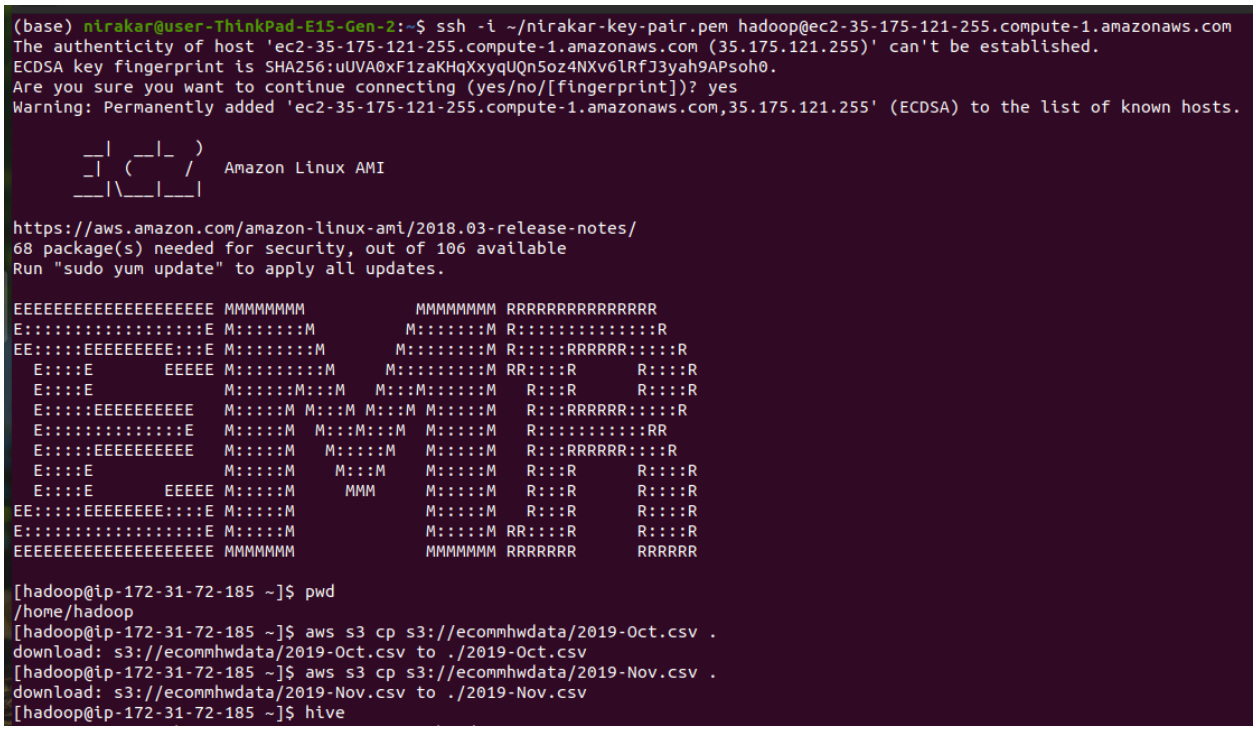
HIVE ECOMM ASSIGNMENT - Nirakar Padhy

nirakarpadhy001@gmail.com

Launching and connecting to an EMR cluster via SSH:
And copying the 2019-Oct.csv and 2019-Nov.csv files to hadoop



Run following Commands on Hadoop EMR:

aws s3 cp <s3_path_to_dataset_bucket> <instance_path_.>

hive # launching hive engine (can also use beeline connector)

Run following Commands on HIVE:

#we could use MR or TEZ exec engine: we use default taz engine

create database ecomm_hw; #creating new database
use ecomm_hw; #using the created db

#enabling dynamic partition for partitioning table using event_type for query optimisation

set hive.exec.dynamic.parition=true;
set hive.exec.dynamic.parition.mode=nonstrict;

#creating two tables for 2019-Oct.csv and 2019-Nov.csv
#can be stored as textfile, parquet or avro format : for simplicity we use textfile
CREATE TABLE table_oct (event_time string, event_type string, product_id string, category_id string, category_code string, brand string, price float, user_id bigint, user_session string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' stored as textfile;

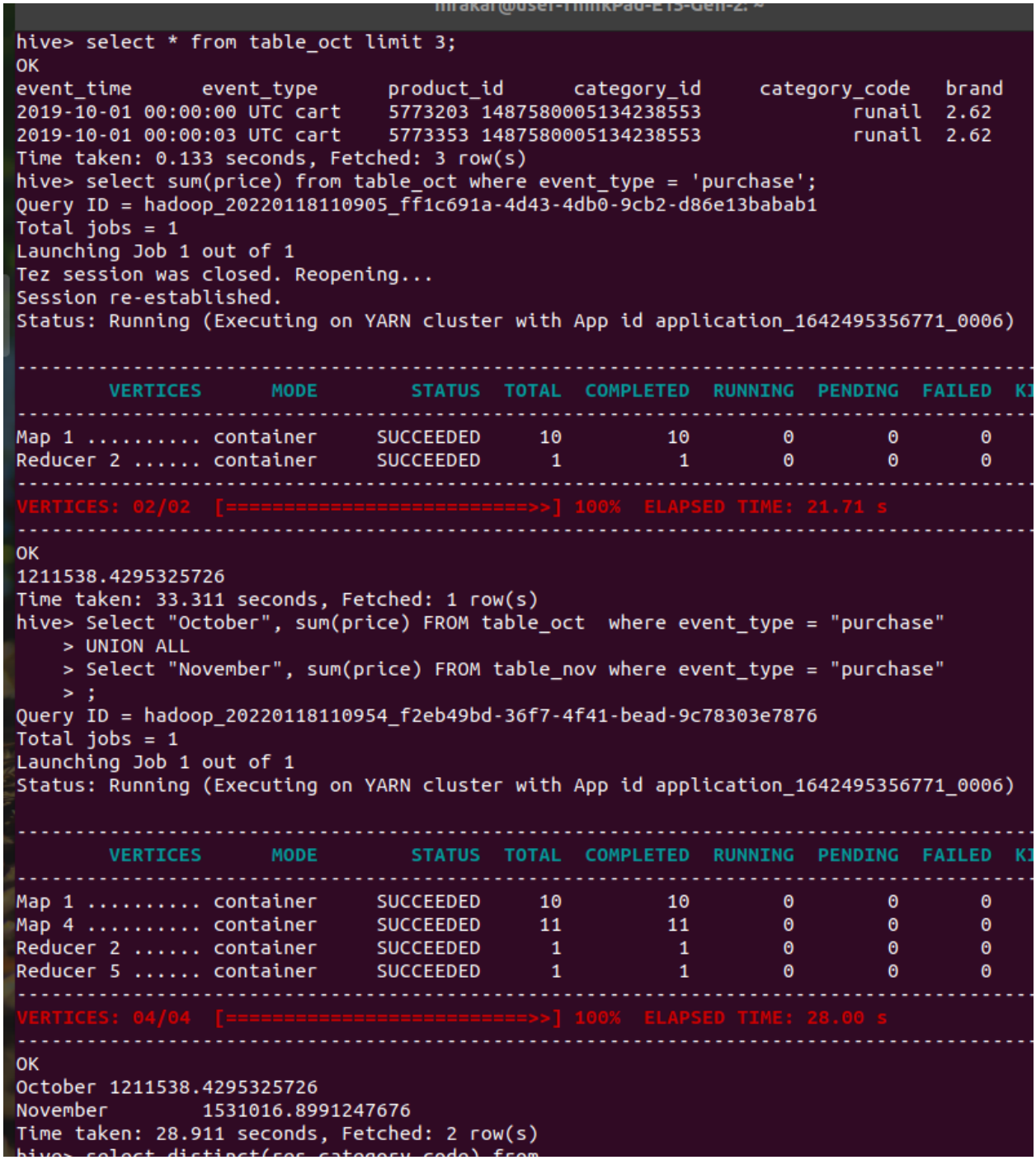
CREATE TABLE table_nov (event_time string, event_type string, product_id string, category_id string, category_code string, brand string, price float, user_id bigint, user_session string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' stored as textfile;

#loading the data from hdfs to hive tables
load data local inpath '/home/hadoop/2019-Oct.csv' into table table_oct;
load data local inpath '/home/hadoop/2019-Nov.csv' into tabletable_nov;

<u>You are required to provide answers to the questions given below.</u>

1. Find the total revenue generated due to the purchases made in October.

select sum(price) from table_oct where event_type = 'purchase';



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

Select "October", sum(price) FROM table_oct where event_type = "purchase"
UNION ALL
Select "November", sum(price) FROM table_nov where event_type = "purchase";

3. Write a query to find the change in the revenue generated due to purchases made from October to November.

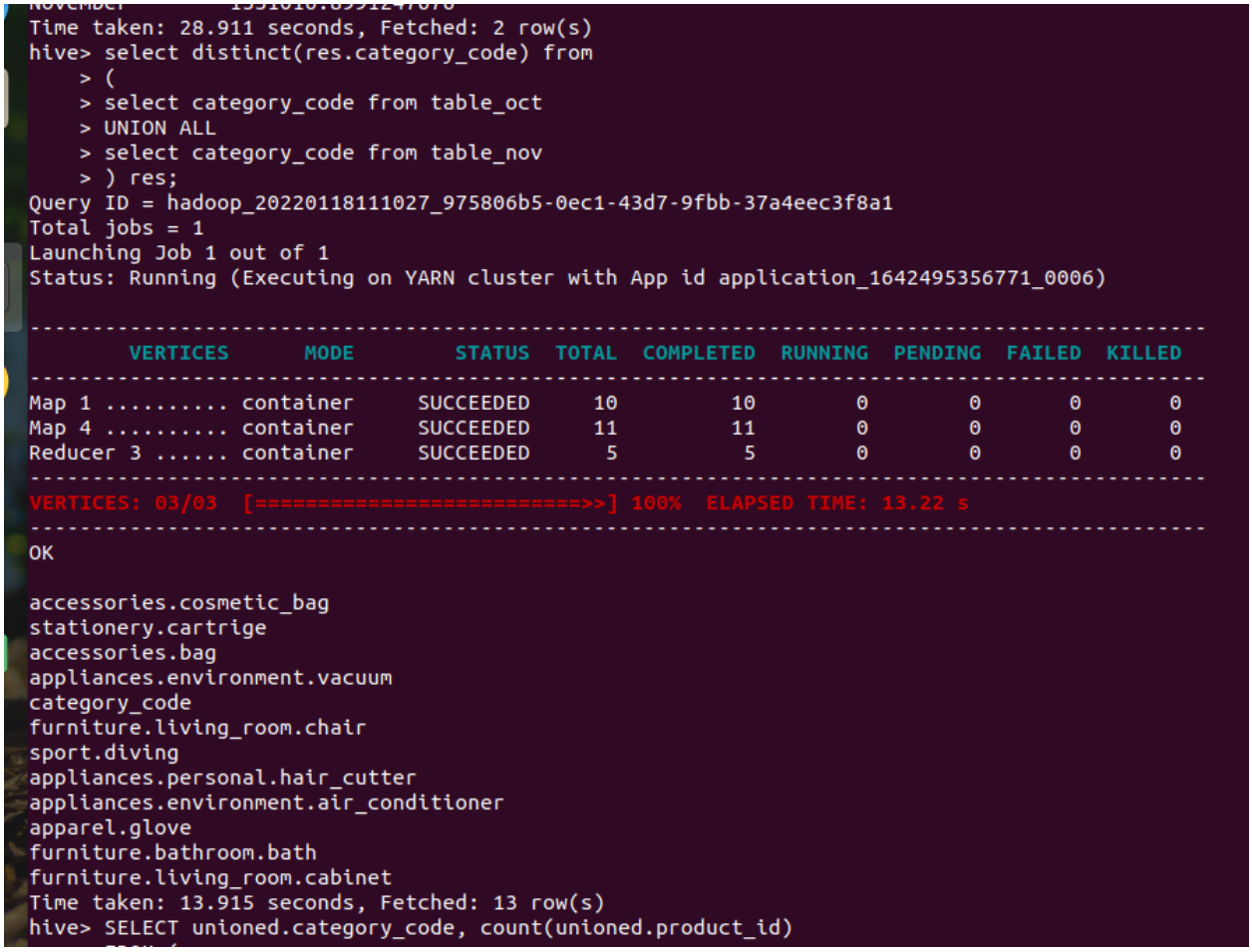
SELECT sum(nov_rev-oct_rev) as Difference
FROM (
select sum(price) as oct_rev, 0 as nov_rev FROM table_oct where event_type = "purchase" UNION ALL
select 0 as oct_rev, sum(price) as nov_rev FROM table_nov where event_type = "purchase") unioned;

```
hive> SELECT sum(nov_rev-oct_rev) as Difference
    > FROM (
    > select sum(price) as oct_rev, 0 as nov_rev FROM table_oct where event_type = "purchase" UNION ALL
    > select 0 as oct_rev, sum(price) as nov_rev FROM table_nov where event_type = "purchase" ) unioned;
Query ID = hadoop_20220119140445_6e023ec6-e2b2-45db-a8d4-6f7a94bd389b
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1642599511867_0004)

--------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED     10         10        0        0       0       0
Map 5 .......... container     SUCCEEDED     10         10        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
Reducer 4 ...... container     SUCCEEDED      1          1        0        0       0       0
Reducer 6 ...... container     SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 05/05  [==========================>>] 100%  ELAPSED TIME: 22.28 s
--------------------------------------------------------------------------------
OK
319478.469592195
Time taken: 24.078 seconds, Fetched: 1 row(s)
hive>
```

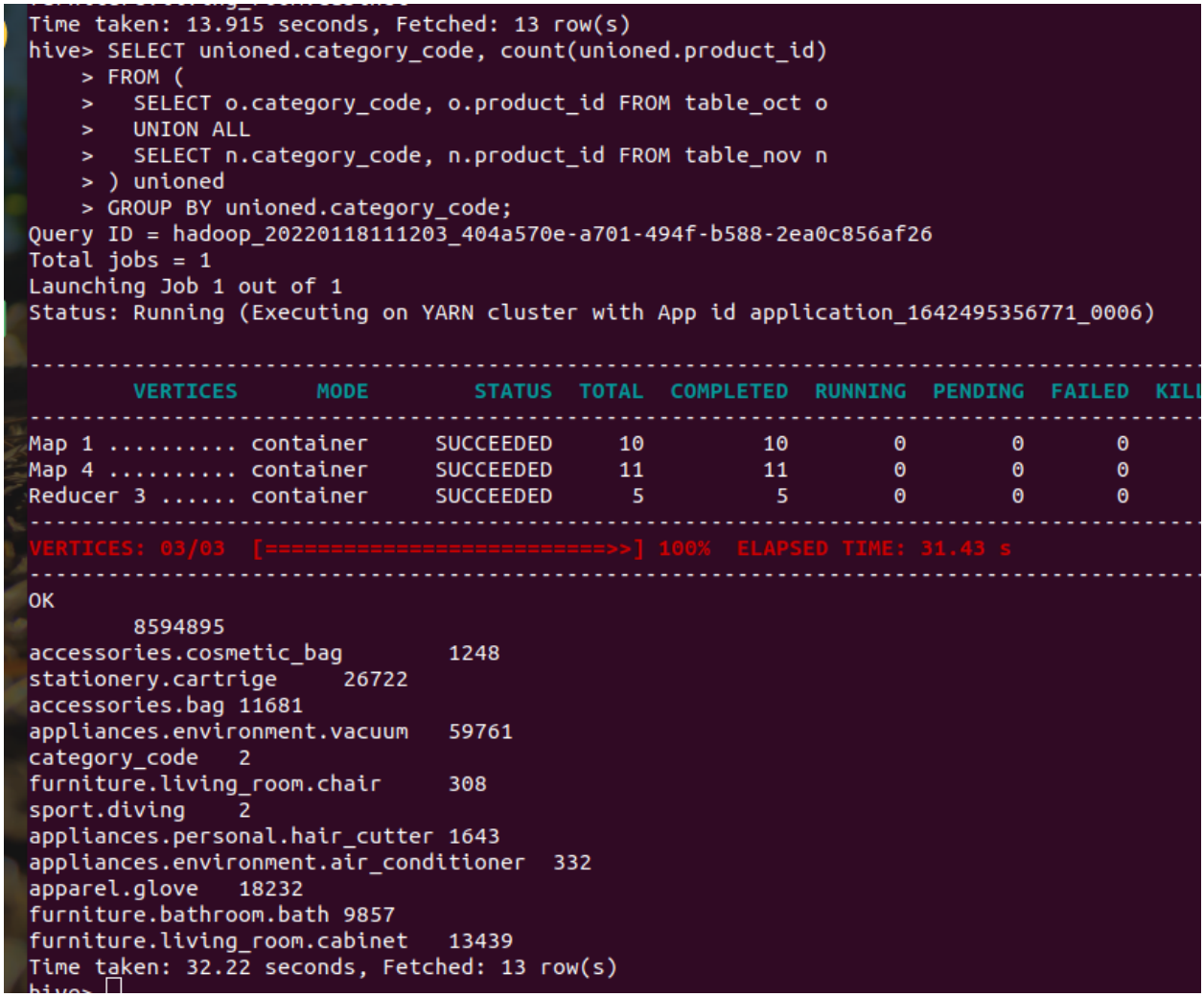
4. Find distinct categories of products.

select distinct(res.category_code) FROM (select category_code from table_oct UNION ALL select category_code from table_nov) res;



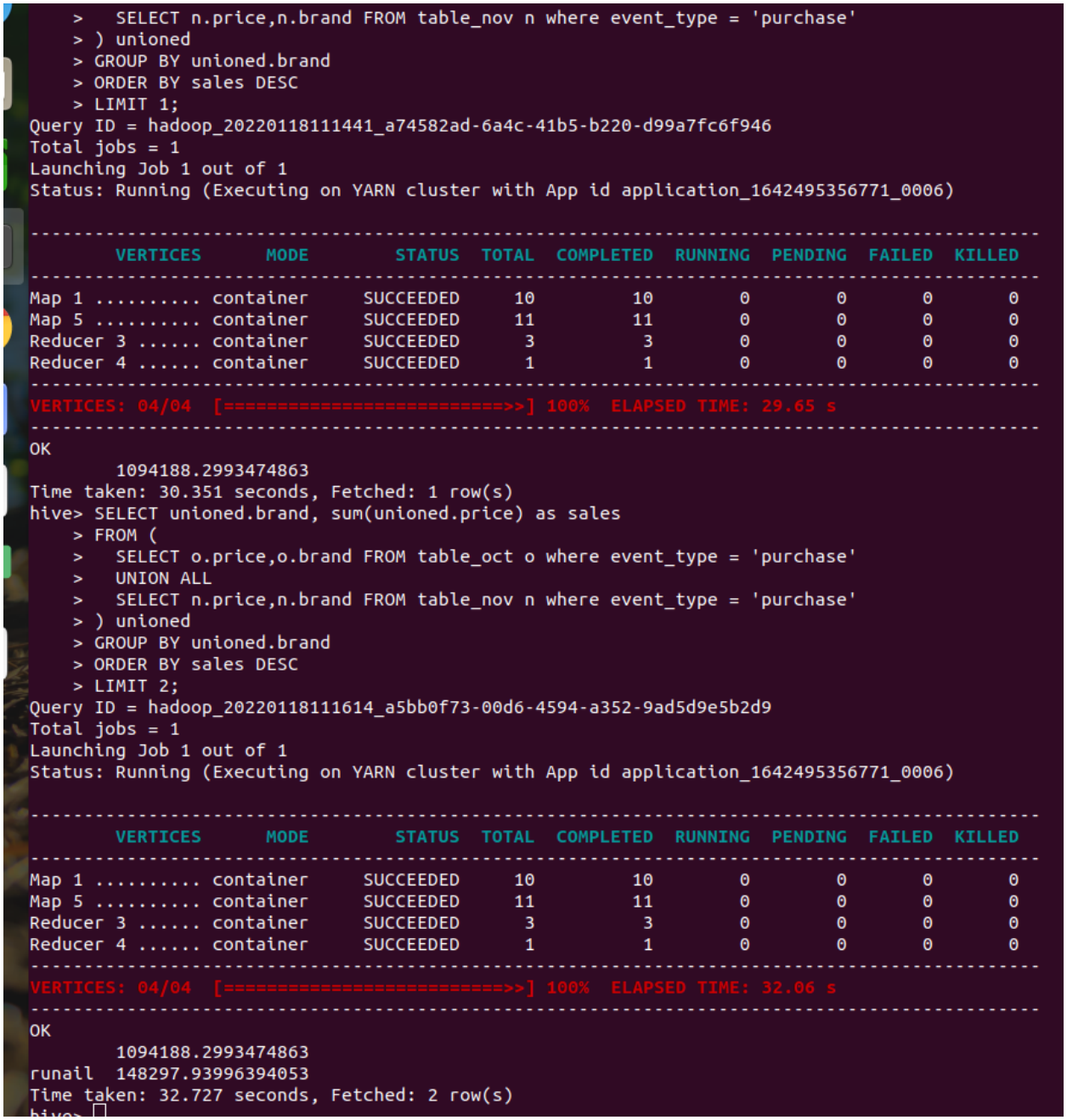
5. Find the total number of products available under each category.

SELECT unioned.category_code, count(unioned.product_id)
FROM (
SELECT o.category_code, o.product_id FROM table_oct o
UNION ALL
SELECT n.category_code, n.product_id FROM table_nov n
) unioned
GROUP BY unioned.category_code;



6. Which brand had the maximum sales in October and November combined?

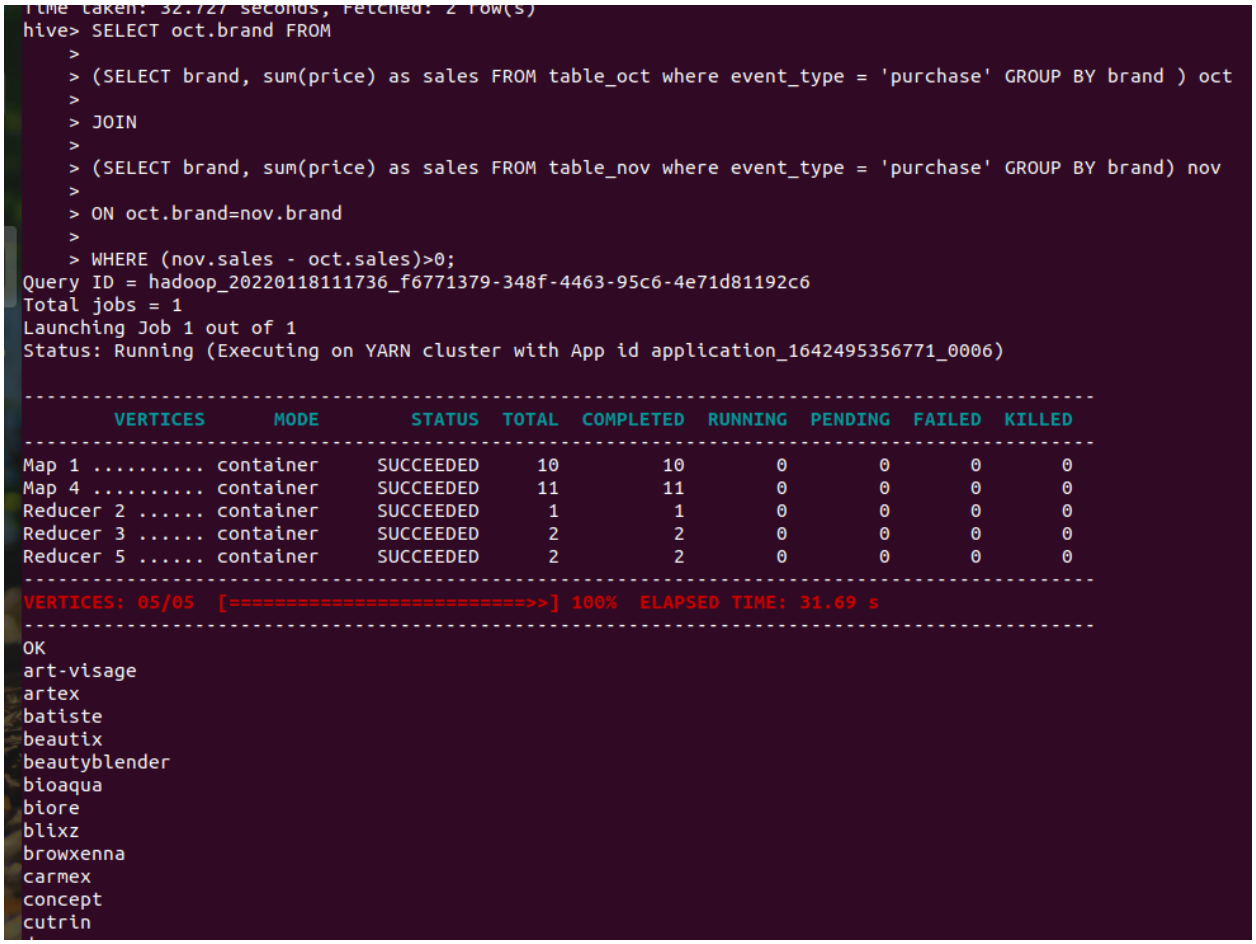
SELECT unioned.brand, sum(unioned.price) as sales
FROM (
SELECT o.price,o.brand FROM table_oct o where event_type = 'purchase'
UNION ALL
SELECT n.price,n.brand FROM table_nov n where event_type = 'purchase'
) unioned
GROUP BY unioned.brand
ORDER BY sales DESC
LIMIT 1;



7. Which brands increased their sales from October to November?

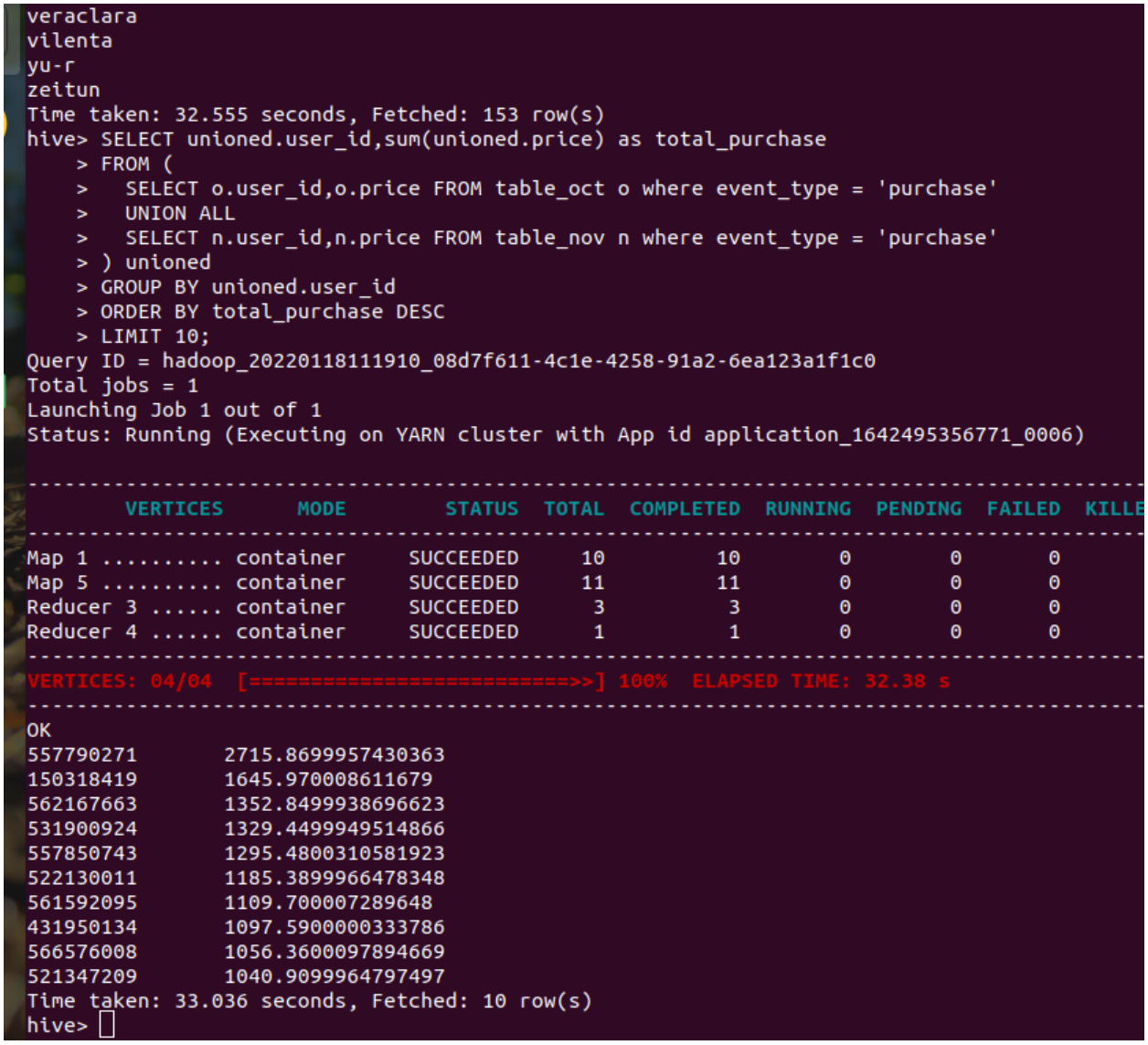
SELECT oct.brand FROM
(SELECT brand, sum(price) as sales FROM table_oct where event_type = 'purchase' GROUP BY brand) oct
JOIN
(SELECT brand, sum(price) as sales FROM table_nov where event_type = 'purchase' GROUP BY brand) nov
ON oct.brand=nov.brand

WHERE (nov.sales - oct.sales)>0;



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 on purchases.

SELECT unioned.user_id,sum(unioned.price) as total_purchase
FROM (
SELECT o.user_id,o.price FROM table_oct o where event_type = 'purchase'
UNION ALL
SELECT n.user_id,n.price FROM table_nov n where event_type = 'purchase'
) unioned
GROUP BY unioned.user_id
ORDER BY total_purchase DESC
LIMIT 10;



OPTIMISATION OF QUERIES:

We can optimize queries by choosing appropriate storage format for tables. Additionally we can use methods such as indexing, partitioning, bucketing, etc. to make our queries faster.

Indexing: When the data set is large in size and a faster query execution is required we can use indexing on columns that are used more frequently. It is wise to use indexing for read-heavy applications, where you need to read the data more frequently and not write heavy-applications.

Hive Partition is a way to organize large tables into smaller logical tables based on values of columns; one logical table (partition) for each distinct value. In Hive, tables are created as a directory on HDFS. A table can have one or more partitions that correspond to a sub-directory for each partition inside a table directory. Hive Bucketing a.k.a (Clustering) is a technique to split the data into more manageable files, (By specifying the number of buckets to create). The value of the bucketing column will be hashed by a user-defined number into buckets. Bucketing can be created on just one column, you can also create bucketing on a partitioned table to further split the data which further improves the query performance of the partitioned table. Each bucket is stored as a file within the table's directory or the partitions directories. Note that partition creates a directory and you can have a partition on one or more columns; these are some of the differences between Hive partition and bucket.

As the size of data increases, partitioning and bucketing yields results much quicker than unoptimised queries.

```
create table table_oct_part (event_time string, product_id string, category_id string,category_code string,brand string, price float, user_id bigint, user_session string) PARTITIONED BY (event_type string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' stored as avro;

insert into table table_oct_part partition(event_type) select event_time, event_type, product_id, category_id, category_code, brand, price, user_id, user_session from table_oct;

create table table_nov_part (event_time string, product_id string, category_id string,category_code string,brand string, price float, user_id bigint, user_session string) PARTITIONED BY (event_type string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' stored as avro;

insert into table table_nov_part partition(event_type) select event_time, event_type, product_id, category_id, category_code, brand, price, user_id, user_session from table_nov;
```

Optimized Query Q8:

```
SELECT unioned.user_id,sum(unioned.price) as total_purchase
FROM (
  SELECT o.user_id,o.price FROM table_oct_part o where event_type = 'purchase'
  UNION ALL
  SELECT n.user_id,n.price FROM table_nov_part n where event_type = 'purchase'
) unioned
GROUP BY unioned.user_id
ORDER BY total_purchase DESC
LIMIT 10;
```

It takes 21.245 secs to run the above query vis-a-vis 33.036 secs in the previous case!