# SCD Type-1,2 Implementation In Pyspark(by Vijay Bhaskar Reddy)

**SCDs** refer to data in dimension tables that changes slowly over time and not at a regular cadence. A common example for SCDs is customer profiles—for example, an email address or the phone number of a customer doesn't change that often, and these are perfect candidates for SCD

- Here are some of the main aspects we will need to consider while designing an SCD:
  - Should we keep track of the changes? If yes, how much of the history should we maintain
  - Or, should we just overwrite the changes and ignore the history

  Based on our requirements for maintaining the history, there are about seven ways in which we can accomplish keeping track of changes. They are named SCD1, SCD2, SCD3, and so on, up to SCD7

## Designing SCD1

In SCD type 1, the values are overwritten and no history is maintained, so once the data is updated, there is no way to find out what the previous value was. The new queries will always return the most recent value. Here is an example of an SCD1 table:

| CustomerID | Name | City | Email | ... |
|---|---|---|---|---|
| 1 | Adam | New York | adam@.... | ... |

| CustomerID | Name | City | Email | ... |
|---|---|---|---|---|
| 1 | Adam | New Jersey | adam@.... | ... |

## Implementation: SCD-1

## Read the Data

```
# list of employee data
data = [["1001", "gaurav", "hyderabad","42000"],
        ["1002", "vijay", "hyderabad","45565"],
        ["1003", "akanksha","hyderabad", "52000"],
        ["1004", "niharika", "hyderabad","35000"]]

# specify column names
columns = ['id','name','location','salry']

# creating a dataframe from the lists of data
df_full = spark.createDataFrame(data, columns)

# list of employee data
data = [ ["1003", "akanksha","delhi", "65000"],
        ["1004", "niharika", "bihar","10000"],
        ["1005", "murali","vijaywada", "80000"],
        ["1002", "vijay", "hyderabad","45565"]
       ]

# specify column names
columns = ['id','name','location','salry']

# creating a dataframe from the lists of data
df_daily_update = spark.createDataFrame(data, columns)

print("Full data...")
df_full.show()
print("daily data...")
df_daily_update.show()
```

Result:

```
Full data...
+----+--------+---------+-----+
|  id|    name| location|salry|
+----+--------+---------+-----+
|1001|  gaurav|hyderabad|42000|
|1002|   vijay|hyderabad|45565|
|1003|akanksha|hyderabad|52000|
|1004|niharika|hyderabad|35000|
+----+--------+---------+-----+

daily data...
+----+--------+---------+-----+
|  id|    name| location|salry|
+----+--------+---------+-----+
|1003|akanksha|    delhi|65000|
|1004|niharika|    bihar|10000|
|1005|  murali|vijaywada|80000|
|1002|   vijay|hyderabad|45565|
+----+--------+---------+-----+
```

- In the Above we read the data into the dataframe
  1. We have a full data/History data and we need to change the data as use or entity requestd
  2. user data will get i.e Daily data and we need to perform operations on that,

a. If any new records are available, we need to insert.

b. If any records are updated, We need to update the History Table.

In the above we have the data and we read into the dataframes.

## Insert & Update Operation

```
from pyspark.sql.functions import coalesce
res=df_full.join(df_daily_update,"id","full_outer").\
select(coalesce(df_full.id,df_daily_update.id).alias("ID"),\
       coalesce(df_daily_update.name,df_full.name).alias("Name"),\
       coalesce(df_daily_update.location,df_full.location).alias("Location"),\
        coalesce(df_daily_update.salry,df_full.salry).alias("Salary")
       )
res.show()
```

Result:

```
+----+--------+---------+------+
|  ID|    Name| Location|Salary|
+----+--------+---------+------+
|1001|  gaurav|hyderabad| 42000|
|1002|   vijay|hyderabad| 45565|
|1003|akanksha|    delhi| 65000|
|1004|niharika|    bihar| 10000|
|1005|  murali|vijaywada| 80000|
+----+--------+---------+------+
```

---

- We have perform the operation that will insert new records and update the records using pyspark code

# SCD Type-2

Type 2 dimensions are always created as a new record. If a detail in the data changes, a new row will be added to the table with a new primary key. However, the natural key would remain the same in order to map a record change to one another. Type 2 dimensions are the most common approach to tracking historical records.

## SCD 2 Implementation

### Read the data into dataframes.

```
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('sparkdf').getOrCreate()

# list of employee data
data = [["1001", "gaurav", "hyderabad","42000"],
        ["1002", "vijay", "hyderabad","45565"],
        ["1003", "akanksha","hyderabad", "52000"],
        ["1004", "niharika", "hyderabad","35000"]]

# specify column names
columns = ['id','name','location','salry']

# creating a dataframe from the lists of data
df_full = spark.createDataFrame(data, columns)
```

```
# list of employee data
data = [ ["1003", "akanksha","delhi", "65000"],
        ["1004", "niharika", "bihar",None],
        ["1005", "murali","vijaywada", "80000"],
        ["1002", "vijay", "hyderabad","45565"]
       ]

# specify column names
columns = ['id','name','location','salry']

# creating a dataframe from the lists of data
df_daily = spark.createDataFrame(data, columns)
```

```
df_full:
+----+--------+---------+-----+
|  id|    name| location|salry|
+----+--------+---------+-----+
|1001|  gaurav|hyderabad|42000|
|1002|   vijay|hyderabad|45565|
|1003|akanksha|hyderabad|52000|
|1004|niharika|hyderabad|35000|
+----+--------+---------+-----+

df_daily:
+----+--------+---------+-----+
|  id|    name| location|salry|
+----+--------+---------+-----+
|1003|akanksha|    delhi|65000|
|1004|niharika|    bihar| null|
|1005|  murali|vijaywada|80000|
|1002|   vijay|hyderabad|45565|
+----+--------+---------+-----+
```

## Adding Additional Columns in both DataFrames

```
from pyspark.sql.functions import *

df_full=df_full.withColumn("Active_Flag",lit("Y")).withColumn("From_date",\
to_date(current_date()))\
.withColumn("To_date",lit("Null"))
df_full.show()

df_daily=df_daily.withColumn("Active_Flage",lit("Y"))\
.withColumn("From_date",to_date(current_date()))\
.withColumn("To_date",lit("Null"))
```

```
df_full.show()
```

```
+----+--------+---------+-----+-----------+----------+-------+
|  id|    name| location|salry|Active_Flag| From_date|To_date|
+----+--------+---------+-----+-----------+----------+-------+
|1001|  gaurav|hyderabad|42000|          Y|2024-04-26|   Null|
|1002|   vijay|hyderabad|45565|          Y|2024-04-26|   Null|
|1003|akanksha|hyderabad|52000|          Y|2024-04-26|   Null|
|1004|niharika|hyderabad|35000|          Y|2024-04-26|   Null|
+----+--------+---------+-----+-----------+----------+-------+
```

```
df_daily.show()
```

```
+----+--------+---------+-----+------------+----------+-------+
|  id|    name| location|salry|Active_Flage| From_date|To_date|
+----+--------+---------+-----+------------+----------+-------+
|1003|akanksha|    delhi|65000|           Y|2024-04-26|   Null|
|1004|niharika|    bihar| null|           Y|2024-04-26|   Null|
|1005|  murali|vijaywada|80000|           Y|2024-04-26|   Null|
|1002|   vijay|hyderabad|45565|           Y|2024-04-26|   Null|
+----+--------+---------+-----+------------+----------+-------+
```

## Create Dataframe by using Updating the Active Flag If any changes in dataframes using Hash

```
update_ds=df_full.join(df_daily,((df_full.id==df_daily.id) & (df_full.Active_Flag =='Y')
"inner")\
    .filter(hash(df_full.name,df_full.location,df_full.salry ) !=
            hash(df_daily.name,df_daily.location,df_daily.salry  ) )\
                .select(df_full.id,
                        df_full.name,
                        df_full.location,
                        df_full.salry,
                        lit("N").alias("Active_Flag"),
                        df_full.From_date,
                        lit(to_date(current_date())).alias("To_Date"))
update_ds.show()
```

```
+----+--------+---------+-----+-----------+----------+----------+
|  id|    name| location|salry|Active_Flag| From_date|   To_Date|
+----+--------+---------+-----+-----------+----------+----------+
|1003|akanksha|hyderabad|52000|          N|2024-04-26|2024-04-26|
|1004|niharika|hyderabad|35000|          N|2024-04-26|2024-04-26|
+----+--------+---------+-----+-----------+----------+----------+
```

## Create a Data Frame with No changes data using the update_ds dataframe

```
no_change=df_full.join(update_ds,((df_full.id==update_ds.id) & (df_full.Active_Flag =='Y
,"left_anti")
no_change.show()
```

```
+----+------+---------+-----+-----------+----------+-------+
|  id|  name| location|salry|Active_Flag| From_date|To_date|
+----+------+---------+-----+-----------+----------+-------+
|1001|gaurav|hyderabad|42000|          Y|2024-04-26|   Null|
|1002| vijay|hyderabad|45565|          Y|2024-04-26|   Null|
+----+------+---------+-----+-----------+----------+-------+
```

### Create data frame using no_change Dataframe and the dataframe consists of the new records that need to insert

```
insert_ds = df_daily.join(no_change,"id","left_anti")
insert_ds.show()
```

```
+----+--------+---------+-----+------------+----------+-------+
|  id|    name| location|salry|Active_Flage| From_date|To_date|
+----+--------+---------+-----+------------+----------+-------+
|1003|akanksha|    delhi|65000|           Y|2024-04-26|   Null|
|1004|niharika|    bihar| null|           Y|2024-04-26|   Null|
|1005|  murali|vijaywada|80000|           Y|2024-04-26|   Null|
+----+--------+---------+-----+------------+----------+-------+
```

### Finally We have to concatenate update_ds , Insert_ds, no_change dataframe by using the union function.

```
df_final=update_ds.union(insert_ds).union(no_change)
df_final.show()
```

```
+----+--------+---------+-----+-----------+----------+----------+
|  id|    name| location|salry|Active_Flag| From_date|   To_Date|
+----+--------+---------+-----+-----------+----------+----------+
|1003|akanksha|hyderabad|52000|          N|2024-04-26|2024-04-26|
|1004|niharika|hyderabad|35000|          N|2024-04-26|2024-04-26|
|1003|akanksha|    delhi|65000|          Y|2024-04-26|      Null|
|1004|niharika|    bihar| null|          Y|2024-04-26|      Null|
|1005|  murali|vijaywada|80000|          Y|2024-04-26|      Null|
|1001|  gaurav|hyderabad|42000|          Y|2024-04-26|      Null|
|1002|   vijay|hyderabad|45565|          Y|2024-04-26|      Null|
+----+--------+---------+-----+-----------+----------+----------+
```

Finally we have achieved the records which have the new updates and new entries.