

E-COMMERCE ASSIGNMENT

(By Suryansh Bhardwaj)

Implementation Phase

1. Importing Data to HDFS
 - a. Launching an EMR Cluster with Hive Services
 - b. Connecting using Putty (Windows)
 - c. Download the data and move to HDFS
2. Creating hive tables
 - a. Using beeline to get started with hive
 - b. Create hive table schema
 - c. Load data from HDFS to hive tables
 - d. Verifying the schema is appropriate and data is loaded correctly.
3. Running Queries for analysis purpose
4. Dropping the database and terminating the cluster.

Setup/Implementation Phase (Before querying)

1. Launched AWS EMR Cluster and connected using Putty
2. Downloading the data and move to HDFS
`[hadoop@ip-172-31-33-172 ~]$ wget https://e-commerce-events-ml.s3.amazonaws.com/2019-Oct.csv`
`[hadoop@ip-172-31-33-172 ~]$ wget https://e-commerce-events-ml.s3.amazonaws.com/2019-Nov.csv`

Comment: wget command is used to download files from these s3 buckets.

3. Move it to a new directory

```
[hadoop@ip-172-31-33-172 ~]$ mkdir hive_cs
```

```
cp 2019-Oct.csv hive_cs/
```

```
cp 2019-Nov.csv hive_cs/
```

```
[hadoop@ip-172-31-33-172 ~]$ cd hive_cs/
```

4. Checking data

Comment: cat command to view the data and verify if it is proper or not.

```
[hadoop@ip-172-31-33-172 hive_cs]$ cat 2019-Oct.csv | head
event_time,event_type,product_id,category_id,category_code,brand,price,user_id,user_session
2019-10-01 00:00:00 UTC,cart,5773203,1487580005134238553,,runail,2.62,463240011,26dd6e6e-4dac-
4778-8d2c-92e149dab885
2019-10-01 00:00:03 UTC,cart,5773353,1487580005134238553,,runail,2.62,463240011,26dd6e6e-4dac-
4778-8d2c-92e149dab885
2019-10-01 00:00:07 UTC,cart,5881589,2151191071051219817,,lovely,13.48,429681830,49e8d843-adf3-
428b-a2c3-fe8bc6a307c9
2019-10-01 00:00:07 UTC,cart,5723490,1487580005134238553,,runail,2.62,463240011,26dd6e6e-4dac-
4778-8d2c-92e149dab885
```

```

2019-10-01 00:00:15 UTC,cart,5881449,1487580013522845895,,lovely,0.56,429681830,49e8d843-adf3-428b-a2c3-fe8bc6a307c9
2019-10-01 00:00:16 UTC,cart,5857269,1487580005134238553,,runail,2.62,430174032,73dea1e7-664e-43f4-8b30-d32b9d5af04f
2019-10-01 00:00:19 UTC,cart,5739055,1487580008246412266,,kapous,4.75,377667011,81326ac6-daa4-4f0a-b488-fd0956a78733
2019-10-01 00:00:24 UTC,cart,5825598,1487580009445982239,,,0.56,467916806,2f5b5546-b8cb-9ee7-7ecd-84276f8ef486
2019-10-01 00:00:25 UTC,cart,5698989,1487580006317032337,,,1.27,385985999,d30965e8-1101-44ab-b45d-cc1bb9fae694

```

```

[hadoop@ip-172-31-33-172 hive_cs]$ cat 2019-Nov.csv | head
event_time,event_type,product_id,category_id,category_code,brand,price,user_id,user_session
2019-11-01 00:00:02 UTC,view,5802432,1487580009286598681,,,0.32,562076640,09fafd6c-6c99-46b1-834f-33527f4de241
2019-11-01 00:00:09 UTC,cart,5844397,1487580006317032337,,,2.38,553329724,2067216c-31b5-455d-a1cc-af0575a34ffb
2019-11-01 00:00:10 UTC,view,5837166,1783999064103190764,,pnb,22.22,556138645,57ed222e-a54a-4907-9944-5a875c2d7f4f
2019-11-01 00:00:11 UTC,cart,5876812,1487580010100293687,,jessnail,3.16,564506666,186c1951-8052-4b37-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,15.71,562076640,09fafd6c-6c99-46b1-834f-33527f4de241
2019-11-01 00:00:32 UTC,view,5837835,1933472286753424063,,,3.49,514649199,432a4e95-375c-4b40-bd36-0fc039e77580
2019-11-01 00:00:34
UTC,remove_from_cart,5870838,1487580007675986893,,milv,0.79,429913900,2f0bff3c-252f-4fe6-afcd-5d8a6a92839a

```

5. Connecting to hive

Comment: We can also use "hive" to use hive. Here I have used beeline to connect to HiveServer2.

```
[hadoop@ip-172-31-33-172 hive_cs]$ beeline -u jdbc:hive2://localhost:10000/default -n hadoop
```

6. Create Database if not exists

```
create database clickstream; use clickstream;
```

7. Create table clickstream_tbl if not exists

```
create table if not exists clickstream_tbl(event_time timestamp, event_type string, product_id string, category_id string, category_code string, brand string, price decimal(10,3), user_id bigint, user_session string) row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde' WITH SERDEPROPERTIES ( "separatorChar" = ",", "quoteChar" = "\"", "escapeChar" = "\\") stored as textfile TBLPROPERTIES ("skip.header.line.count"="1");
```

Comment: Table clickstream_tbl with original schema when loaded, format = OpenCSVSerde

8. Load Data into table clickstream_tbl and verify

load data local inpath '/home/hadoop/hive_cs/2019-Oct.csv' into table clickstream_tbl;

load data local inpath '/home/hadoop/hive_cs/2019-Nov.csv' into table clickstream_tbl;

9. Setting up dynamic partitioning, run these 2 commands

set hive.exec.dynamic.partition.mode = true;

set hive.exec.dynamic.partition.mode = nonstrict;

10. Creating dynamically partitioned table, Partition on month(event_time) and bucketing on event_type (4 buckets).

create table if not exists part_dyn(event_time string,event_type string,product_id string,category_id string,category_code string,brand string,price decimal(10,3),user_id bigint,user_session string) partitioned by (mnth int) clustered by (event_type) into 4 buckets row format delimited fields terminated by '|' lines terminated by "\n" stored as textfile;

Comment: Table part_dyn is partitioned on month(event_time) and bucket on event_type.

I've proceeded with this approach because on analysis of queries we have to run for problems provided, columns event_time and event_type are used most frequently. Also we know there are only 2 months of Data (October and November) and there are only 4 distinct event_types, so bucketing on event_type would be perfect.

11. Insert values into this dynamic partition table from our original clickstream_tbl

insert into table part_dyn partition(mnth) select *,month(event_time) as mnth from clickstream_tbl where month(event_time) in (10,11);

12. Enforce Bucketing

set hive.enforce.bucketing = true;

13. The tables are ready, one is normal table (clickstream_tbl) and other is partitioned table (part_dyn). Now we will compare first 2 queries and see which queries requires less execution time.

Questions and Answers:

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

- a. Running Query using normal table – clickstream_tbl

Query : ***select sum(price) as Total_Oct_Revenue from clickstream_tbl where month(event_time) = 10 and event_type='purchase';***

Output:

```
+-----+
| total_oct_revenue |
+-----+
| 1211538.4299997438 |
+-----+
```

```
INFO : Map 1: 1(+1)/2 Reducer 2: 0/1
INFO : Map 1: 1(+1)/2 Reducer 2: 0(+1)/1
INFO : Map 1: 1(+1)/2 Reducer 2: 0(+1)/1
INFO : Map 1: 2/2 Reducer 2: 0(+1)/1
INFO : Map 1: 2/2 Reducer 2: 1/1
INFO : Completed executing command(queryId=hive_20220720102010)
INFO : OK
+-----+
| total_oct_revenue |
+-----+
| 1211538.4299997438 |
+-----+
1 row selected (76.926 seconds)
```

- b. Running query using dynamically partitioned table – part_dyn

Query : ***select sum(price) as Total_Oct_Revenue from part_dyn where mnth = 10 and event_type='purchase';***

Output:

```
+-----+
| total_oct_revenue |
+-----+
| 1211538.430 |
+-----+
```

```
INFO : Map 1: 4(+2)/6 Reducer 2: 0(+1)/1
INFO : Map 1: 6/6 Reducer 2: 0(+1)/1
INFO : Map 1: 6/6 Reducer 2: 1/1
INFO : Completed executing command(queryId=hive_20220720102010)
INFO : OK
+-----+
| total_oct_revenue |
+-----+
| 1211538.430 |
+-----+
1 row selected (34.828 seconds)
```

CONCLUSION: Total Revenue Generated in the month of October due to Purchase = 1211538.429

Comment: When we used dynamically partitioned table (part_dyn) on a query that has conditions related to event_time and event_type, there is a massive difference in time taken to execute the query in comparison to the normal clickstream_tbl table.

The execution time went from 76.926 seconds to 34.828 seconds (Massive Improvement)

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

a. Running Query using normal table – clickstream_tbl

Query: ***select month(event_time) as Month, sum(price) as Price from clickstream_tbl where event_type = 'purchase' group by month(event_time);***

Output:

```
+-----+-----+
| month | price          |
+-----+-----+
| 11    | 1531016.900000122 |
| 10    | 1211538.4299997438 |
+-----+-----+
```

```
INFO : Map 1: 2/2      Reducer 2: 6/6
INFO : Completed executing command(queryId=hive_20220720110347_9
INFO : OK
+-----+-----+
| month | price          |
+-----+-----+
| 11    | 1531016.900000122 |
| 10    | 1211538.4299997438 |
+-----+-----+
2 rows selected (72.223 seconds)
```

b. Running Query using dynamically partitioned table – part_dyn

Query : ***select mnth, sum(price) as Price from part_dyn where event_type = 'purchase' group by mnth;***

Output:

```
+-----+-----+
| mnth | price          |
+-----+-----+
| 10    | 1211538.430    |
| 11    | 1531016.900    |
+-----+-----+
```

```
INFO : Map 1: 8/8      Reducer 2: 0(+2)/2
INFO : Map 1: 8/8      Reducer 2: 2/2
INFO : Completed executing command(queryId=hive_202
INFO : OK
+-----+-----+
| mnth | price          |
+-----+-----+
| 10    | 1211538.430    |
| 11    | 1531016.900    |
+-----+-----+
2 rows selected (30.915 seconds)
```

CONCLUSION :

October Revenue generated due to Purchase = 1211538.430

November Revenue generated due to Purchase = 1531016.900

Comment: When we used dynamically partitioned table (part_dyn) on a query that has conditions related to event_time and event_type, there is a massive difference in time taken to execute the query in comparison to the normal clickstream_tbl table.

The execution time went from 72.223 seconds to 30.915 seconds (Improvement)

As the dynamically partitioned table is much more faster (takes less execution time) than normal table, we will be using dynamic partition table for the rest of the queries.

Table Name/ Query No.	clickstream_tbl	part_dyn (partitioned)
1	76.926 seconds	34.828 seconds
2	72.223 seconds	30.915 seconds

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

Query : ***select October_price, November_price, November_price - October_price as Difference from(select sum(case when mnth=10 then price else 0 end) as October_price, sum(case when mnth=11 then price else 0 end) as November_price from part_dyn WHERE mnth in (10,11) and event_type='purchase')s;***

Comment: Approach is to find November Revenue generated – October Revenue generated. Revenue means event_type = 'purchase' because other event type don't produce revenue.

Output:

```
+-----+-----+-----+
| october_price | november_price | difference |
+-----+-----+-----+
| 1211538.430   | 1531016.900   | 319478.470 |
+-----+-----+-----+
1 row selected (40.385 seconds)
```

```
INFO : Map 1: 1(+3)/8 Reducer 2: 0/1
INFO : Map 1: 2(+3)/8 Reducer 2: 0/1
INFO : Map 1: 3(+3)/8 Reducer 2: 0/1
INFO : Map 1: 5(+1)/8 Reducer 2: 0/1
INFO : Map 1: 5(+3)/8 Reducer 2: 0/1
INFO : Map 1: 6(+2)/8 Reducer 2: 0/1
INFO : Map 1: 6(+2)/8 Reducer 2: 0(+1)/1
INFO : Map 1: 7(+1)/8 Reducer 2: 0(+1)/1
INFO : Map 1: 8/8 Reducer 2: 0(+1)/1
INFO : Map 1: 8/8 Reducer 2: 1/1
INFO : Completed executing command(queryId=hive_20220720112506_d68d7fd6-bc22-45d2-a429-ef30e795c1e9)
INFO : OK
+-----+-----+-----+
| october_price | november_price | difference |
+-----+-----+-----+
| 1211538.430   | 1531016.900   | 319478.470 |
+-----+-----+-----+
1 row selected (40.385 seconds)
```

CONCLUSION: Revenue generated due to purchases increased by 319478.470 from month October to November.

4. Find distinct categories of products.

Query : ***select distinct(category_code) from part_dyn where category_code!="" and category_code is not null;***

Output:

```
+-----+
|      category_code      |
+-----+
| accessories.bag          |
| accessories.cosmetic_bag |
| apparel.glove           |
| appliances.environment.air_conditioner |
| appliances.environment.vacuum |
| appliances.personal.hair_cutter |
| furniture.bathroom.bath |
| furniture.living_room.cabinet |
| furniture.living_room.chair |
| sport.diving            |
| stationery.cartridge     |
+-----+
```

11 rows selected (30.172 seconds)

```
INFO : Completed compiling command(queryId=hive_20220720112956_459a84dc-d52d-4a10-b44a-3
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Executing command(queryId=hive_20220720112956_459a84dc-d52d-4a10-b44a-3f4d3970ddc
INFO : Query ID = hive_20220720112956_459a84dc-d52d-4a10-b44a-3f4d3970ddcf
INFO : Total jobs = 1
INFO : Launching Job 1 out of 1
INFO : Starting task [Stage-1:MAPRED] in serial mode
INFO : Session is already open
INFO : Dag name: select distinct(category_code) from p...null(Stage-1)
INFO : Status: Running (Executing on YARN cluster with App id application_1658307022784_

INFO : Map 1: 0/8      Reducer 2: 0/10
INFO : Map 1: 0/8      Reducer 2: 0/10
INFO : Map 1: 0(+1)/8   Reducer 2: 0/10
INFO : Map 1: 0(+2)/8   Reducer 2: 0/10
INFO : Map 1: 0(+3)/8   Reducer 2: 0/10
INFO : Map 1: 0(+3)/8   Reducer 2: 0/10
INFO : Map 1: 0(+3)/8   Reducer 2: 0/10
INFO : Map 1: 0(+3)/8   Reducer 2: 0/10
INFO : Map 1: 0(+3)/8   Reducer 2: 0/10
INFO : Map 1: 1(+3)/8   Reducer 2: 0/10
INFO : Map 1: 2(+3)/8   Reducer 2: 0/10
INFO : Map 1: 3(+3)/8   Reducer 2: 0/10
INFO : Map 1: 3(+3)/8   Reducer 2: 0/10
INFO : Map 1: 4(+2)/8   Reducer 2: 0/10
INFO : Map 1: 6(+0)/8   Reducer 2: 0/1
INFO : Map 1: 6(+2)/8   Reducer 2: 0/1
INFO : Map 1: 6(+2)/8   Reducer 2: 0(+1)/1
INFO : Map 1: 7(+1)/8   Reducer 2: 0(+1)/1
INFO : Map 1: 8/8      Reducer 2: 0(+1)/1
INFO : Map 1: 8/8      Reducer 2: 1/1
INFO : Completed executing command(queryId=hive_20220720112956_459a84dc-d52d-4a10-b44a-3
INFO : OK

+-----+
|      category_code      |
+-----+
| accessories.bag          |
| accessories.cosmetic_bag |
| apparel.glove           |
| appliances.environment.air_conditioner |
| appliances.environment.vacuum |
| appliances.personal.hair_cutter |
| furniture.bathroom.bath |
| furniture.living_room.cabinet |
| furniture.living_room.chair |
| sport.diving            |
| stationery.cartridge     |
+-----+
11 rows selected (30.172 seconds)
```

CONCLUSION : There are 11 distinct categories which are **bag, cosmetic_bag, glove, air_conditioner, vacuum, hair_cutter, bath, cabinet, chair, diving, cartridge.**

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

Query : ***select count(product_id) as Product_Count, category_code from part_dyn group by category_code;***

Output:

product_count	category_code
11681	accessories.bag
1248	accessories.cosmetic_bag
18232	apparel.glove
332	appliances.environment.air_conditioner
59761	appliances.environment.vacuum
1643	appliances.personal.hair_cutter
9857	furniture.bathroom.bath
13439	furniture.living_room.cabinet
308	furniture.living_room.chair
2	sport.diving
26722	stationery.cartridge

11 rows selected (39.944 seconds)

```
INFO : Map 1: 1(+3)/8 Reducer 2: 0/10
INFO : Map 1: 2(+3)/8 Reducer 2: 0/10
INFO : Map 1: 3(+3)/8 Reducer 2: 0/10
INFO : Map 1: 5(+1)/8 Reducer 2: 0/10
INFO : Map 1: 6(+0)/8 Reducer 2: 0/1
INFO : Map 1: 6(+2)/8 Reducer 2: 0/1
INFO : Map 1: 6(+2)/8 Reducer 2: 0(+1)/1
INFO : Map 1: 6(+2)/8 Reducer 2: 0(+1)/1
INFO : Map 1: 8/8 Reducer 2: 0(+1)/1
INFO : Map 1: 8/8 Reducer 2: 1/1
INFO : Completed executing command(queryId=hive_20220720114433_67f59086-53fe-4d71
INFO : OK

+-----+-----+
| product_count | category_code |
+-----+-----+
| 11681         | accessories.bag |
| 1248          | accessories.cosmetic_bag |
| 18232         | apparel.glove |
| 332           | appliances.environment.air_conditioner |
| 59761         | appliances.environment.vacuum |
| 1643          | appliances.personal.hair_cutter |
| 9857          | furniture.bathroom.bath |
| 13439         | furniture.living_room.cabinet |
| 308           | furniture.living_room.chair |
| 2             | sport.diving |
| 26722        | stationery.cartridge |
+-----+-----+
11 rows selected (39.944 seconds)
```

CONCLUSION: These are the product counts of individual categories (except the missing value category).

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

Query: *select brand,sum(price) as Brand_Price from part_dyn where brand!= "" and event_type='purchase' group by brand order by Brand_Price desc limit 1;*

Comment: Approach is to find total sales for each brand and then find the maximum out of them. Sales means event_type = 'purchase' and we need to ignore the brand records with empty value.

Output:

```
+-----+-----+
| brand | brand_price |
+-----+-----+
| runail | 148297.940 |
+-----+-----+
```

1 row selected (42.359 seconds)

```
INFO : Completed compiling command(queryId=hive_20220720120821_2b9a3e31-c7de-47f3-bcf2-bfefbe058323); Time taken: 0.191 seconds
INFO : Concurrency mode is disabled, not creating a lock manager
INFO : Executing command(queryId=hive_20220720120821_2b9a3e31-c7de-47f3-bcf2-bfefbe058323): select brand,sum(price) as Brand_Price from part_dyn
INFO : Query ID = hive_20220720120821_2b9a3e31-c7de-47f3-bcf2-bfefbe058323
INFO : Total jobs = 1
INFO : Launching Job 1 out of 1
INFO : Starting task [Stage-1:MAPRED] in serial mode
INFO : Session is already open
INFO : Dag name: select brand,sum(price) as Brand_Price f...1(Stage-1)
INFO : Tez session was closed. Reopening...
INFO : Session re-established.
INFO : Status: Running (Executing on YARN cluster with App id application_1658307022784_0010)

INFO : Map 1: 0/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 0/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 0/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 0(+2)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 0(+3)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 0(+3)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 0(+3)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 0(+3)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 0(+3)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 0(+2)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 1(+3)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 2(+3)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 3(+3)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 4(+2)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 4(+3)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 6(+1)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 6(+2)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 6(+2)/8 Reducer 2: 0(+1)/6 Reducer 3: 0/1
INFO : Map 1: 8/8 Reducer 2: 0(+2)/6 Reducer 3: 0/1
INFO : Map 1: 8/8 Reducer 2: 0(+3)/6 Reducer 3: 0/1
INFO : Map 1: 8/8 Reducer 2: 1(+3)/6 Reducer 3: 0/1
INFO : Map 1: 8/8 Reducer 2: 2(+3)/6 Reducer 3: 0/1
INFO : Map 1: 8/8 Reducer 2: 3(+3)/6 Reducer 3: 0/1
INFO : Map 1: 8/8 Reducer 2: 5(+1)/6 Reducer 3: 0(+1)/1
INFO : Map 1: 8/8 Reducer 2: 6/6 Reducer 3: 0(+1)/1
INFO : Map 1: 8/8 Reducer 2: 6/6 Reducer 3: 1/1
INFO : Completed executing command(queryId=hive_20220720120821_2b9a3e31-c7de-47f3-bcf2-bfefbe058323); Time taken: 42.152 seconds
INFO : OK
+-----+-----+
| brand | brand_price |
+-----+-----+
| runail | 148297.940 |
+-----+-----+
1 row selected (42.359 seconds)
```

CONCLUSION: Brand = runail has the maximum sales in October and November combined.

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

Query : *create view part_monthly_brand_price as (select mnth, brand, sum(price) as Monthly_Price from part_dyn where event_type = 'purchase' and brand != "" group by mnth,brand order by mnth,brand);*

select brand from (select brand,sum(case when mnth = 11 then Monthly_Price else -Monthly_Price end) as Increased_Sales from part_monthly_brand_price group by brand having Increased_Sales>0);

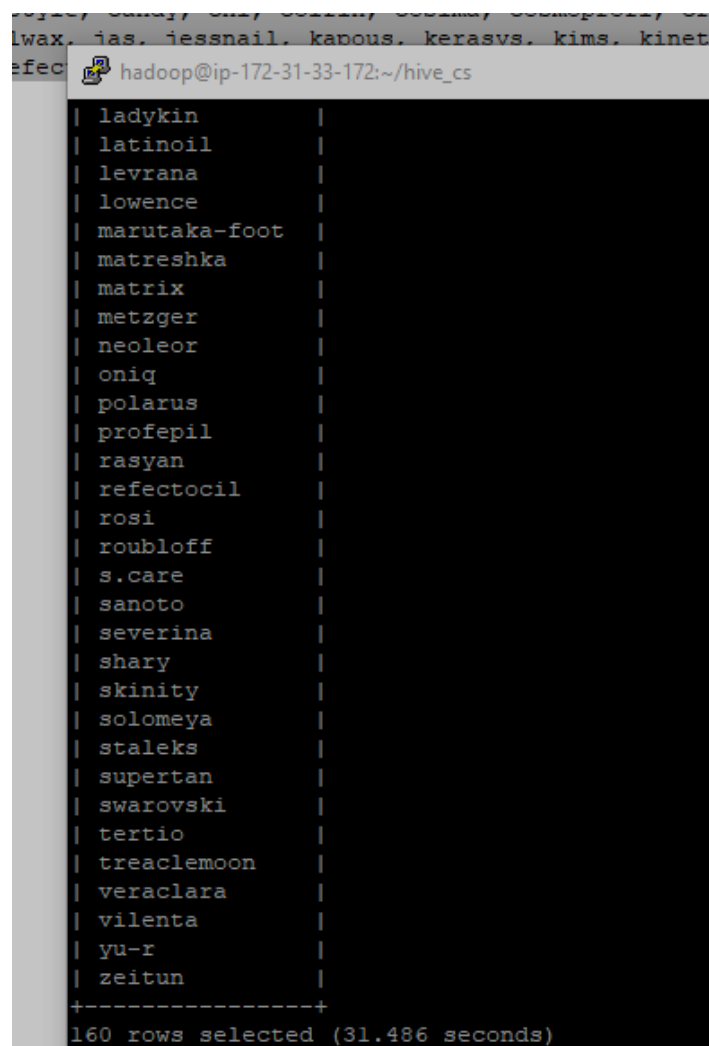
Comment: To simplify this query. I created a view that provides per month, for each brand what are the total sales. Then I use that view, to calculate the increased sales by subtracting November Sales – October Sales for each brand. I subtract them by adding negative of October sales to November sales. (November Sales + (-

October Sales). This way I get the increased sales from October to November if the value is positive. Then filter out the positive values and that gives us the brands with increased sales.

Also, I've written the output (brands with increased sales) in normal format otherwise it takes up lot of space as shown in the screenshot.

Output:

```
+-----+ brand, +-----+
art-visage, artex, barbie, batiste, beautix, beautyblender, bioaqua, biore, blixz, browxenna, carmex, concept,
cutrin, deoproce, domix, ecocraft, ecolab, egomania, ellips, elskin, entity, eos, f.o.x, farmavita, fedua, fly,
freshbubble, gehwol, glysolid, greymy, happyfons, haruyama, helloganic, inm, insight, jaguar, joico, juno,
kaaral, kamill, kares, kaypro, keen, konad, laboratorium, levissime, lianail, likato, limoni, lovely, mane,
marathon, markell, masura, mavala, milv, miskin, missha, moyou, nagaraku, naomi, nefertiti, nirvel, nitrile,
orly, osmo, ovale, plazan, profhenna, protokeratin, provoc, runail, shik, skinlite, smart, soleo, sophin, strong,
trind, uno, uskusi, yoko, airnails, aura, balbcare, beauty-free, beauugreen, benovy, binacil, bluesky, bodyton,
bpw.style, candy, chi, coifin, cosima, cosmoprofi, cristalinas, de.lux, depilflax, , dewal, dizao, elizavecca,
enjoy, estel, estelare, farmona, finish, foamie, freedecor, godefroy, grace, grattol, igrobeauty, ingarden, irisk,
italwax, jas, jessnail, kapous, kerasys, kims, kinetics, kiss, kocostar, koelcia, koelf, kosmekka, lador, ladykin,
latinoil, levrana, lowence, marutaka-foot, matreshka, matrix, metzger, neoleor, oniq, polarus, profepil,
rasyan, refectocil, rosi, roubloff, s.care, sanoto, severina, shary, skinity, solomeya, staleks, supertan,
swarovski, tertio, treaclemoon, veraclara, vilenta, yu-r, zeitun
+-----+160rowsselected(31.486seconds)
```



```
ladykin
latinoil
levrana
lowence
marutaka-foot
matreshka
matrix
metzger
neoleor
oniq
polarus
profepil
rasyan
refectocil
rosi
roubloff
s.care
sanoto
severina
shary
skinity
solomeya
staleks
supertan
swarovski
tertio
treaclemoon
veraclara
vilenta
yu-r
zeitun
+-----+
160 rows selected (31.486 seconds)
```

CONCLUSION: There are total 160 brands that have increased sales from October to November, listed above.

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.

Query: *select user_id,sum(price) as Purchase_Amt from part_dyn where event_type = 'purchase' group by user_id order by Purchase_Amt desc limit 10;*

Output:

```
+-----+-----+
| user_id | purchase_amt |
+-----+-----+
| 557790271 | 2715.870 |
| 150318419 | 1645.970 |
| 562167663 | 1352.850 |
| 531900924 | 1329.450 |
| 557850743 | 1295.480 |
| 522130011 | 1185.390 |
| 561592095 | 1109.700 |
| 431950134 | 1097.590 |
| 566576008 | 1056.360 |
| 521347209 | 1040.910 |
+-----+-----+
```

10 rows selected (43.447 seconds)

```
INFO : Map 1: 0(+3)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 1(+3)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 2(+3)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 3(+3)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 4(+2)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 5(+2)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 5(+3)/8 Reducer 2: 0/6 Reducer 3: 0/1
INFO : Map 1: 6(+2)/8 Reducer 2: 0(+1)/6 Reducer 3: 0/1
INFO : Map 1: 7(+1)/8 Reducer 2: 0(+2)/6 Reducer 3: 0/1
INFO : Map 1: 8/8 Reducer 2: 0(+3)/6 Reducer 3: 0/1
INFO : Map 1: 8/8 Reducer 2: 1(+3)/6 Reducer 3: 0/1
INFO : Map 1: 8/8 Reducer 2: 2(+3)/6 Reducer 3: 0/1
INFO : Map 1: 8/8 Reducer 2: 3(+3)/6 Reducer 3: 0/1
INFO : Map 1: 8/8 Reducer 2: 4(+2)/6 Reducer 3: 0(+1)/1
INFO : Map 1: 8/8 Reducer 2: 5(+1)/6 Reducer 3: 0(+1)/1
INFO : Map 1: 8/8 Reducer 2: 6/6 Reducer 3: 0(+1)/1
INFO : Map 1: 8/8 Reducer 2: 6/6 Reducer 3: 1/1
INFO : Completed executing command(queryId=hive_20220720131918_72b164db-5aea-48e1-bb37-2f
.227 seconds
INFO : OK
+-----+-----+
| user_id | purchase_amt |
+-----+-----+
| 557790271 | 2715.870 |
| 150318419 | 1645.970 |
| 562167663 | 1352.850 |
| 531900924 | 1329.450 |
| 557850743 | 1295.480 |
| 522130011 | 1185.390 |
| 561592095 | 1109.700 |
| 431950134 | 1097.590 |
| 566576008 | 1056.360 |
| 521347209 | 1040.910 |
+-----+-----+
10 rows selected (43.447 seconds)
0: jdbc:hive2://localhost:10000/default>
```

Activate V
Go to Setting

CONCLUSION: These are the top 10 user_ids who spend the most on purchases.