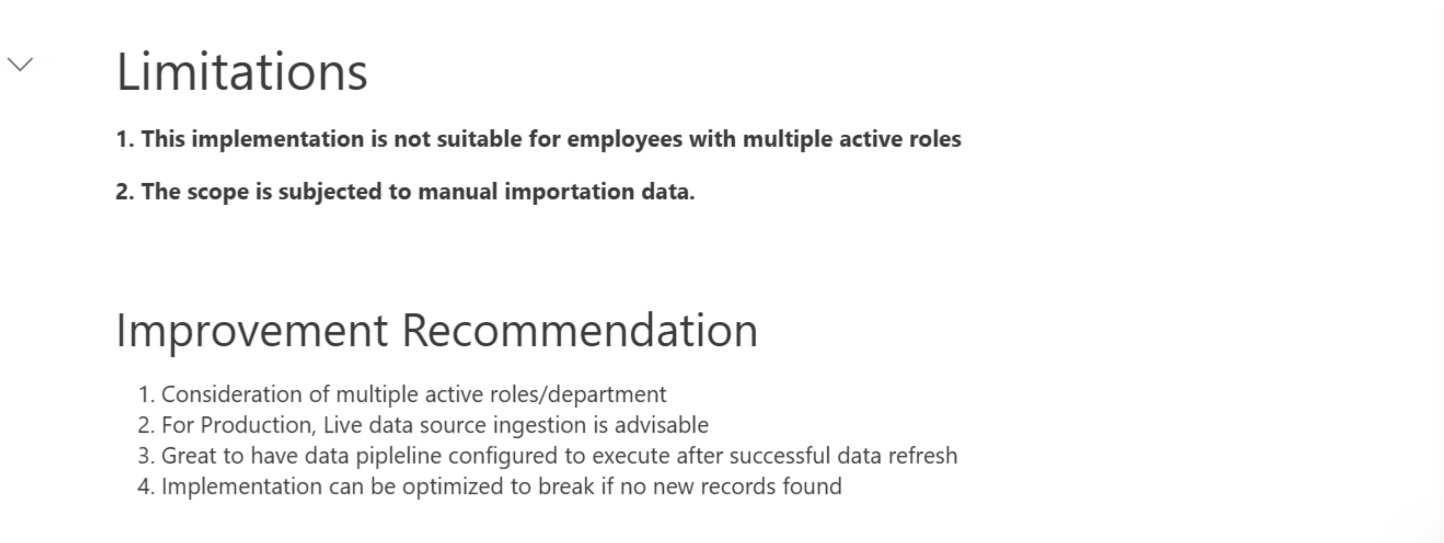
Solution Architecture:


Fabric
Lakehouse Schema
PySpark & SQL
Lakehouse Schema
CSV
Ingestion
SCD Type 2 Processing
Output
Source: scd_data.csv
Bronze_layer: master_employee
Notebook: Scd_type2 Implementation
Silver_layer: employee_history
Implemented by: Muftahu Abdulrahman



Implementation capabilities

1. **Read Source data from bronze_layer schema**
2. **Eliminated Duplicate Records**
3. **Backfill to assign start date for NULL LoadDate records**
4. **Lead approach to assign StartDate of earlier records as EndDate of Previous ones**
5. **Compare source data with scd type2 historical data to identify changes**
6. **Extract Records to Insert & Retired**
7. **Apply changes to silver_layer schema scd type2 historical data**
8. **Identify active and inactive records in ranking of their versions**
9. **Keeps seemless records of historical changes on employee history data**
10. **Adapt medallion layer method (bronze to silver)**

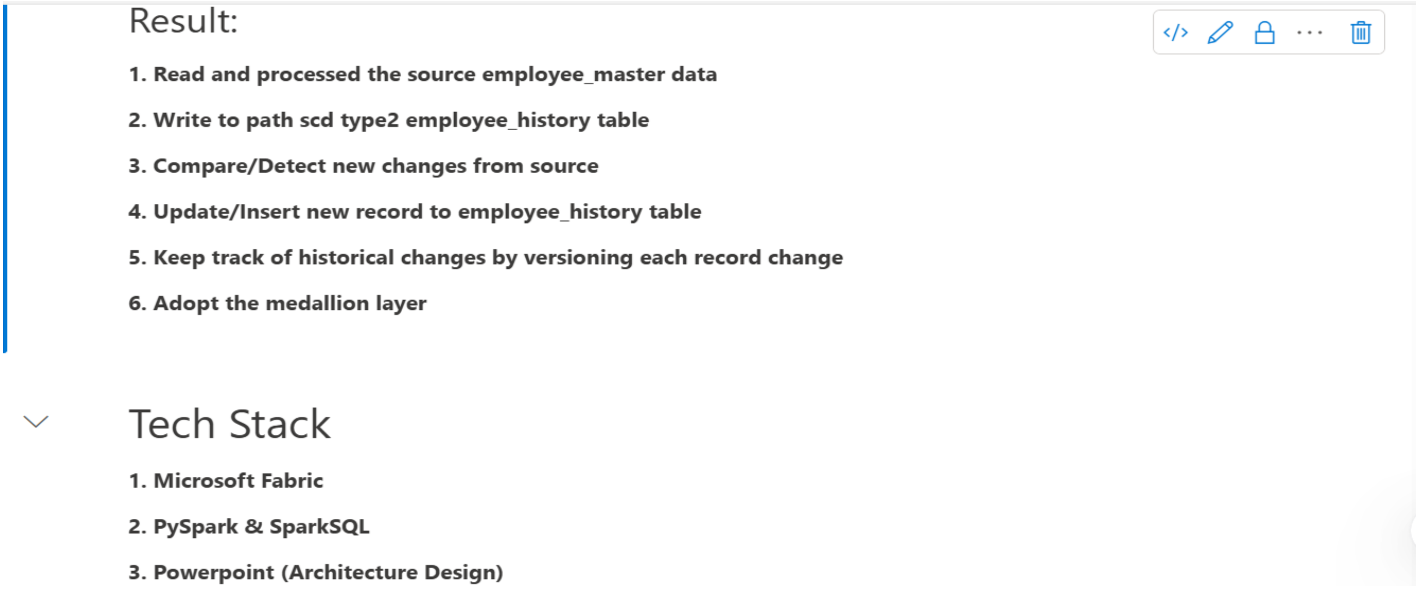


Limitations

1. **This implementation is not suitable for employees with multiple active roles**
2. **The scope is subjected to manual importation data.**

Improvement Recommendation

1. Consideration of multiple active roles/department
2. For Production, Live data source ingestion is advisable
3. Great to have data pipleline configured to execute after successful data refresh
4. Implementation can be optimized to break if no new records found

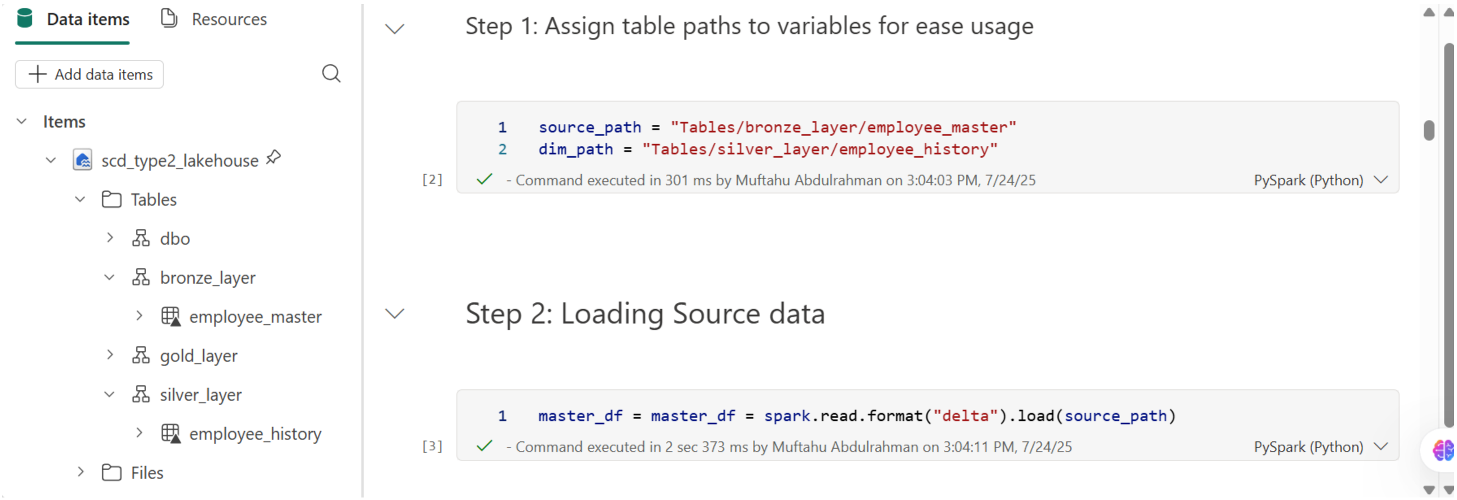


Result:

1. **Read and processed the source employee_master data**
2. **Write to path scd type2 employee_history table**
3. **Compare/Detect new changes from source**
4. **Update/Insert new record to employee_history table**
5. **Keep track of historical changes by versioning each record change**
6. **Adopt the medallion layer**

Tech Stack

1. **Microsoft Fabric**
2. **PySpark & SparkSQL**
3. **Powerpoint (Architecture Design)**



Step 1: Assign table paths to variables for ease usage

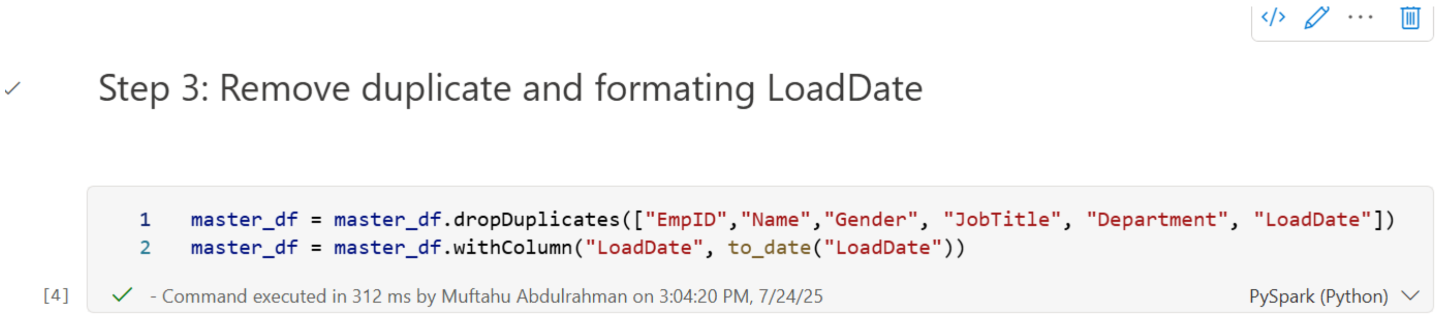
```
source_path = "Tables/bronze_layer/employee_master"
dim_path = "Tables/silver_layer/employee_history"
```

[2] ✓ - Command executed in 301 ms by Muftahu Abdulrahman on 3:04:03 PM, 7/24/25 PySpark (Python)

Step 2: Loading Source data

```
master_df = master_df = spark.read.format("delta").load(source_path)
```

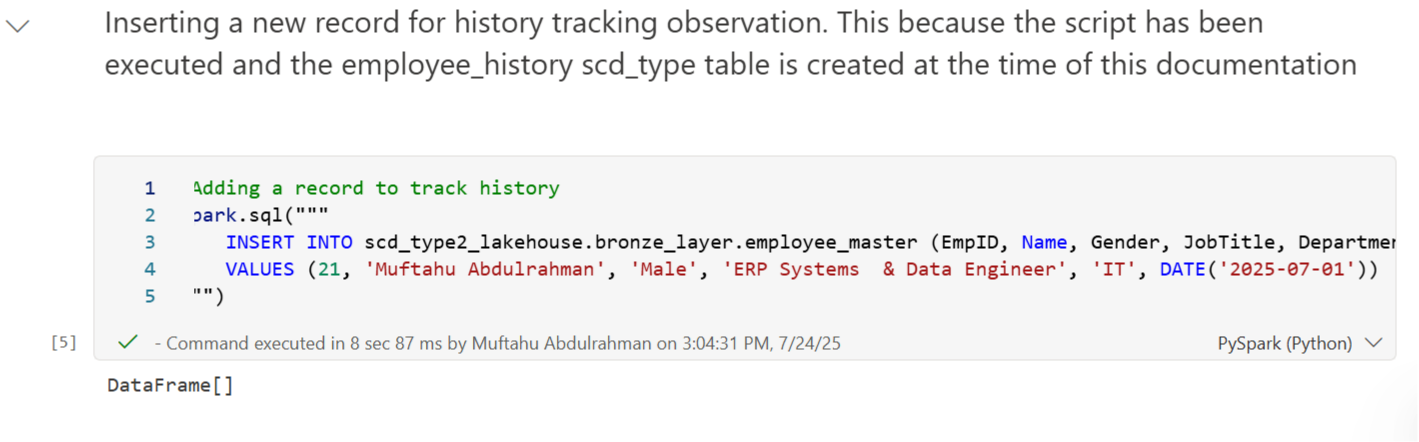
[3] ✓ - Command executed in 2 sec 373 ms by Muftahu Abdulrahman on 3:04:11 PM, 7/24/25 PySpark (Python)



Step 3: Remove duplicate and formating LoadDate

```
master_df = master_df.dropDuplicates(["EmpID","Name","Gender", "JobTitle", "Department", "LoadDate"])
master_df = master_df.withColumn("LoadDate", to_date("LoadDate"))
```

[4] ✓ - Command executed in 312 ms by Muftahu Abdulrahman on 3:04:20 PM, 7/24/25 PySpark (Python)

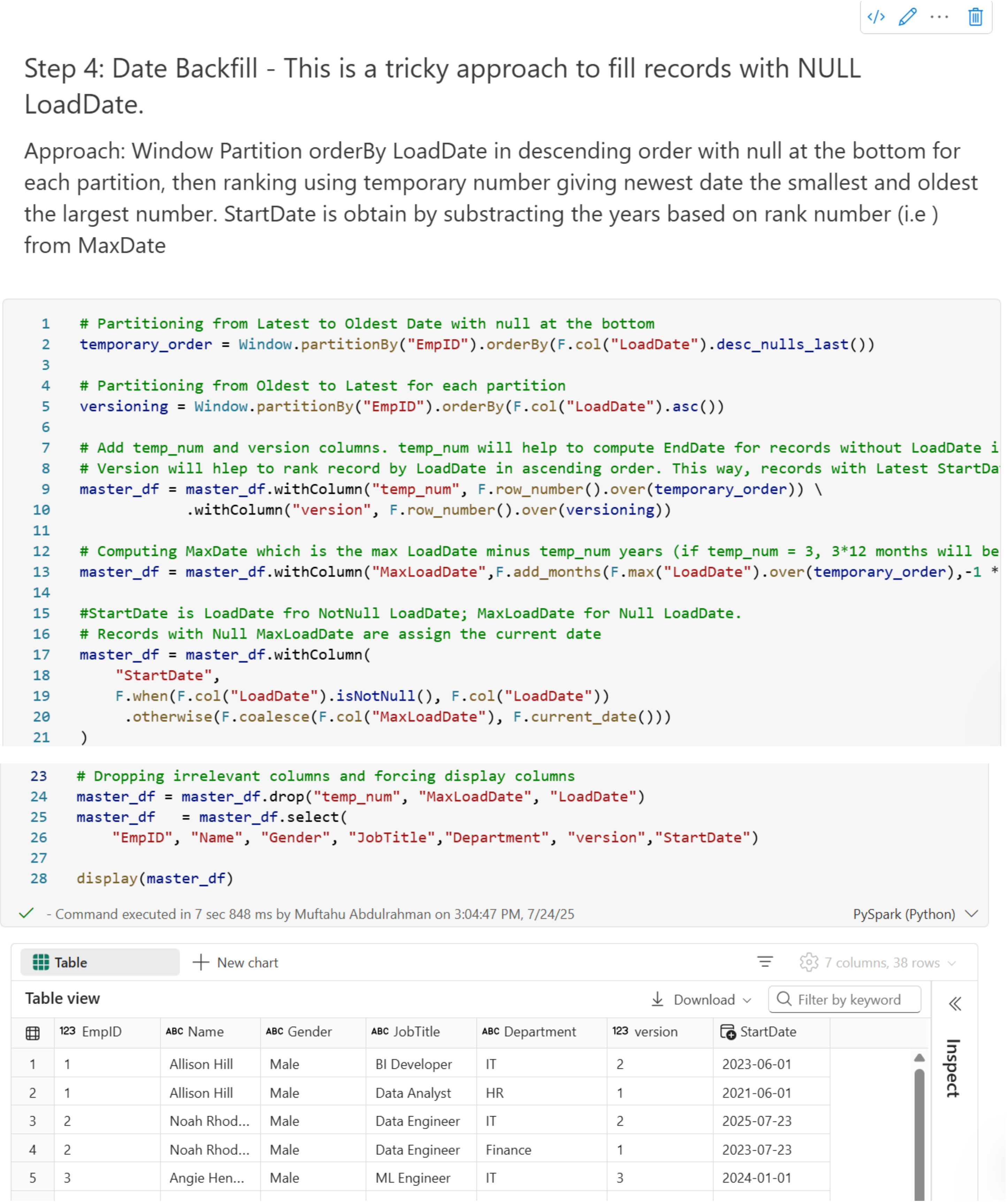


Inserting a new record for history tracking observation. This because the script has been executed and the employee_history scd_type table is created at the time of this documentation

```
Adding a record to track history
park.sql("""
   INSERT INTO scd_type2_lakehouse.bronze_layer.employee_master (EmpID, Name, Gender, JobTitle, Departmen
   VALUES (21, 'Muftahu Abdulrahman', 'Male', 'ERP Systems  & Data Engineer', 'IT', DATE('2025-07-01'))
""")
```

[5] ✓ - Command executed in 8 sec 87 ms by Muftahu Abdulrahman on 3:04:31 PM, 7/24/25 PySpark (Python)

```
DataFrame[]
```



Step 4: Date Backfill - This is a tricky approach to fill records with NULL LoadDate.

Approach: Window Partition orderBy LoadDate in descending order with null at the bottom for each partition, then ranking using temporary number giving newest date the smallest and oldest the largest number. StartDate is obtain by substracting the years based on rank number (i.e) from MaxDate

```python
# Partitioning from Latest to Oldest Date with null at the bottom
temporary_order = Window.partitionBy("EmpID").orderBy(F.col("LoadDate").desc_nulls_last())

# Partitioning from Oldest to Latest for each partition
versioning = Window.partitionBy("EmpID").orderBy(F.col("LoadDate").asc())

# Add temp_num and version columns. temp_num will help to compute EndDate for records without LoadDate i
# Version will hlep to rank record by LoadDate in ascending order. This way, records with Latest StartDa
master_df = master_df.withColumn("temp_num", F.row_number().over(temporary_order)) \
            .withColumn("version", F.row_number().over(versioning))

# Computing MaxDate which is the max LoadDate minus temp_num years (if temp_num = 3, 3*12 months will be
master_df = master_df.withColumn("MaxLoadDate",F.add_months(F.max("LoadDate").over(temporary_order),-1 *

#StartDate is LoadDate fro NotNull LoadDate; MaxLoadDate for Null LoadDate.
# Records with Null MaxLoadDate are assign the current date
master_df = master_df.withColumn(
    "StartDate",
    F.when(F.col("LoadDate").isNotNull(), F.col("LoadDate"))
     .otherwise(F.coalesce(F.col("MaxLoadDate"), F.current_date()))
)
```

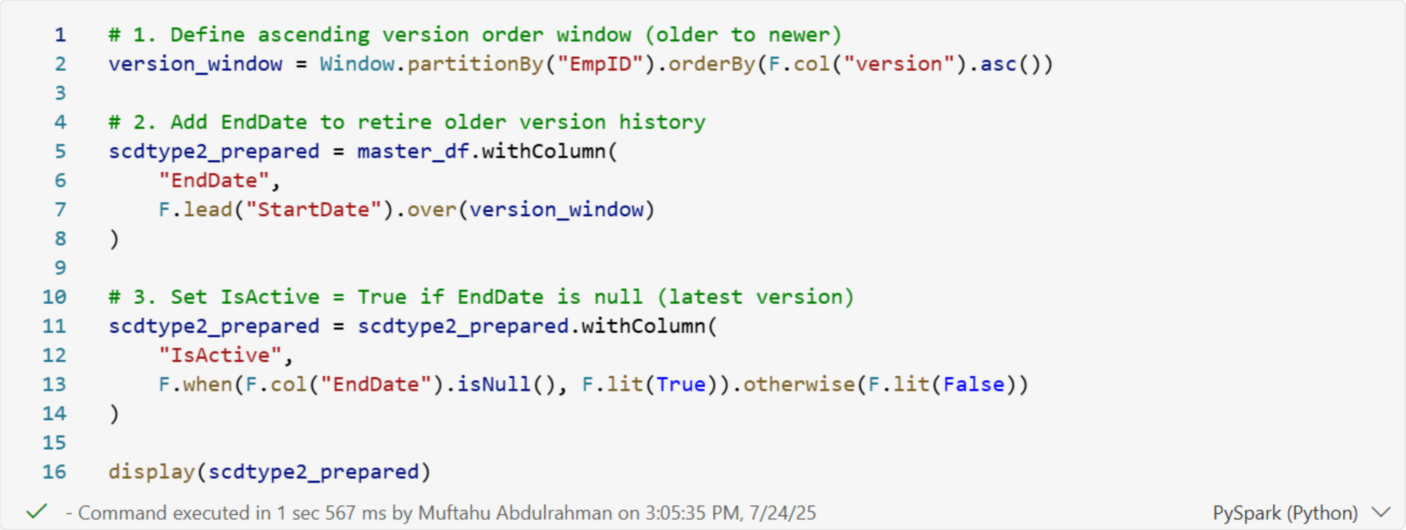
```python
# Dropping irrelevant columns and forcing display columns
master_df = master_df.drop("temp_num", "MaxLoadDate", "LoadDate")
master_df   = master_df.select(
    "EmpID", "Name", "Gender", "JobTitle","Department", "version","StartDate")

display(master_df)
```

✓ - Command executed in 7 sec 848 ms by Muftahu Abdulrahman on 3:04:47 PM, 7/24/25

	123 EmpID	ABC Name	ABC Gender	ABC JobTitle	ABC Department	123 version	StartDate
1	1	Allison Hill	Male	BI Developer	IT	2	2023-06-01
2	1	Allison Hill	Male	Data Analyst	HR	1	2021-06-01
3	2	Noah Rhod...	Male	Data Engineer	IT	2	2025-07-23
4	2	Noah Rhod...	Male	Data Engineer	Finance	1	2023-07-23
5	3	Angie Hen...	Male	ML Engineer	IT	3	2024-01-01

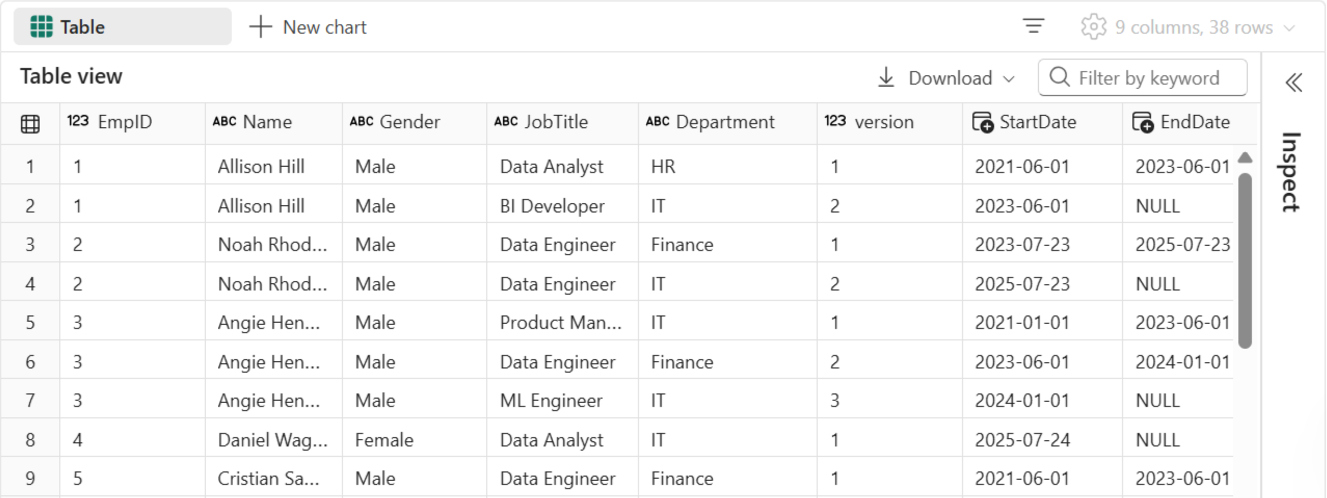
Step 5: Retiring Older versions and Assigning IsActive = True for active version records

Another tricky approach: Window partition to order version in ascending so that the last StartDate of the last row in a partition become the EndDate of the row before it using lead() function



```python
# 1. Define ascending version order window (older to newer)
version_window = Window.partitionBy("EmpID").orderBy(F.col("version").asc())

# 2. Add EndDate to retire older version history
scdtype2_prepared = master_df.withColumn(
    "EndDate",
    F.lead("StartDate").over(version_window)
)

# 3. Set IsActive = True if EndDate is null (latest version)
scdtype2_prepared = scdtype2_prepared.withColumn(
    "IsActive",
    F.when(F.col("EndDate").isNull(), F.lit(True)).otherwise(F.lit(False))
)

display(scdtype2_prepared)
```

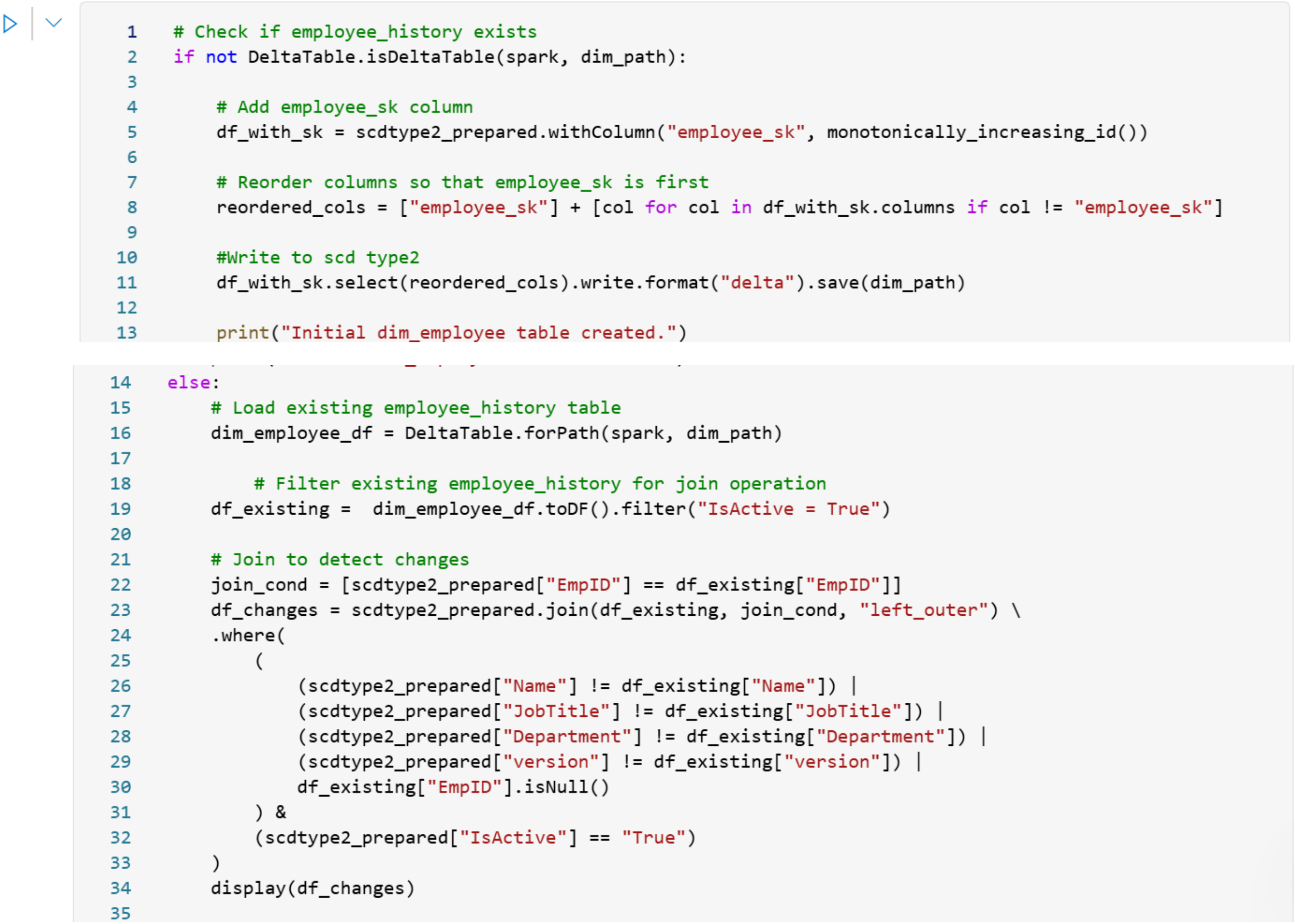
✓ - Command executed in 1 sec 567 ms by Muftahu Abdulrahman on 3:05:35 PM, 7/24/25



	123 EmpID	ABC Name	ABC Gender	ABC JobTitle	ABC Department	123 version	StartDate	EndDate
1	1	Allison Hill	Male	Data Analyst	HR	1	2021-06-01	2023-06-01
2	1	Allison Hill	Male	BI Developer	IT	2	2023-06-01	NULL
3	2	Noah Rhod...	Male	Data Engineer	Finance	1	2023-07-23	2025-07-23
4	2	Noah Rhod...	Male	Data Engineer	IT	2	2025-07-23	NULL
5	3	Angie Hen...	Male	Product Man...	IT	1	2021-01-01	2023-06-01
6	3	Angie Hen...	Male	Data Engineer	Finance	2	2023-06-01	2024-01-01
7	3	Angie Hen...	Male	ML Engineer	IT	3	2024-01-01	NULL
8	4	Daniel Wag...	Female	Data Analyst	IT	1	2025-07-24	NULL
9	5	Cristian Sa...	Male	Data Engineer	Finance	1	2021-06-01	2023-06-01

Step 6: Write processed data to SCD Type 2 employee_history table if not exist

If table exist then read the table and proceed to detect changes.



```python
# Check if employee_history exists
if not DeltaTable.isDeltaTable(spark, dim_path):

    # Add employee_sk column
    df_with_sk = scdtype2_prepared.withColumn("employee_sk", monotonically_increasing_id())

    # Reorder columns so that employee_sk is first
    reordered_cols = ["employee_sk"] + [col for col in df_with_sk.columns if col != "employee_sk"]

    #Write to scd type2
    df_with_sk.select(reordered_cols).write.format("delta").save(dim_path)

    print("Initial dim_employee table created.")
else:
    # Load existing employee_history table
    dim_employee_df = DeltaTable.forPath(spark, dim_path)

        # Filter existing employee_history for join operation
    df_existing =  dim_employee_df.toDF().filter("IsActive = True")

    # Join to detect changes
    join_cond = [scdtype2_prepared["EmpID"] == df_existing["EmpID"]]
    df_changes = scdtype2_prepared.join(df_existing, join_cond, "left_outer") \
    .where(
        (
            (scdtype2_prepared["Name"] != df_existing["Name"]) |
            (scdtype2_prepared["JobTitle"] != df_existing["JobTitle"]) |
            (scdtype2_prepared["Department"] != df_existing["Department"]) |
            (scdtype2_prepared["version"] != df_existing["version"]) |
            df_existing["EmpID"].isNull()
        ) &
        (scdtype2_prepared["IsActive"] == "True")
    )
    display(df_changes)

```

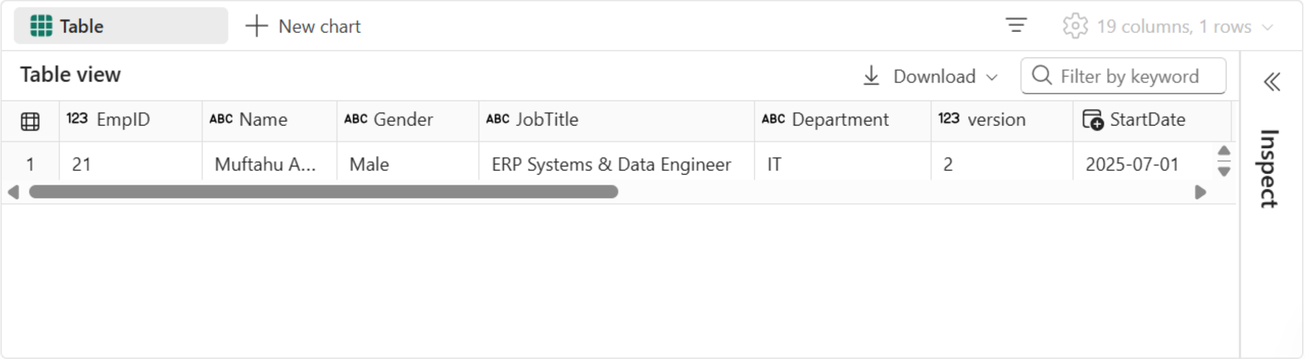
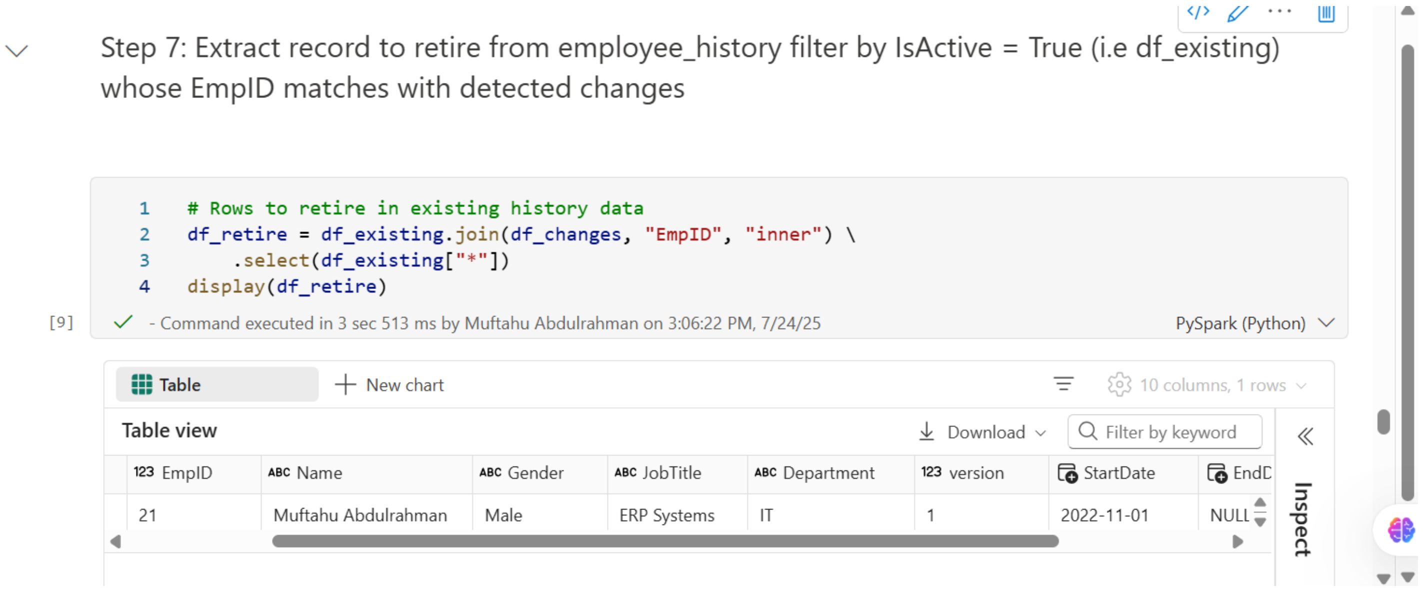


Table view

	123 EmpID	ABC Name	ABC Gender	ABC JobTitle	ABC Department	123 version	StartDate
1	21	Muftahu A...	Male	ERP Systems & Data Engineer	IT	2	2025-07-01

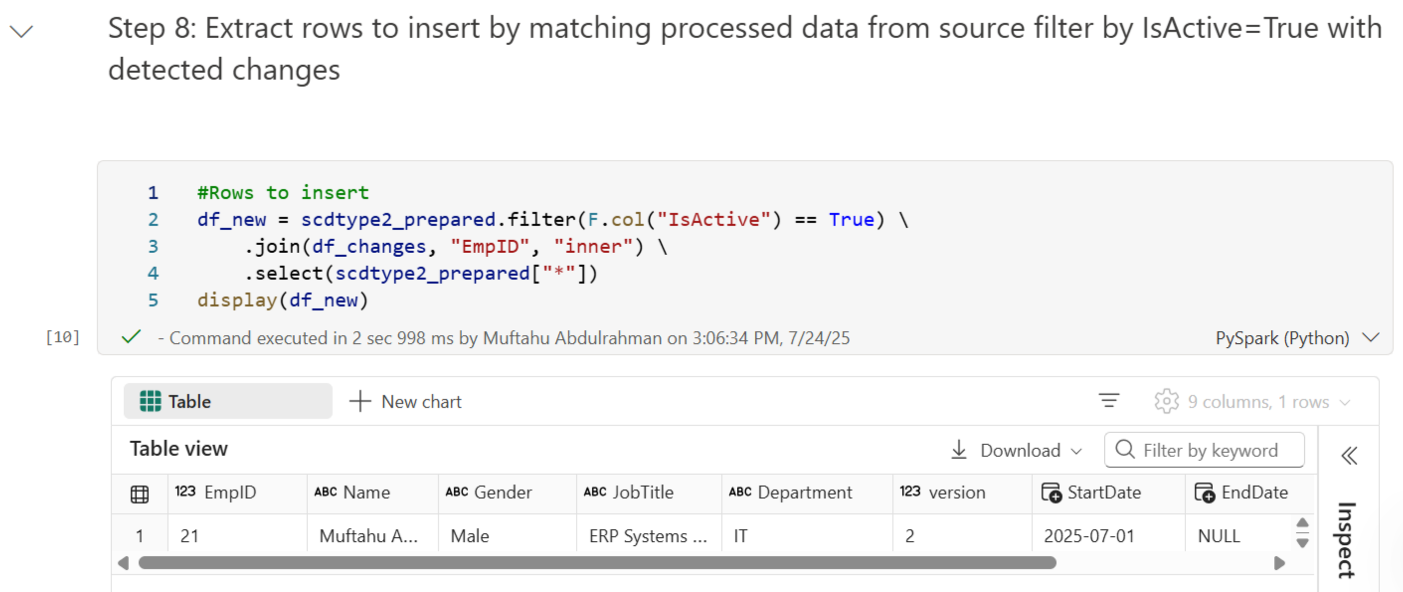


Step 7: Extract record to retire from employee_history filter by IsActive = True (i.e df_existing) whose EmpID matches with detected changes

```python
# Rows to retire in existing history data
df_retire = df_existing.join(df_changes, "EmpID", "inner") \
    .select(df_existing["*"])
display(df_retire)
```

[9] ✓ - Command executed in 3 sec 513 ms by Muftahu Abdulrahman on 3:06:22 PM, 7/24/25

EmpID	Name	Gender	JobTitle	Department	version	StartDate	EndD
21	Muftahu Abdulrahman	Male	ERP Systems	IT	1	2022-11-01	NULL

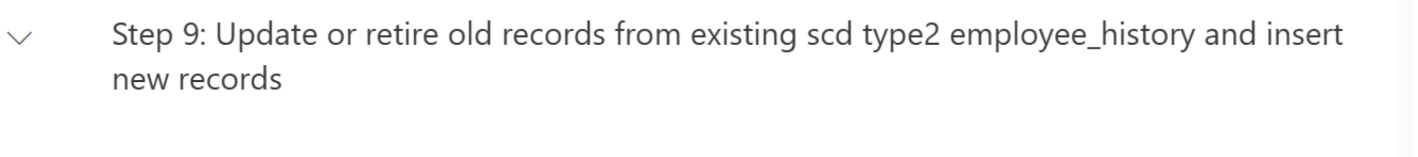


Step 8: Extract rows to insert by matching processed data from source filter by IsActive=True with detected changes

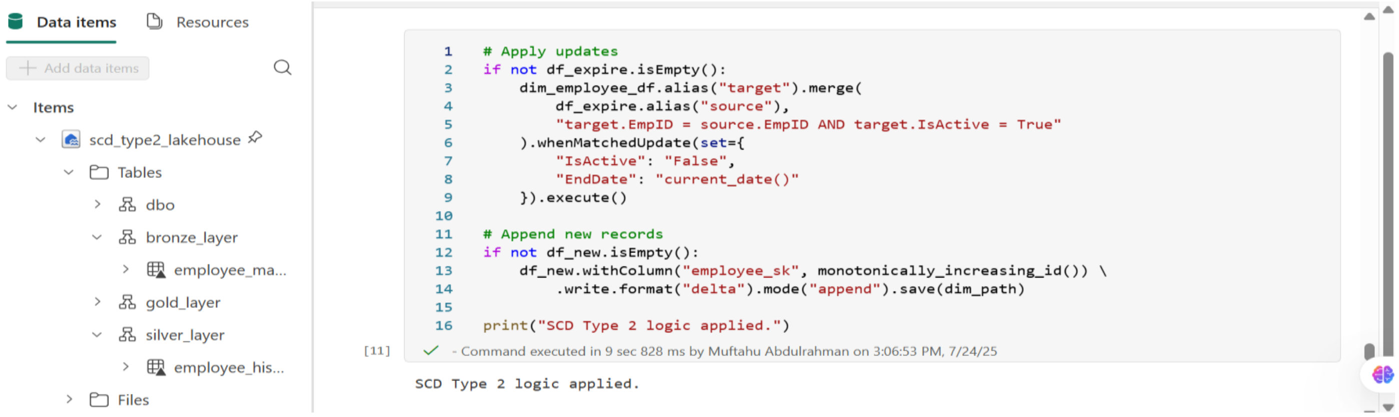
```python
#Rows to insert
df_new = scdtype2_prepared.filter(F.col("IsActive") == True) \
    .join(df_changes, "EmpID", "inner") \
    .select(scdtype2_prepared["*"])
display(df_new)
```

[10] ✓ - Command executed in 2 sec 998 ms by Muftahu Abdulrahman on 3:06:34 PM, 7/24/25

	EmpID	Name	Gender	JobTitle	Department	version	StartDate	EndDate
1	21	Muftahu A...	Male	ERP Systems ...	IT	2	2025-07-01	NULL



Step 9: Update or retire old records from existing scd type2 employee_history and insert new records

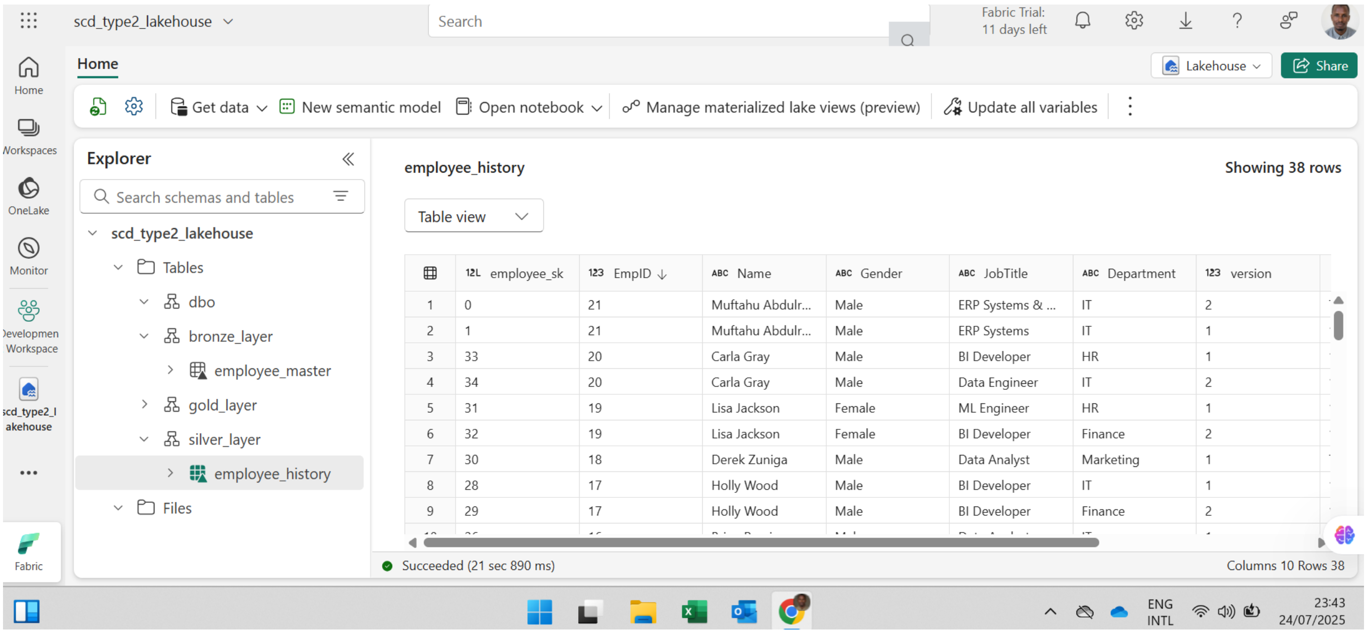


```python
# Apply updates
if not df_expire.isEmpty():
    dim_employee_df.alias("target").merge(
        df_expire.alias("source"),
        "target.EmpID = source.EmpID AND target.IsActive = True"
    ).whenMatchedUpdate(set={
        "IsActive": "False",
        "EndDate": "current_date()"
    }).execute()

# Append new records
if not df_new.isEmpty():
    df_new.withColumn("employee_sk", monotonically_increasing_id()) \
        .write.format("delta").mode("append").save(dim_path)

print("SCD Type 2 logic applied.")
```

[11] ✓ - Command executed in 9 sec 828 ms by Muftahu Abdulrahman on 3:06:53 PM, 7/24/25

```
SCD Type 2 logic applied.
```

Final employee_history table (dim_employee)



employee_history

Showing 38 rows

	employee_sk	EmpID	Name	Gender	JobTitle	Department	version
1	0	21	Muftahu Abdulr...	Male	ERP Systems & ...	IT	2
2	1	21	Muftahu Abdulr...	Male	ERP Systems	IT	1
3	33	20	Carla Gray	Male	BI Developer	HR	1
4	34	20	Carla Gray	Male	Data Engineer	IT	2
5	31	19	Lisa Jackson	Female	ML Engineer	HR	1
6	32	19	Lisa Jackson	Female	BI Developer	Finance	2
7	30	18	Derek Zuniga	Male	Data Analyst	Marketing	1
8	28	17	Holly Wood	Male	BI Developer	IT	1
9	29	17	Holly Wood	Male	BI Developer	Finance	2

Succeeded (21 sec 890 ms)

Columns 10 Rows 38