Operation Analytics and Investigating Metric Spike

The Operational Analytics project aims to leverage advanced SQL skills to analyse various datasets and tables to derive valuable insights for a company. As a Lead Data Analyst, the primary goal is to improve the company's operations by investigating and explaining sudden changes in key metrics. The project involves close collaboration with different departments, including operations, support, and marketing, to provide actionable insights that contribute to overall business enhancement.

Tech-Stack Used : MySQL 8.0 CE

1. User Friendly
2. Performance
3. Open Source

Job Data Analysis

A. Jobs Reviewed Over Time:

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

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

SQL CODE:

```
SELECT
  DAY (STR_TO_DATE(ds, '%m/%d/%Y')) AS `Day`,
  COUNT (*) AS `Jobs Per Day`,
  SUM (time_spent) / 3600 AS `Hours Spent Per Day`
FROM
  job_data
GROUP BY `Day`;
```

Insights:

The result of this query would be a table with three columns: Day, Jobs Per Day, and Hours Spent Per Day. Each row in the result represents a day, and the columns provide insights into the number of jobs and the total hours spent on jobs for each day. Helps us in analyzing the distribution of work over time and understanding the workload on different days.

B. Throughput Analysis:

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

Your 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 CODE:

```
SELECT ds, event_per_day,AVG(event_per_day) OVER (ORDER BY ds ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS 7_day_rolling_avg

FROM  (

SELECT ds, COUNT(DISTINCT event) AS event_per_day

FROM job_data

GROUP BY ds

ORDER BY ds) as Eventperday;
```

Insights:

This query gives a result with three columns: ds, event_per_day, and 7_day_rolling_avg. The 7_day_rolling_avg column contains the 7-day rolling average of the daily count of distinct events. It's a useful way to smooth out fluctuations and identify trends over a weekly period.

The preference between using a daily metric or a 7-day rolling average for throughput analysis depends on the specific goals. A daily metric offers high granularity and is suitable for real-time monitoring and operational decision-making, while a 7-day rolling average smoothens fluctuations, providing a more stable view for strategic planning and long-term trends. The choice should align with the nature of the data, the desired level of detail, and the analytical focus on short-term fluctuations or broader performance trends. A combination of approaches can offer a better understanding of throughput.

C. Language Share Analysis:

Objective: Calculate the percentage share of each language in the last 30 days.

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

SQL CODE:

```
SELECT `language`,
                  count(*) * 100/ sum(count(*)) over() as `Percetage Share`
FROM job_data
GROUP BY `language`;
```

Insights:

This information can be valuable for understanding which languages are most commonly used or encountered in the dataset.

Languages with a higher percentage share are more dominant in the dataset. This can be useful for decision-making, resource allocation, or targeting specific language-related tasks.

D. Duplicate Rows Detection:

Objective: Identify duplicate rows in the data.

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

SQL CODE:

```
SELECT ds, job_id, actor_id, `event`, `language`, time_spent, org
FROM      (
SELECT *,
```

ROW_NUMBER () OVER (PARTITION BY ds,job_id,actor_id,`event`,`language`,time_spent,org) AS rep_count

FROM job_data

) AS DuplicateCount

WHERE rep_count > 1;

Insights:

The SQL query you provided is used to identify and retrieve rows in the job_data table that are considered duplicates based on specific columns (ds, job_id, actor_id, event, language, time_spent, and org) , i.e. a row is a duplicate if another row has all the same values in all the columns.

Investigating Metric Spike

A. Weekly User Engagement:

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

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

SQL CODE:

```
SELECT

   EXTRACT(WEEK FROM STR_TO_DATE(occurred_at, ‘%d-%m-%Y %H:%i’)) AS `Week`,

   COUNT(DISTINCT user_id) AS User_count,

   COUNT(event_type) AS Engagement_count

FROM

   `events`

WHERE

   event_type = 'engagement'

GROUP BY `Week`

ORDER BY `Week`;
```

Insights:

By grouping events into weeks, the query enables the identification of trends or patterns in user engagement over time.

The analysis allows for insights into how user engagement fluctuates on a weekly basis, helping to understand user behavior and preferences.

B. User Growth Analysis:

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

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

SQL CODE:

```
SELECT `year`, `month`, active_users,
            sum(active_users) OVER(ROWS BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW) AS user_growth
FROM (
      SELECT EXTRACT(year FROM str_to_date(activated_at, ‘%d-
%m-%Y %H:%i’)) AS `year`,
                  EXTRACT(MONTH FROM
str_to_date(activated_at, ‘%d-%m-%Y %H:%i’)) AS `month`,
                        COUNT(DISTINCT user_id) AS
active_users
            FROM users
            WHERE state = 'active'
            GROUP BY `year`,`month`
            ) as Active_Users_per_Month
ORDER BY `year`;
```

Insights:

The query offers insights into monthly user acquisition, active user counts, and the overall growth trajectory, aiding strategic planning and decision-making.

The cumulative sum enables a deeper understanding of overall user growth, highlighting the aggregated impact of each month on the total user base.

C. Weekly Retention Analysis:

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

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

SQL CODE:

```
SELECT
   signup_week,
   COUNT(DISTINCT user_id) AS total_users,
   COUNT(DISTINCT CASE WHEN Week_1_Active > 0 THEN
user_id END) AS Week_1_Retained,
   COUNT(DISTINCT CASE WHEN Week_2_Active > 0 THEN
user_id END) AS Week_2_Retained,
   COUNT(DISTINCT CASE WHEN Week_3_Active > 0 THEN
user_id END) AS Week_3_Retained,
   COUNT(DISTINCT CASE WHEN Week_4_Active > 0 THEN
user_id END) AS Week_4_Retained,
   COUNT(DISTINCT CASE WHEN Week_5_Active > 0 THEN
user_id END) AS Week_5_Retained,
   COUNT(DISTINCT CASE WHEN Active_for_more_than_5_weeks
> 0 THEN user_id END) AS
Active_for_more_than_5_weeks_Retained
FROM (
   SELECT
```

```
        user_id,
        EXTRACT(WEEK FROM STR_TO_DATE(occurred_at, '%d-%m-
%Y %H:%i')) AS signup_week,
        COUNT(DISTINCT EXTRACT(WEEK FROM
STR_TO_DATE(occurred_at, '%d-%m-%Y %H:%i'))) AS
number_of_weeks_active
    FROM
        `events`
    GROUP BY
        user_id, signup_week
) AS user_activity
GROUP BY
    signup_week;
```

Insights:

The columns like Week_1_Retained, Week_2_Retained, etc., represent the number of users who remained active in the subsequent weeks after signing up. This can help you analyze user retention over time.

The column Active_for_more_than_5_weeks_Retained shows the number of users who remained active for more than 5 weeks. This metric can provide insights into long-term user engagement.

By analyzing these retention metrics, you can identify patterns, trends, and potential areas for improvement in retaining users over time.

D. Weekly Engagement Per Device:

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

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

SQL CODE:

```
SELECT EXTRACT(YEAR FROM str_to_date(occurred_at, ‘%d-%m-%Y %H:%i’)) AS `Year`,
    EXTRACT(WEEK FROM str_to_date(occurred_at, ‘%d-%m-%Y %H:%i’)) AS `Week`,
    COUNT(DISTINCT user_id) AS UserCount,
    device AS DeviceName
FROM `events`
WHERE event_type = 'engagement'
GROUP BY `year`,`week`,device
ORDER BY `year`, `week`,UserCount DESC ;
```

Insights:

By analysing engagement events on a weekly basis, you can identify trends and patterns over time. Seasonal variations or specific events may be visible in the data.

The breakdown by device allows you to see which devices are most used for engagement. This information is valuable for optimizing user experiences on specific devices.

E. Email Engagement Analysis:

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

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

SQL CODE:

```
SELECT user_id, `Emails Sent`, `Emails Opened`, `Emails Clicked`,
            CASE WHEN `Emails Clicked` = 0 THEN 0
  ELSE
```

```
                    ( sum(`Emails Clicked`)/sum(`Emails Sent` )
OVER(PARTITION BY user_id)) *100

       END AS `Success Percentage`

       FROM (

          SELECT user_id,

                    sum(CASE WHEN `action` = "sent_weekly_digest"
THEN 1 ELSE 0 END) as `Emails Sent`,

                    sum(CASE WHEN `action` = "email_open" THEN 1
ELSE 0 END) as `Emails Opened`,

                    sum(CASE WHEN `action` = "email_clickthrough"
THEN 1 ELSE 0 END ) as `Emails Clicked`

          FROM email_events

          GROUP BY user_id) as dtable

    GROUP BY user_id;
```

Insights:

The query calculates metrics for email campaigns, including the number of emails sent, emails opened, and emails clicked, for each user.

The Success Percentage column is calculated as the percentage of emails clicked relative to the total number of emails sent. This metric is only calculated if there is at least one email clicked (Emails Clicked > 0). Otherwise, the success percentage is set to 0.

This analysis can guide business decisions related to optimizing email campaigns, identifying highly engaged users, and improving overall campaign effectiveness.