## Logic For Final Submission

**Task 5: Calculate the total number of different drivers for each customer.**

**select customer\_id , count(distinct driver\_id) from booking\_data group by customer\_id order by customer\_id asc;**

This query will group the results by customer\_id and then order the output by customer\_id in ascending order.

Explaination of each part:

SELECT customer\_id: This specifies the column customer\_id that you want to retrieve in the results.

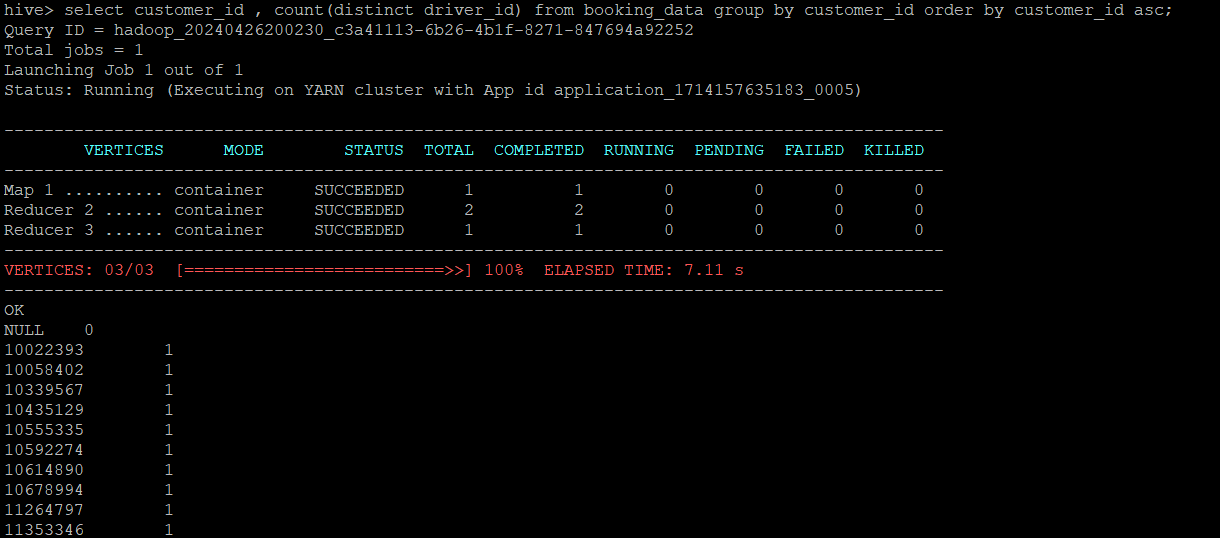
COUNT(DISTINCT driver\_id) AS num\_unique\_drivers: This calculates the count of distinct driver\_id values for each customer\_id group. The alias num\_unique\_drivers gives a name to this computed value in the result set.

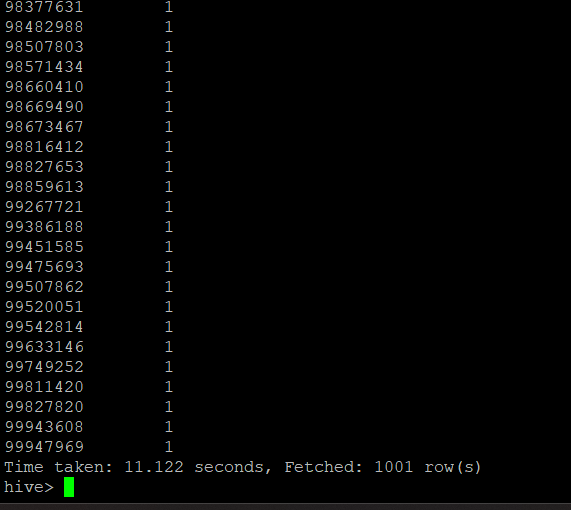
FROM booking\_data: This specifies the table booking\_data from which the data is being queried.

GROUP BY customer\_id: This groups the rows of the booking\_data table by customer\_id.

ORDER BY customer\_id ASC: This orders the result set by customer\_id in ascending order (ASC).

screenshot:





**Task 6: Calculate the total rides taken by each customer.**

**select customer\_id ,count(distinct booking\_id) from booking\_data group by customer\_id order by customer\_id asc;**

This SQL query is structured to count the distinct booking\_id values for each customer\_id in the booking\_data table, grouping the results by customer\_id and then ordering them in ascending order based on customer\_id.

Explanation of each part:

SELECT customer\_id: This specifies the column customer\_id that you want to include in the output.

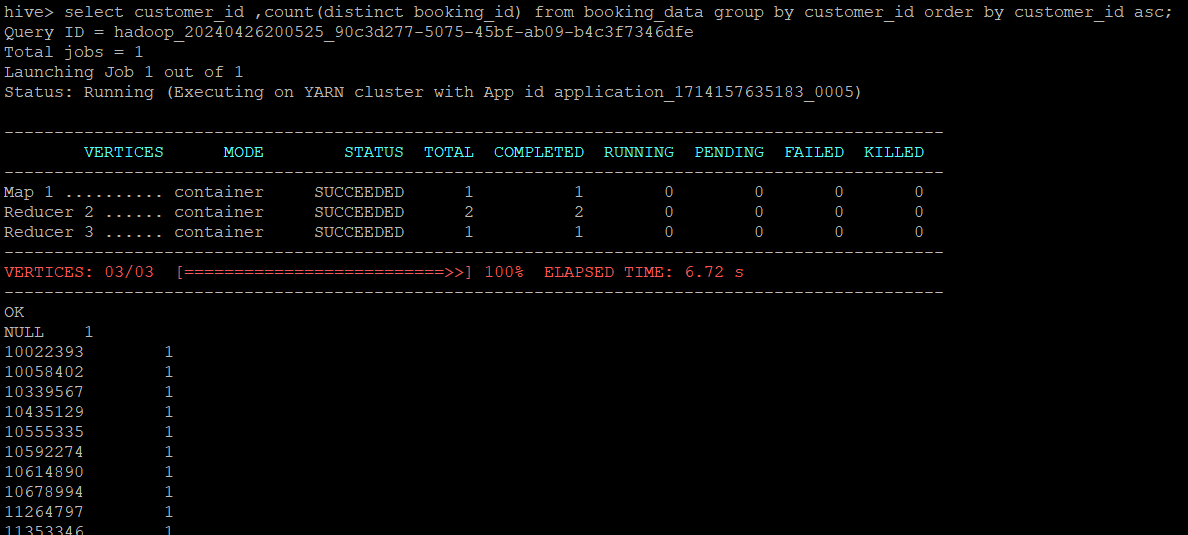
COUNT(DISTINCT booking\_id) AS num\_unique\_bookings: This calculates the count of distinct booking\_id values for each customer\_id group. The result of this count is given an alias num\_unique\_bookings in the output.

FROM booking\_data: This indicates the source table from which data is being queried, which is booking\_data in this case.

GROUP BY customer\_id: This clause groups the data by customer\_id, so the COUNT(DISTINCT booking\_id) is calculated per customer\_id.

ORDER BY customer\_id ASC: This specifies that the result set should be ordered by customer\_id in ascending (ASC) order.

screenshot:



**Task 7: Find the total visits made by each customer on the booking page and the total ‘Book Now’ button presses. This can show the conversion ratio.**

**select (sum(case when button\_id = "fcba68aa-1231-11eb-adc1-0242ac120002" and**

**is\_button\_click = 'Yes' then 1 end) / sum(case when page\_id = "e7bc5fb2-1231-11eb-adc1-0242ac120002" and is\_page\_view = 'Yes' then 1 end)) as conversion\_ratio from clickstream\_data;**

This query will calculate the conversion ratio, which typically represents the ratio of button clicks (meeting specific criteria) to page views (meeting specific criteria) within your clickstream\_data table. The result will be a single value representing this conversion ratio.

Explanation of each part:

SELECT: This indicates that we want to select and calculate a value based on the subsequent expression.

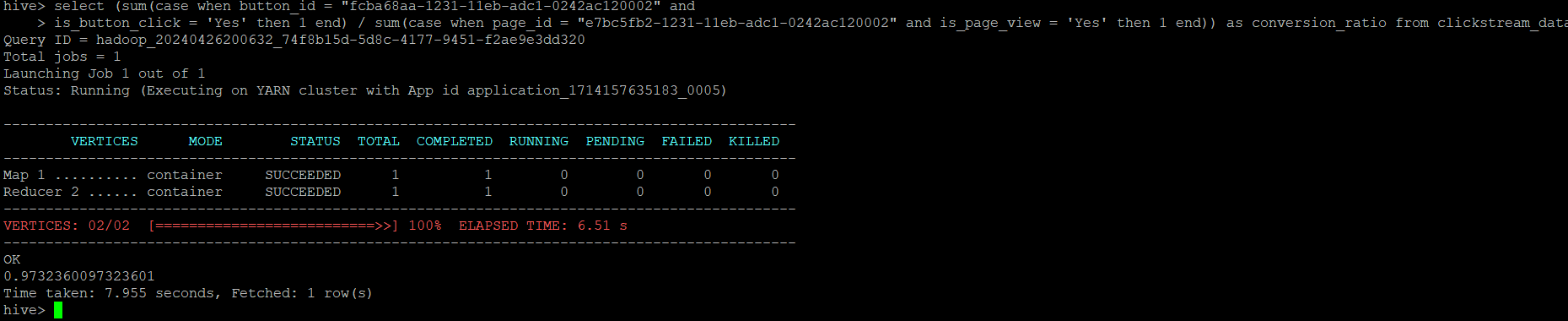
SUM(CASE WHEN button\_id = 'fcba68aa-1231-11eb-adc1-0242ac120002' AND is\_button\_click = 'Yes' THEN 1 END): This part of the query calculates the total count of records where button\_id matches 'fcba68aa-1231-11eb-adc1-0242ac120002' and is\_button\_click is 'Yes'. The CASE statement assigns a value of 1 for each matching record, and SUM adds these values together.

SUM(CASE WHEN page\_id = 'e7bc5fb2-1231-11eb-adc1-0242ac120002' AND is\_page\_view = 'Yes' THEN 1 END): This part of the query calculates the total count of records where page\_id matches 'e7bc5fb2-1231-11eb-adc1-0242ac120002' and is\_page\_view is 'Yes'. Similarly, the CASE statement assigns a value of 1 for each matching record, and SUM adds these values together.

The division (/) operator is used to divide the sum of button clicks (is\_button\_click = 'Yes') by the sum of page views (is\_page\_view = 'Yes') to compute the conversion ratio.

AS conversion\_ratio: This assigns an alias conversion\_ratio to the calculated value in the result set.

screenshot:



**Task 8: Calculate the count of all trips done on black cabs.**

**select cab\_color ,count(distinct driver\_id ) from booking\_data where cab\_color in ('black') group by cab\_color ;**

This query will provide you with the count of distinct driver\_id values for bookings associated with black-colored cabs (cab\_color = 'black'), grouped by cab\_color. The result will show the number of unique drivers for black-colored cab bookings.

Explanation of each part:

SELECT cab\_color: This specifies the cab\_color column that you want to include in the output.

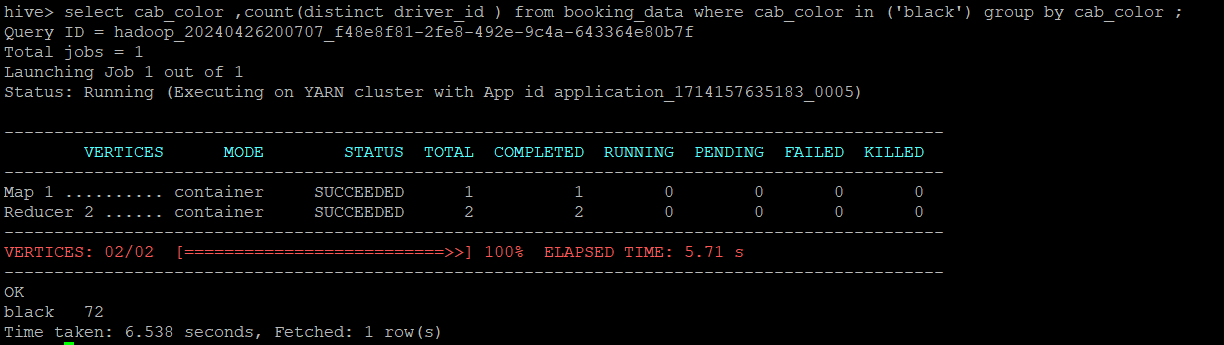
COUNT(DISTINCT driver\_id) AS num\_unique\_drivers: This calculates the count of distinct driver\_id values for bookings where the cab\_color is 'black'. The alias num\_unique\_drivers gives a name to this computed value in the result set.

FROM booking\_data: This indicates the source table from which data is being queried, which is booking\_data in this case.

WHERE cab\_color = 'black': This filters the rows to only consider bookings where the cab\_color is 'black'.

GROUP BY cab\_color: This groups the data by cab\_color, so the COUNT(DISTINCT driver\_id) is calculated for each distinct cab\_color group.

screenshot:



**Task 9: Calculate the total amount of tips given date wise to all drivers by customers.**

**select pickup\_date, sum(tip\_amount) from booking\_data group by pickup\_date order by pickup\_date asc;**

This query will provide you with the total tip amount (total\_tips) for each pickup\_date, sorted in chronological order by pickup\_date. Each row in the result set will show a pickup\_date along with the corresponding sum of tip\_amount for bookings on that date.

Explanation of each part:

SELECT pickup\_date: This specifies the pickup\_date column that you want to include in the output.

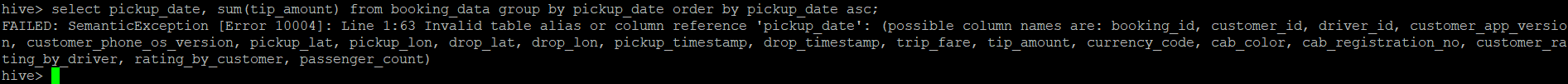
SUM(tip\_amount) AS total\_tips: This calculates the sum of tip\_amount for each pickup\_date group. The alias total\_tips assigns a name to this aggregated value in the result set.

FROM booking\_data: This indicates the source table from which data is being queried (booking\_data in this case).

GROUP BY pickup\_date: This groups the data by pickup\_date, so the SUM(tip\_amount) is calculated for each unique pickup\_date.

ORDER BY pickup\_date ASC: This orders the result set by pickup\_date in ascending (ASC) order.

screenshot:



**Task 10: Calculate the total count of all the bookings with ratings lower than 2 as given by customers in a particular month.**

**select date\_format(pickup\_timestamp,'yyyy-MM') ,count( rating\_by\_customer) from booking\_data where rating\_by\_customer < 2 group by date\_format(pickup\_timestamp,'yyyy-MM');**

This query will provide you with the count of ratings by customers that are less than 2 (rating\_by\_customer < 2) for each month (pickup\_month), formatted as year and month (yyyy-MM). The result set will show each month along with the corresponding count of low ratings for that month**.**

Explanation of each part:

SELECT DATE\_FORMAT(pickup\_timestamp, 'yyyy-MM') AS formatted\_date: This part of the query selects the pickup\_timestamp column from the booking\_data table and formats it using the DATE\_FORMAT function to display the date in the format 'yyyy-MM', which represents the year and month (e.g., '2024-05'). The AS formatted\_date clause assigns an alias formatted\_date to this formatted result.

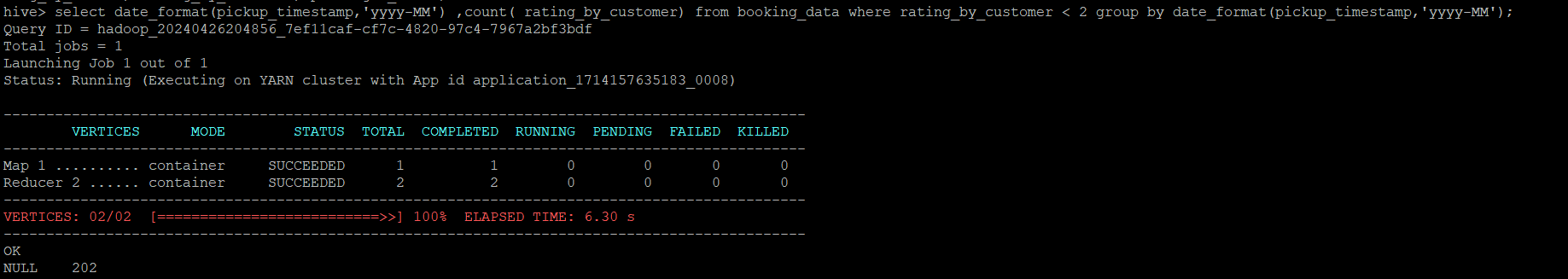
COUNT(rating\_by\_customer): This counts the number of rows where the rating\_by\_customer column has a value less than 2 (rating\_by\_customer < 2). The COUNT() function is used here without an alias, so the resulting count will be displayed in the output without a specific column name.

FROM booking\_data: This specifies the source table booking\_data from which data is being queried.

WHERE rating\_by\_customer < 2: This filters the rows of booking\_data to include only those where the rating\_by\_customer value is less than 2.

GROUP BY DATE\_FORMAT(pickup\_timestamp, 'yyyy-MM'): This part of the query groups the results based on the formatted pickup\_timestamp values ('yyyy-MM'). This means that rows with the same year and month will be grouped together.

screenshot:



**Task 11: Calculate the count of total iOS users.**

**select os\_version ,count(distinct customer\_id) from clickstream\_data where os\_version in ('iOS') group by os\_version;**

This query is useful for analyzing the distribution of unique customers across different versions of the iOS operating system recorded in your clickstream data.

Explanation of each part:

SELECT os\_version, COUNT(DISTINCT customer\_id): This part of the query selects the os\_version column and counts the distinct customer\_id values from the clickstream\_data table.

os\_version: This is the column name representing the operating system version.

COUNT(DISTINCT customer\_id): This function calculates the number of unique customer\_id values for each os\_version.

FROM clickstream\_data: This specifies the source table clickstream\_data from which data is being queried.

WHERE os\_version IN ('iOS'): This part of the query filters the rows to include only those where the os\_version is 'iOS'. The IN ('iOS') condition checks if the os\_version matches the specified value 'iOS'.

GROUP BY os\_version: This groups the results by os\_version, meaning that the counts will be aggregated and displayed for each unique os\_version value ('iOS' in this case).

screenshot:

