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.

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

SQL CODE:

```
SELECT
   DAY(ds) AS `Date`,
   COUNT(*) / (SUM(time_spent) / 3600) AS `Jobs_per_hour`
FROM
```

```
  job_data
WHERE
        ds BETWEEN '2020-11-01' AND '2020-11-30'
GROUP BY `Date`;
```

Insights:

The result of this query would be a table with two columns: Date, and the Jobs reviewed per hour for each day. The Date 28th has the most Jobs reviewed per hour with 218 and Date 29th has the least with 35. This helps us in analyzing the distribution of work overtime and understanding the workload on different days.

B. Throughput Analysis:

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

Task: Calculate the 7-day rolling average of throughput. Additionally, explain whether using the daily metric or the 7-day rolling average for throughput is preferred, and why.

SQL CODE:

```
WITH daily_throughput AS (
                SELECT ds, COUNT(*)/SUM(time_spent) AS throughput
                FROM job_data
                GROUP BY ds
    )
SELECT ds,
        ROUND(AVG(throughput) OVER(ORDER BY ds ROWS BETWEEN 6 PRECEDING
AND CURRENT ROW),3) AS 7_day_throughput_rolling_average
FROM daily_throughput;
```

Insights:

Throughput is a measure of the number of activities processed in a given amount of time, i.e. in this case the number of events per time taken.

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.

C. Language Share Analysis:

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

Task: Calculate the percentage share of each language over the last 30 days.

SQL CODE:

```
SELECT language,
        100*COUNT(*)/SUM(COUNT(*)) OVER() AS `Percetage Share`
FROM job_data
GROUP BY `language`;
```

Insights:

This information can be valuable for understanding which languages are most 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.

Persian has the highest percentage share among other languages with 37.5 %.

D. Duplicate Rows Detection:

Objective: Identify duplicate rows in the data.

Task: To display duplicate rows from the job_data table.

SQL CODE:

```
SELECT *

FROM job_data

GROUP BY ds, job_id, actor_id, event, language, time_spent, org

HAVING COUNT(*) > 1;
```

Insights:

The SQL query provided is used to identify and retrieve rows in the job_data table that are considered duplicates by grouping the rows on each data essentially querying only rows with the same values in each of the columns, ie. Duplicate rows.

There appears to be no duplicate rows.

Investigating Metric Spike

A. Weekly User Engagement:

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

Task: Calculate the weekly user engagement.

SQL CODE:

```
SELECT
   EXTRACT(WEEK FROM occurred_at) AS Week,
   COUNT(DISTINCT user_id) AS User_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.

The results show that the user count kept increasing from week 17 till the week 30 with a user count of 1467 and then started falling off very quickly and reached to a stooping low of 104 users in week 35.

B. User Growth Analysis:

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

Task: 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,
  CASE
    WHEN LAG(active_users) OVER (ORDER BY year, month) IS NULL
THEN 0
    ELSE ROUND(((active_users - LAG(active_users) OVER (ORDER BY
year, month)) / LAG(active_users) OVER (ORDER BY year, month)) * 100,
2)
  END AS growth_percentage
FROM (
      SELECT YEAR(activated_at) AS year,
                    EXTRACT(MONTH FROM activated_at) AS month,
                        COUNT(DISTINCT user_id) AS active_users
              FROM users
              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.

The growth is expressed as a percentage in increase or decrease from the previous months active user count.

C. Weekly Retention Analysis:

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

Task: Calculate the weekly retention of users based on their sign-up cohort.

SQL CODE:

```
WITH user_signups AS (
  SELECT
    user_id,
    MIN(occurred_at) AS signup_date,
    YEAR(MIN(occurred_at)) AS signup_year,
    WEEK(MIN(occurred_at)) AS signup_week
  FROM
    events
  WHERE
    event_name = 'complete_signup'
  GROUP BY
```

```
        user_id
),

user_engagements AS (
    SELECT
        e.user_id,
        c.signup_date,
        c.signup_year,
        c.signup_week,
        TIMESTAMPDIFF(WEEK, c.signup_date, e.occurred_at) AS
weeks_since_signup
    FROM
        events e
    JOIN
        user_signups c ON e.user_id = c.user_id
    WHERE
        e.occurred_at >= c.signup_date
)
SELECT
    signup_year,
    signup_week,
    weeks_since_signup AS week_number,
    COUNT(DISTINCT u.user_id) AS retained_users
FROM
    user_engagements u
GROUP BY
    signup_year,
```

```
    signup_week,
    weeks_since_signup
ORDER BY
    signup_year,
    signup_week,
    weeks_since_signup;
```

Insights:

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

User retained in each week based on their signup cohort is found.

D. Weekly Engagement Per Device:

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

Task: Calculate the weekly engagement per device.

SQL CODE:

```
SELECT device AS DeviceName,
                YEAR(occurred_at) AS Year,
                WEEK(occurred_at) AS Week,
                COUNT(DISTINCT user_id) AS UserCount
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.

Task: Calculate the email engagement metrics.

SQL CODE:

```
SELECT user_id, Emails_sent, Emails_opened, Emails_clicked,
    ROUND(SUM(Emails_opened)/SUM(Emails_sent),2)*100 AS Open_rate,
    ROUND(SUM(Emails_clicked)/SUM(Emails_opened),2)*100 AS Click_through_rate
    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 email_engagement
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.

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

The Open rate calculates the percentage of emails opened to emails sent.

The Click through rate calculates the percentage of emails clicked to emails opened.