



Spark Type 2 and type 6 implementation

By Abdelrahman Omar



Content

Introduction.....	3
Authenticate Kerberos.....	6
Developing SCD type 6 Script.....	8
Type 6 Logic Diagram.....	11
Developing SCD type 2 Script.....	11
Type 2 Logic Diagram.....	13



- [Introduction](#)

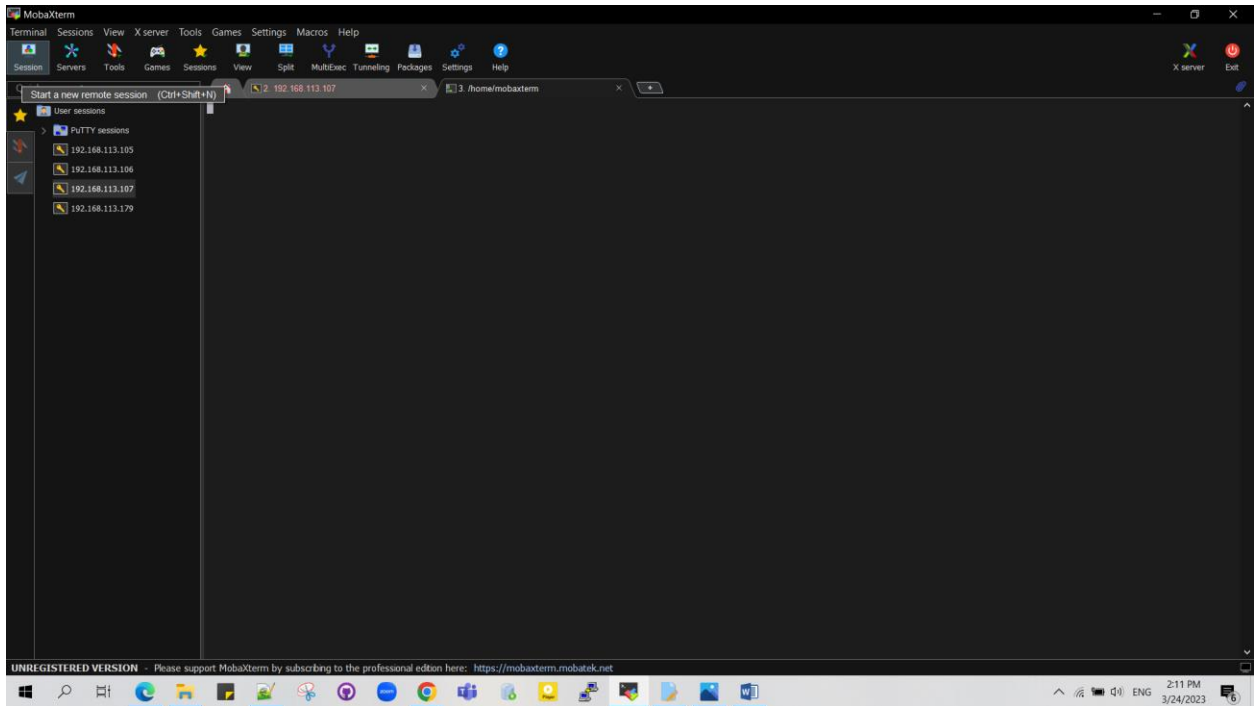
Slowly Changing Dimension (SCD) Type 6. This was an exciting project that allowed me to apply my knowledge of big data and Spark to a real-world problem.

SCD Type 6 is a complex data warehousing concept that tracks changes to data over time. In this type of dimension, the system keeps track of all the changes made to a record, even after it has been updated or deleted. This is useful in scenarios where historical data is important, such as in financial reporting or compliance.

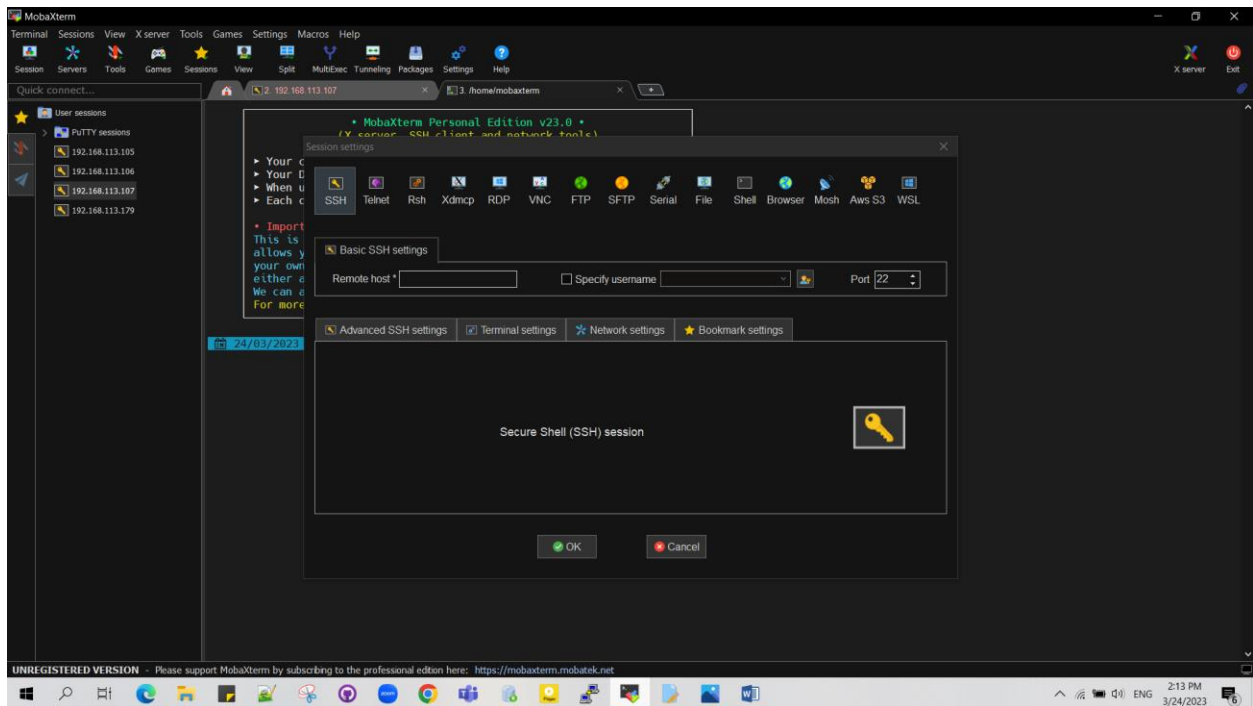
To implement SCD Type 6 in Spark, I had to first identify the fields in the data that needed to be tracked for changes. I then designed a schema to store the historical data on a hive table, created a temporary table to hold the current data, and used Spark Python Dataframes to compare the two tables and identify any changes.

[Connect to VM](#)

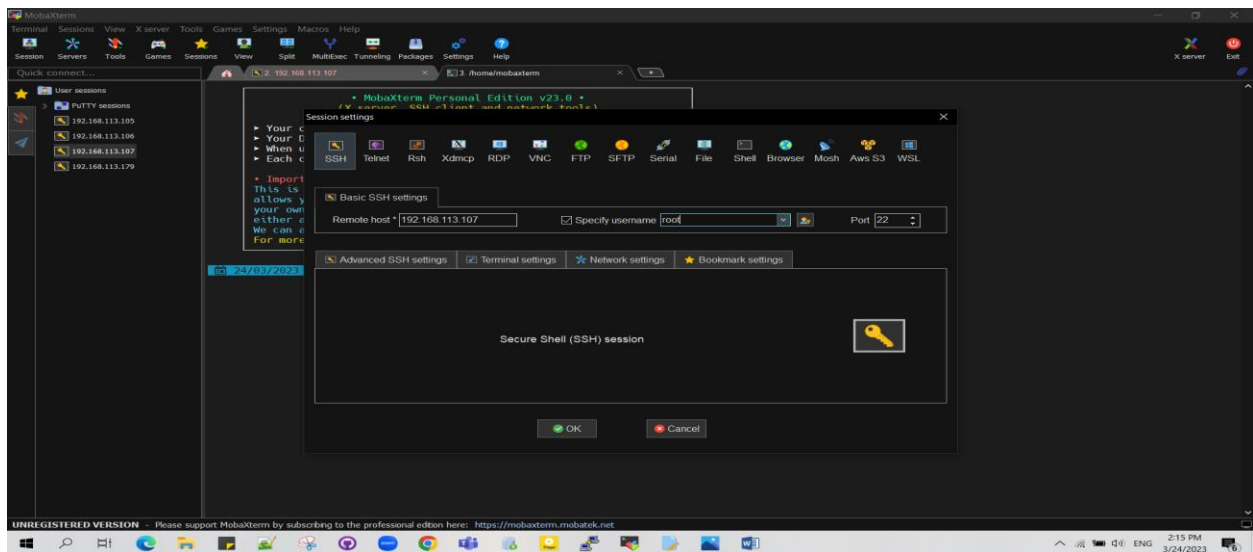
1. Open MobaXterm on your local machine.
2. Click on the "Session" button in the top left corner of the window.



3. In the "Session Settings" window, select "SSH" as the protocol.



4. Enter the IP address of the VM in the "Remote host" field.



5. Double-click on the session in the "Sessions" tab to connect to the VM.



If the connection is successful, you will see a terminal window with a command prompt for the VM.

The screenshot displays the MobaXterm Personal Edition v23.0 interface. The main terminal window shows the results of an SSH session to root@192.168.113.107. A status box indicates that the connection was successful, with Direct SSH, SSH compression, and SSH-browser all checked, while X11-forwarding is disabled. Below this, the terminal output shows the last login time (Fri Mar 24 13:25:43 2023) and a warning from SLF4J about multiple bindings in the classpath. The prompt is [root@bbi-gateway ~]#.

```
• MobaXterm Personal Edition v23.0 •
(SSH client, X server and network tools)

▶ SSH session to root@192.168.113.107
  • Direct SSH : ✓
  • SSH compression : ✓
  • SSH-browser : ✓
  • X11-forwarding : ✗ (disabled or not supported by server)

▶ For more info, ctrl+click on help or visit our website.

Last login: Fri Mar 24 13:25:43 2023 from 10.253.252.5
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/cloudera/parcels/CDH-7.1.8-1.cdh7.1.8.p0.30998532/jars/log4j-slf4j-impl-2.17.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/cloudera/parcels/CDH-7.1.8-1.cdh7.1.8.p0.30998532/jars/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
^C
[root@bbi-gateway ~]#
```

At the bottom of the window, a status bar shows system information: 6.23 GB / 31.24 GB memory usage, 5.00 MB/s network speed, and disk usage for various partitions (root: 36%, /opt: 79%, /logs: 1%, /boot: 12%, /home: 1%, /var: 24%). The taskbar at the very bottom shows the Windows taskbar with the time 2:19 PM on 3/24/2023.



Authenticate Kerberos

Kerberos is a widely-used authentication protocol that can be used to secure distributed computing environments, such as Apache Hadoop and its related projects, including Spark and Hive. Kerberos provides a way for users to securely authenticate with a distributed system, and to access resources on that system based on their authenticated identity.

In the context of Spark and Hive, Kerberos authentication can be used to secure access to sensitive data and ensure that only authorized users are able to perform certain actions. For example, Kerberos can be used to authenticate users who want to access a Hive table or run a Spark job, and to ensure that only authorized users can read or modify that data.

To use Kerberos authentication with Spark and Hive, several configuration steps are required. These steps typically involve configuring Kerberos on the cluster, configuring Kerberos authentication for the Spark and Hive services, and configuring user accounts to use Kerberos authentication. Additionally, users may need to obtain Kerberos tickets and set up the appropriate environment variables in order to authenticate with the cluster.

Overall, Kerberos authentication provides an important layer of security for distributed computing environments like Spark and Hive, helping to ensure that sensitive data is only accessible by authorized users and reducing the risk of data breaches and unauthorized access.

1. Identify the Spark service principal name (SPN) that you want to use for Kerberos authentication. The SPN typically takes the form of "spark/[hostname]@REALM".



[spark/bbi-gateway.test.local@TEST.LOCAL](#)

2. Generate a Kerberos keytab file for the SPN using the kadmin command. For example, the command "kadmin -q 'addprinc -randkey spark/[hostname]@REALM'" will generate a keytab file for the "spark/[hostname]@REALM" principal.

Kadmin -g 'addprinc -randkey [spark/bbi-gateway.test.local@TEST.LOCAL](#)'

3. Save the keytab file in a secure location on the Spark server.

4. Configure the Spark service to use Kerberos authentication by setting the "spark.authenticate" configuration property to "true" in the Spark configuration file.

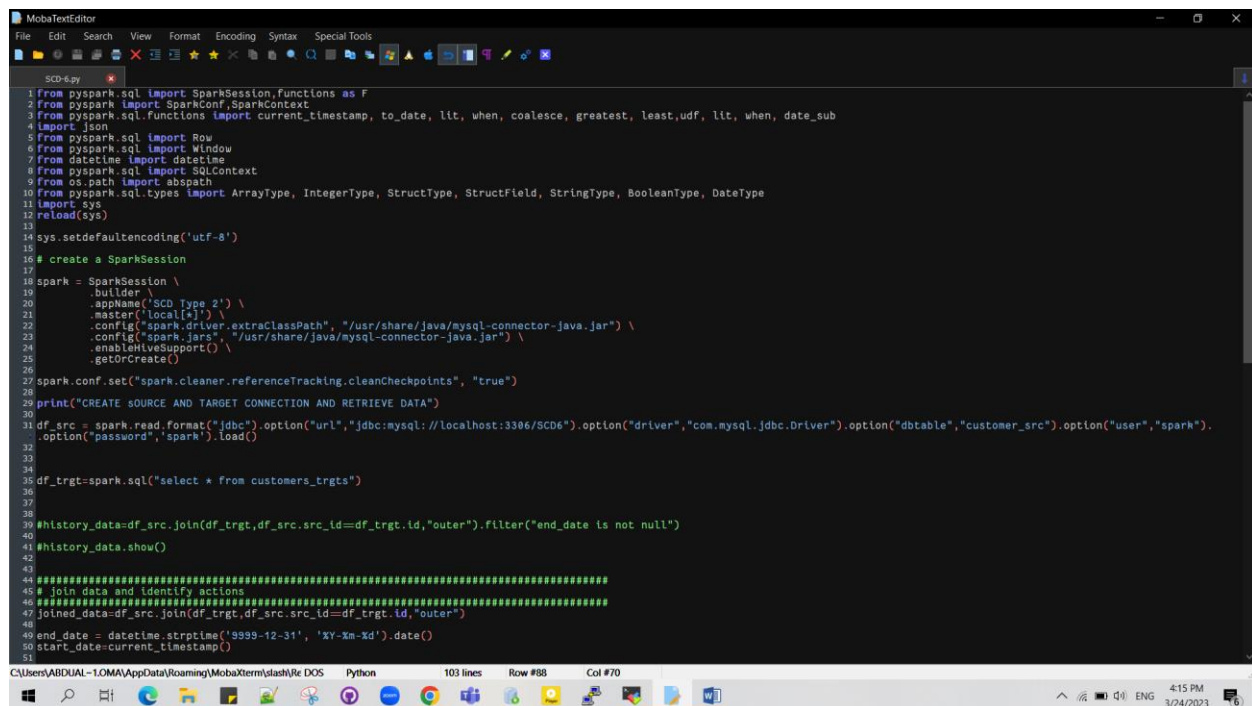
```
kinit -kt /var/run/cloudera-scm-agent/process/738-spark_on_yarn-  
SPARK_YARN_HISTORY_SERVER/spark_on_yarn.keytab spark/bbi-  
gateway.test.local@TEST.LOCAL
```

```
[root@bbi-gateway 910-spark_on_yarn-SPARK_YARN_HISTORY_SERVER]# pwd  
/var/run/cloudera-scm-agent/process/910-spark_on_yarn-SPARK_YARN_HISTORY_SERVER  
[root@bbi-gateway 910-spark_on_yarn-SPARK_YARN_HISTORY_SERVER]# ls -la  
bash: ls: command not found  
[root@bbi-gateway 910-spark_on_yarn-SPARK_YARN_HISTORY_SERVER]# kinit -kt /var/run/cloudera-scm-agent/process/910-spark_on_yarn-SPARK_YARN_HISTORY_SERVER/spark_on_yarn.keytab spark/bbi-gateway.test.local@TEST.LOCAL  
kinit: Client 'spark_on_yarn.keytab@TEST.LOCAL' not found in Kerberos database while getting initial credentials  
[root@bbi-gateway 910-spark_on_yarn-SPARK_YARN_HISTORY_SERVER]# clear  
[root@bbi-gateway 910-spark_on_yarn-SPARK_YARN_HISTORY_SERVER]# spark-submit /root/test.py  
2/3/2023 13:17:00 INFO spark.SparkContext: Running Spark version 2.4.0-7.1.0-0-901  
2/3/2023 13:17:00 INFO LoggingDriverLogger: Added a local log appender at: /tmp/spark-94dad1e2-2b08-4b22-be7d-509f0745666f/_driver_logs/_driver.log
```




Developing SCD type 6 Script

1. Open mobaxtreme to write the python script



```
SCD6.py
1 from pyspark.sql import SparkSession, functions as F
2 from pyspark import SparkConf, SparkContext
3 from pyspark.sql.functions import current_timestamp, to_date, lit, when, coalesce, greatest, least, udf, lit, when, date_sub
4 import json
5 from pyspark.sql import Row
6 from pyspark.sql import Window
7 from datetime import datetime
8 from pyspark.sql import SQLContext
9 from os.path import abspath
10 from pyspark.sql.types import ArrayType, IntegerType, StructType, StructField, StringType, BooleanType, DateType
11 import sys
12 reload(sys)
13
14 sys.setdefaultencoding('utf-8')
15
16 # create a SparkSession
17
18 spark = SparkSession \
19     .builder \
20     .appName('SCD Type 2') \
21     .master('local[*]') \
22     .config("spark.driver.extraClassPath", "/usr/share/java/mysql-connector-java.jar") \
23     .config("spark.jars", "/usr/share/java/mysql-connector-java.jar") \
24     .enableHiveSupport() \
25     .getOrCreate()
26
27 spark.conf.set("spark.cleaner.referenceTracking.cleanCheckpoints", "true")
28
29 print("CREATE SOURCE AND TARGET CONNECTION AND RETRIEVE DATA")
30
31 df_src = spark.read.format("jdbc").option("url", "jdbc:mysql://localhost:3306/SCD6").option("driver", "com.mysql.jdbc.Driver").option("dbtable", "customer_src").option("user", "spark").\
32     .option("password", "spark").load()
33
34
35 df_trgt=spark.sql("select * from customers_trgts")
36
37
38
39 #history_data=df_src.join(df_trgt,df_src.src_id==df_trgt.id,"outer").filter("end_date is not null")
40
41 #history_data.show()
42
43
44 #####
45 # join data and identify actions
46 #####
47 joined_data=df_src.join(df_trgt,df_src.src_id==df_trgt.id,"outer")
48
49 end_date = datetime.strptime('9999-12-31', '%Y-%m-%d').date()
50 start_date=current_timestamp()
51
```

2. First thing before start write the ETL logic we must import all libraries we will use in our program

```
from pyspark.sql import SparkSession, functions as F
from pyspark import SparkConf, SparkContext
from pyspark.sql.functions import current_timestamp, to_date, lit, when, coalesce, greatest, least, udf, lit, when, date_sub
import json
from pyspark.sql import Row
from pyspark.sql import Window
from datetime import datetime
from pyspark.sql import SQLContext
from os.path import abspath
from pyspark.sql.types import ArrayType, IntegerType, StructType, StructField, StringType, BooleanType, DateType
import sys
reload(sys)

sys.setdefaultencoding('utf-8')
```



3. Create spark session with DB connector and hive context to connect to MySQL and hive table.

```
# create a SparkSession

spark = SparkSession \
    .builder \
    .appName('SCD Type 2') \
    .master('local[*]') \
    .config("spark.driver.extraClassPath", "/usr/share/java/mysql-connector-java.jar") \
    .config("spark.jars", "/usr/share/java/mysql-connector-java.jar") \
    .enableHiveSupport() \
    .getOrCreate()

spark.conf.set("spark.cleaner.referenceTracking.cleanCheckpoints", "true")
```

4. Start implement Type 6 logic

```
println("CREATE SOURCE AND TARGET CONNECTION AND RETRIEVE DATA")

df_src = spark.read.format("jdbc").option("url", "jdbc:mysql://localhost:3306/SCD6").option("driver", "com.mysql.jdbc.Driver").option("dbtable", "customer_src").option("user", "spark").option("password", "spark").load()

df_trgt=spark.sql("select * from customers_trgts")

#history_data=df_src.join(df_trgt,df_src.src_id=df_trgt.id,"outer").filter("end_date is not null")
#history_data.show()

#####
# join data and identify actions
#####
joined_data=df_src.join(df_trgt,df_src.src_id=df_trgt.id,"outer")

end_date = datetime.strptime('9999-12-31', '%Y-%m-%d').date()
start_date=current_timestamp()

joined_data = joined_data.withColumn('action',
    when(joined_data.key.isNull(), 'I')
    .when((joined_data.src_address != joined_data.current_address) & (joined_data.end_date = '9999-12-31'), 'EI')
    .when((joined_data.src_address != joined_data.current_address) & (joined_data.end_date != '9999-12-31'), 'U')
    .otherwise('no_action')
)

joined_data.show()

df_insert=joined_data.filter(joined_data.action == "I").select(joined_data.src_id.alias("id"),joined_data.src_name.alias("name"),joined_data.src_address.alias("current_address"),
    joined_data.src_address.alias("prev_address")).withColumn("start_date",lit(start_date)).withColumn("end_date",lit(end_date))

df_ext_insert=joined_data.filter(joined_data.action == "EI").select(joined_data.src_id.alias("id"),joined_data.src_name.alias("name"),joined_data.src_address.alias("current_address"),
    joined_data.src_address.alias("prev_address")).withColumn("start_date",lit(start_date)).withColumn("end_date",lit(end_date))

df_ext_update=joined_data.filter(joined_data.action == "EI").select(joined_data.id,joined_data.name,joined_data.src_address.alias("current_address"),joined_data.prev_address,joined_data.start_date,joined_data.start_date.alias("end_date"))

df_update=joined_data.filter(joined_data.action == "U").select(joined_data.id,joined_data.name,joined_data.src_address.alias("start_date"),joined_data.prev_address,joined_data.start_date,joined_data.end_date)

df_no_action=joined_data.filter(joined_data.action == "no_action").select(joined_data.id,joined_data.name,joined_data.current_address,joined_data.prev_address,joined_data.start_date,joined_data.end_date)
```

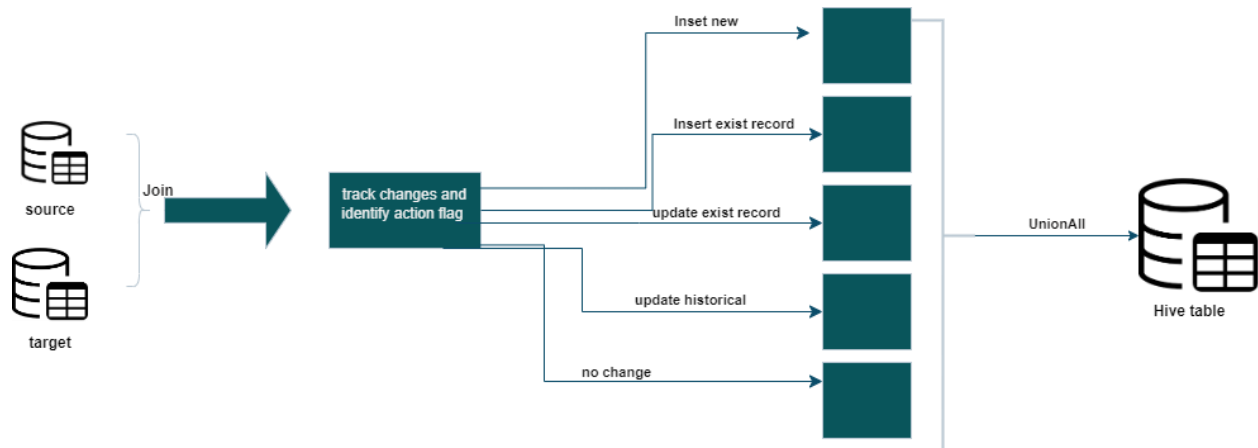
5. Submit the script and start the ETL Job using “spark-submit” command

```
^C
[root@bbi-gateway ~]# spark-submit --jars /usr/share/java/mysql-connector-java.jar SCD-6.py
```

When submitting the script we must specify the directory of the MySQL connector using
The argument --jars



Type 6 diagram



Developing Type 2 script

For developing Type 2 script it is the same steps as developing type 6 and this is the logic
For the script



```
MobaTextEditor
File Edit Search View Format Encoding Syntax Special Tools
SCD-6.py SCD-2.py
56
57
58 #
59 df_src.select("name").show()
60
61
62
63 history_data=spark.sql("select * from customer_trgt where end_date is not null").distinct().select("id","name","prev_address","start_date","end_date")
64
65 history_data.show()
66 spark.stop()
67
68 joined_data=df_src.join(df_trgt,df_src.id==df_trgt.id,"outer").filter("end_date is null")
69
70
71 change_type= when(joined_data["key"].isNull(),"I").when(joined_data["address"] != joined_data["prev_address"],"IU").otherwise("no_change")
72
73
74 joined_data = joined_data.withColumn("change_type",change_type)
75
76
77
78 #x = spark.sql("select * from customer_src")
79 start_date=when(joined_data["change_type"]=="I",current_timestamp())
80 #end_date=when(joined_data["change_type"]=="IU",df_trgt.start_date)
81 #joined_data = joined_data.withColumn("start_date", start_date)
82 #start_date=when(joined_data["change_type"]=="IU",df_trgt.start_date)
83 #joined_data = joined_data.withColumn("start_date", start_date)
84 #joined_data = joined_data.withColumn("end_date", end_date)
85
86 #joined_data.show()
87
88 df_insert = joined_data.filter(joined_data.change_type == "I").select(joined_data.src_id.alias("id"),joined_data.src_name.alias("name"),joined_data.address.alias("prev_address"),
89 ,start_date.alias("start_date"),joined_data.end_date)
90
91
92
93 #df_insert.show()
94
95 df_upsert = joined_data.filter(joined_data.change_type == "IU").select(joined_data.src_id.alias("id"),joined_data.src_name.alias("name"),joined_data.address.alias("prev_address"),
96 ,joined_data.start_date,joined_data.end_date)
97
98 df_upsert=df_upsert.withColumn("start_date",lit(joined_data.start_date)).withColumn("end_date",lit(null).cast("string"))
99
100 #df_upsert.show()
101
102 df_update = joined_data.filter(joined_data.change_type == "IU").select(joined_data.id.alias("id"),joined_data.name.alias("name"),joined_data.prev_address,joined_data.start_date,
103 ,joined_data.start_date.alias("end_date"))
104
105 #df_update.show()
106
C:\Users\ABDUAL-1.OMAI\AppData\Roaming\MobaXterm\glash\lx UNIX Python 171 lines Row #1 Col #1
5:13 PM 3/24/2023
```



Type 2 Diagram

