Case Study - Hive

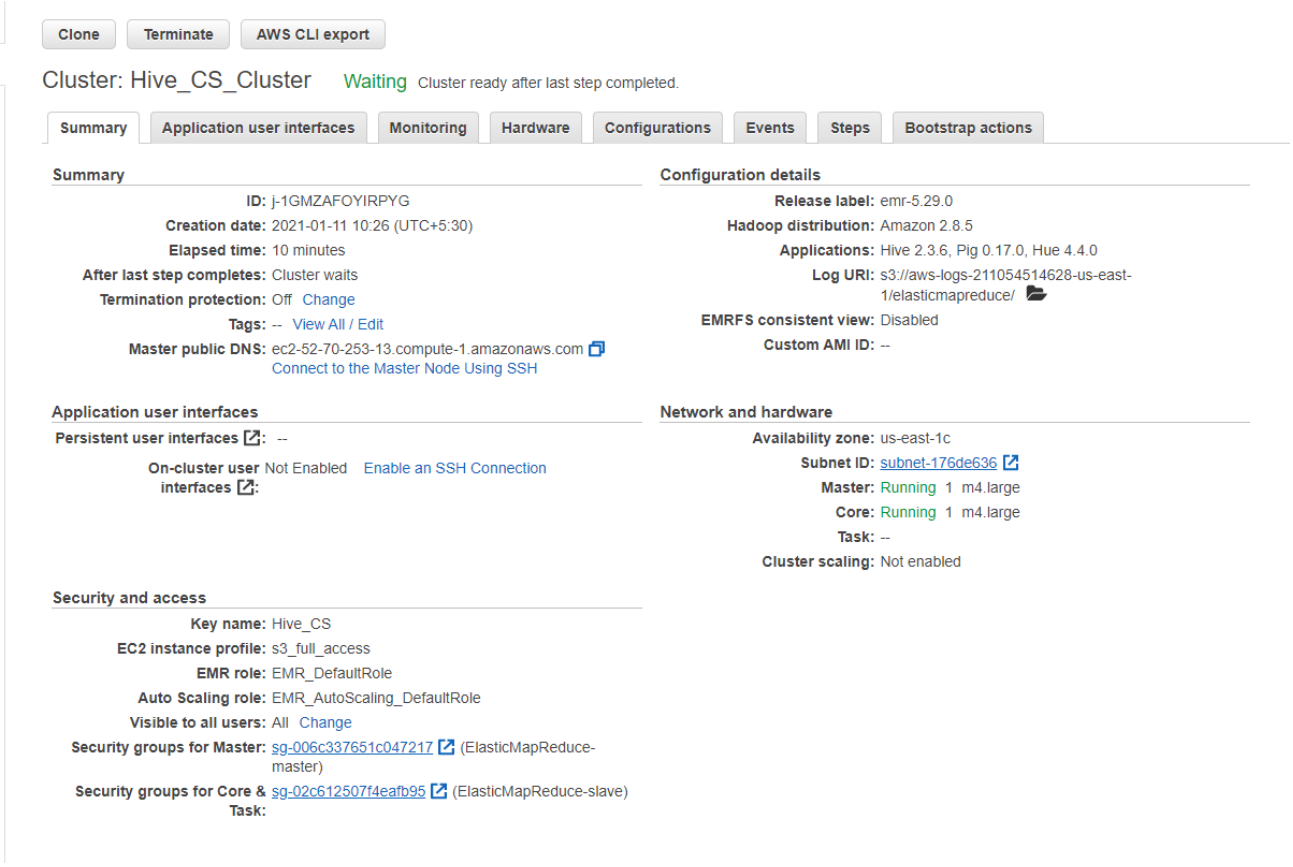
Clickstream Data Analysis

By- Abhishek Hegde

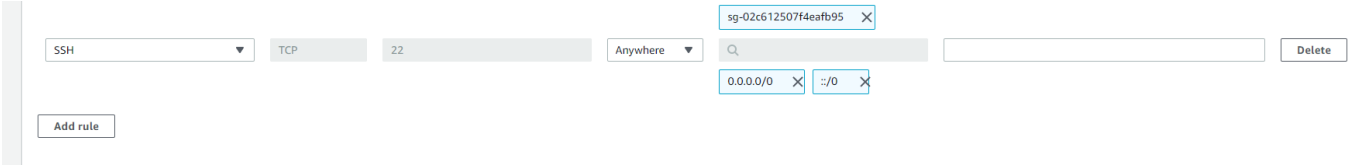
Steps Followed:

❖ Starting an EMR Cluster:

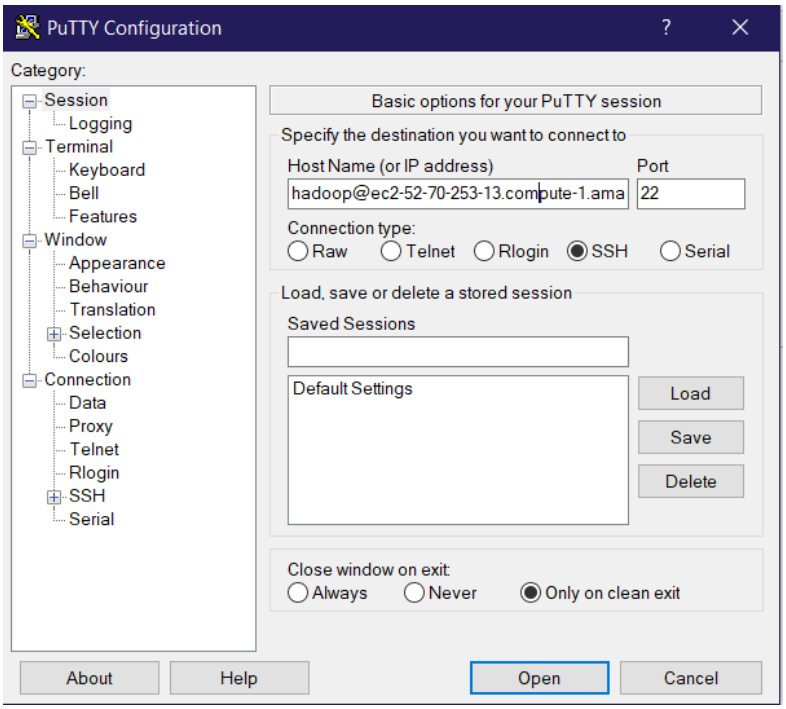
- Select region as N. Virginia (us-east-1c).
- Create a key pair.
- Go to your AWS Account -> Management Console -> EMR.
- Create a new cluster with properties as follows:
  - 1 m4.large master instance
  - 1 m4.large core instance
  - emr-5.29.0, Hive 2.3.6
  - Select the key created initially in 'Key Name'



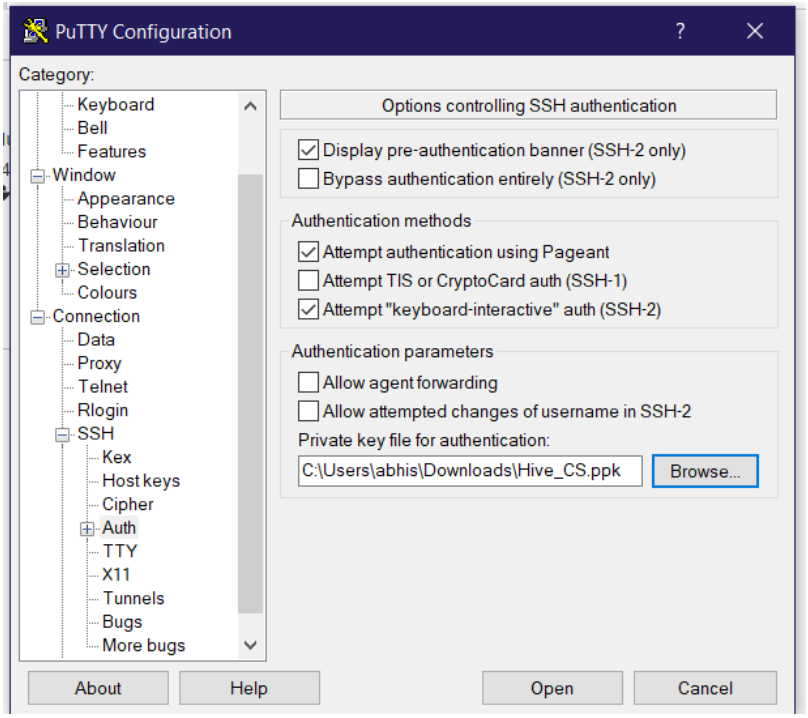
- Enable SSH from anywhere in the inbound rules setting of master node security group.



❖ Putty SSH on port 22: Give hostname as the master node public DNS.



- Make sure the .ppk key is added in the SSH -> Auth section.



❖ Hadoop Commands using Putty Terminal:

- Step 1: Browse to the dataset bucket which we will be using for our case study using the command:

```
aws s3 ls e-commerce-events-ml
```

```
[hadoop@ip-172-31-93-152 ~]$ aws s3 ls e-commerce-events-ml
2020-03-17 11:47:09  545839412 2019-Nov.csv
2020-03-17 11:37:31  482542278 2019-Oct.csv
```

As we can see we have clickstream data from 2 months in form of 2 .csv files.

- Step 2: Create a directory in HDFS into which we shall move the dataset using the command:

```
hadoop fs -mkdir /user/hive/CaseStudy
```

```
[hadoop@ip-172-31-93-152 ~]$ hadoop fs -mkdir /user/hive/CaseStudy
```

- Step 3: Move the dataset from s3 bucket to our HDFS directory using hadoop distcp:

```
hadoop distcp 's3://e-commerce-events-ml/* '/user/hive/CaseStudy'
```

```
[hadoop@ip-172-31-93-152 ~]$ hadoop distcp 's3://e-commerce-events-ml/*' '/user/
hive/CaseStudy'
21/01/11 05:10:42 INFO tools.DistCp: Input Options: DistCpOptions{atomicCommit=f
alse, syncFolder=false, deleteMissing=false, ignoreFailures=false, overwrite=fal
se, skipCRC=false, blocking=true, numListstatusThreads=0, maxMaps=20, mapBandwid
th=100, sslConfigurationFile='null', copyStrategy='uniformsize', preserveStatus=
[], preserveRawXattrs=false, atomicWorkPath=null, logPath=null, sourceFileListin
g=null, sourcePaths=[s3://e-commerce-events-ml/*], targetPath=/user/hive/CaseStu
dy, targetPathExists=true, filtersFile='null'}
21/01/11 05:10:43 INFO client.RMProxy: Connecting to ResourceManager at ip-172-3
1-93-152.ec2.internal/172.31.93.152:8032
21/01/11 05:10:47 INFO tools.SimpleCopyListing: Paths (files+dirs) cnt = 2; dirC
nt = 0
21/01/11 05:10:47 INFO tools.SimpleCopyListing: Build file listing completed.
21/01/11 05:10:47 INFO Configuration.deprecation: io.sort.mb is deprecated. Inst
ead, use mapreduce.task.io.sort.mb
21/01/11 05:10:47 INFO Configuration.deprecation: io.sort.factor is deprecated.
Instead, use mapreduce.task.io.sort.factor
21/01/11 05:10:47 INFO tools.DistCp: Number of paths in the copy list: 2
21/01/11 05:10:47 INFO tools.DistCp: Number of paths in the copy list: 2
21/01/11 05:10:47 INFO client.RMProxy: Connecting to ResourceManager at ip-172-3
1-93-152.ec2.internal/172.31.93.152:8032
21/01/11 05:10:48 INFO mapreduce.JobSubmitter: number of splits:2
21/01/11 05:10:48 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_16
10341391246_0001
21/01/11 05:10:49 INFO impl.YarnClientImpl: Submitted application application_16
10341391246_0001
21/01/11 05:10:49 INFO mapreduce.Job: The url to track the job: http://ip-172-31
-93-152.ec2.internal:20888/proxy/application_1610341391246_0001/
21/01/11 05:10:49 INFO tools.DistCp: DistCp job-id: job_1610341391246_0001
21/01/11 05:10:49 INFO mapreduce.Job: Running job: job_1610341391246_0001
```

- Step 4: Check whether data got moved or not:

```
hadoop fs -ls '/user/hive/Casestudy'
```

```
[hadoop@ip-172-31-93-152 ~]$ hadoop fs -ls '/user/hive/CaseStudy'
Found 2 items
-rw-r--r--   1 hadoop hadoop  545839412 2021-01-11 05:11 /user/hive/CaseStudy/20
19-Nov.csv
-rw-r--r--   1 hadoop hadoop  482542278 2021-01-11 05:11 /user/hive/CaseStudy/20
19-Oct.csv
```

As we can see, the 2 .csv files are successfully moved from the s3 bucket to our HDFS directory.

❖ Launch Hive Console:

- Step 1: Launch the Hive CLI using the command:

```
hive
```

```
[hadoop@ip-172-31-93-152 ~]$   hive

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j2.
properties Async: false
```

Hive service us running now.

- Step 2: Create the database in which we will be creating our tables.

```
create database if not exists CS_DB;
```

```
hive> create database if not exists CS_DB;
OK
Time taken: 1.049 seconds
```

- Step 3: Check database creation:

```
show databases;
```

```
hive> show databases;
OK
cs_db
default
Time taken: 0.148 seconds, Fetched: 2 row(s)
```

```
describe database CS_DB;
```

```
hive> describe database CS_DB;
OK
cs_db           hdfs://ip-172-31-93-152.ec2.internal:8020/user/hive/warehouse/cs
_db.db  hadoop  USER
Time taken: 0.043 seconds, Fetched: 1 row(s)
```

As we can see, our database CS_DB is created successfully in the location /user/hive/warehouse/cs_db.db.

Type `use CS_DB;` to start working on this database.

- Step 4: Create a base table to load our dataset into.

    - We will be creating an external table here because we don't want the HDFS data to be deleted in case of a cluster failure/crash as the data is of large size.
    - We will use the OpenCSVSerde library to load the .csv files into our base_table.
    - We don't want to import the first line as it contains the headers of the file, hence we skip line count 1.

```
CREATE EXTERNAL TABLE IF NOT EXISTS base_table
(
event_time          TIMESTAMP,
event_type          STRING,
product_id          STRING,
category_id         STRING,
category_code       STRING,
```

```
brand                   STRING,
price                   FLOAT,
user_id                 BIGINT,
user_session            STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
STORED AS TEXTFILE
LOCATION '/user/hive/CaseStudy/'
TBLPROPERTIES ("skip.header.line.count"="1");
```

```
hive> CREATE EXTERNAL TABLE IF NOT EXISTS base_table(event_time timestamp , event_type string
 , product_id string, category_id string , category_code string , brand string , price float
, user_id bigint , user_session string)ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCS
VSerde' STORED AS TEXTFILE LOCATION '/user/hive/CaseStudy/' TBLPROPERTIES("skip.header.line.c
ount"="1")
    > ;
OK
Time taken: 0.332 seconds
```

- Step 5: Check table creation:

```
show tables;
```

```
hive> show tables;
OK
base_table
Time taken: 0.041 seconds, Fetched: 1 row(s)
```

base_table has been successfully created.

- Step 6: Check data in table:

```
SELECT  *
FROM   base_table
LIMIT   10;
```

```
hive> SELECT *
    > FROM base_table
    > LIMIT 10;
OK
2019-11-01 00:00:02 UTC view    5802432 1487580009286598681                     0.32    56207
6640    09fafd6c-6c99-46b1-834f-33527f4de241
2019-11-01 00:00:09 UTC cart    5844397 1487580006317032337                     2.38    55332
9724    2067216c-31b5-455d-a1cc-af0575a34ffb
2019-11-01 00:00:10 UTC view    5837166 1783999064103190764             pnb     22.22   55613
8645    57ed222e-a54a-4907-9944-5a875c2d7f4f
2019-11-01 00:00:11 UTC cart    5876812 1487580010100293687             jessnail        3.165
64506666        186c1951-8052-4b37-adce-dd9644b1d5f7
2019-11-01 00:00:24 UTC remove_from_cart        5826182 1487580007483048900                 3
.33     553329724       2067216c-31b5-455d-a1cc-af0575a34ffb
2019-11-01 00:00:24 UTC remove_from_cart        5826182 1487580007483048900                 3
.33     553329724       2067216c-31b5-455d-a1cc-af0575a34ffb
2019-11-01 00:00:25 UTC view    5856189 1487580009026551821             runail  15.71   56207
6640    09fafd6c-6c99-46b1-834f-33527f4de241
2019-11-01 00:00:32 UTC view    5837835 1933472286753424063                     3.49    51464
9199    432a4e95-375c-4b40-bd36-0fc039e77580
2019-11-01 00:00:34 UTC remove_from_cart        5870838 1487580007675986893             milv0
.79     429913900       2f0bff3c-252f-4fe6-afcd-5d8a6a92839a
2019-11-01 00:00:37 UTC view    5870803 1487580007675986893             milv    0.79    42991
3900    2f0bff3c-252f-4fe6-afcd-5d8a6a92839a
Time taken: 2.177 seconds, Fetched: 10 row(s)
```

First 10 records. *Took 2.177 s to fetch*.

- Step 7: Now we shall create our optimized_table:
    - To create optimized table, we first need to enable partitioning(dynamic) and bucketing:

- set hive.exec.dynamic.partition=true;
- set hive.exec.dynamic.partition.mode= nonstrict;
- set hive.enforce.bucketing=true ;

- We will use 'event_type' as our field for partitioning as event_type has low cardinality (of 4) and most of our queries will be around purchases and 'purchase' is an event type.
- Partitioning helps speed up the indexing for WHERE clause, so every time an 'event_type' would be used in WHERE clause, we will essentially be saving a lot of time.
- Bucketing will be done on the 'price' field as it has high cardinality and also since most of our queries contain some kind of aggregations on the 'price' column to calculate sales, revenue, etc. We'll create 50 of these.
- This table will be an internal table as data is the same as our base_table and that data, is already secure.

```
CREATE TABLE IF NOT EXISTS optimized_table
(
event_time           TIMESTAMP,
product_id           STRING,
category_id          STRING,
category_code        STRING,
brand                STRING,
price                FLOAT,
user_id              BIGINT,
user_session         STRING
)
PARTITIONED BY (event_type string)
CLUSTERED BY (price) INTO 50 BUCKETS
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' STORED AS
TEXTFILE LOCATION '/user/hive/CaseStudy/'
TBLPROPERTIES ("skip.header.line.count"="1");
```

```
hive> CREATE TABLE IF NOT EXISTS optimized_table
    >   (
    >      event_time    TIMESTAMP,
    >      product_id    STRING,
    >      category_id   STRING,
    >      category_code STRING,
    >      brand         STRING,
    >      price         FLOAT,
    >      user_id       BIGINT,
    >      user_session  STRING
    >   ) PARTITIONED BY (event_type string) CLUSTERED BY (price) INTO 50 BUCKET
S ROW
    > FORMAT SERDE'org.apache.hadoop.hive.serde2.OpenCSVSerde' STORED AS TEXTFIL
E
    > LOCATION'/user/hive/CaseStudy/' TBLPROPERTIES("skip.header.line.count"="1"
);
OK
Time taken: 0.125 seconds
```

- Step 8: Check table creation:

```
show tables;
```

```
hive> show tables;
OK
base_table
optimized_table
Time taken: 0.058 seconds, Fetched: 2 row(s)
```

Both tables are created successfully.

- Step 9: Insert data into the optimized_table using data from base_table creating the partitions :

```
INSERT INTO  table optimized_table
PARTITION    (event_type)
SELECT       event time, product_id, category_id, category_code, brand,
price, user_id, user_session event_type
FROM         base_table;
```

```
hive> INSERT INTO table optimized_table PARTITION
    >           (
    >                    event_type
    >           )
    > SELECT event_time ,
    >       product_id ,
    >       category_id ,
    >       category_code,
    >       brand ,
    >       price,
    >       user_id,
    >       user_session ,
    >       event_type
    > FROM   base_table;
Query ID = hadoop_20210111055230_dae76c26-34fa-40ba-9cef-8128f5445779
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1610341391246_0005)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      2          2        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      5          5        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 161.86 s
--------------------------------------------------------------------------------
Loading data to table default.optimized_table partition (event_type=null)

Loaded : 4/4 partitions.
        Time taken to load dynamic partitions: 0.922 seconds
        Time taken for adding to write entity : 0.005 seconds
OK
Time taken: 174.765 seconds
```

- Step 10: Check partitions::

```
hadoop fs -ls '/user/hive/CaseStudy/'
```

```
[hadoop@ip-172-31-93-152 ~]$ hadoop fs -ls '/user/hive/CaseStudy/'
Found 6 items
-rw-r--r--   1 hadoop hadoop  545839412 2021-01-11 05:11 /user/hive/CaseStudy/2019-Nov.csv
-rw-r--r--   1 hadoop hadoop  482542278 2021-01-11 05:11 /user/hive/CaseStudy/2019-Oct.csv
drwxr-xr-x   - hadoop hadoop          0 2021-01-11 05:55 /user/hive/CaseStudy/event_type=cart
drwxr-xr-x   - hadoop hadoop          0 2021-01-11 05:55 /user/hive/CaseStudy/event_type=purchase
drwxr-xr-x   - hadoop hadoop          0 2021-01-11 05:55 /user/hive/CaseStudy/event_type=remove_from_cart
drwxr-xr-x   - hadoop hadoop          0 2021-01-11 05:55 /user/hive/CaseStudy/event_type=view
```

As we can see 4 partitions have been dynamically created on the field event_type in the form of directories with name format ‘event_type=value’

- Step 11: Check buckets of any one partition:

```
hadoop fs -ls '/user/hive/CaseStudy/event_type=cart'
```

```
[hadoop@ip-172-31-93-152 ~]$ hadoop fs -ls '/user/hive/CaseStudy/event_type=cart'
Found 50 items
-rwxr-xr-x   1 hadoop hadoop    2054764 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000000_0
-rwxr-xr-x   1 hadoop hadoop    3705883 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000001_0
-rwxr-xr-x   1 hadoop hadoop   12579390 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000002_0
-rwxr-xr-x   1 hadoop hadoop    3617299 2021-01-11 05:55 /user/hive/CaseStudy/event_type=cart/000003_0
-rwxr-xr-x   1 hadoop hadoop    4426146 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000004_0
-rwxr-xr-x   1 hadoop hadoop    7573931 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000005_0
-rwxr-xr-x   1 hadoop hadoop    7350044 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000006_0
-rwxr-xr-x   1 hadoop hadoop    3028836 2021-01-11 05:55 /user/hive/CaseStudy/event_type=cart/000007_0
-rwxr-xr-x   1 hadoop hadoop    7028319 2021-01-11 05:55 /user/hive/CaseStudy/event_type=cart/000008_0
-rwxr-xr-x   1 hadoop hadoop    6282173 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000009_0
-rwxr-xr-x   1 hadoop hadoop    4240684 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000010_0
-rwxr-xr-x   1 hadoop hadoop    4822418 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000011_0
-rwxr-xr-x   1 hadoop hadoop    3002275 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000012_0
-rwxr-xr-x   1 hadoop hadoop    2750488 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000013_0
-rwxr-xr-x   1 hadoop hadoop    2720374 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000014_0
-rwxr-xr-x   1 hadoop hadoop   11330763 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000015_0
-rwxr-xr-x   1 hadoop hadoop    3202119 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000016_0
-rwxr-xr-x   1 hadoop hadoop    5746591 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000017_0
-rwxr-xr-x   1 hadoop hadoop    2559685 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000018_0
-rwxr-xr-x   1 hadoop hadoop   10987168 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000019_0
-rwxr-xr-x   1 hadoop hadoop    8688698 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000020_0
-rwxr-xr-x   1 hadoop hadoop    9592355 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000021_0
-rwxr-xr-x   1 hadoop hadoop    3241246 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000022_0
-rwxr-xr-x   1 hadoop hadoop    5565655 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000023_0
-rwxr-xr-x   1 hadoop hadoop    9369608 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000024_0
-rwxr-xr-x   1 hadoop hadoop    3486334 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000025_0
-rwxr-xr-x   1 hadoop hadoop    8442021 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000026_0
-rwxr-xr-x   1 hadoop hadoop    3774273 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000027_0
-rwxr-xr-x   1 hadoop hadoop   10254871 2021-01-11 05:55 /user/hive/CaseStudy/event_type=cart/000028_0
-rwxr-xr-x   1 hadoop hadoop    7172671 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000029_0
-rwxr-xr-x   1 hadoop hadoop   10452691 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000030_0
-rwxr-xr-x   1 hadoop hadoop    9367203 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000031_0
-rwxr-xr-x   1 hadoop hadoop    6937884 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000032_0
-rwxr-xr-x   1 hadoop hadoop   12346395 2021-01-11 05:55 /user/hive/CaseStudy/event_type=cart/000033_0
-rwxr-xr-x   1 hadoop hadoop   11051492 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000034_0
-rwxr-xr-x   1 hadoop hadoop    5106528 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000035_0
-rwxr-xr-x   1 hadoop hadoop   13248816 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000036_0
-rwxr-xr-x   1 hadoop hadoop    3095884 2021-01-11 05:55 /user/hive/CaseStudy/event_type=cart/000037_0
-rwxr-xr-x   1 hadoop hadoop    4743238 2021-01-11 05:55 /user/hive/CaseStudy/event_type=cart/000038_0
-rwxr-xr-x   1 hadoop hadoop    3279586 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000039_0
-rwxr-xr-x   1 hadoop hadoop    2075742 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000040_0
-rwxr-xr-x   1 hadoop hadoop    4702588 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000041_0
-rwxr-xr-x   1 hadoop hadoop    7542010 2021-01-11 05:55 /user/hive/CaseStudy/event_type=cart/000042_0
-rwxr-xr-x   1 hadoop hadoop    8322186 2021-01-11 05:55 /user/hive/CaseStudy/event_type=cart/000043_0
-rwxr-xr-x   1 hadoop hadoop    6536512 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000044_0
-rwxr-xr-x   1 hadoop hadoop    5040957 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000045_0
-rwxr-xr-x   1 hadoop hadoop    7014340 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000046_0
-rwxr-xr-x   1 hadoop hadoop    9180203 2021-01-11 05:55 /user/hive/CaseStudy/event_type=cart/000047_0
-rwxr-xr-x   1 hadoop hadoop    3918374 2021-01-11 05:55 /user/hive/CaseStudy/event_type=cart/000048_0
-rwxr-xr-x   1 hadoop hadoop    7688717 2021-01-11 05:54 /user/hive/CaseStudy/event_type=cart/000049_0
[hadoop@ip-172-31-93-152 ~]$
```

50 buckets have been created on the 'price' field for each partition. (00-49)

- Step 12: Check data in table:

```
SELECT  *
FROM   optimized_table
LIMIT   10;
```

```
hive> SELECT *
    > FROM optimized_table
    > LIMIT 10;
OK
2019-10-09 18:22:53 UTC 5622687 1487580007281722301            severina        1.81    411454474       07c1d0e5-86f3-4925-b8a3-2a92aa4b5210    cart
2019-10-09 12:19:37 UTC 5813313 1842735760499802745            kinetics        8.57    545556413       b0be43c1-1fdf-2c83-cc94-28001ce0e4a8    cart
2019-10-09 16:58:08 UTC 5783000 1638456119066100510            pole    3.70    557782796       cc13e0b1-f26d-499f-a115-5588bf6fb9c6    cart
2019-10-09 16:58:13 UTC 5813102 1842735760499802745            kinetics        8.57    558526472       c514843a-1bdd-483a-9c2e-43ca4c4c17a9    cart
2019-10-09 10:32:22 UTC 5622687 1487580007281722301            severina        1.81    555920023       06e1a6dc-ae2f-43ce-a654-885667b49395    cart
2019-10-10 11:17:06 UTC 5868459 1487580013069861041            italwax 3.25    520356921       9c116887-1efa-47d2-a875-4dd464fdd208    cart
2019-10-09 16:21:01 UTC 5556128 1487580011853512836                    8.57    495440704       d1775379-2c48-42c6-9607-d673b174f246    cart
2019-10-08 08:12:07 UTC 5851892 1542195323827388674            yu-r    33.33   557982045       276156e0-5261-4619-8033-3ac88ddae8d9    cart
2019-10-09 12:19:52 UTC 5813108 1842735760499802745            kinetics        8.57    545556413       b0be43c1-1fdf-2c83-cc94-28001ce0e4a8    cart
2019-10-08 05:52:04 UTC 5882380 1487580013338296510            bespecial       8.57    525189629       220076a5-afaa-4480-b459-ea3849fd8b28    cart
Time taken: 0.285 seconds, Fetched: 10 row(s)
```

First 10 records. *Took 0.285 s to fetch as opposed to 2.177s on the base table.*

❖ Running Hive Queries:

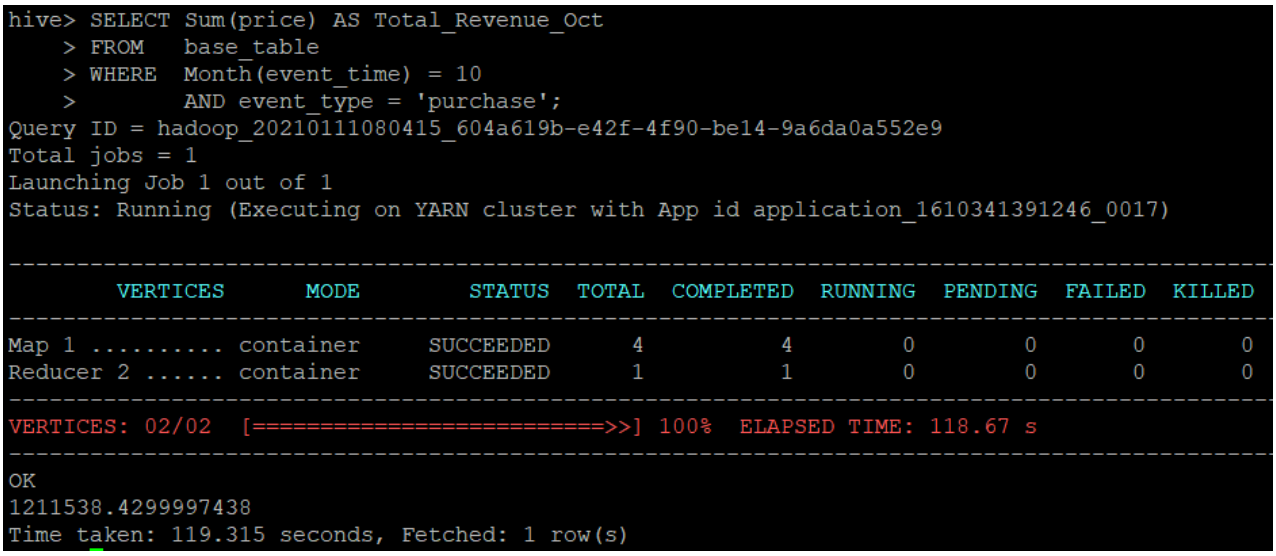
- Query 1: Find the total revenue generated due to purchases made in October:

- This is a good query for comparing performance of the base table and the optimized table as both 'event_type' as well as aggregation on 'price' are being done here.

Query on base_table:

```
SELECT   Sum(price) AS Total_Revenue_Oct
FROM     base_table
WHERE    Month(event_time) = 10
         AND event_type = 'purchase' ;
```

Screenshot:



Observations and Reasoning:

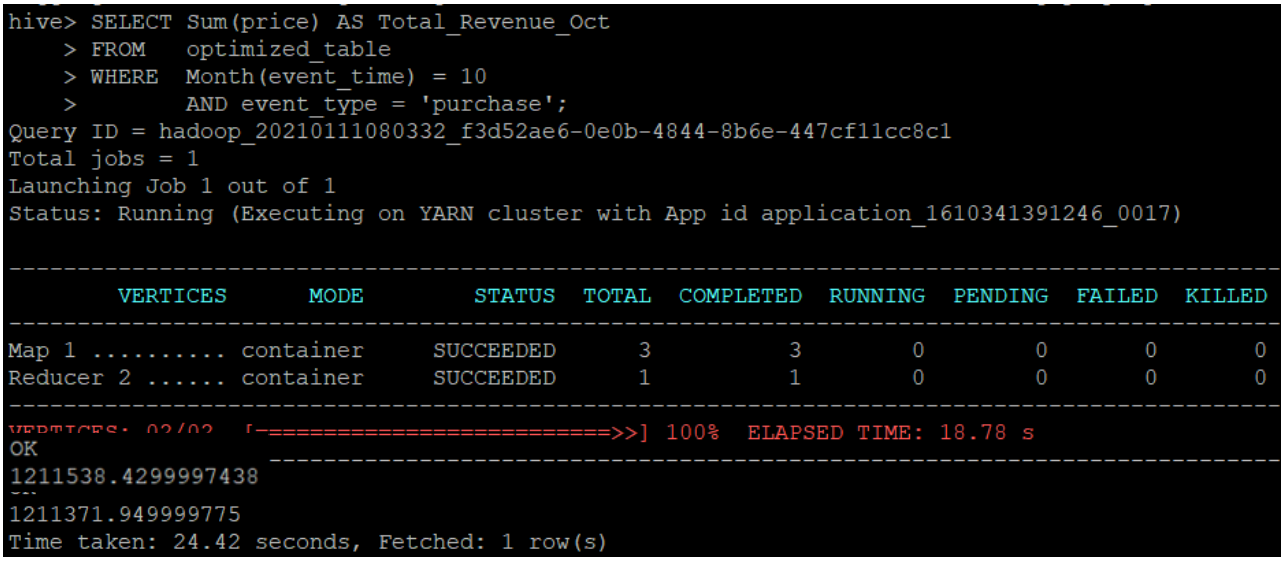
The total revenue generated due to purchases made in October is: 1211538.4299.

As we can see, the Map Reduce job has to run through the entire table to filter out purchase records from October and hence, the time taken is on the higher side.

Query on optimized_table:

```
SELECT   Sum(price) AS Total_Revenue_Oct
FROM     optimized_table
WHERE    Month(event_time) = 10
         AND event_type = 'purchase' ;
```

Screenshot:



Observations and Reasoning:

As expected the bucketing on 'price' field and the partitioning on the 'event_type' field helped speed up the querying by almost 7 folds. (18s vs 118s)

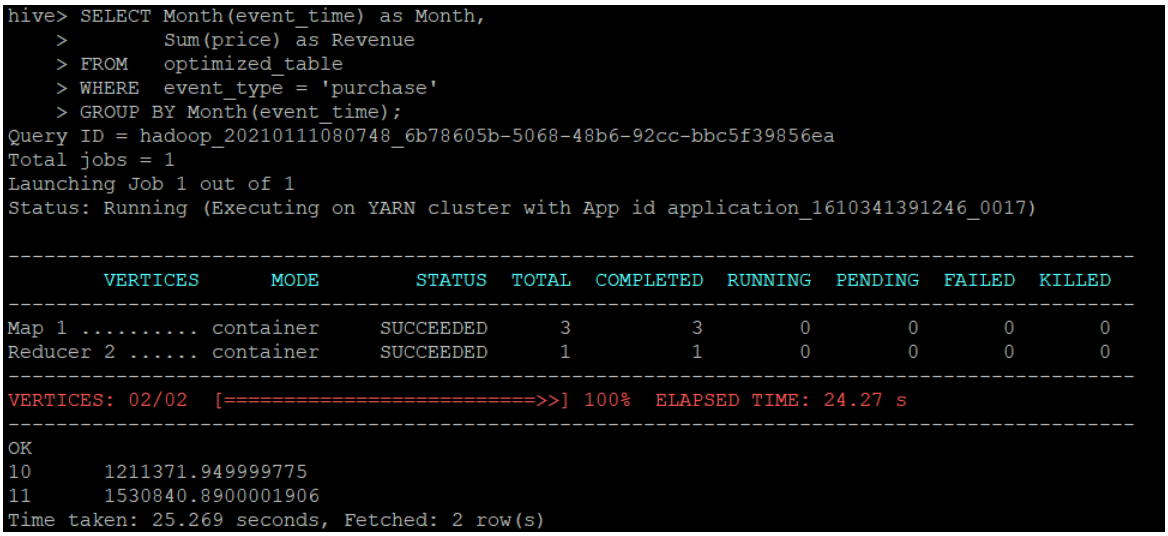
NOTE: As the performance on the optimized_table is much better, we will be running rest of our queries on this table itself.

- Query 2: Write a query to yield the total sum of purchases per month in a single output:

  Query:

```
SELECT   Month(event_time) as Month,
         Sum(price) as Revenue
FROM     optimized_table
WHERE    event_type = 'purchase'
GROUP BY Month(event_time);
```

  Screenshot:



Observations and Reasoning:

As seen, the total sum of purchases for October stand at 1211371.949 and November at 1530840.89.

We have used GROUP BY here to output the sums of both months together.

It can also be seen that the partitioning and bucketing have again come into effect here to keep the query execution time to just under 25s.

- Query 3: Write a query to find the change in revenue generated due to purchases from October to November:

  Query:

```
WITH revenue_cte AS
  (
SELECT Sum (CASE WHEN Month(event_time) = 10
                 THEN price
                 ELSE 0
                 END) AS OCT_SALES,

        Sum (CASE WHEN Month(event_time) = 11
                 THEN price
                 ELSE 0
                 END) AS NOV_SALES
FROM        optimized_table
WHERE     event_type = 'purchase'
)
SELECT NOV_SALES - OCT_SALES
FROM revenue_cte;
```

  Screenshot:

```
hive> WITH revenue_cte AS
    > (
    >
    > SELECT Sum (CASE
    >                 WHEN Month(event_time) = 10
    >                      THEN price
    >                 ELSE 0
    >             END) AS OCT_SALES,
    >        Sum (CASE
    >                 WHEN Month(event_time) = 11
    >                      THEN price
    >                 ELSE 0
    >             END) AS NOV_SALES
    >          FROM   optimized_table
    >          WHERE  event_type = 'purchase'
    > )
    >
    > SELECT NOV_SALES - OCT_SALES
    > FROM   revenue_cte;
Query ID = hadoop_20210111092005_4415c353-0c25-42fb-8436-1254d83427c7
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1610341391246_0019)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      3          3        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 23.62 s
--------------------------------------------------------------------------------
OK
319468.94000041555
Time taken: 24.228 seconds, Fetched: 1 row(s)
```

  Observations and Reasoning:

In this query, firstly we are using a CTE which contains 2 conditional sum variables, one sums prices only when month is October and the other sums only when its November, This is implemented using CASE. Once we have these 2 values, we are then going to query this CTE and retrieve the difference in the 2 calculated sums. Prices are summed only when 'event_type' is a purchase.

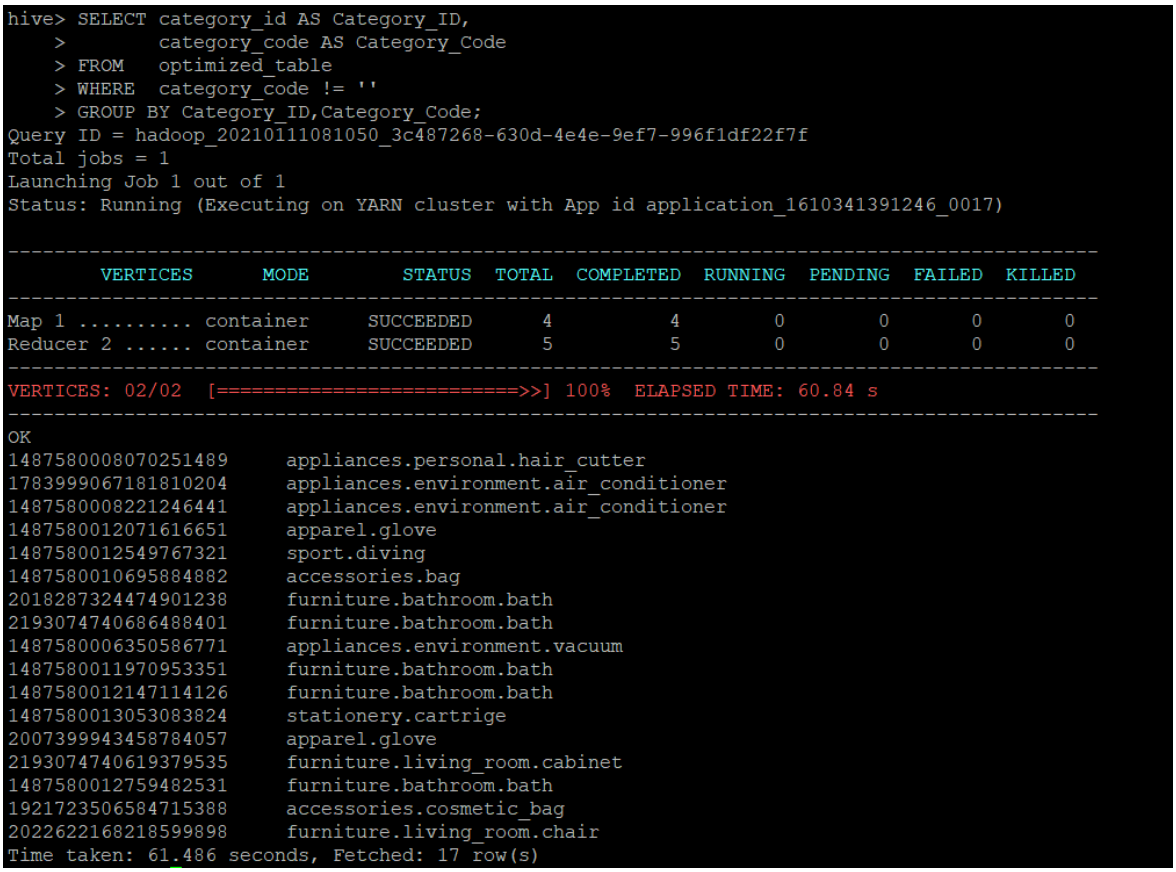
It can be seen that purchases in Nov have increased by an amount of 319468.94 over October.

- Query 4: Find distinct categories of products. Categories with null category code can be ignored:

  Query:

```
SELECT        category_id AS Category_ID,
              category_code AS Category_code
FROM          optimized_table
WHERE         category_code ! = “”
GROUP BY      category_id, category_code;
```

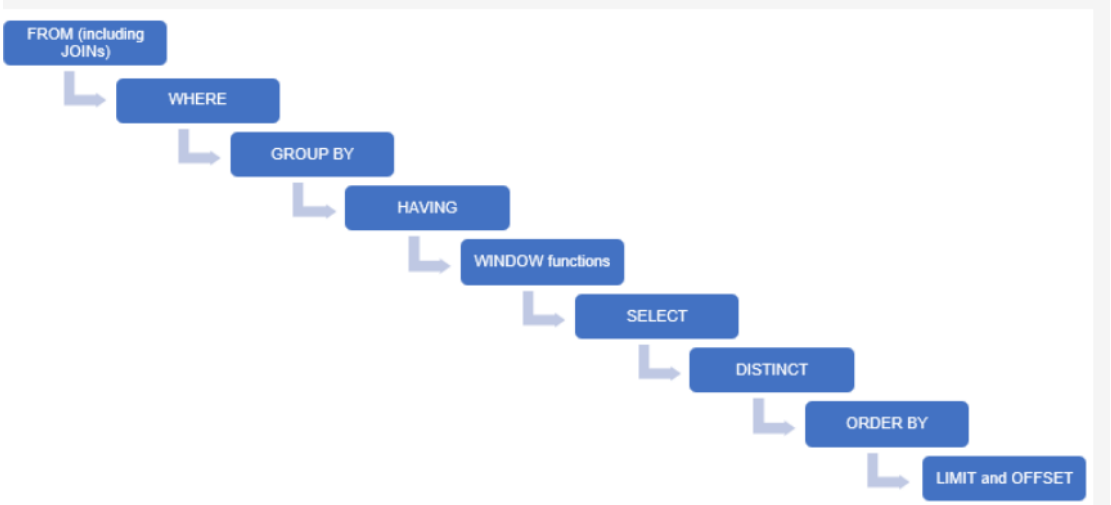
  Screenshot:



```
hive> SELECT category_id AS Category_ID,
    >        category_code AS Category_Code
    > FROM   optimized_table
    > WHERE  category_code != ''
    > GROUP BY Category_ID,Category_Code;
Query ID = hadoop_20210111081050_3c487268-630d-4e4e-9ef7-996f1df22f7f
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1610341391246_0017)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      4          4        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      5          5        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 60.84 s
--------------------------------------------------------------------------------
OK
1487580008070251489     appliances.personal.hair_cutter
1783999067181810204     appliances.environment.air_conditioner
1487580008221246441     appliances.environment.air_conditioner
1487580012071616651     apparel.glove
1487580012549767321     sport.diving
1487580010695884882     accessories.bag
2018287324474901238     furniture.bathroom.bath
2193074740686488401     furniture.bathroom.bath
1487580006350586771     appliances.environment.vacuum
1487580011970953351     furniture.bathroom.bath
1487580012147114126     furniture.bathroom.bath
1487580013053083824     stationery.cartrige
2007399943458784057     apparel.glove
2193074740619379535     furniture.living_room.cabinet
1487580012759482531     furniture.bathroom.bath
1921723506584715388     accessories.cosmetic_bag
2022622168218599898     furniture.living_room.chair
Time taken: 61.486 seconds, Fetched: 17 row(s)
```

  Observations and Reasoning:

  We have chosen not to use the DISTINCT function here on category_id as would usually be the general approach. Instead we're using GROUP BY to mimic the output of DISTINCT. The reason for this is:



Order of Query Execution

GROUP BY is executed much earlier as compared to DISTINCT, hence the time complexity of the query reduces (verified by executing query with 'distinct') and becomes more optimal.

17 distinct product categories exist.

- Query 5: Find the total number of products available under each category.

  Query:

```
SELECT        category_id AS Category_ID,
              category_code AS Category_Code,
              Count(product_id) AS Number_Of_Products
FROM          optimized_table
WHERE         category_code ! = ""
GROUP BY      category_id, category_code;
```

  Screenshot:

```
hive> SELECT category_id AS Category_ID,
    >        category_code AS Category_Code,
    >        Count(product_id) AS Number_of_Products
    > FROM   optimized_table
    > WHERE  category_code != ''
    > GROUP BY category_id, category_code;
Query ID = hadoop_20210111082407_af120db0-bf39-402b-97fd-ad19106b6446
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1610341391246_0018)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      6          6        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      5          5        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 64.01 s
--------------------------------------------------------------------------------
OK
1487580008070251489     appliances.personal.hair_cutter 1643
1783999067181810204     appliances.environment.air_conditioner  286
1487580008221246441     appliances.environment.air_conditioner  46
1487580012071616651     apparel.glove   162
1487580012549767321     sport.diving    2
1487580010695884882     accessories.bag 11681
2018287324474901238     furniture.bathroom.bath 1401
2193074740686488401     furniture.bathroom.bath 3712
1487580006350586771     appliances.environment.vacuum   59759
1487580011970953351     furniture.bathroom.bath 4128
1487580012147114126     furniture.bathroom.bath 367
1487580013053083824     stationery.cartrige     26720
2007399943458784057     apparel.glove   18070
2193074740619379535     furniture.living_room.cabinet   13438
1487580012759482531     furniture.bathroom.bath 249
1921723506584715388     accessories.cosmetic_bag        1248
2022622168218599898     furniture.living_room.chair     308
Time taken: 64.689 seconds, Fetched: 17 row(s)
```

Observations and Reasoning:

We are counting the product_ids under each category_id here.

We chose to display both category ID as well as the code as only ID doesn't tell much about what category the product belongs to.

- Query 6: Which brand had the maximum sales in October and November combined?

Query:

```
WITH brand_rank_cte AS
(
SELECT      brand,
            Sum (price) AS Sales,
            RANK () OVER (ORDER BY Sum (Price) DESC) AS brand_rank

FROM        optimized_table
WHERE       event_type = 'purchase'
AND         brand != ""
GROUP BY    brand
)
SELECT      brand, sales, brand_rank
FROM        brand_rank_cte
WHERE       brand_rank =1;
```

Screenshot:

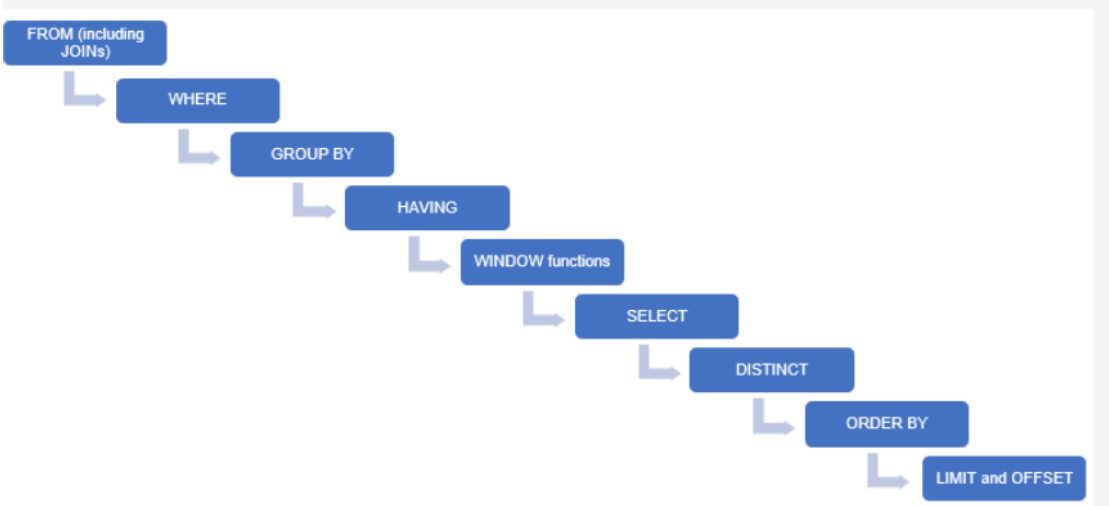
```
hive> WITH brand_rank_cte AS(
    >
    >      SELECT brand,
    >             Sum(price) AS Sales,
    >             rank() over(order by Sum(Price) desc) AS brand_rank
    >      FROM   optimized_table
    >      WHERE  event_type = 'purchase'
    >             AND brand != ''
    >      GROUP  BY brand
    >      )
    >
    >      select brand, sales, brand_rank
    >      from brand_rank_cte
    >      where brand_rank =1;
Query ID = hadoop_20210111082552_babdbdeb-acd3-4f3c-baa3-65ba9235be56
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1610341391246_0018)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      3          3        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 21.80 s
--------------------------------------------------------------------------------
OK
runail  148233.39999997825      1
Time taken: 22.571 seconds, Fetched: 1 row(s)
```

Observations and Reasoning:

Runail has the most sales both months combined at 148233.399.

A simple order brands by descending Sum(price) and then doing LIMIT 1 would also do the job, but again, LIMIT is executed last as compared to WINDOW functions - RANK() OVER- which are executed much earlier, hence we felt using the RANK() along with a CTE is a better approach.

Refer:



Order of Query Execution

- Query 7: Which brands increased their sales from October to November?

Query:

```
WITH brand_revenue_cte AS
(
SELECT     brand,
           Sum (CASE WHEN Month (event_time) = 10
           THEN price
           ELSE 0
           END) AS OCT_SALES,
           Sum (CASE WHEN Month (event_time) = 11
           THEN price
           ELSE 0
           END) AS NOV_SALES
FROM       optimized_table
WHERE      event_type = 'purchase'
GROUP BY   brand
)
SELECT     brand
FROM       brand_revenue_cte
WHERE      NOV_SALES > OCT_SALES;
```

Screenshot:

```
hive> WITH brand_revenue_cte AS
    > (
    >
    > SELECT brand,
    >        Sum (CASE
    >                WHEN Month(event_time) = 10
    >                     THEN price
    >                ELSE 0
    >             END) AS OCT_SALES,
    >        Sum (CASE
    >                WHEN Month(event_time) = 11
    >                     THEN price
    >                ELSE 0
    >             END) AS NOV_SALES
    > FROM   optimized_table
    > WHERE  event_type = 'purchase'
    > GROUP BY brand
    > )
    >
    > SELECT brand
    > FROM   brand_revenue_cte
    > WHERE  NOV_SALES > OCT_SALES;
Query ID = hadoop_20210111092734_5209033e-2fe2-4a5b-9adb-4f25adaa0ea7
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.
Status: Running (Executing on YARN cluster with App id application_1610341391246_0020)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      3          3        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 26.55 s
--------------------------------------------------------------------------------
OK

airnails
art-visage
nefertiti
neoleor
nirvel
nitrile
oniq
orly
osmo
ovale
plazan
polarus
profepil
profhenna
protokeratin
provoc
rasyan
refectocil
rosi
roubloff
runail
s.care
sanoto
severina
shary
shik
skinity
skinlite
smart
soleo
solomeya
sophin
staleks
strong
supertan
swarovski
tertio
treaclemoon
trind
uno
uskusi
veraclara
vilenta
yoko
yu-r
zeitun
Time taken: 34.285 seconds, Fetched: 161 row(s)
hive>
```

Observations and Reasoning:

In this query, firstly we are using a CTE which contains 2 conditional sum variables, one sums prices only when month is October and the other sums only when its November, This is implemented using CASE. Once we have these 2 values FOR EACH BRAND, we are then going to query this CTE and retrieve those brands whose NOV_SALES value is greater than their OCT_SALES value. Prices are summed only when 'event_type' is a purchase.

We can see totally 161 increased their revenue from Oct to Nov.

- Query 8: Your company wants to reward the top 10 users of its website with a Golden Customer plan. Write a query to generate a list of top 10 users who spend the most:

  Query:

```
WITH user_rank_cte AS
(
SELECT     user_id,
           Sum (price) AS Amount_Spent,
           RANK () OVER (ORDER BY Sum (price) DESC) AS user_rank
FROM       optimized_table
WHERE      event_type = 'purchase'
GROUP BY   user_id
)
SELECT     user_rank, user_id, Amount_Spent
FROM       user_rank_cte
WHERE      user_rank <=10;
```

  Screenshot:

```
hive> WITH user_rank_cte as(
    >
    >      SELECT user_id, Sum(price) AS Amount_Spent,
    >             rank() over(order by Sum(price) desc) as user_rank
    >      FROM   optimized_table
    >      WHERE  event_type = 'purchase'
    >      GROUP  BY user_id
    >
    >      )
    >      SELECT user_rank, user_id, Amount_Spent
    >      FROM   user_rank_cte
    >      WHERE  user_rank <=10;
Query ID = hadoop_20210111083049_759bcab0-de21-4641-8954-ce0129b7c9e8
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1610341391246_0018)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      3          3        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 24.96 s
--------------------------------------------------------------------------------
OK
1       557790271       2715.8699999999963
2       150318419       1645.9700000000003
3       562167663       1352.85
4       531900924       1329.45
5       557850743       1295.48
6       522130011       1185.3900000000008
7       561592095       1109.7000000000003
8       431950134       1097.5899999999997
9       566576008       1056.3600000000001
10      521347209       1040.91
Time taken: 25.544 seconds, Fetched: 10 row(s)
hive>
```

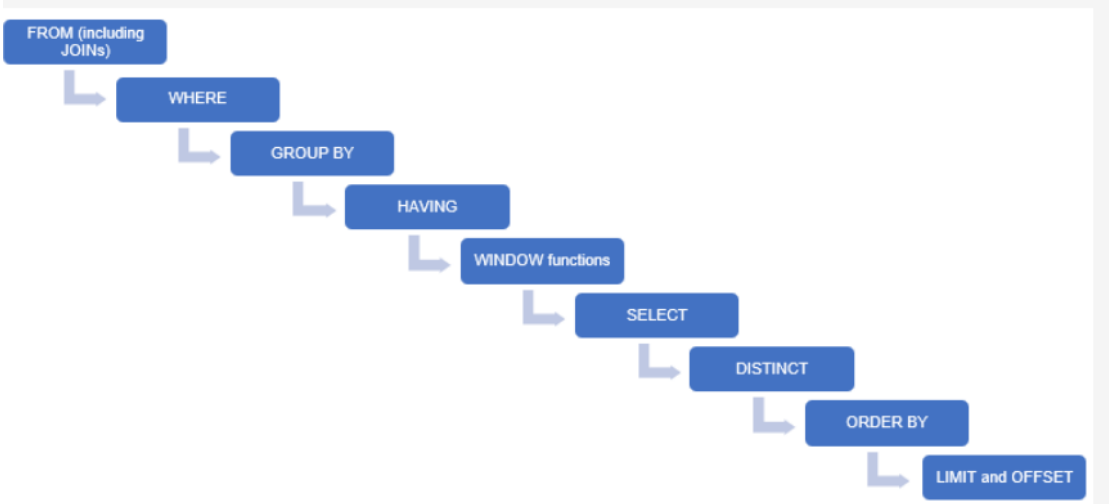
  Observations and Reasoning:

  The Top 10 users are listed as above.

  Same as before, a simple order users by descending Sum(price) and then doing LIMIT 10 would also do the job, but again, LIMIT is executed last as compared to WINDOW functions - RANK()

OVER- which are executed much earlier, hence we felt using the RANK() along with a CTE is a better approach.

Refer:



Order of Query Execution

❖ Clean-up : Last Steps:

- Drop the tables:

```
hive> drop table base_table;
OK
Time taken: 0.274 seconds
hive> drop table optimized_table;
OK
Time taken: 0.171 seconds
hive> show tables;
OK
Time taken: 0.014 seconds
```

No more tables left.

- Drop the database:

```
hive> drop database if exists cs_db;
OK
Time taken: 0.23 seconds
hive> show databases;
OK
default
Time taken: 0.016 seconds, Fetched: 1 row(s)
```

Our database is dropped.

- Terminate the database: Go to the Cluster's page -> Click on Terminate -> Disable the Termination protection -> Terminate the cluster.

Clone | Terminate | AWS CLI export

Cluster: Hive_CS_Cluster Terminating Terminated by user request

Summary | Application user interfaces | Monitoring | Hardware | Configurations | Events | Steps | Bootstrap actions

Summary

ID: j-1GMZAFOYIRPYG
Creation date: 2021-01-11 10:26 (UTC+5:30)
Elapsed time: 4 hours, 47 minutes
After last step completes: Cluster waits
Termination protection: Off
Tags: --
Master public DNS: ec2-52-70-253-13.compute-1.amazonaws.com
Connect to the Master Node Using SSH

Configuration details

Release label: emr-5.29.0
Hadoop distribution: Amazon 2.8.5
Applications: Hive 2.3.6, Pig 0.17.0, Hue 4.4.0
Log URI: s3://aws-logs-211054514628-us-east-1/elasticmapreduce/
EMRFS consistent view: Disabled
Custom AMI ID: --

Application user interfaces

Persistent user interfaces: --
On-cluster user interfaces: --

Network and hardware

Availability zone: us-east-1c
Subnet ID: subnet-176de636
Master: Terminating 1 m4.large
Core: Terminating 1 m4.large
Task: --
Cluster scaling: Not enabled

Security and access

Key name: Hive_CS
EC2 instance profile: s3_full_access
EMR role: EMR_DefaultRole
Auto Scaling role: EMR_AutoScaling_DefaultRole
Visible to all users: All Change
Security groups for Master: sg-006c337651c047217 (ElasticMapReduce-master)
Security groups for Core & Task: sg-02c612507f4eafb95 (ElasticMapReduce-slave)

THANK YOU ☺