**Hive Assignment**

Step 1 : Get data files in Master node

wget <https://e-commerce-events-ml.s3.amazonaws.com/2019-Oct.csv>

wget <https://e-commerce-events-ml.s3.amazonaws.com/2019-Nov.csv>

Step 2 : remove UTC from the timestamp column before loading the data

sed -i -e 's/ UTC//g' 2019-Oct.csv

sed -i -e 's/ UTC//g' 2019-Nov.csv

Step 3 : Import csv files in HDFS by running following steps

hdfs dfs -mkdir /user/hive\_assignment

hdfs dfs -put 2019-Nov.csv /user/hive\_assignment/

hdfs dfs -put 2019-Oct.csv /user/hive\_assignment/

Step 4 : Verify if the files are imported in HDFS succesfully.

hdfs dfs -ls /user/hive\_assignment/

hdfs dfs -cat /user/hive\_assignment/2019-Nov.csv | head

hdfs dfs -cat /user/hive\_assignment/2019-Oct.csv | head

Step 5 : Enter into Hive console by typing ‘hive’

Step 6 : Create database and required tables using following commands

create database if not exists ECom comment "Created to store events done on website" with dbproperties('creator'='Prakash','date'='30-01-2021');

show databases;

use ecom;

create external table if not exists event\_nov (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 delimited fields terminated by ',' lines terminated by '\n' stored as textfile tblproperties("skip.header.line.count"="1") ;

create external table if not exists event\_oct (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 delimited fields terminated by ',' lines terminated by '\n' stored as textfile tblproperties("skip.header.line.count"="1") ;

show tables;

Step 7 : Load data to both tables created for October and November

load data inpath '/user/hive\_assignment/2019-Nov.csv' into table event\_nov;

load data inpath '/user/hive\_assignment/2019-Oct.csv' into table event\_oct;

set hive.cli.print.header=true;

select \* from event\_nov limit 10;

select \* from event\_nov limit 10;

Step 8 : Consolidate the data of October and November in a single table

create table events

as

(

select 'Nov' as Month,\* from event\_nov

union all

select 'Oct' as Month,\* from event\_oct

)

Step 9 : Create partitioning(Static partitioning was used) and bucketing. (I have chosen month and event\_type for partitioning and product\_id for bucketing. I chose this because most of the queries were based on event type and month. Both columns were having low cardinality)

create table if not exists part\_events (event\_time timestamp,product\_id string,category\_id string,category\_code string,brand string,price float,user\_id bigint,user\_session string) partitioned by (month string,event\_type string) clustered by (product\_id) into 5 buckets row format delimited fields terminated by ',' lines terminated by '\n';

insert into part\_events partition( month='Oct', event\_type='cart') select event\_time,product\_id,category\_id,category\_code,brand,price,user\_id, user\_session from events where month='Oct' and event\_type='cart';

insert into part\_events partition( month='Oct', event\_type='remove\_from\_cart') select event\_time,product\_id,category\_id,category\_code,brand,price,user\_id, user\_session from events where month='Oct' and event\_type='remove\_from\_cart';

insert into part\_events partition( month='Oct' ,event\_type='view') select event\_time,product\_id,category\_id,category\_code,brand,price,user\_id, user\_session from events where month='Oct' and event\_type='view';

insert into part\_events partition( month='Oct' ,event\_type='purchase') select event\_time,product\_id,category\_id,category\_code,brand,price,user\_id, user\_session from events where month='Oct' and event\_type='purchase';

insert into part\_events partition( month='Nov', event\_type='cart') select event\_time,product\_id,category\_id,category\_code,brand,price,user\_id, user\_session from events where month='Nov' and event\_type='cart';

insert into part\_events partition(month='Nov',event\_type='remove\_from\_cart') select event\_time,product\_id,category\_id,category\_code,brand,price,user\_id, user\_session from events where month='Nov' and event\_type='remove\_from\_cart';

insert into part\_events partition(month='Nov',event\_type='view') select event\_time,product\_id,category\_id,category\_code,brand,price,user\_id, user\_session from events where month='Nov' and event\_type='view';

insert into part\_events partition(month='Nov',event\_type='purchase') select event\_time,product\_id,category\_id,category\_code,brand,price,user\_id, user\_session from events where month='Nov' and event\_type='purchase';

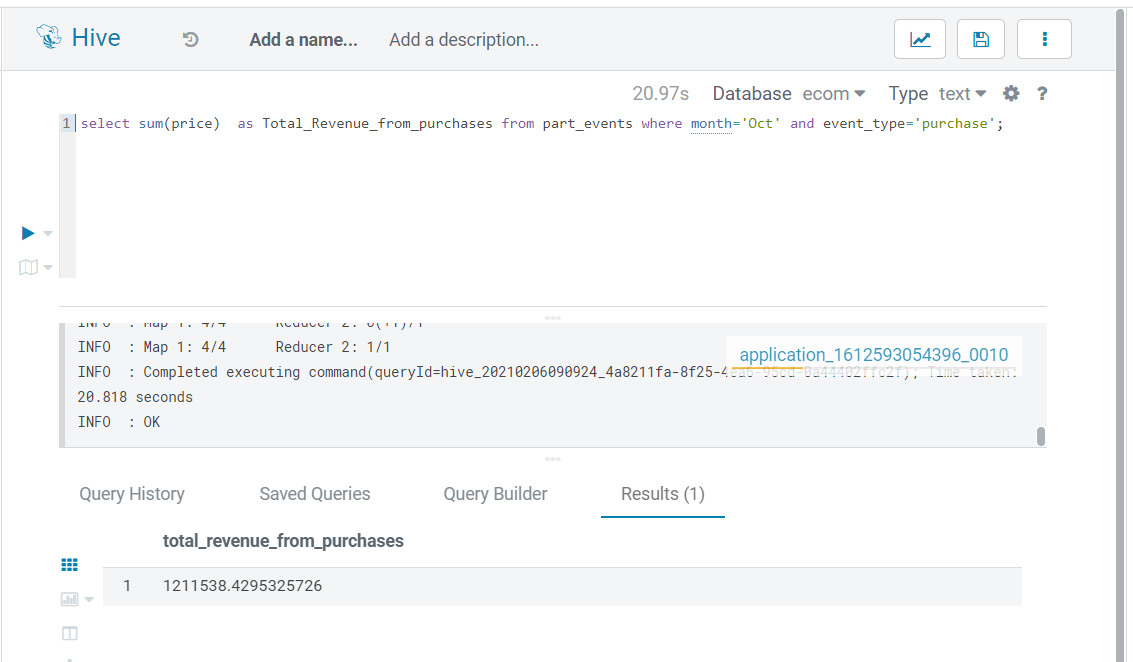
Step 10 : Verify if partitioning and bucketing is done

hdfs dfs -ls /user/hive/warehouse/ecom.db/part\_events

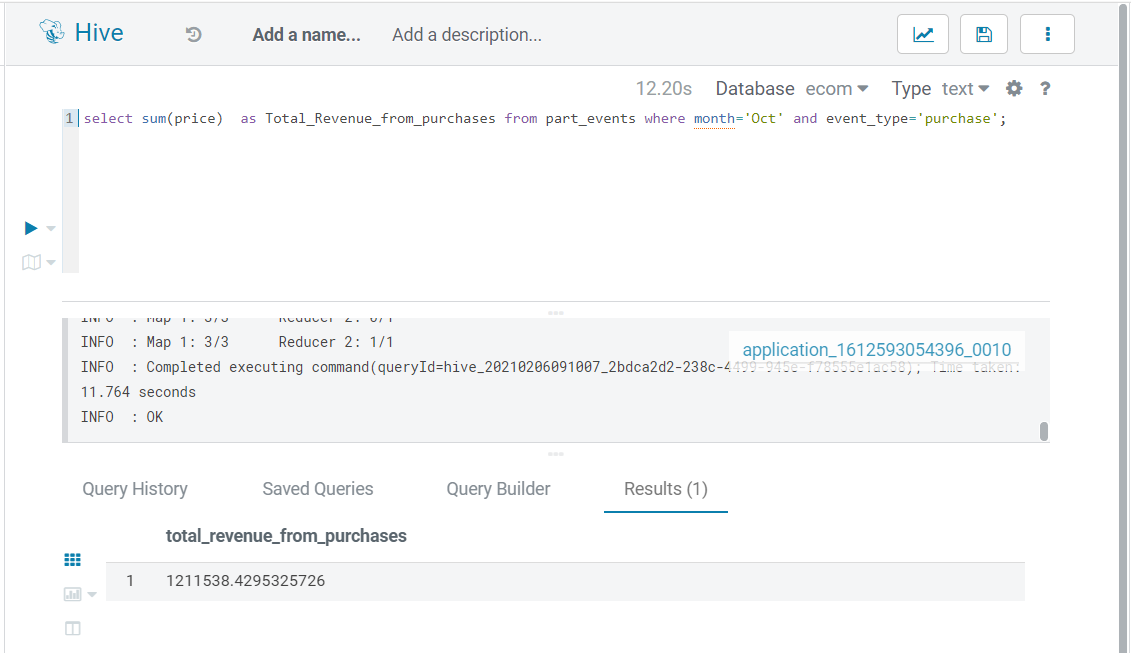
Step 11 : Run a test if the query will be optimized after partitioning.(I have checked the performance for question #1 )

I have queried the events table directly which is not partitioned and it is taking 21 seconds. However when I used the partitioned table it is giving result in 12 seconds. There is a improvement of 43 percent. Please find the screenshot below.

select sum(price) as Total\_Revenue\_from\_purchases from events where month='Oct' and event\_type='purchase';



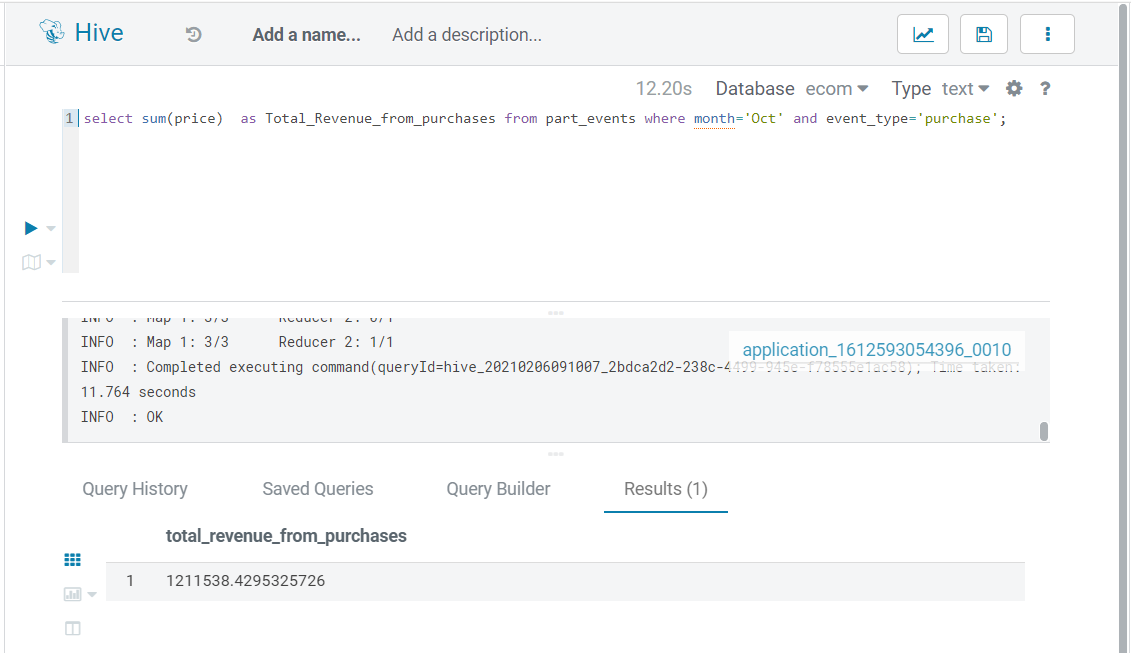
select sum(price) as Total\_Revenue\_from\_purchases from part\_events where month='Oct' and event\_type='purchase';



Step 12 : Answer the Questions asked. Please find below the Answers of all queries/questions.

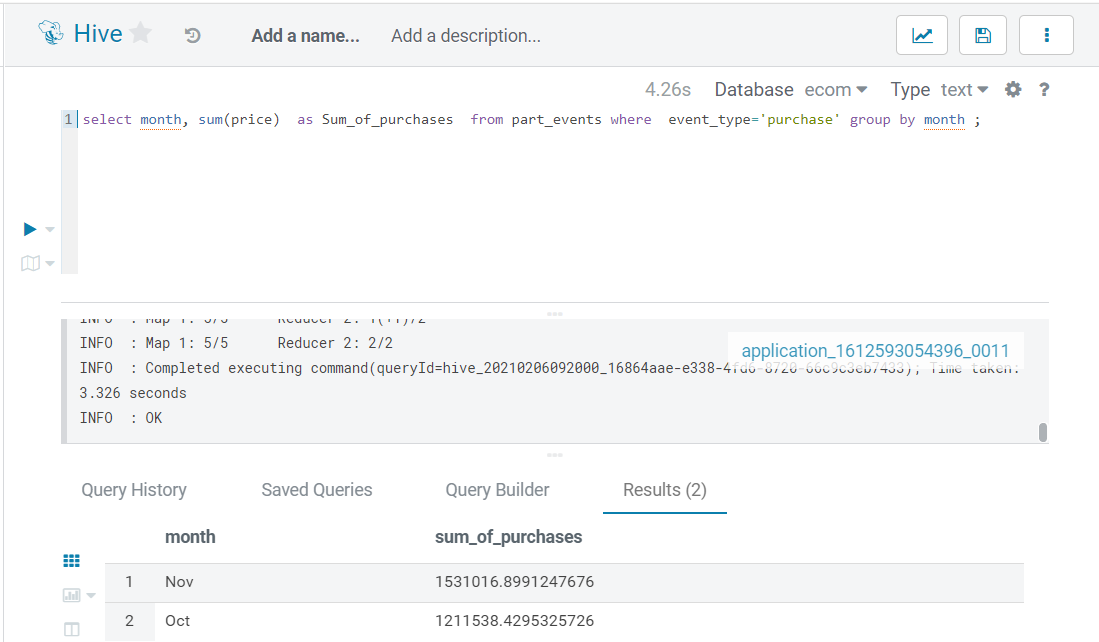
1. Find the total revenue generated due to the purchases made in October.

select sum(price) as Total\_Revenue\_from\_purchases from part\_events where month='Oct' and event\_type='purchase';



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

select month, sum(price) as Sum\_of\_purchases from part\_events where event\_type='purchase' group by month ;



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

with MonthlyRevenue as

(select month, sum(price) as Revenue\_purchases

from part\_events

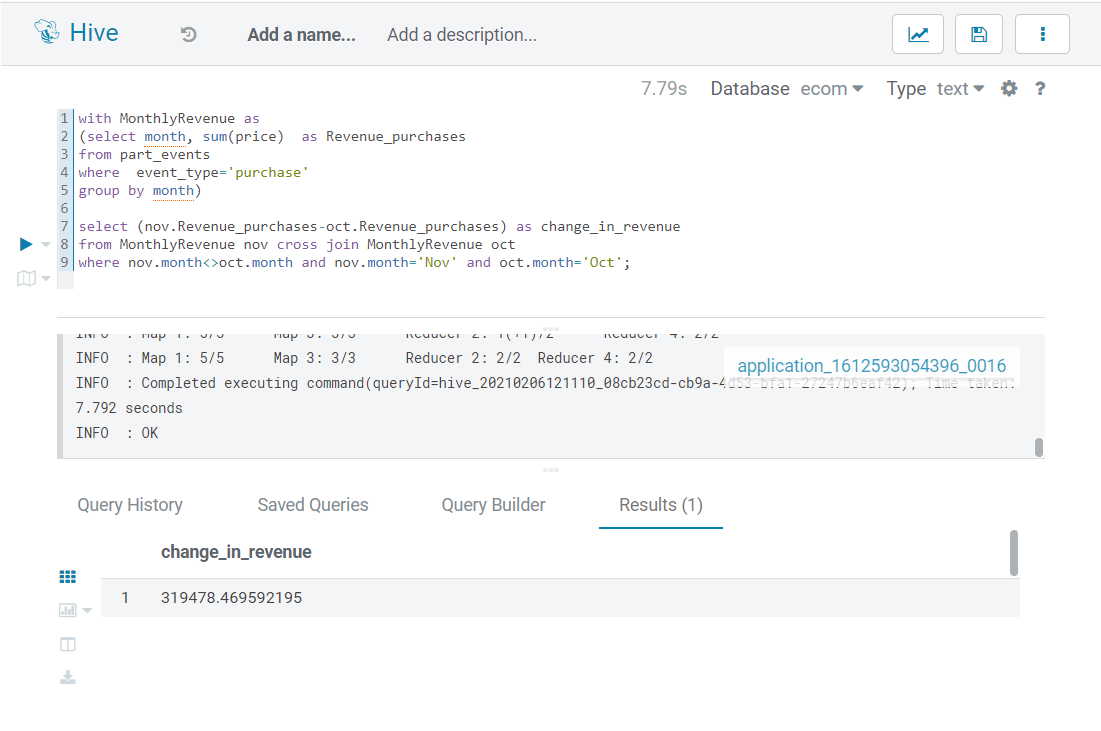
where event\_type='purchase'

group by month)

select (nov.Revenue\_purchases-oct.Revenue\_purchases) as change\_in\_revenue

from MonthlyRevenue nov cross join MonthlyRevenue oct

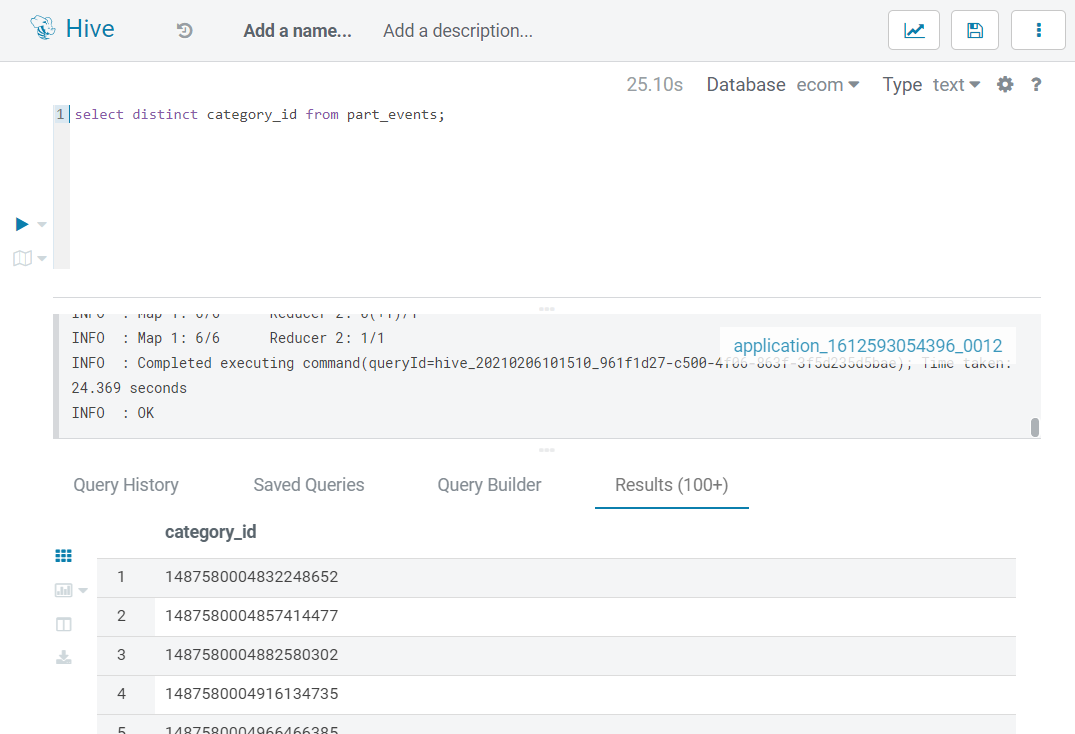
where nov.month<>oct.month and nov.month='Nov' and oct.month='Oct';



Note: use set hive.strict.checks.cartesian.product = False for cross product.

1. Find distinct categories of products.

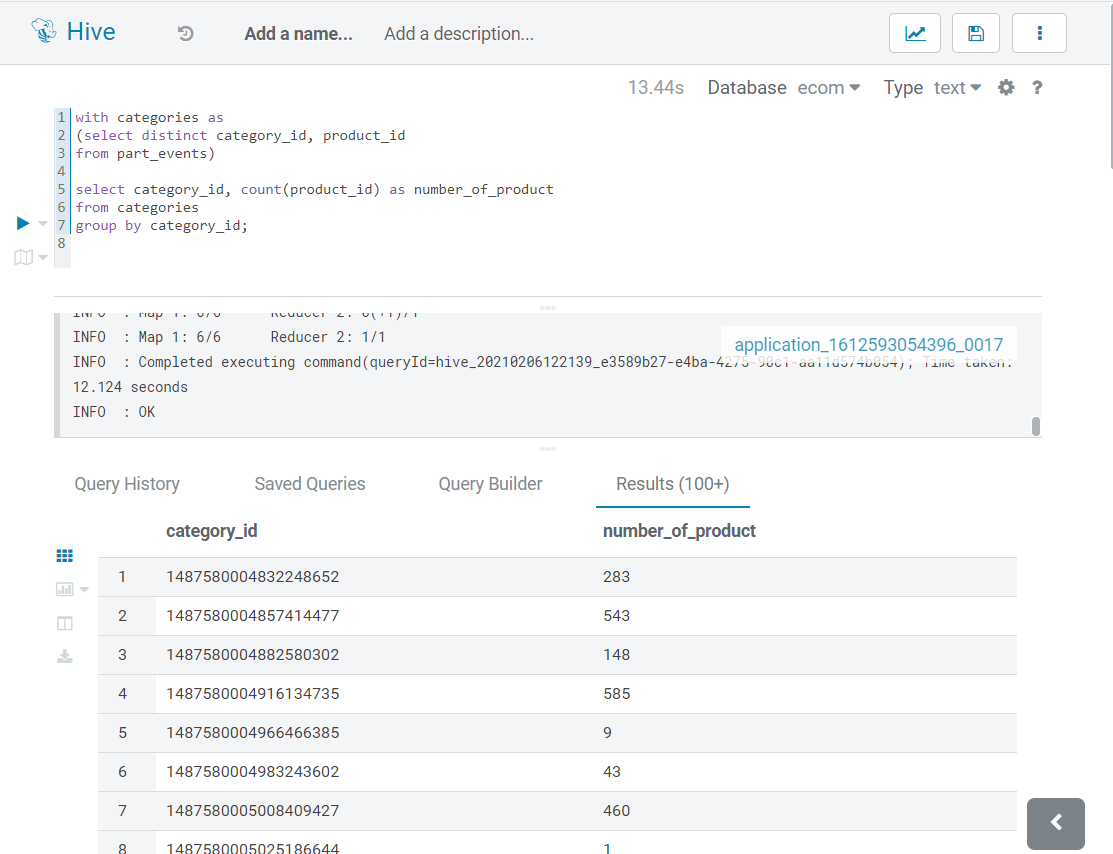
select distinct category\_id from part\_events;



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

with categories as (select distinct category\_id, product\_id from part\_events)

select category\_id, count(category\_id) from categories group by category\_id;



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

select brand, sum(price) as revenue from part\_events where event\_type='purchase' and brand <> ""

group by brand order by revenue desc limit 1;



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

with BrandRevenue as

(select month, brand ,sum(price) as sales\_per\_brand

from part\_events

where event\_type='purchase'

and brand <> ""

group by month,brand)

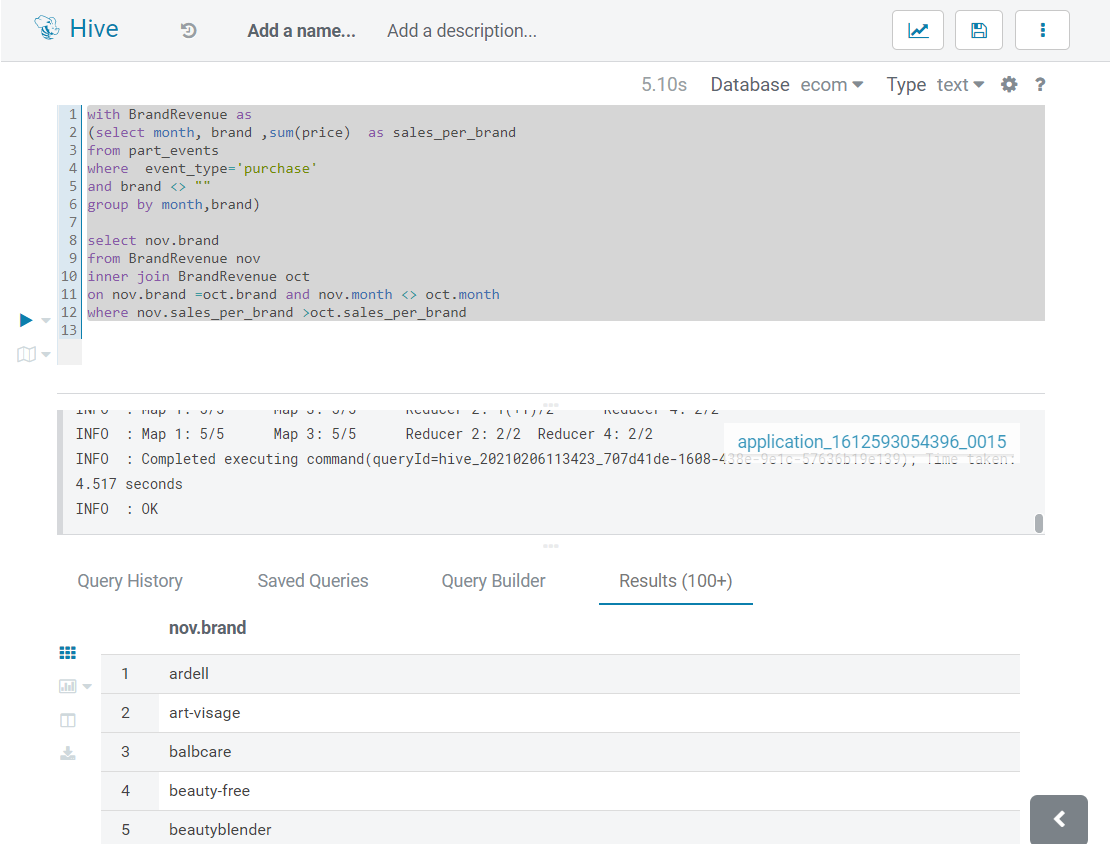
select nov.brand

from BrandRevenue nov

inner join BrandRevenue oct

on nov.brand =oct.brand and nov.month <> oct.month

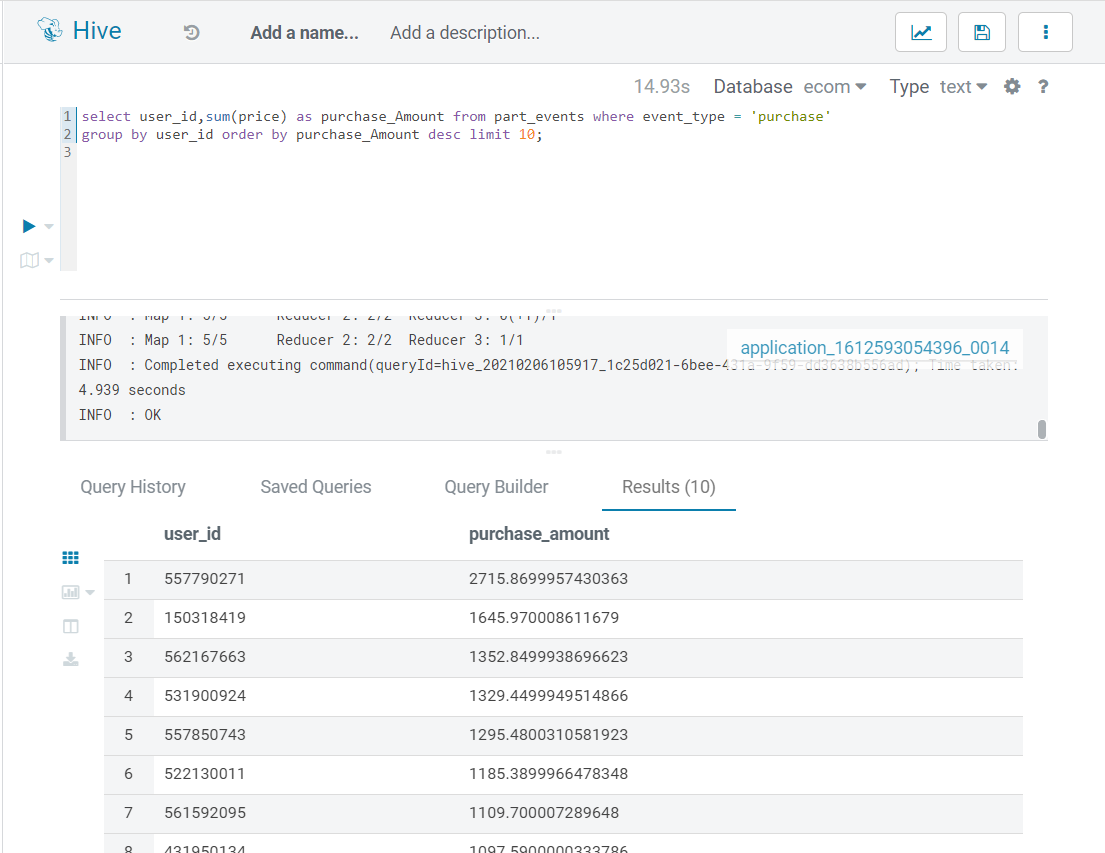
where nov.sales\_per\_brand >oct.sales\_per\_brand



1. 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 user\_id,sum(price) as purchase\_Amount from part\_events where event\_type = 'purchase'

group by user\_id order by purchase\_Amount desc limit 10;



Step 13 : Dropped all tables and database and Terminated the cluster. I have tried to do all this in one go so as to prevent heavy costing due to extended use.