Operation Analytics and Investigating Metric Spike

Project Description :

The **"Operation Analytics and Investigating Metric Spike"** project aims to analyze operational data and investigate spikes or anomalies in key metrics. The purpose of this project is to gain insights into the performance and behavior of systems or processes, identify any unusual patterns or trends, and take appropriate actions to address them.

Tech-Stack :

MySQL Workbench: Version 8.0.26

MySQL Workbench was selected as the IDE for database development and administration. It provides a user-friendly interface for designing, modelling , and managing MySQL databases, along with tools for writing and executing SQL queries, making it an ideal choice for this project.

Approach:

Understanding Requirements: Initially, I thoroughly reviewed the project requirements and objectives to gain a clear understanding of the goals and expectations.

Data Collection: Utilizing SQL queries, I extracted relevant data from the Instagram database, including user profiles, posts, likes, comments, and engagement metrics. This involved connecting to the MySQL database using MySQL Workbench and executing SELECT queries to retrieve the required data.

User Segmentation: Utilize SQL queries to segment users based on demographics, interests, and engagement levels, enabling the identification of distinct audience groups.

Content Analysis: Analyze content characteristics such as post types, captions, hashtags, and visual elements using SQL queries to understand what resonates most with the audience.

Engagement Analysis: Utilize SQL queries to examine factors influencing user engagement, such as posting frequency, timing, and interaction with followers, to optimize engagement strategies.

Sentiment Analysis: Perform sentiment analysis on comments using SQL queries to gauge audience sentiment and preferences.

SQL Tasks :

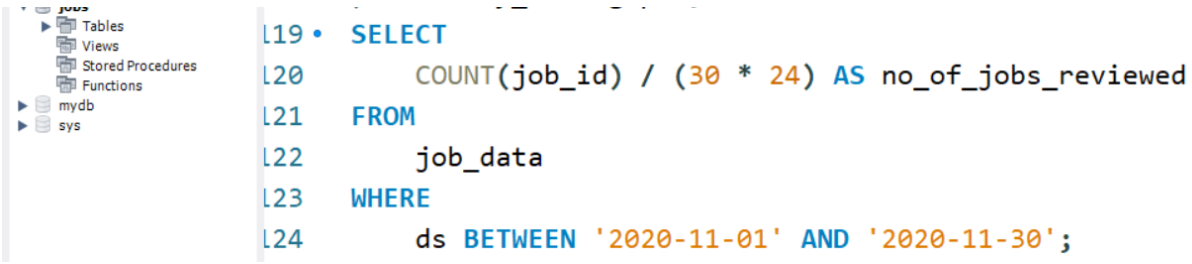
Case Study 1: Job Data Analysis

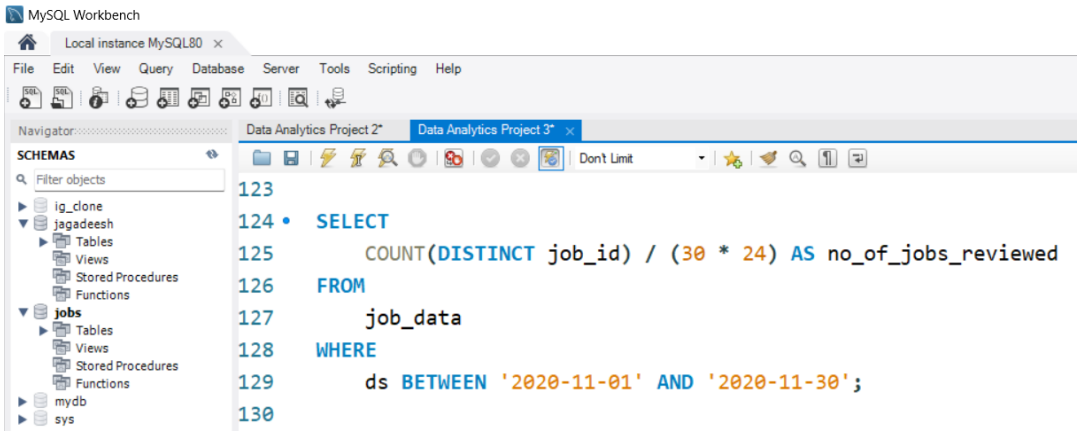
1.Jobs Reviewed Over Time:

Objective: Calculate the number of jobs reviewed per hour for each day in November 2020.

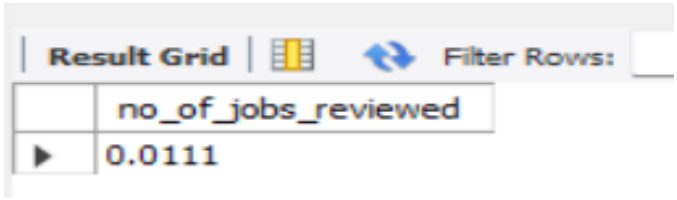
Task: Write an SQL query to calculate the number of jobs reviewed per hour for each day in November 2020.

SQL Query:

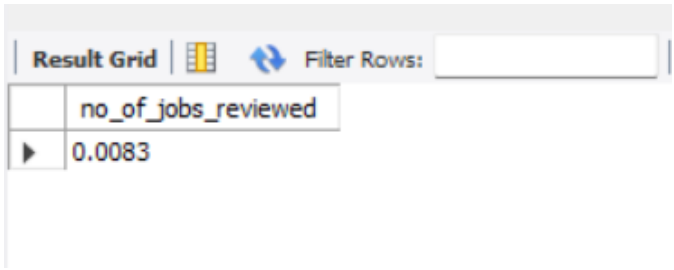




Output:



This output shows the distinct result



Explanation:

1. **SELECT COUNT(DISTINCT job_id):**This part of the query calculates the count of distinct job_id values. **COUNT(DISTINCT)** is an aggregate function that counts the number of unique occurrences of the specified column (job_id in this case).
2. **(30 * 24):**After counting the distinct job_id values, the result is divided by (30 * 24). This calculation represents the total number of hours in the specified date range, which is November 2020.
3. **AS no_of_jobs_reviewed:**This part of the query assigns a label **(no_of_jobs_reviewed)** to the calculated result, which represents the average number of jobs reviewed per hour over the specified date range.

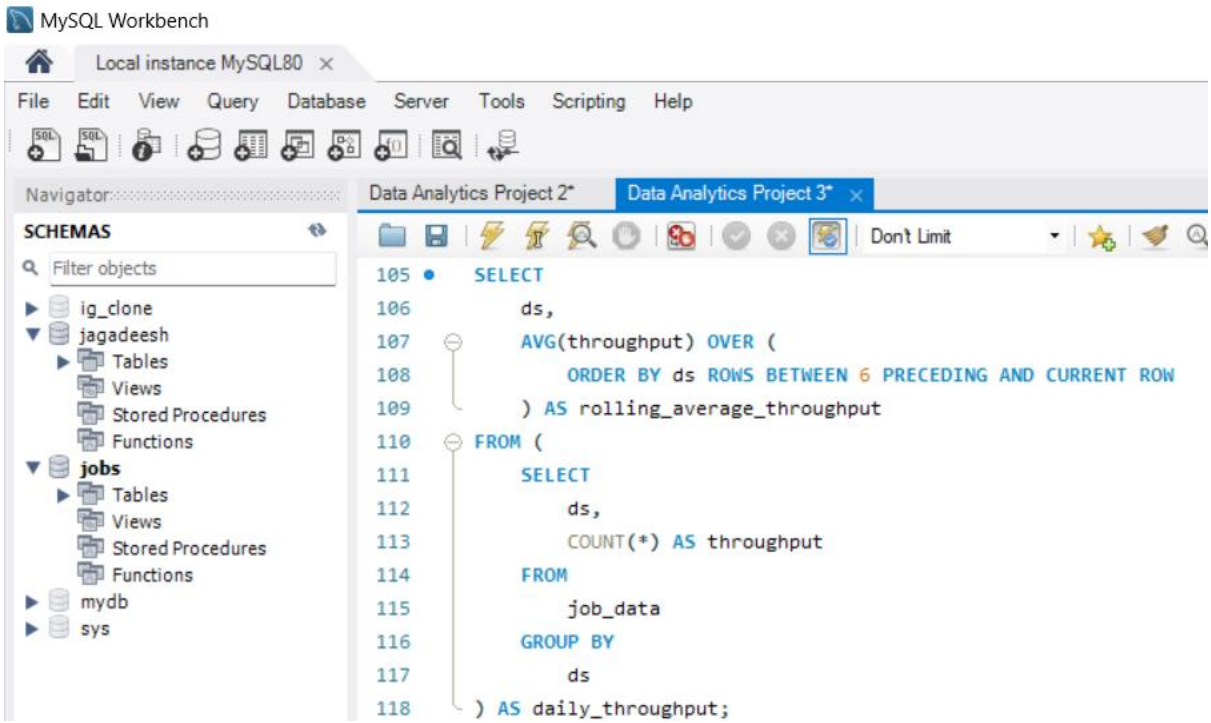
4. **FROM job_data WHERE ds BETWEEN '2020-11-01' AND '2020-11-30':**The FROM clause specifies the table from which the data is retrieved **(job_data).**
5. The **WHERE** clause filters the data based on the ds (date) column, restricting it to dates between November 1st, 2020, and November 30th, 2020.

2. Throughput Analysis:

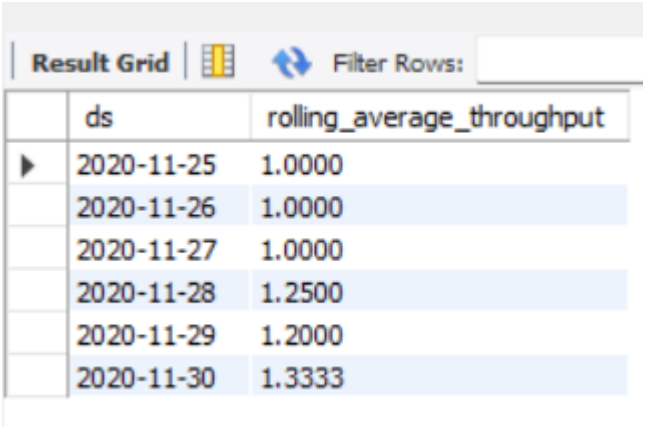
Objective: Calculate the 7-day rolling average of throughput (number of events per second).

Task: Write an SQL query to calculate the 7-day rolling average of throughput. Additionally, explain whether you prefer using the daily metric or the 7-day rolling average for throughput, and why.

SQL Query:



Output:



Result Grid | Filter Rows:

ds	rolling_average_throughput
2020-11-25	1.0000
2020-11-26	1.0000
2020-11-27	1.0000
2020-11-28	1.2500
2020-11-29	1.2000
2020-11-30	1.3333

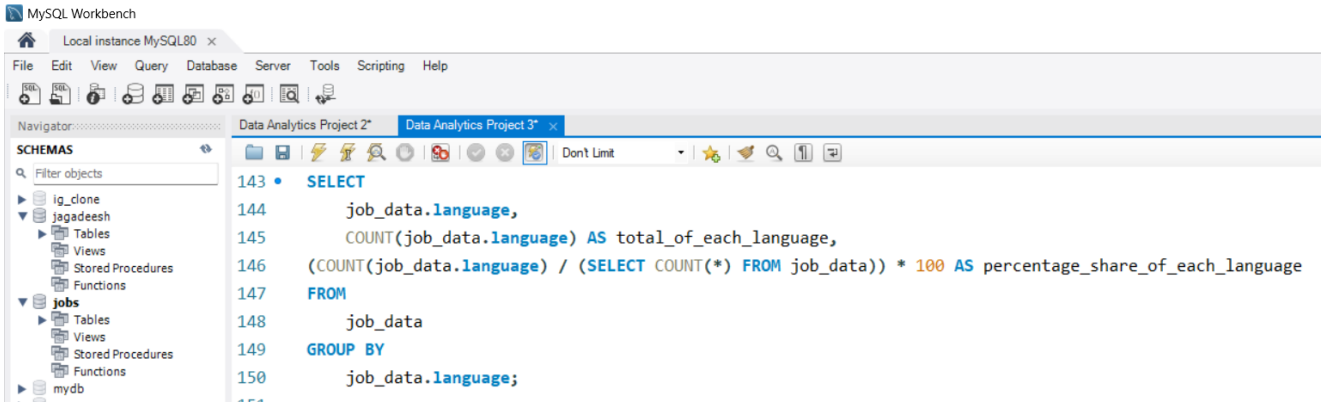
Explanation:

1. **Inner Subquery (SELECT ds, COUNT(*)** AS throughput **FROM job_data GROUP BY ds):**This subquery calculates the total throughput (number of jobs) for each date (ds) in the **job_data** table. It counts the number of jobs reviewed on each day.
2. **Outer Query** (SELECT ds, AVG(throughput) **OVER** (ORDER BY ds ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) **AS** rolling_average_throughput **FROM** AS daily_throughput):
3. The outer query selects the date (ds) from the inner subquery and applies a window function **AVG**(throughput) **OVER** to calculate the 7-day rolling average of throughput.
4. The **OVER** clause defines the **window frame** as the current row and the preceding 6 rows (i.e., the 7-day window).
5. The **ORDER BY** ds ensures that the window frame is ordered by date (ds).

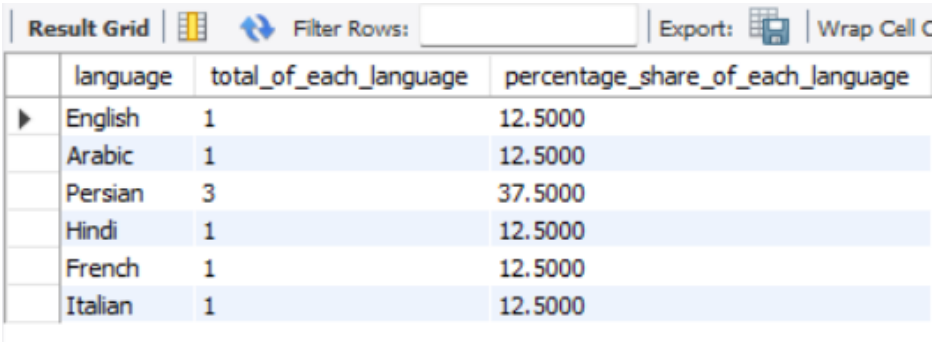
3.Language Share Analysis:

Objective: Calculate the percentage share of each language in the last 30 days.
Task: Write an SQL query to calculate the percentage share of each language over the last 30 days.

SQL Query:



Output:



language	total_of_each_language	percentage_share_of_each_language
English	1	12.5000
Arabic	1	12.5000
Persian	3	37.5000
Hindi	1	12.5000
French	1	12.5000
Italian	1	12.5000

Explanation:

1. The **GROUP BY** job_data.language clause ensures that the **COUNT** functions are applied for each distinct language.
2. We removed **job_data.job_id** from the **SELECT** list because it was not included in the GROUP BY clause, and it's not being used in any aggregate functions.
3. We directly use **COUNT(job_data.language)** to count the occurrences of each language.
4. The percentage share is calculated by dividing the count of each language by the total count of all records in the **job_data** table.

4.Duplicate Rows Detection:

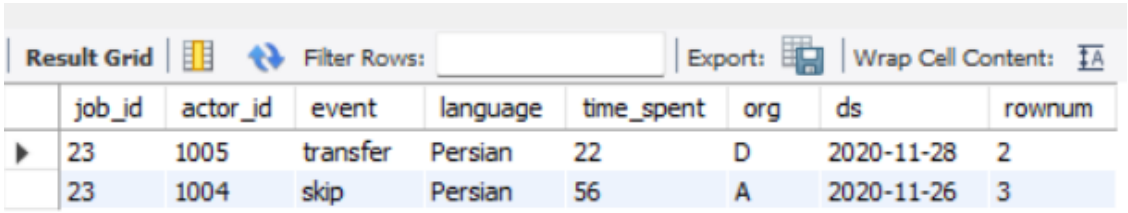
Objective: Identify duplicate rows in the data.

Task: Write an SQL query to display duplicate rows from the job_data table.

SQL Query:

```
select * from
( select *, row_number()over(partition by job_id) as rownum from job_data )a
where rownum>1;
```

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

job_id	actor_id	event	language	time_spent	org	ds	rownum
23	1005	transfer	Persian	22	D	2020-11-28	2
23	1004	skip	Persian	56	A	2020-11-26	3

Explanation:

1. **Inner Subquery:** The inner subquery selects all columns (*) from the **job_data** table and adds a new column **rownum** using the **ROW_NUMBER()** window function.
2. The **ROW_NUMBER()** function assigns a unique sequential number to each row within a partition defined by **job_id.** This means that rows with the same **job_id** will have consecutive row numbers.
3. **Outer Query:** The outer query selects all columns from the result of the inner subquery **(SELECT * FROM a).**
4. The **WHERE** clause filters the results to only include rows where the **rownum** is greater than 1. This effectively selects rows with duplicate **job_id**, as they would have **rownum** values greater than 1.

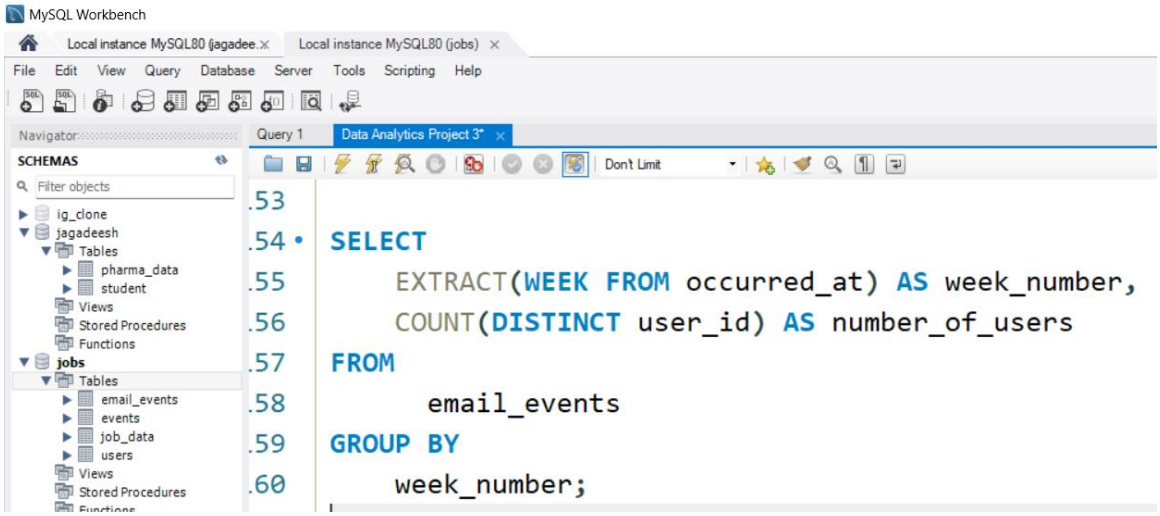
Case Study 2: Investigating Metric Spike

1.Weekly User Engagement:

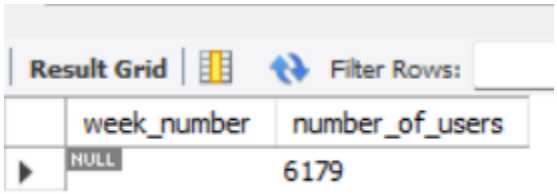
Objective: Measure the activeness of users on a weekly basis.

Task: Write an SQL query to calculate the weekly user engagement.

SQL Query:



Output:



Explanation:

1. **SELECT EXTRACT(WEEK FROM occurred_at) AS week_number:** This part of the query selects the week number from the **occurred_at** column in the **email_events table**. The **EXTRACT** function is used to extract the week component from a date. The **AS** week_number renames the extracted week number to **week_number** for clarity in the result set.
2. **COUNT(DISTINCT user_id) AS number_of_users:** This part of the query counts the number of distinct **user_id** values in the dataset. Each row represents a unique user.

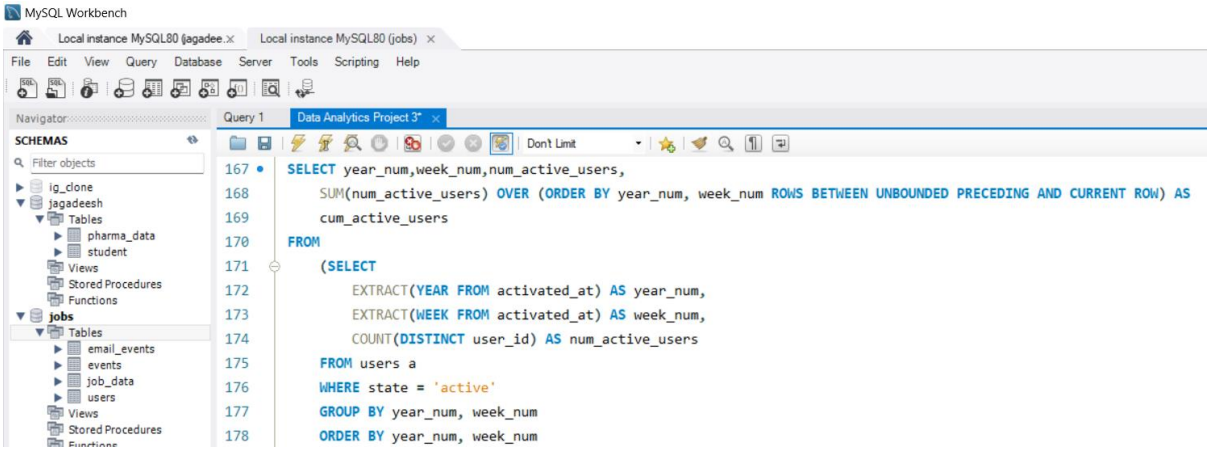
3. **FROM events:** This specifies the source table from which the data is being retrieved. In this case, it's the **events** table in the tutorial database.
4. **GROUP BY week_number:** This clause groups the results by the week_number column. It means that the count of distinct users will be aggregated for each unique week.

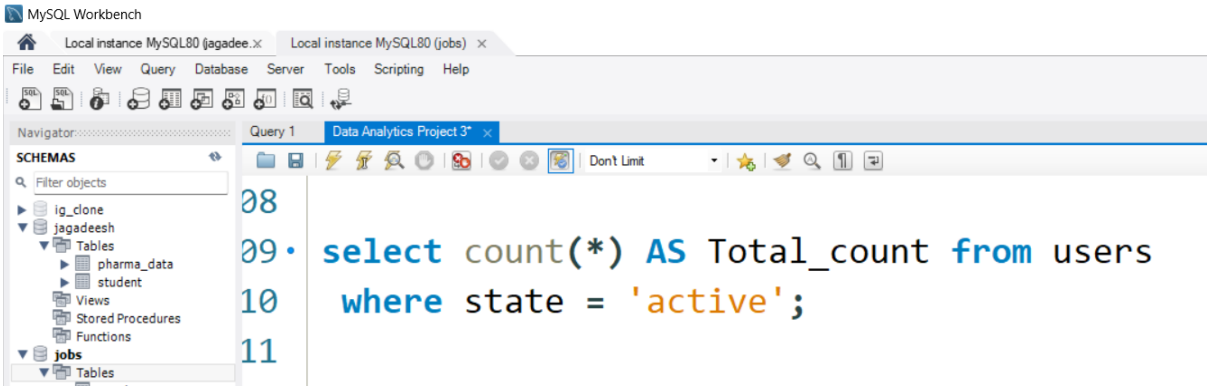
2. User Growth Analysis:

Objective: Analyze the growth of users over time for a product.

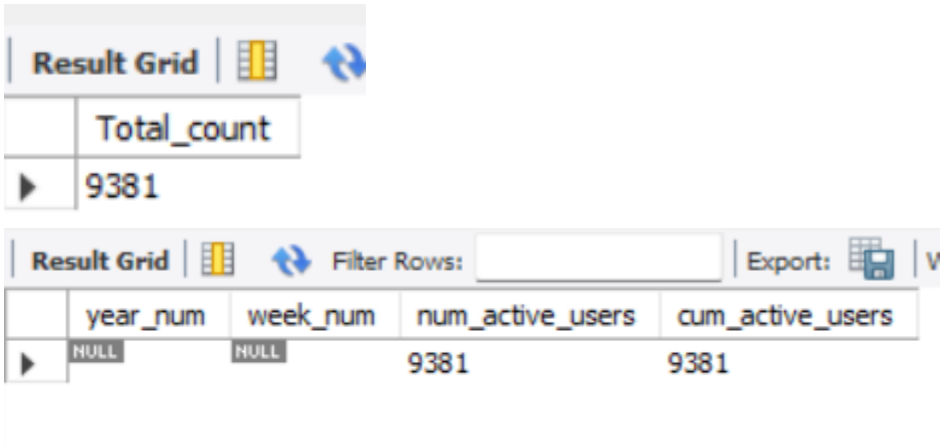
Task: Write an SQL query to calculate the user growth for the product.

SQL Query:





Output:



Explanation:

1. **SELECT COUNT(*) AS Total_count:** This part of the query selects the count of all rows (* represents all columns) in the users table where the condition specified in the WHERE clause is met. The **AS Total_count** alias renames the count result as **Total_count** for clarity in the result set.
2. **FROM users:** This specifies the source table from which the data is being retrieved. In this case, it's the users table.
3. **WHERE state = 'active':** This part of the query filters the rows from the users table where the state column is equal to **'active'.** This condition ensures that only users with an active state are counted.
4. This **inner query** selects the year and week number from the **activated_at** column in the users table, extracts the year and week using the **EXTRACT** function, and counts the distinct number of active users for each year and week combination.
5. The **WHERE** clause filters the rows where the state column is equal to **'active'.**
6. The results are grouped by **year_num** and **week_num** and then ordered by **year_num** and **week_num.**
7. This part of the query calculates the cumulative sum of **num_active_users** over the ordered set of rows by **year_num** and week_num.
8. **ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW** specifies the window frame for the cumulative sum. It includes all rows from the beginning of the ordered set to the current row.

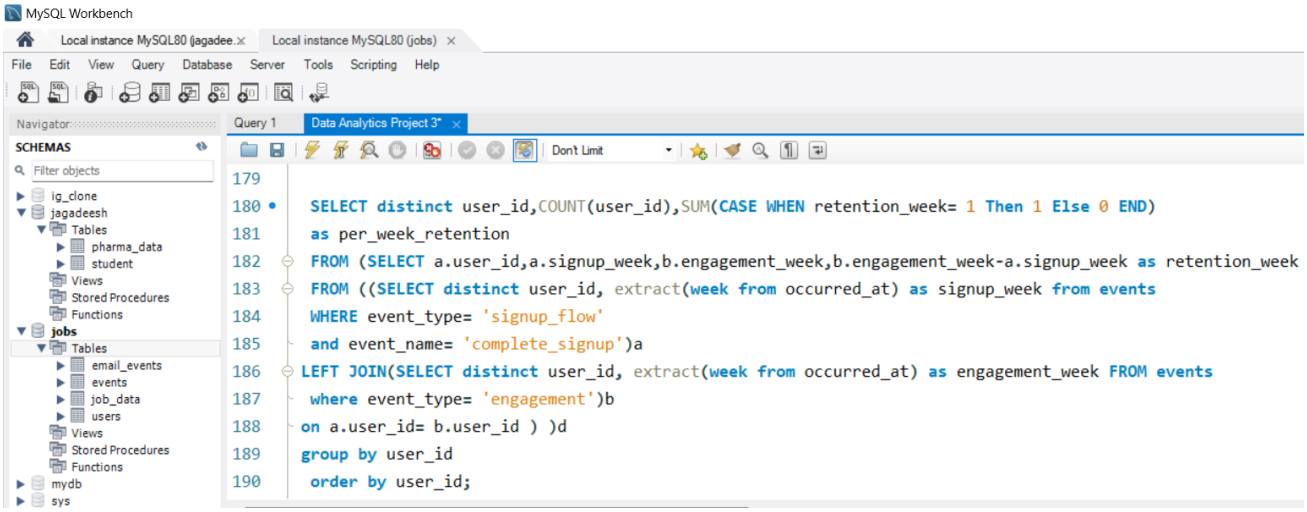
9. This **outer query** selects the **year_num, week_num, num_active_users,** and **cum_active_users** columns from the results of the inner query.

3.Weekly Retention Analysis:

Objective: Analyze the retention of users on a weekly basis after signing up for a product.

Task: Write an SQL query to calculate the weekly retention of users based on their sign-up cohort.

SQL Query:



Output:

https://drive.google.com/file/d/11fyTnJ3MfcMnqDT9w11uSWDnUBs-Wbwl/view?usp=sharing

Note: The output for this query is too big so I attached the output in Google drive link kindly refer this.

Explanation:

1. This part of the query selects distinct **user_id** values from the dataset.
2. It counts the occurrences of each **user_id**.
3. It sums the occurrences where **retention_week** is equal to 1, effectively counting the number of users who were retained in the first week after signup.

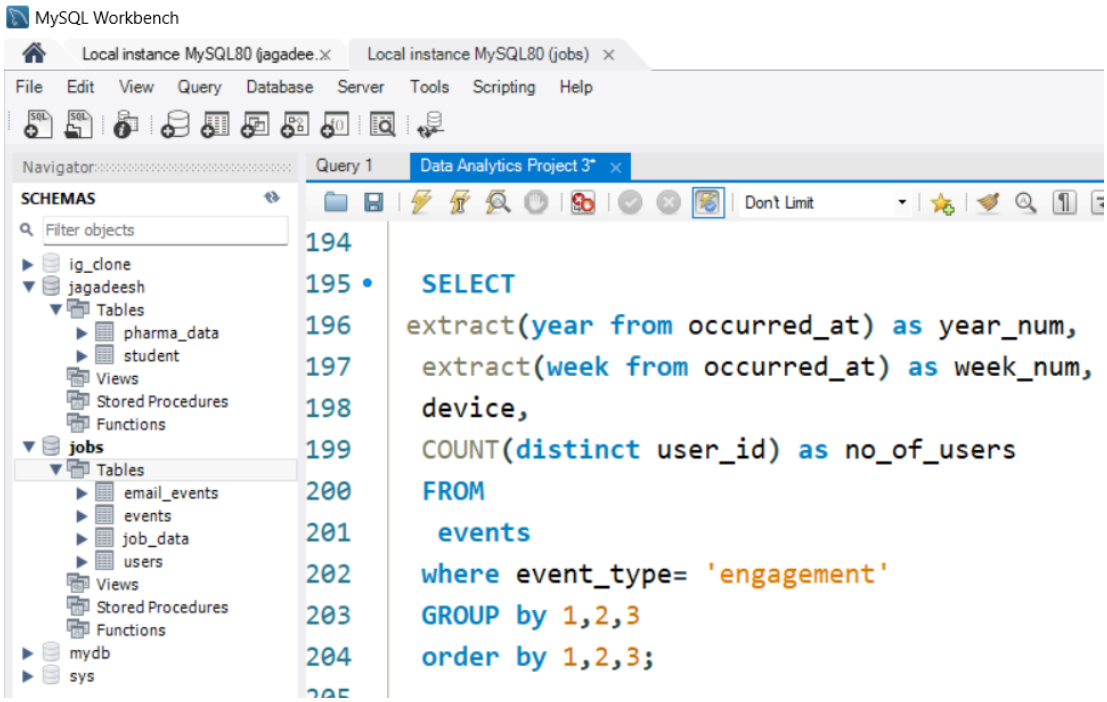
4. This subquery retrieves the **user_id, signup_week, engagement_week,** and calculates the **retention_week** (the difference between engagement_week and signup_week) for each user.
5. It first selects distinct **user_id** values along with the week of sign-up (signup_week) from the events table where the event type is **'signup_flow'** and the event name is **'complete_signup'**.
6. It then left joins this result with another subquery that selects distinct **user_id** values along with the week of engagement **(engagement_week)** from the events table where the event type is 'engagement'. This allows matching sign-up events with engagement events.
7. The difference between the **engagement_week** and **signup_week** calculates the retention week.
8. Finally, the results are **grouped by** user_id, and the result set is **ordered by** user_id.

4. Weekly Engagement Per Device:

Objective: Measure the activeness of users on a weekly basis per device.

Task: Write an SQL query to calculate the weekly engagement per device.

SQL Query:



Output:

https://drive.google.com/file/d/1hFeNXrismr2_Ch9rvShPq3HHdqJ_6r8H/view?usp=sharing

Note: The output for this query is too big so I attached the output in Google drive link kindly refer this.

Explanation:

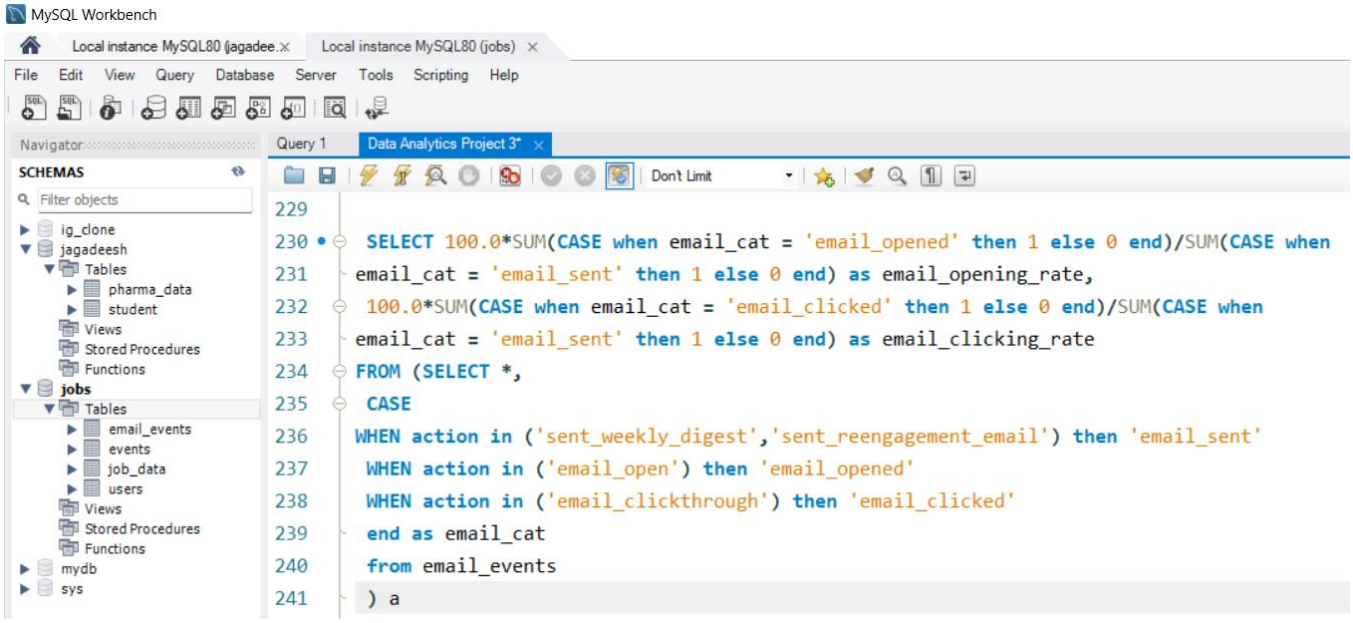
1. **STR_TO_DATE(occurred_at, '%d-%m-%Y %H:%i')** converts the string occurred_at to a valid date-time format using the format specifier **%d-%m-%Y %H:%i**, which matches the format "02-05-2014 11:02".
2. Then, **EXTRACT(YEAR)** and **EXTRACT(WEEK)** are used to extract the year and week from the converted date-time value.
3. The results are grouped by the extracted **year_num, week_num**, and device, ensuring accurate grouping and counting.
4. Finally, the results are ordered by **year_num, week_num,** and device.

5.Email Engagement Analysis:

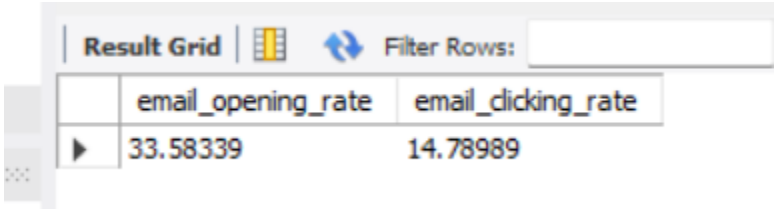
Objective: Analyze how users are engaging with the email service.

Task: Write an SQL query to calculate the email engagement metrics.

SQL Query:



Output:



Explanation:

1. The **inner subquery** selects data from the **email_events table** and categorizes email events into three categories: **'email_sent'**, **'email_opened'**, and **'email_clicked'**.
2. This categorization is done using a **CASE** statement based on the values in the **'action' column**. Each email event is assigned a category **(email_cat)** depending on the action type.
3. The outer query performs conditional aggregation using the **SUM()** function with a **CASE** statement.
4. It calculates the count of occurrences for each email action type **('email_opened', 'email_clicked', and 'email_sent').** The counts are then used to calculate the email opening rate and the email clicking rate as

percentages by dividing the counts of **'email_opened'** and **'email_clicked'** by the count of ***'email_sent'***. Multiplying by **100.0** converts the result to a percentage.

5. The query includes a safeguard against division by zero. It checks if the count of **'email_sent'** records is zero. If it is, the result is set to **NULL.** This prevents division by zero errors and ensures the query returns meaningful results even when there are no sent emails.

Result:

The project **"Operation Analytics and Investigating Metric Spike"** involved analyzing operational data to understand performance metrics and investigate sudden spikes or anomalies.

Through this project:

1. **Performance Analysis:** We analyzed metrics like throughput, latency, and error rates to understand system performance.
2. **Anomaly Detection:** We identified abnormal spikes in metrics, indicating underlying issues or exceptional events.
3. **Root Cause Analysis:** Investigating these spikes involved pinpointing the underlying issues or triggers.
4. **Decision Support:** Insights from the analysis supported informed decision-making, such as optimizing system configurations or allocating resources.
5. **Continuous Improvement:** The project fostered a culture of continuous improvement by addressing identified issues over time.
6. **Cross-Functional Collaboration:** Collaboration between teams from different domains enhanced understanding and decision-making.
7. **Knowledge Transfer:** Lessons learned and insights gained were shared across the organization, enriching collective knowledge.