## 

Table of Contents

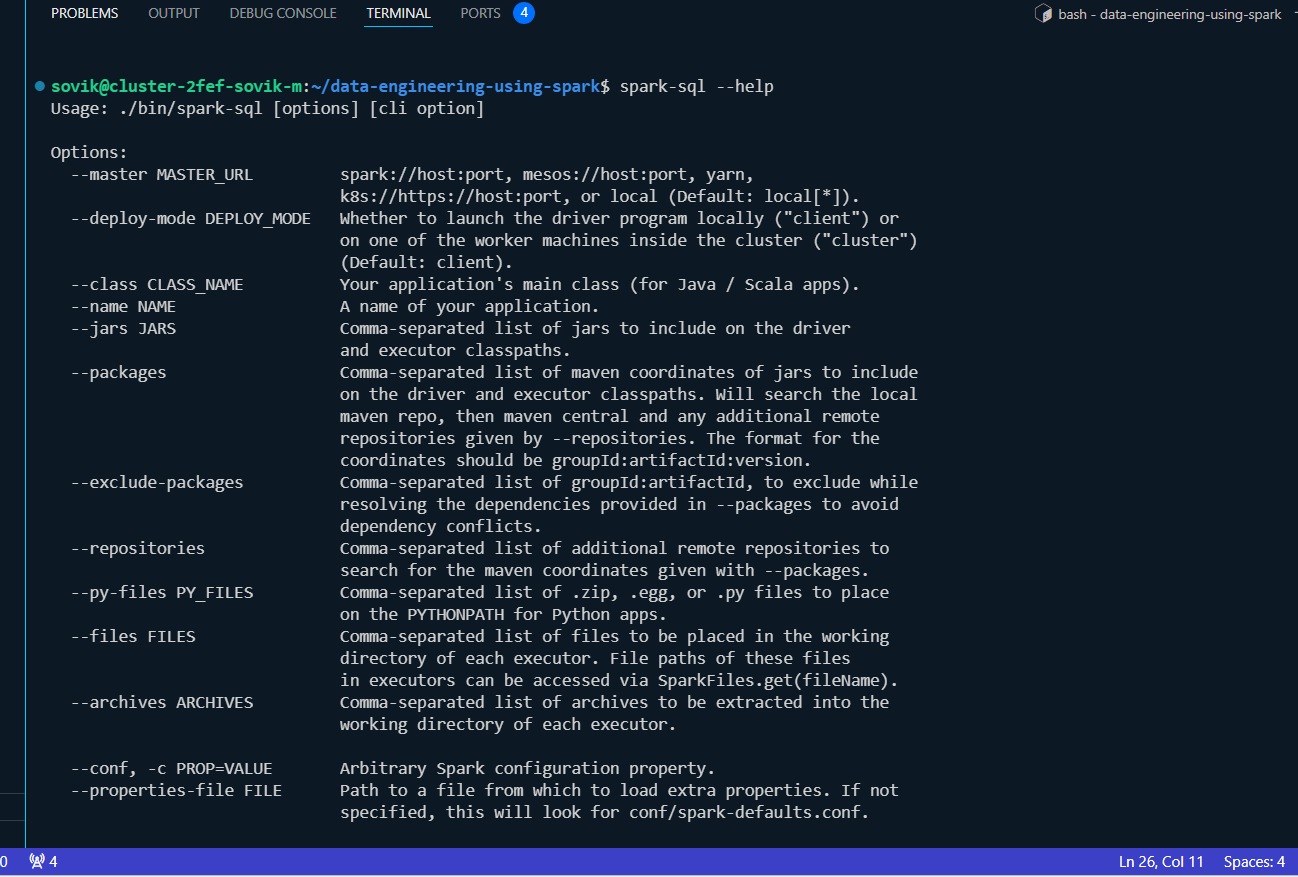
[1.Spark SQL and related commands: 1](#_Toc166397776)

[2. Use spark cli with deltalakes: 3](#_Toc166397777)

[4. Run Spark SQL CLI commands individually 9](#_Toc166397778)

## 1.Spark SQL and related commands:

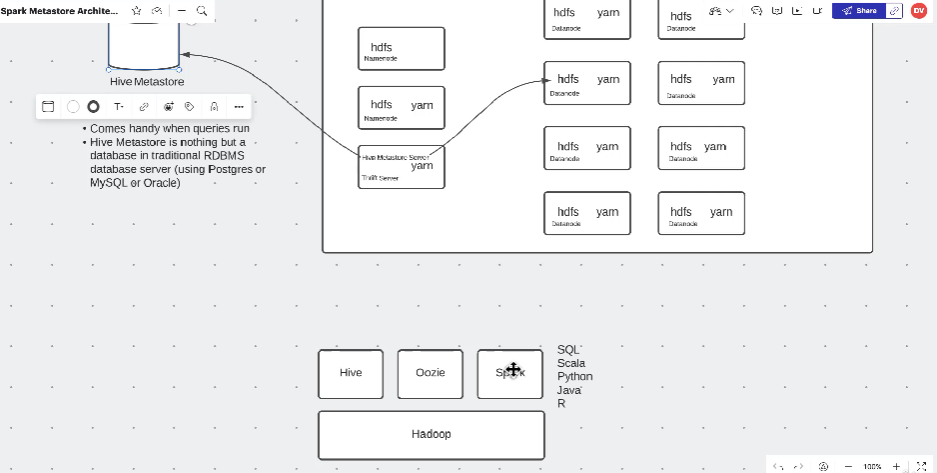
Spark-sql –help



Getiing aquainted:



Spark SQL flow:



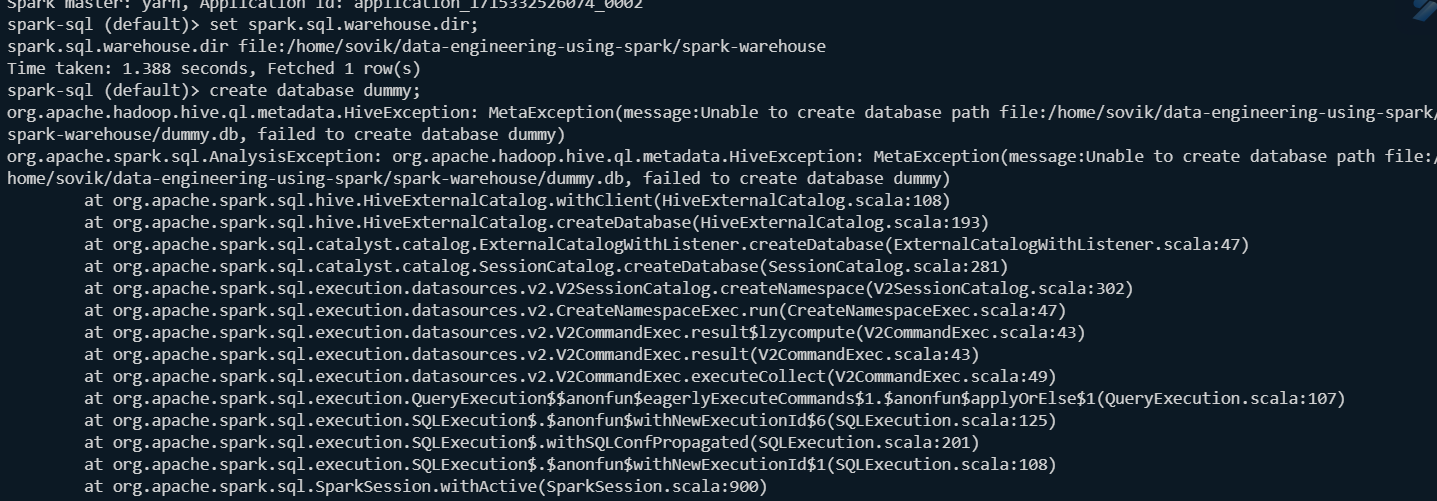
Spark and hive generally shares same metastore and commands are also similar, difference is hive only supports sql, spark supports sql,scala,java,R,python and sql.

Hive does not support delta format but spark does;

Hive executes using tez engine which uses Map reduce in BTS , spark uses spark engine;

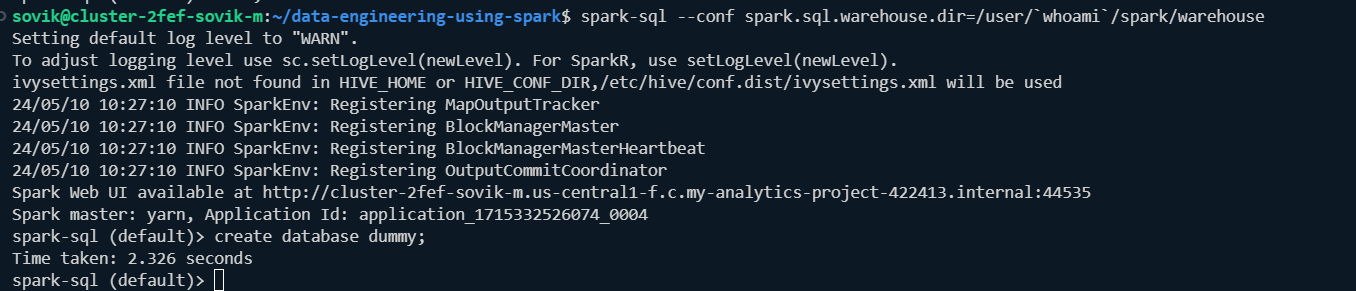
Spark sql is much faster than hive on tez, but hive sql on llap claims it is at par with spark sql but it is limited to Cloudera platform.

**Set the warehouse directory for spark as it will not work if we create a new db**

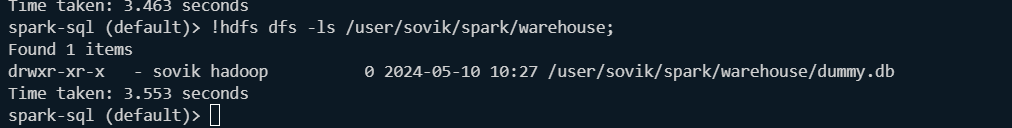
****

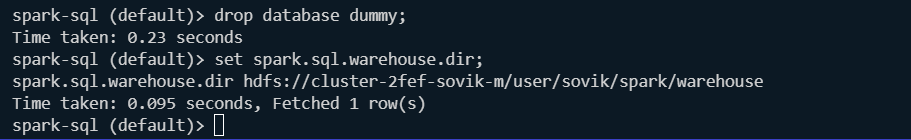
**To do it you must do it during launch of spark-sql cli:**

**Spark-sql –conf spark.sql.warehouse.dir=/user/Sovik/spark/warehouse/**

****

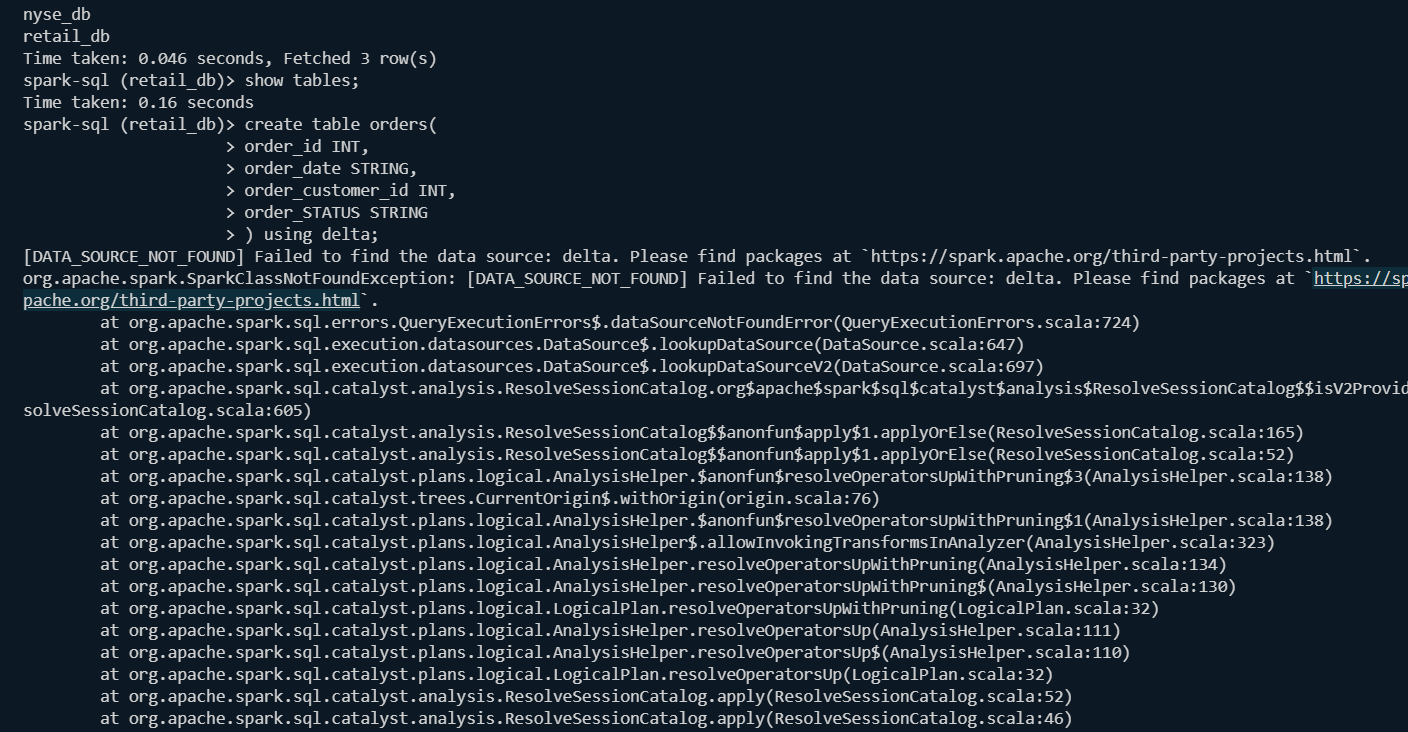
Validate:



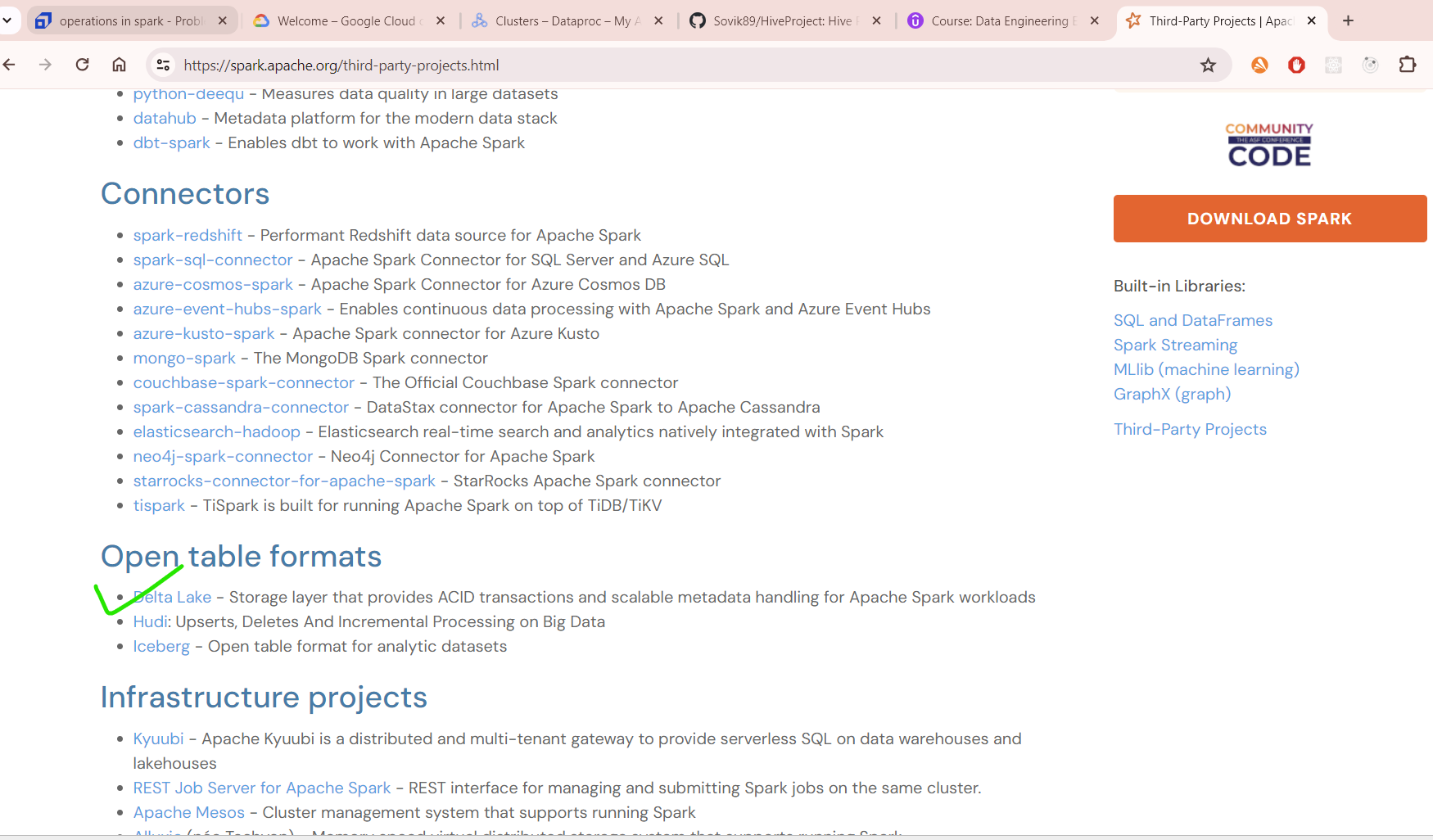


## 2. Use spark cli with deltalakes:

Cannot createdelta table:



Go to <https://spark.apache.org/third-party-projects.html>:



Command:

spark-sql --conf spark.sql.warehouse.dir=/user/`whoami`/spark/warehouse --packages io.delta:delta-spark\_2.12:3.0.0 --conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension --conf spark.sql.catalog.spark\_catalog=org.apache.spark.

sql.delta.catalog.DeltaCatalog

It will download package from the internet(.jar file)

Validate:

create table orders\_v2(

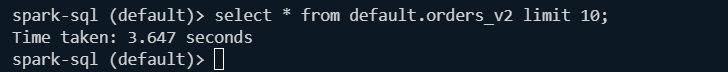
order\_id INT,

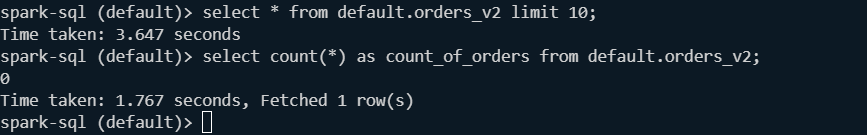
order\_date STRING,

order\_customer\_id INT,

order\_STATUS STRING

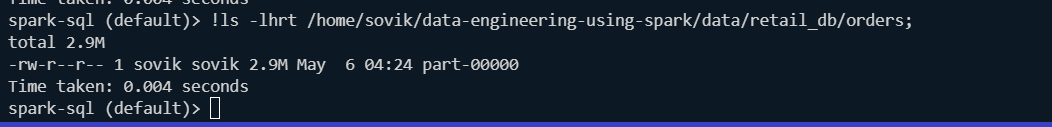
) using delta;



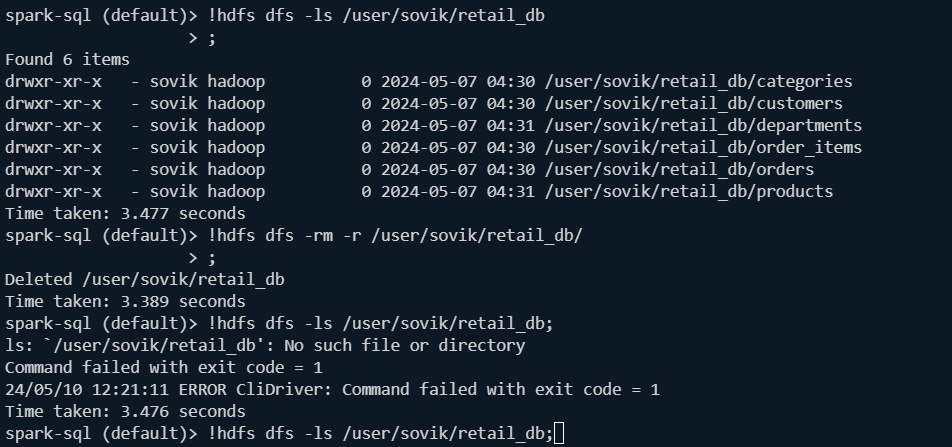


Delta lake docs: <https://docs.delta.io/3.0.0/delta-batch.html#create-a-table>

Now check;



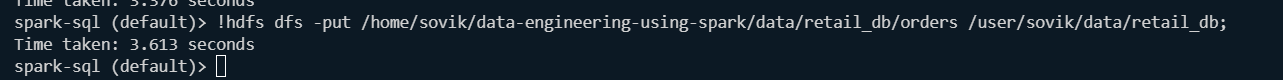
Cleanup:



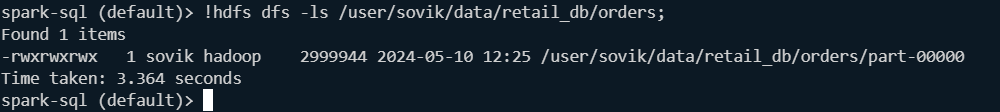
Command:

!hdfs dfs -put /home/sovik/data-engineering-using-spark/data/retail\_db/orders /user/sovik/data/retail\_db;

Put data into hdfs from local:



Validate:



Now we create a temporary view with csv:

spark-sql (default)> CREATE OR REPLACE TEMPORARY VIEW order\_view(

> order\_id INT,

> order\_date STRING,

> order\_customer\_id INT,

> order\_status STRING

> ) USING CSV

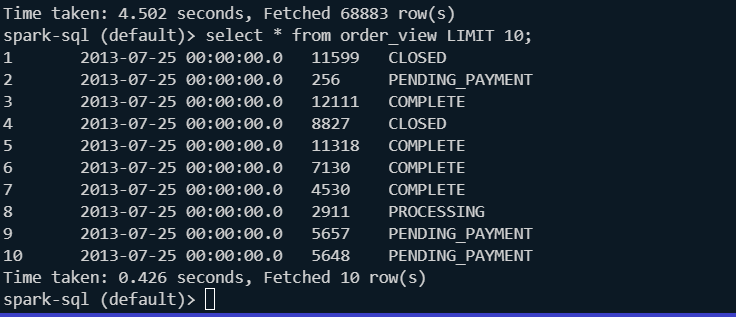
> OPTIONS(

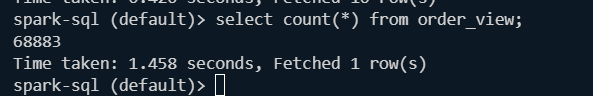
> Path='/user/sovik/data/retail\_db/orders'

> );

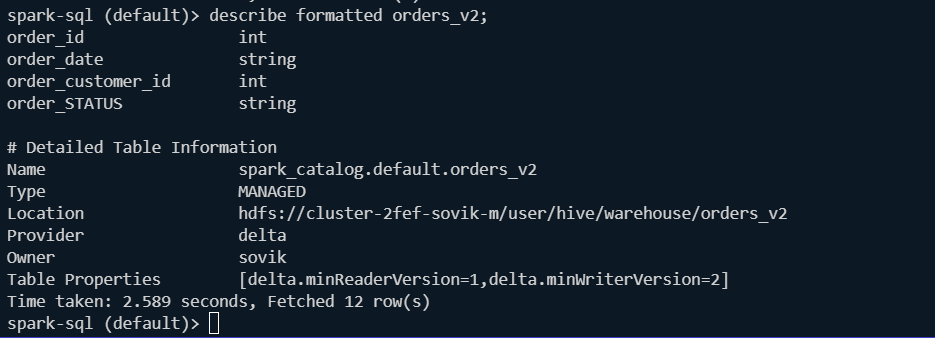
Time taken: 2.009 seconds

Validate:

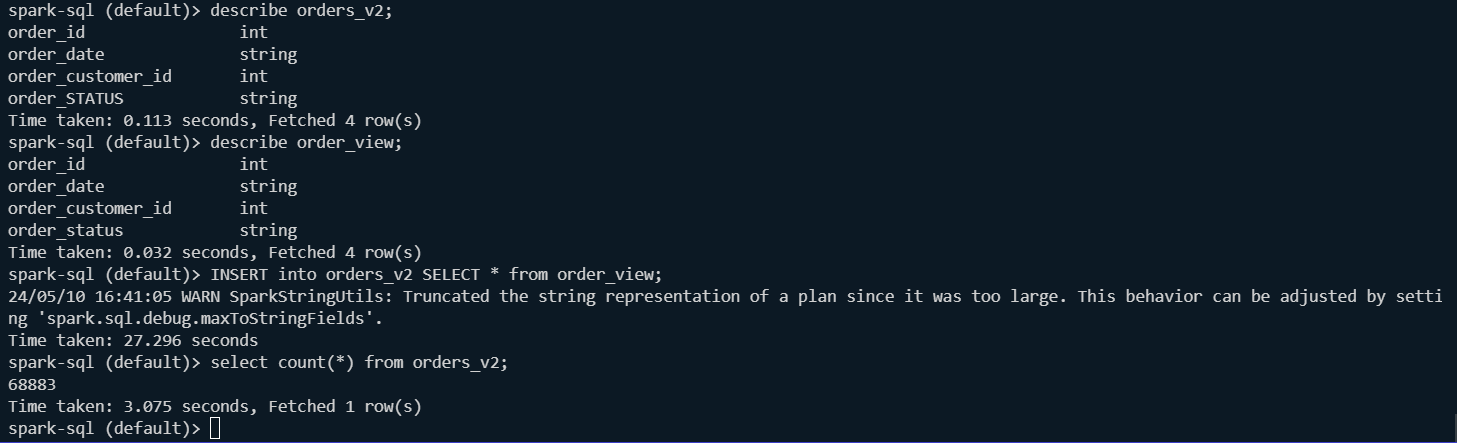




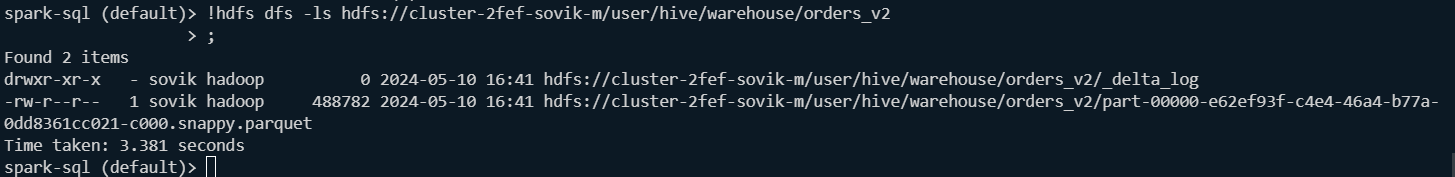
Now we can convert the order\_view to orders delta table;



INSERT into orders\_v2 SELECT \* from order\_view;

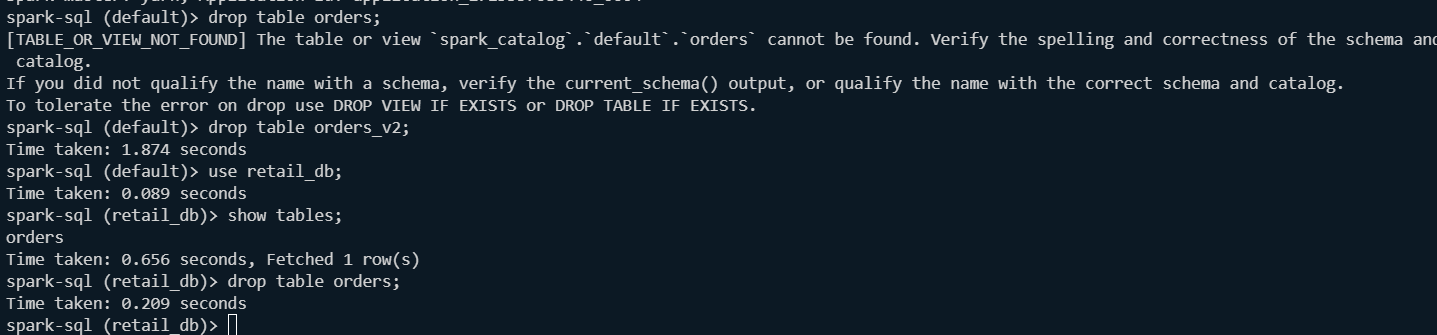


To check the actual file format in delta:

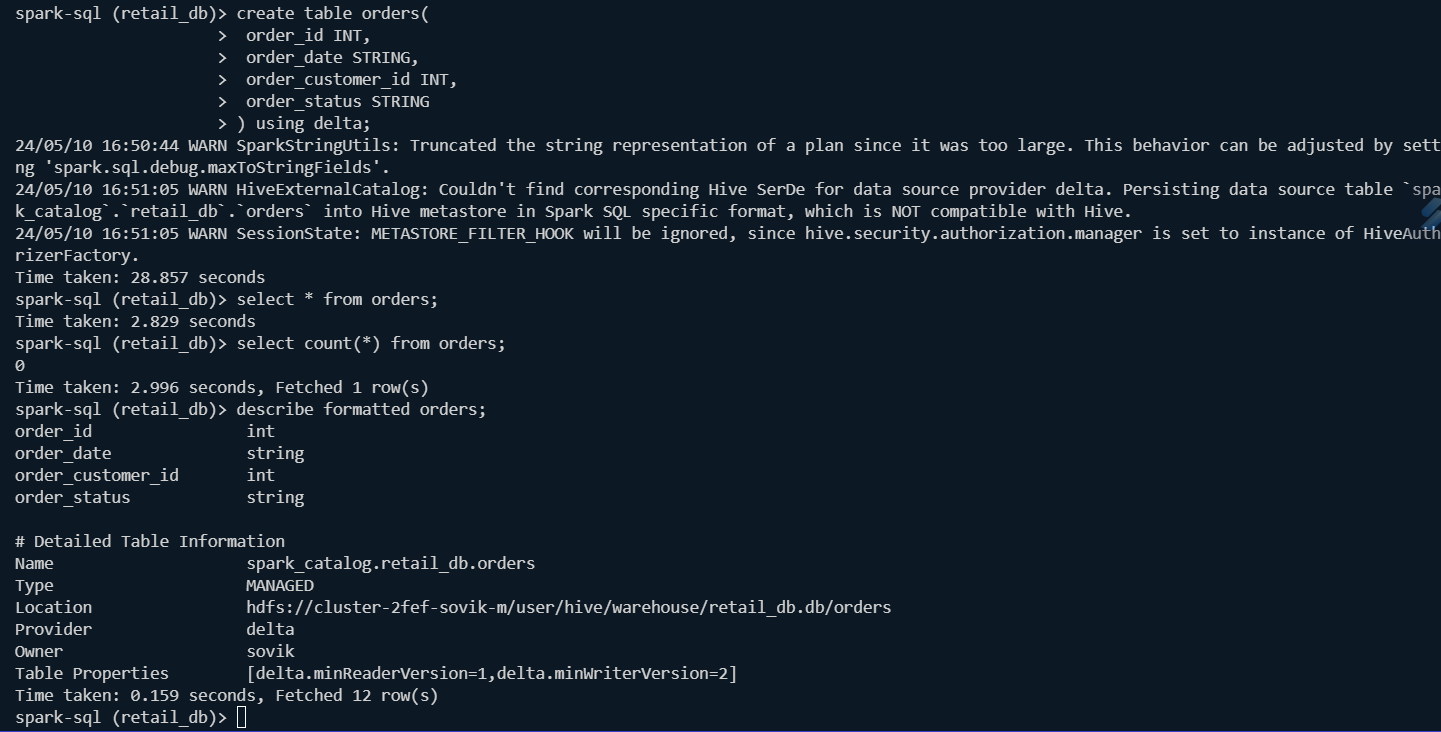


Parquet is the file format and the algorithm for partitioning in parquet is snappy. hdfs://cluster-2fef-sovik-m/user/hive/warehouse/orders\_v2/\_delta\_log 🡪 this for ACID operations

Now we go to retail\_db database and do necessary cleanup;



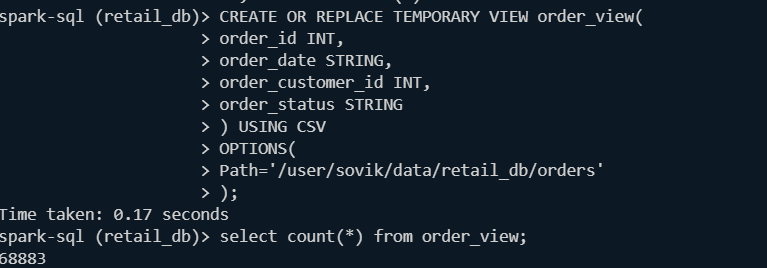
We create the orders table:

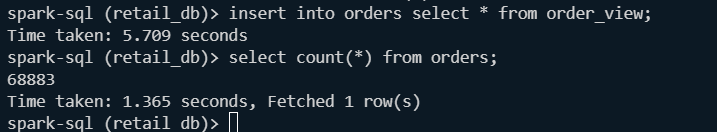


We recheck it from hive to see what is the format of the orders table;



Now populate data into the orders table via the orders view and validate:

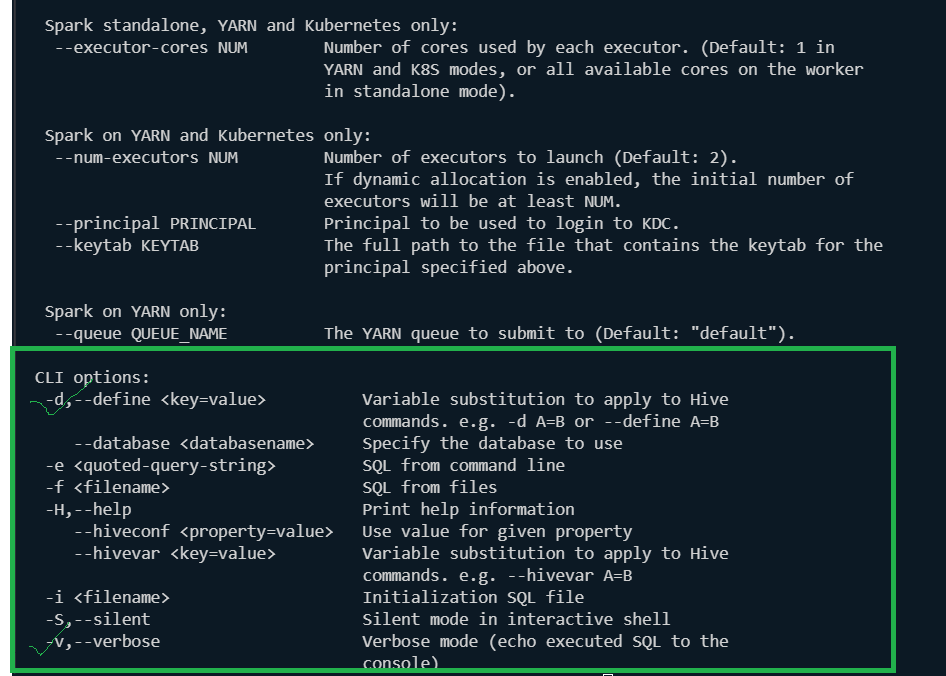




## 3. Run Spark SQL CLI commands individually

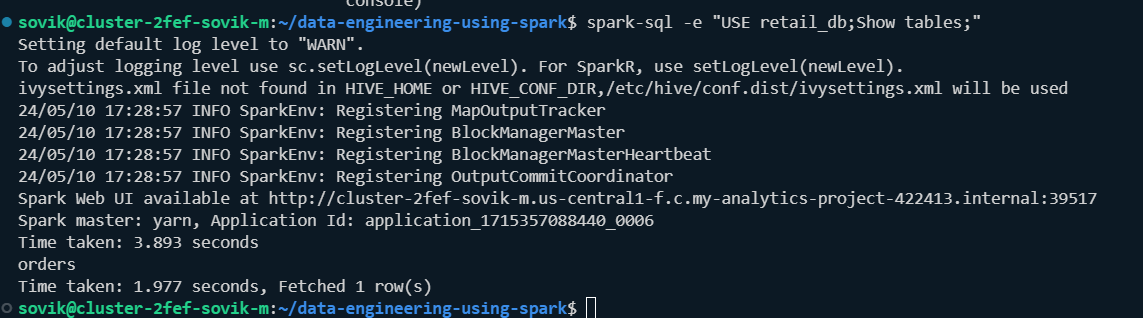
Options related to spark sql

Command: spark-sql –help

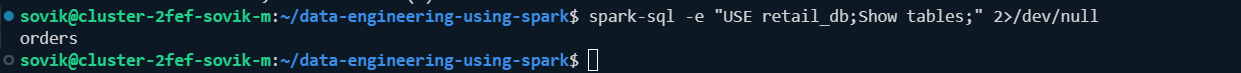


Normal commands:

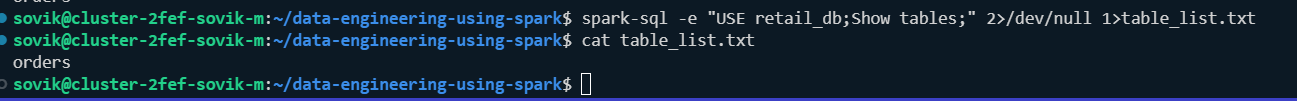
1.



If no additional info needed:



If we want to keep the output in a table:



Note:

1> is used to redirect the standard output (stdout) of the spark-sql command to the file table\_list.txt. So, any normal output from the command will be saved in the file table\_list.txt.

2> is used to redirect the standard error (stderr) of the spark-sql command to /dev/null, which is a special file that discards any data written to it. This means any error messages from the command will be discarded and will not be visible to the user.

Now we create the script:

Script:

db\_name=*${1}*

table\_name=*${2}*

spark-sql --conf spark.sql.warehouse.dir=/user/`whoami`/spark/warehouse \

          --packages io.delta:delta-spark\_2.12:3.0.0 \

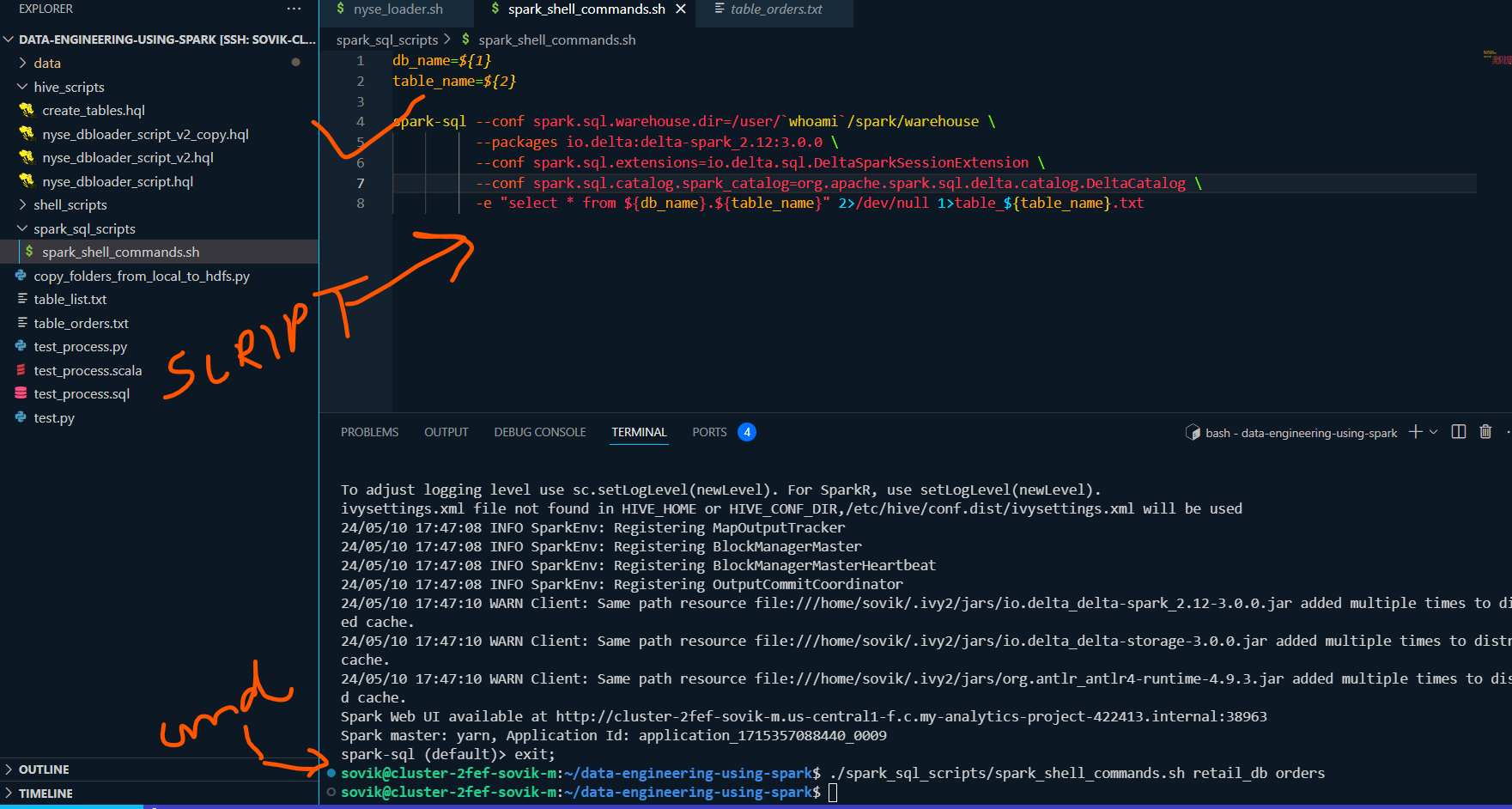
          --conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension \

          --conf spark.sql.catalog.spark\_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog \

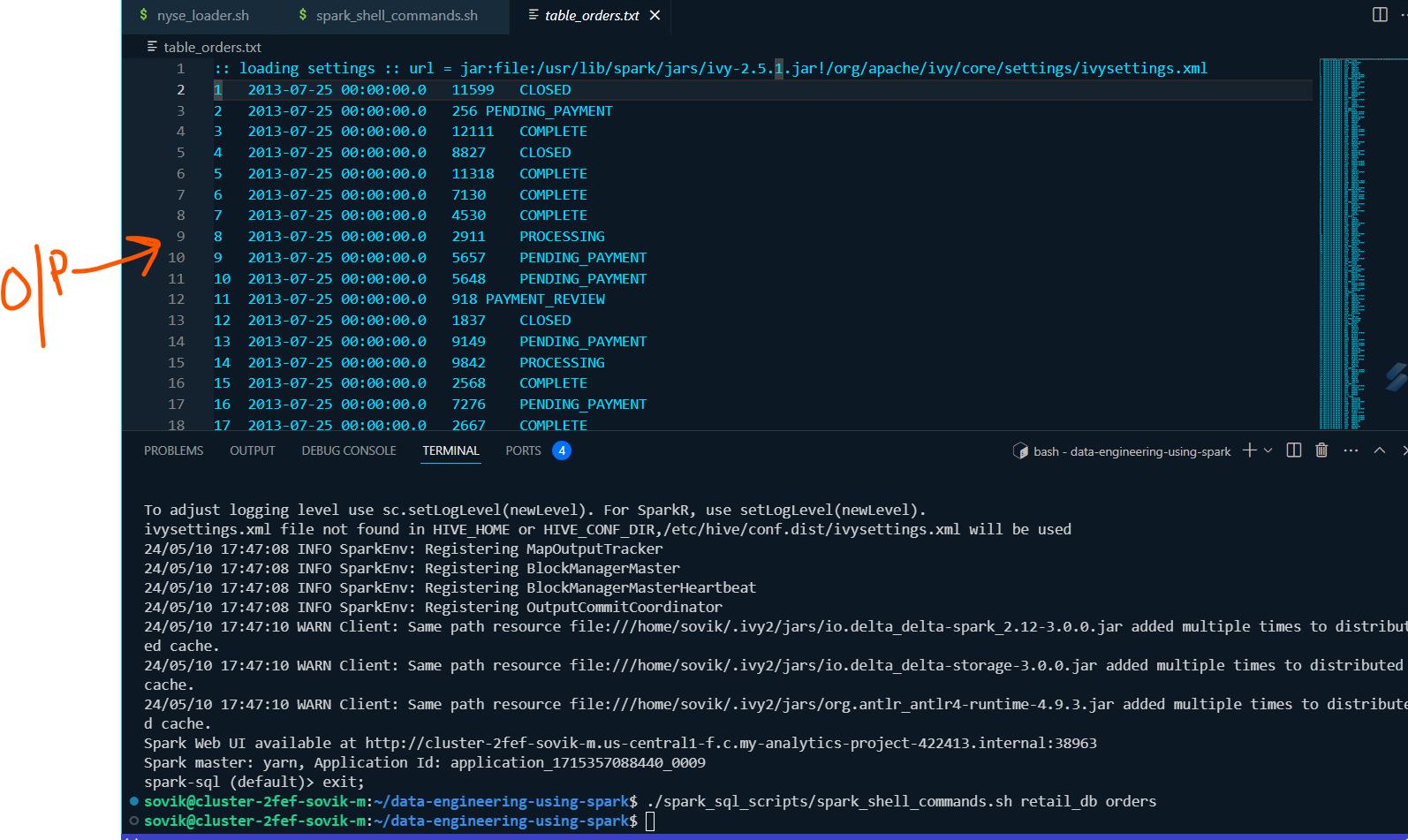
          -e "select \* from ${db\_name}.${table\_name}" 2>/dev/null 1>table\_${table\_name}.txt

Cmd:

./spark\_sql\_scripts/spark\_shell\_commands.sh retail\_db orders



O/P:



Complex script:

db\_name=*${1}*

table\_name=*${2}*

date=*${3}*

spark-sql --conf spark.sql.warehouse.dir=/user/`whoami`/spark/warehouse \

          --packages io.delta:delta-spark\_2.12:3.0.0 \

          --conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension \

          --conf spark.sql.catalog.spark\_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog \

          -f spark\_sql\_scripts/getting\_started.sh \

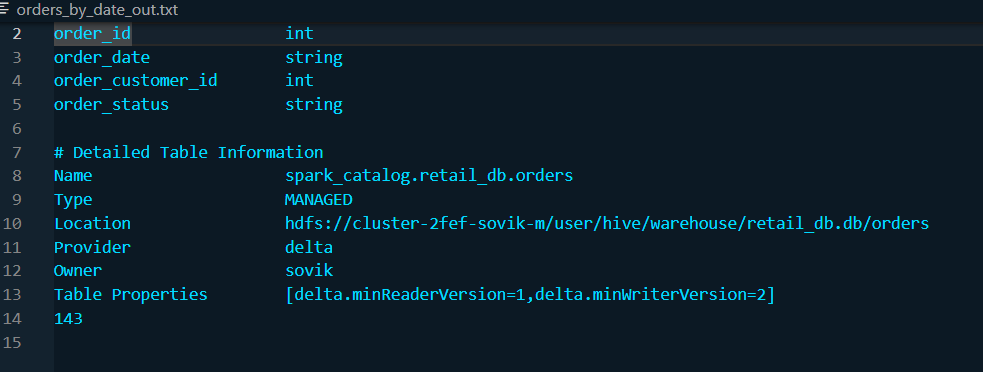
          -d DB\_NAME="${db\_name}" \

          -d TABLE\_NAME="${table\_name}" \

          -d ORDER\_DATE="${date}" 1>${table\_name}\_by\_date\_out.txt 2>${table\_name}\_by\_date\_log.log

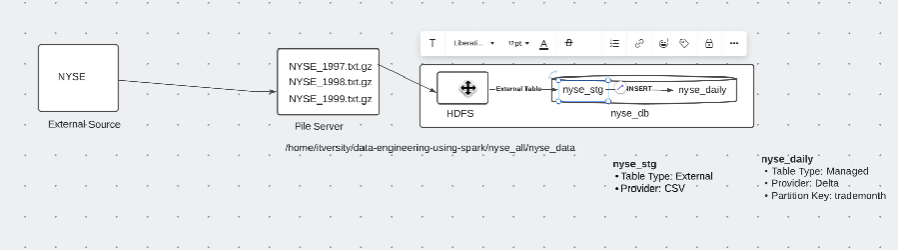
Command:

./spark\_sql\_scripts/spark\_shell\_command\_by\_date.sh retail\_db orders 2013-07-25 00:00:00



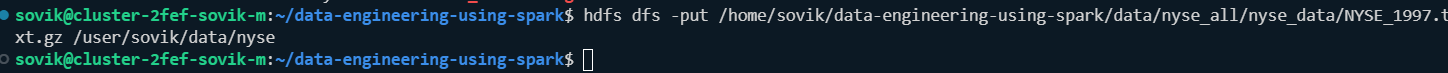
## 4. Build Realtime application with spark SQL with shell wrappers

### 4.1. Design:



### 4.2 One off build before standardization

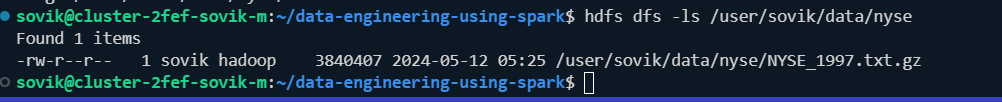
Let’s work with 1997 data



Note: omit -p in the command of put

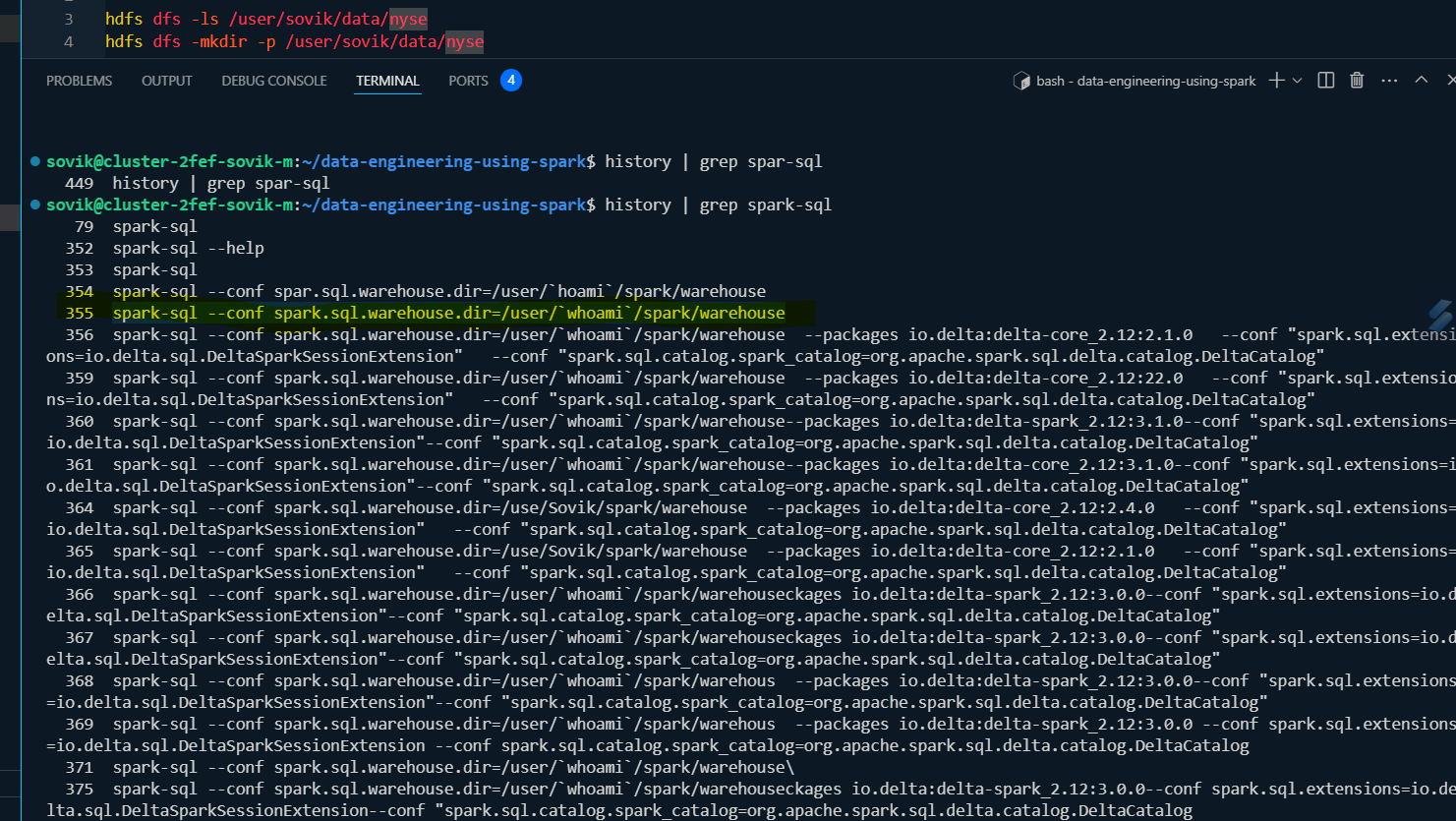


Check:



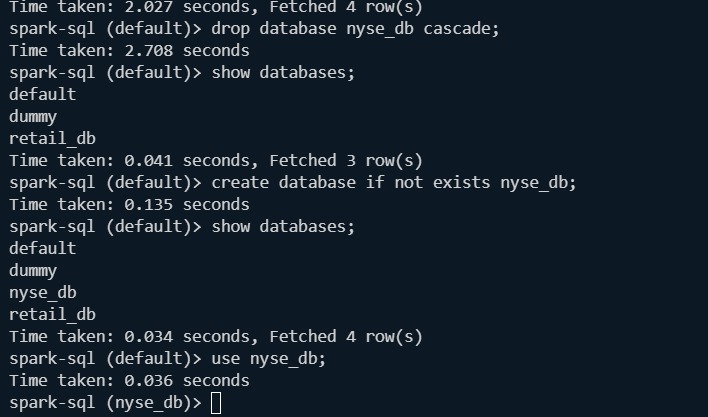
Find from history all the commands related to **spark-sql:**

**Cmd> history | spark-sql**

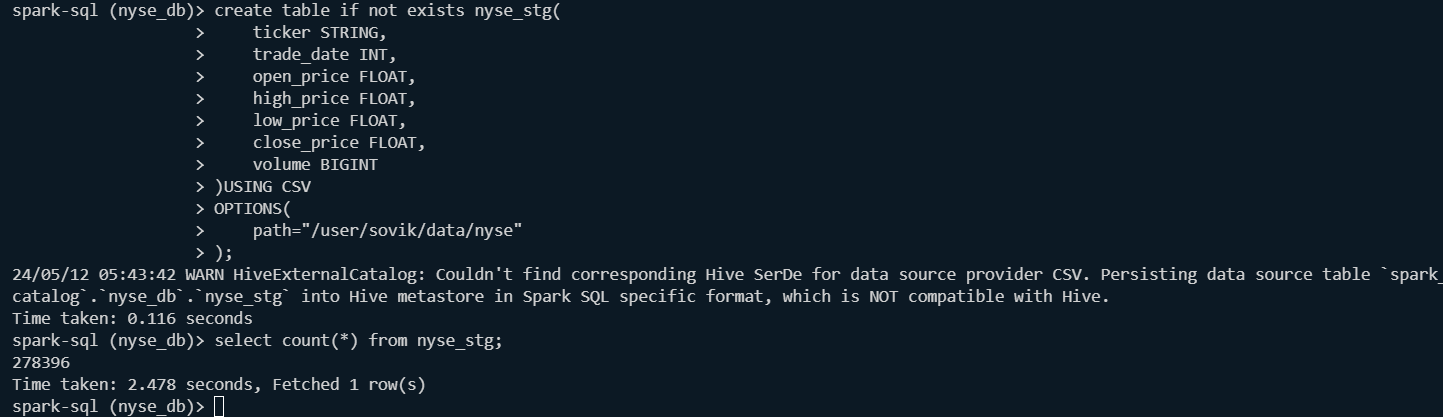


spark-sql --conf spark.sql.warehouse.dir=/user/`whoami`/spark/warehouse

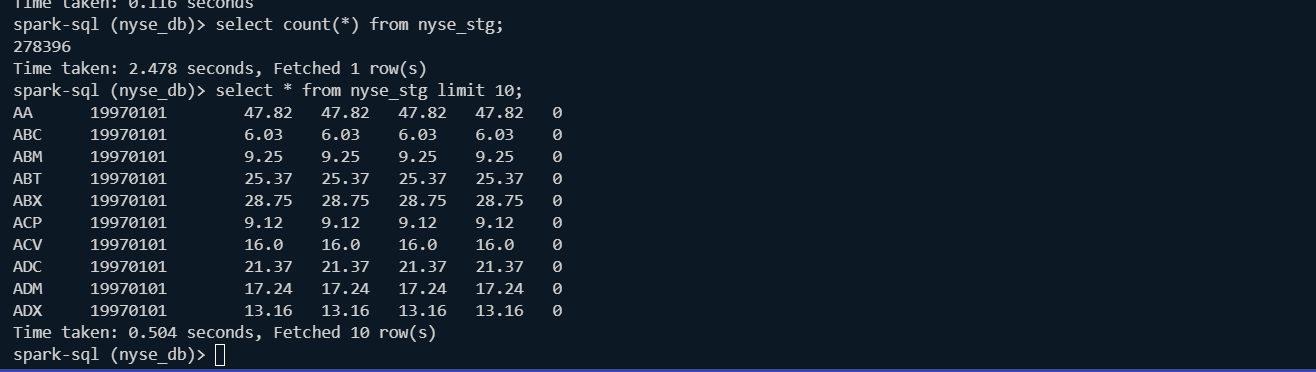
cleanups:



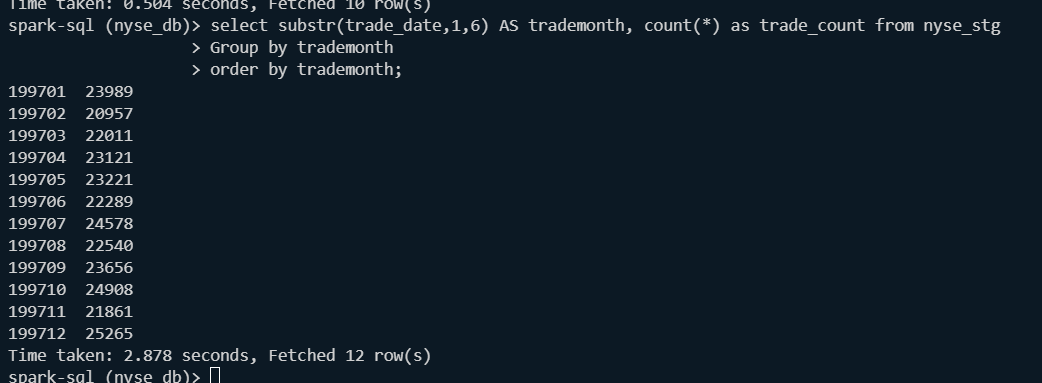
Create staging table:



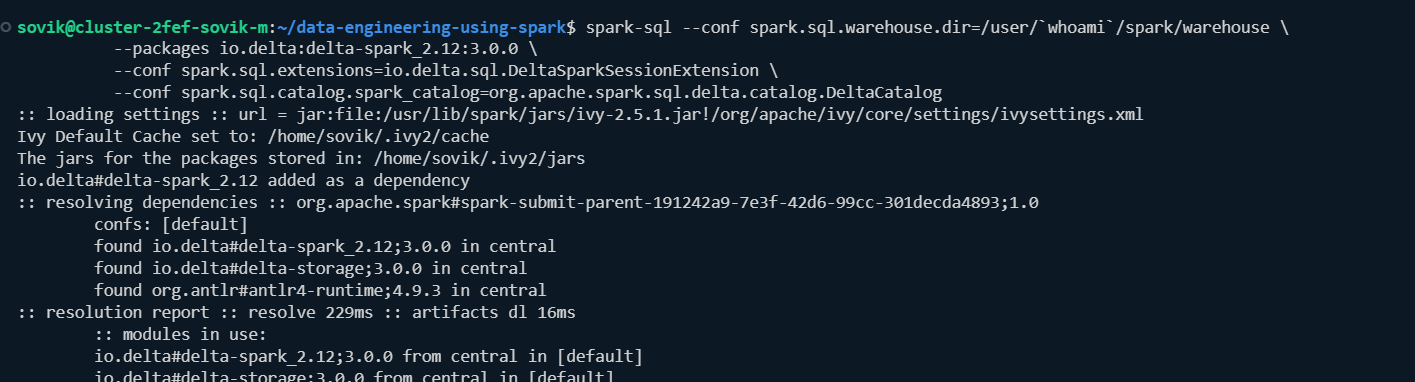
Validate:



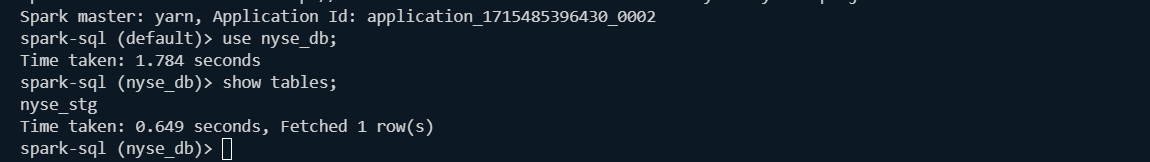
Query on stage:



For target we have to launch spark-sql using delta format;



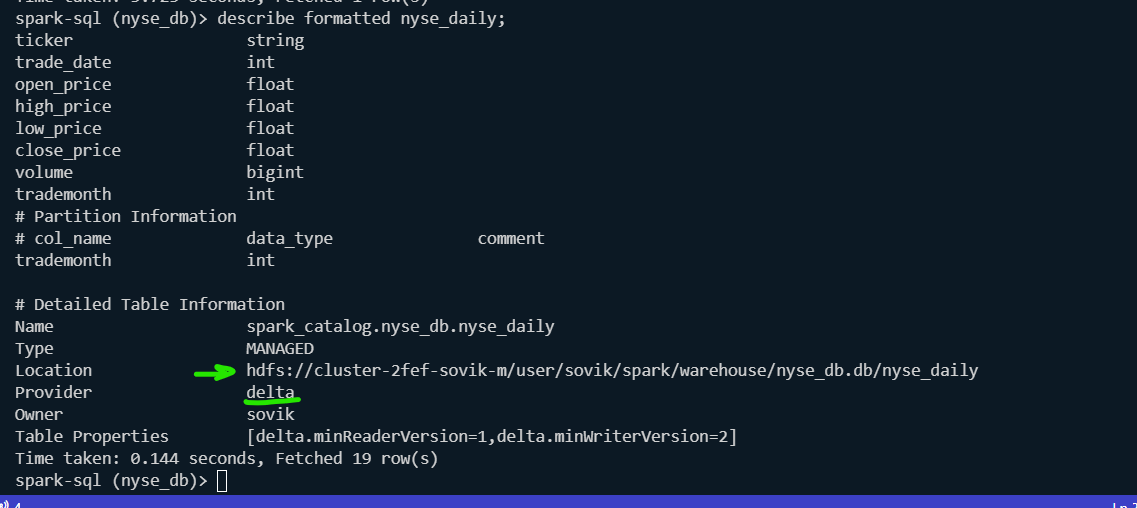
Config and validate:

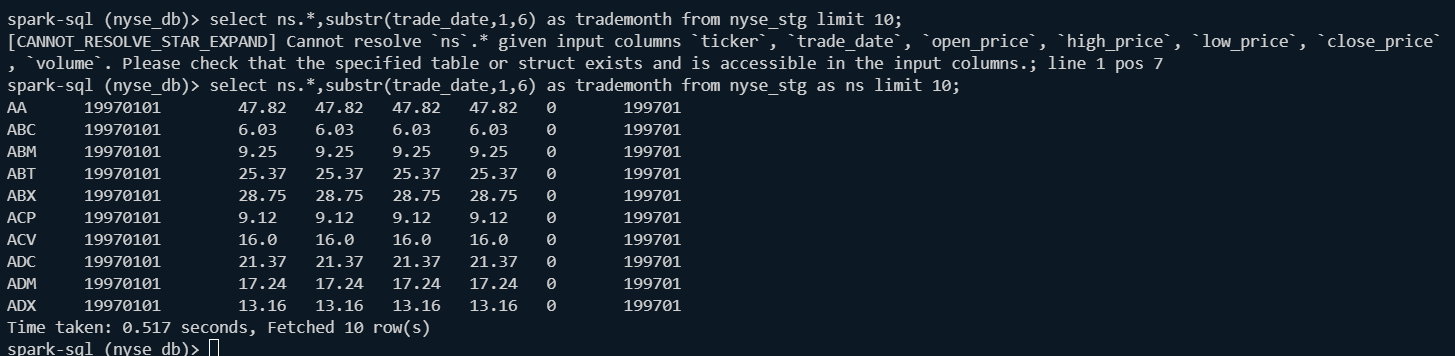


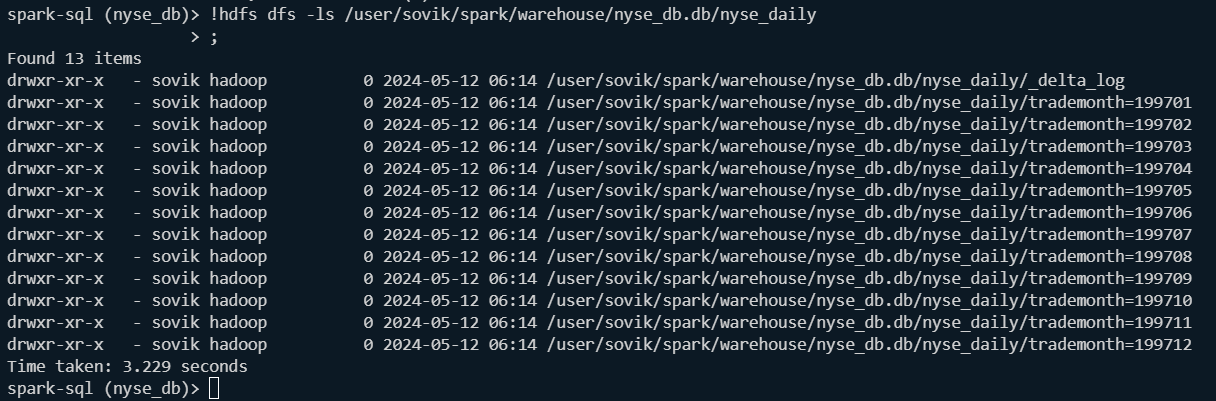
Create target table:



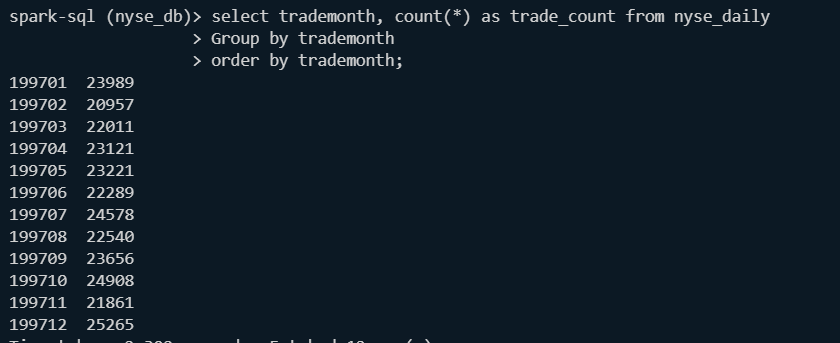
Validate:







Query:



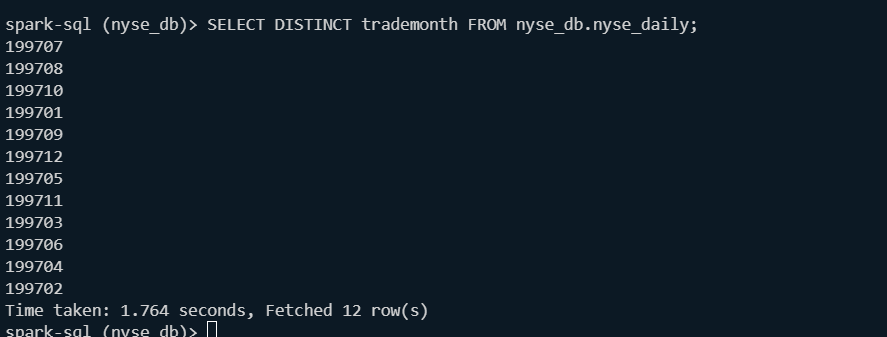
It seems you're trying to execute **the SHOW PARTITIONS** command on a Delta table named nyse\_daily in the nyse\_db database. However, Delta tables managed by Spark SQL do not support direct partition management through commands like SHOW PARTITIONS.

Instead, Delta tables often manage partitions automatically based on partition columns defined during table creation or through other Delta-specific mechanisms. To view the partitions of a Delta table, you usually interact with the table through standard SQL queries or Spark DataFrame API.

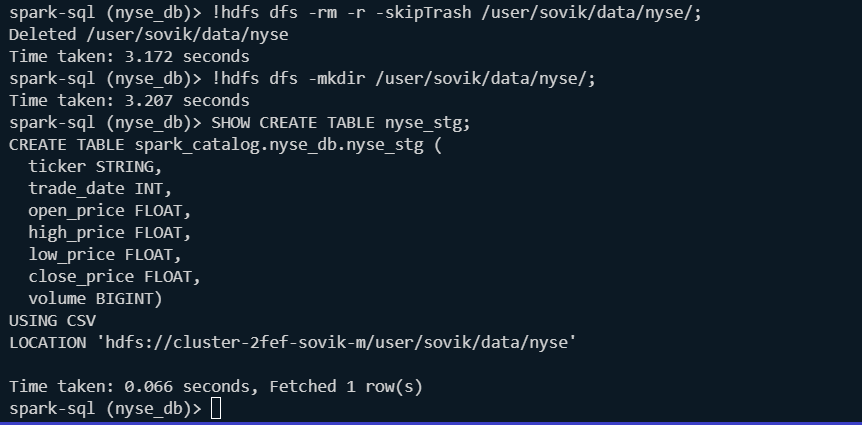
If you need to inspect partition information for a Delta table, you might consider querying the table to retrieve the distinct values of partition columns or exploring the Delta table metadata programmatically using Spark APIs.

For example, to get distinct values of partition columns, you can run SQL queries like:

**SELECT DISTINCT trademonth FROM nyse\_db.nyse\_daily;**

****

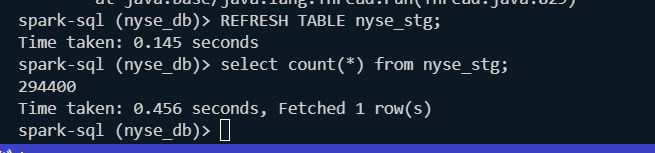
**Now lets cleanup data for the staging table for population of more data;**

****

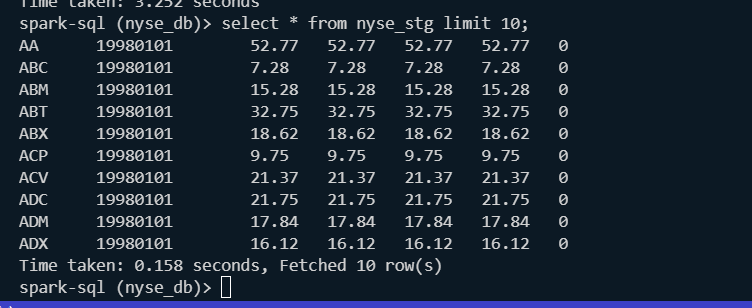
**Lets upload 1998 data;**

****

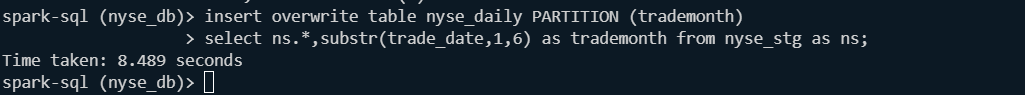
**Refresh using new table:**

****

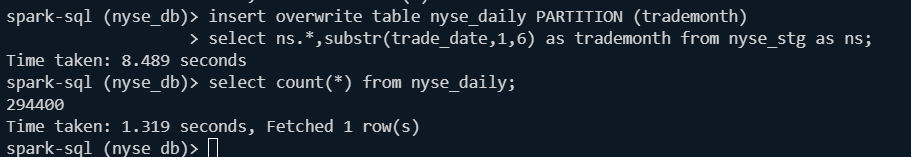
**Validate stg table:**

****

**Insert into nyse\_daily:**

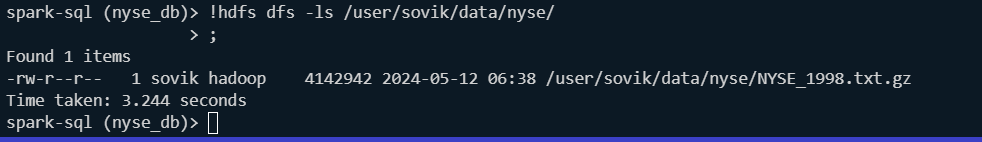
****

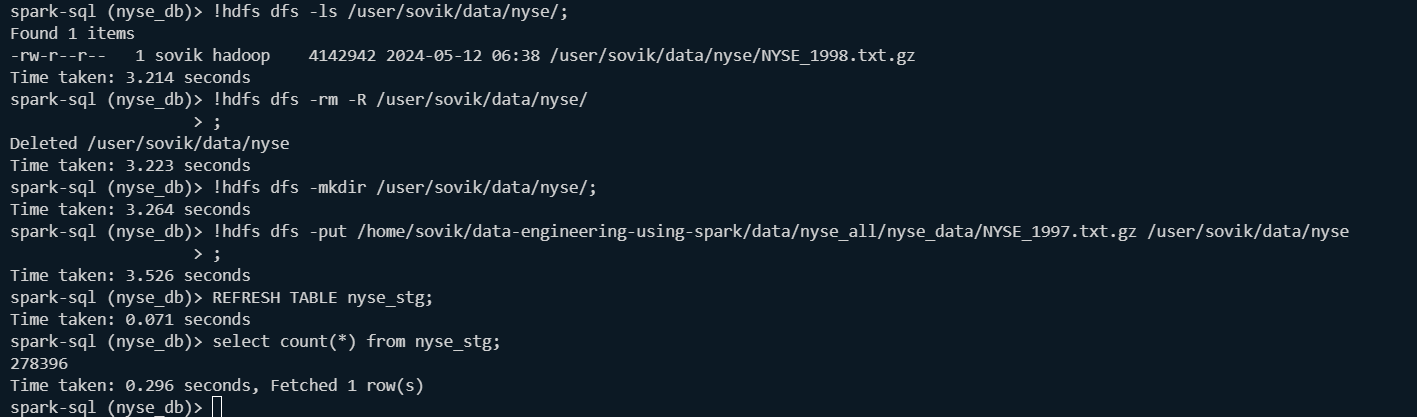
**Validate:**

****

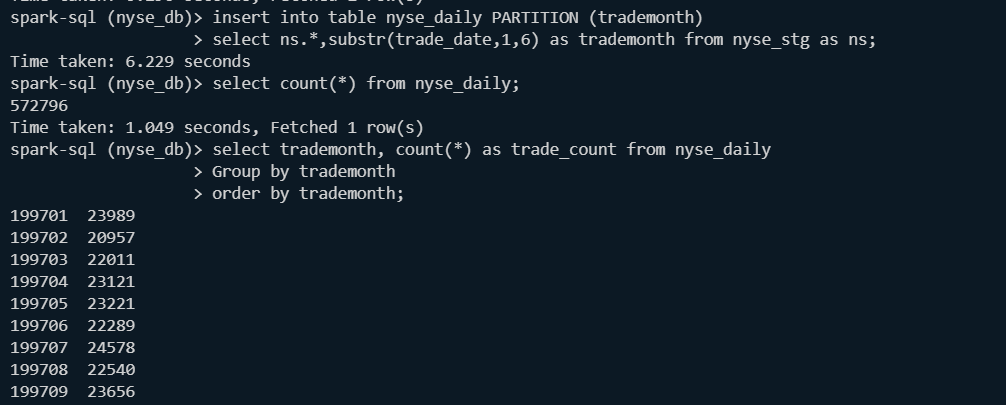
**The previous datas were deleted as we used insert overwrite statement;**

**We restablish 1997 data;**

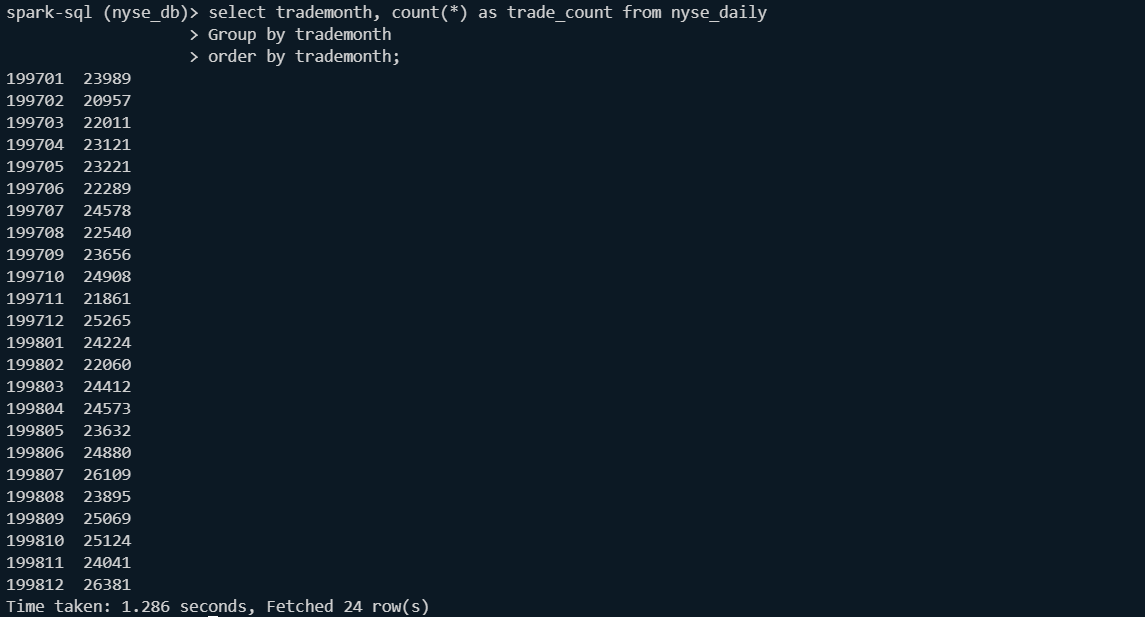
****

****

**Insert:**

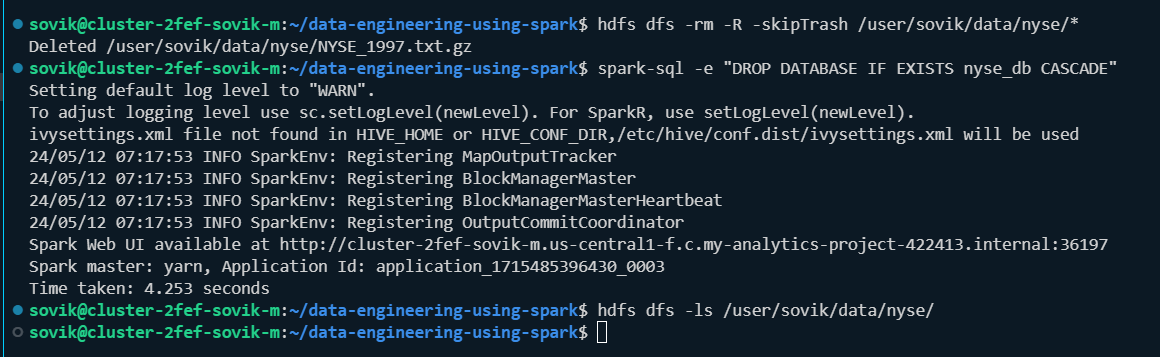
****

**Validate:**

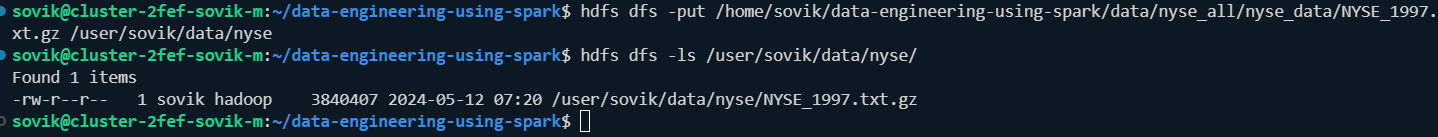
****

### 4.3 Develop the app using script with shell wrappers

Cleanup:



Copy data from local to hdfs:



Create final script: (Only issue is if we try to upload one file multiple time there is no mechanism to upsert for that particular file)

create database if not exists nyse\_db;

use nyse\_db;

create table if not exists nyse\_stg(

    ticker STRING,

    trade\_date INT,

    open\_price FLOAT,

    high\_price FLOAT,

    low\_price FLOAT,

    close\_price FLOAT,

    volume BIGINT

)USING CSV

OPTIONS(

    path="/user/sovik/data/nyse"

);

create table if not exists nyse\_daily(

    ticker STRING,

    trade\_date INT,

    open\_price FLOAT,

    high\_price FLOAT,

    low\_price FLOAT,

    close\_price FLOAT,

    volume BIGINT

)USING DELTA

PARTITIONED BY (trademonth INT);

REFRESH TABLE nyse\_stg;

insert into table nyse\_daily PARTITION (trademonth)

select ns.\*,substr(trade\_date,1,6) as trademonth from nyse\_stg as ns;

select count(\*) from nyse\_stg;

select count(\*) from nyse\_daily;

select substr(trade\_date,1,4) AS tradeyear, count(\*) as trade\_count from nyse\_stg

Group by tradeyear

order by tradeyear;

Validate command: 1997

spark-sql --conf spark.sql.warehouse.dir=/user/`whoami`/spark/warehouse \

          --packages io.delta:delta-spark\_2.12:3.0.0 \

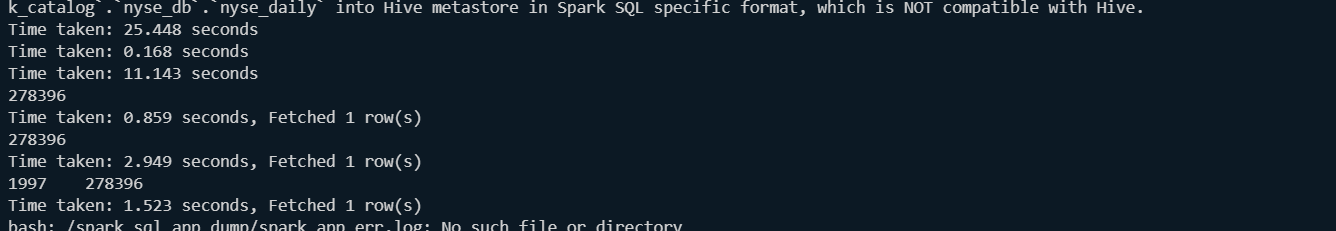
          --conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension \

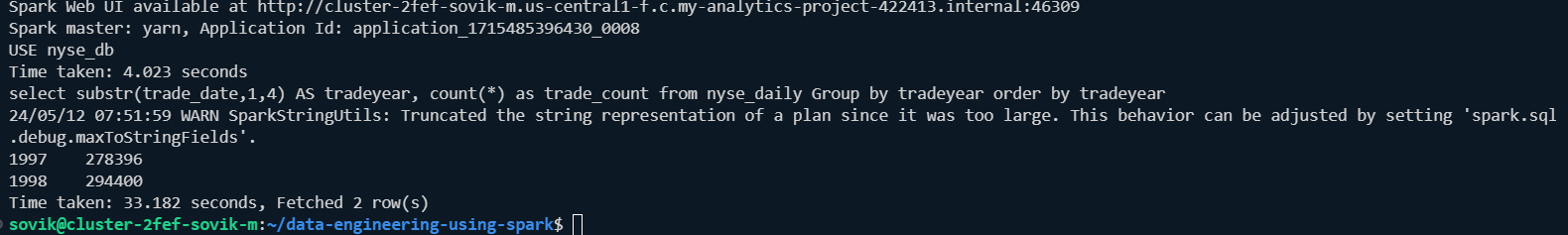
          --conf spark.sql.catalog.spark\_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog \

          -f /user/sovik/data-engineering-using-spark/spark\_sql\_scripts/nyse\_converter\_script\_v\_final.sql

          --verbose

O/P:





### 4.4. Shell wrapper:

spark-sql --conf spark.sql.warehouse.dir=/user/`whoami`/spark/warehouse \

          --packages io.delta:delta-spark\_2.12:3.0.0 \

          --conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension \

          --conf spark.sql.catalog.spark\_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog \

          -f /home/`whoami`/data-engineering-using-spark/spark\_sql\_scripts/nyse\_converter\_script\_v\_final.sql \

          --verbose 2>/dev/null

hdfs dfs -rm -R -skipTrash /user/`whoami`/data/nyse/\*

Cmd:



Give execute permission to the file using chmod 770 -R file\_name.sh and run.