Operation Analytics and Investigating Metric Spike

Project Description

Operational Analytics plays a crucial role in enhancing a company's operational efficiency by analyzing its end-to-end processes. This project involves analyzing operational data to provide insights into performance metrics and investigate any sudden changes or anomalies.

Case Study 1: Job Data Analysis

We will analyze a dataset containing job review events to understand job processing patterns, throughput, language distribution, and identify duplicate entries.

Case Study 2: Investigating Metric Spike

We will analyze user engagement metrics, user growth, retention, and email engagement to uncover patterns and explain any sudden changes in key metrics.

Tech-Stack Used

• MySQL Workbench

• SQL

Analysis and Insights

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.

Approach: To calculate the jobs reviewed per hour for each day we need to add the time spent on job and divide it by 3600s (no. of seconds in an hour).

SQL Query:

```
SELECT

    ds,

    COUNT(job_id) AS no_of_jobs_reviewed,

    SUM(time_spent) / 3600 AS reviewed_per_hour

FROM
```

```
    job_data2
WHERE
    ds BETWEEN '2020-11-01' AND '2020-11-30'
GROUP BY ds;
```

ds	no_of_jobs_reviewed	reviewed_per_hour
2020-11-30	4	0.0222
2020-11-29	2	0.0111
2020-11-28	4	0.0183
2020-11-27	2	0.0578
2020-11-26	2	0.0311
2020-11-25	2	0.0250

Insights: This query calculates the number of job reviews on an hourly basis throughout November 2020, providing granular insights into daily review patterns and peak times.

- The number of jobs reviewed daily varies between 2 and 4 jobs.
- November 27 has the highest throughput (0.0578 reviews per second) despite only 2 jobs being reviewed. This indicates a higher review speed on that day.
- The lowest throughput is on November 29 (0.0111 reviews per second), with 2 jobs reviewed, suggesting a slower review process or more time spent per job.

2. Throughput Analysis

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

Approach: I have used CTE to get the daily throughput then implemented the 7 day rolling throughput taking the value from CTE then using OVER() that includes the current day and the six preceding days

SQL Query:

```
WITH daily_throughput AS (
    SELECT
        ds,
        COUNT(event_name) / SUM(time_spent) AS throughput
```

```sql
    FROM
        job_data2
    WHERE
        ds BETWEEN '2020-11-24' AND '2020-11-30'
    GROUP BY
        ds
    ORDER BY
        ds
)
SELECT
    ds,
    throughput,
    AVG(throughput) OVER (
        ORDER BY ds
        ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
    ) AS rolling_avg_throughput
FROM
    daily_throughput;
```

ds	throughput	rolling_avg_throughput
2020-11-25	0.0222	0.02220000
2020-11-26	0.0179	0.02005000
2020-11-27	0.0096	0.01656667
2020-11-28	0.0606	0.02757500
2020-11-29	0.0500	0.03206000
2020-11-30	0.0500	0.03505000

Insights:

Daily Throughput: The number of events processed per second on each day, which helps in understanding daily operational performance.

7-day Rolling Average Throughput: A smoother measure that averages the daily throughput over the past seven days, providing a clearer view of trends and reducing the impact of daily variability.

- The daily throughput shows significant variability, especially between November 27 and November 28.
- The 7-day rolling average provides a clearer trend by mitigating the effect of these daily fluctuations. It shows an overall increase in throughput from November 27

3. Language Share Analysis

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

SQL Query:

```
SELECT
    language_name,
    COUNT(job_id) AS jobs,
     COUNT(job_id) * 100 /sum( count(job_id))over()
              AS percentage_share
FROM
    job_data2
GROUP BY language_name
order by percentage_share desc;
```

language_name	jobs	percentage_share
Persian	6	37.5000
English	2	12.5000
Arabic	2	12.5000
Hindi	2	12.5000
French	2	12.5000
Italian	2	12.5000

Insights: This query determines the language distribution of reviewed jobs over the past 30 days, helping to understand language preferences and potential localization needs.

- **Persian** stands out with the highest percentage share of 37.50%, indicating that it is the most frequently reviewed language in the dataset over the last 30 days.

4. Duplicate Rows Detection

Objective: Identify duplicate rows in the data.

Approach: Implemented having clause with group by function to count the number of duplicate entries

SQL Query:

```
select job_id,count(*) as duplicate_count

from job_data2

group by job_id

having

count(*)>1

order by duplicate_count desc;
```

job_id	duplicate_count
23	6
21	2
22	2
25	2
11	2
20	2

Insights: This query identifies duplicate rows by grouping all columns and counting occurrences, helping to clean the data and ensure accuracy in further analysis.

- Job_id 26 has maximum duplicate count.

Case Study 2: Investigating Metric Spike

1. Weekly User Engagement

Objective: Measure the activeness of users on a weekly basis.
Write an SQL query to calculate the weekly user engagement.

Approach:

- I implemented the extract function to extract the year and the week and grouped them to get activeness of users on weekly basis.
- Further I have taken the toatal activities per week to get the average number of activities per user

SQL Query:

```
SELECT

    EXTRACT(YEAR FROM occurred_at) AS activity_year,

    EXTRACT(WEEK FROM occurred_at) AS activity_week,

    COUNT(DISTINCT user_id) AS active_users,

    COUNT(*) AS total_activities,

    COUNT(*) / COUNT(DISTINCT user_id) AS avg_activities_per_user

FROM

    events

GROUP BY

    EXTRACT(YEAR FROM occurred_at),

    EXTRACT(WEEK FROM occurred_at)

ORDER BY

    activity_year,

    activity_week;
```

	activity_year	activity_week	active_users	total_activities	avg_activities_per_user
▶	2014	17	663	8091	12.2036
	2014	18	1068	17504	16.3895
	2014	19	1113	17409	15.6415
	2014	20	1154	18087	15.6733
	2014	21	1121	17334	15.4630
	2014	22	1186	18609	15.6906
	2014	23	1232	18476	14.9968
	2014	24	1275	19281	15.1224

Insights:

- From week 17 to week 30, there is a clear upward trend in the number of users, peaking at 1467 users in week 30.
- The total number of activities also shows an upward trend, reaching a peak of 21771 in week 30.
- The average activities per user varied from a high of 16.39 in week 18 to a low of 7.71 in week 35.

2. User Growth Analysis

Objective: Analyze the growth of users over time for a product.
Write an SQL query to calculate the user growth for the product.

Approach:

- I have calculated the user growth on monthly basis by extracting the month.
- Implemented over() to get the cumulative user growth

SQL Query:

```
SELECT

    month(created_at) AS month,

    COUNT(user_id) AS new_users,

    SUM(COUNT(user_id)) OVER (ORDER BY month(created_at)) AS cumulative_users

FROM

    users

GROUP BY
```

```
    month(created_at)
ORDER BY
    month;
```

month	new_users	cumulative_users
1	712	712
2	685	1397
3	765	2162
4	907	3069
5	993	4062

Insights:

- The number of new users increased steadily each month, reaching a peak in August with 1347 new users.
- The number of new users declined to 330in september, a stark contrast to the growth trend.
- The number of new users starts to recover slowly, with 390 in October, 399 in November, and 486 in December.
- By December, the cumulative user base reaches 9381, reflecting a strong overall growth despite the mid-year dip

3. Weekly Retention Analysis

Objective: calculate the weekly retention of users based on their sign-up cohort

Approach:

- Firstly, I extracted the date from created_at column then added an interval of 7,14,21,28 using date_add()to get the start of the week .
- Implement a left join operation on user and events table on user_id column
- Implemented count() and checked whether the event occurred date is >=user created_at.for interval 7,14,21,28 If condition implies to true then it will be counted in the particular interval.

SQL Query:

```
SELECT
```

```
    DATE(u.created_at) AS signup_date,

    DATE_ADD(DATE(u.created_at), INTERVAL 7 DAY) AS week1_start,

    DATE_ADD(DATE(u.created_at), INTERVAL 14 DAY) AS week2_start,

    DATE_ADD(DATE(u.created_at), INTERVAL 21 DAY) AS week3_start,

    DATE_ADD(DATE(u.created_at), INTERVAL 28 DAY) AS week4_start,

    COUNT(DISTINCT CASE WHEN e.occurred_at >= DATE_ADD(DATE(u.created_at),
INTERVAL 7 DAY) THEN u.user_id END) AS week1_retention,

    COUNT(DISTINCT CASE WHEN e.occurred_at >= DATE_ADD(DATE(u.created_at),
INTERVAL 14 DAY) THEN u.user_id END) AS week2_retention,

    COUNT(DISTINCT CASE WHEN e.occurred_at >= DATE_ADD(DATE(u.created_at),
INTERVAL 21 DAY) THEN u.user_id END) AS week3_retention,

    COUNT(DISTINCT CASE WHEN e.occurred_at >= DATE_ADD(DATE(u.created_at),
INTERVAL 28 DAY) THEN u.user_id END) AS week4_retention

FROM

    users u

LEFT JOIN

    events e ON u.user_id = e.user_id

GROUP BY

    signup_date, week1_start, week2_start, week3_start, week4_start

ORDER BY

    signup_date;
```

signup_date	week1_start	week2_start	week3_start	week4_start	week1_retention	week2_retention	week3_retention	week4_retention
2014-01-22	2014-01-29	2014-02-05	2014-02-12	2014-02-19	4	4	4	4
2014-01-23	2014-01-30	2014-02-06	2014-02-13	2014-02-20	2	2	2	2
2014-01-24	2014-01-31	2014-02-07	2014-02-14	2014-02-21	7	7	7	7
2014-01-25	2014-02-01	2014-02-08	2014-02-15	2014-02-22	2	2	2	2
2014-01-26	2014-02-02	2014-02-09	2014-02-16	2014-02-23	4	4	4	4
2014-01-27	2014-02-03	2014-02-10	2014-02-17	2014-02-24	10	10	10	10
2014-01-28	2014-02-04	2014-02-11	2014-02-18	2014-02-25	12	12	12	12
2014-01-29	2014-02-05	2014-02-12	2014-02-19	2014-02-26	11	11	11	11
2014-01-30	2014-02-06	2014-02-13	2014-02-20	2014-02-27	9	9	9	9
2014-01-31	2014-02-07	2014-02-14	2014-02-21	2014-02-28	9	9	9	9

Insights:

- This query measures weekly user retention by comparing user activity weeks to their sign-up weeks, helping to understand long-term engagement and retention patterns.
- Almost all the user who signed up for the product were using it in subsequent weeks.

4. Weekly Engagement Per Device

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

Approach: I implemented this query using where clause having type as engagement and grouping it by devices

SQL Query:

```
SELECT
    EXTRACT(YEAR FROM occurred_at) AS activity_year,
    EXTRACT(WEEK FROM occurred_at) AS activity_week,
    device,
    COUNT(DISTINCT user_id) AS active_users,
    COUNT(*) AS total_activities,
    COUNT(*) / COUNT(DISTINCT user_id) AS avg_activities_per_user
FROM
    events where event_type="engagement"
GROUP BY
      device,
    EXTRACT(YEAR FROM occurred_at),
    EXTRACT(WEEK FROM occurred_at)


ORDER BY
    activity_year,
    activity_week;
```

activity_year	activity_week	device	active_users	total_activities	avg_activities_per_use
2014	17	acer aspire desktop	9	67	7.4444
2014	17	acer aspire notebook	20	206	10.3000
2014	17	amazon fire phone	4	83	20.7500
2014	17	asus chromebook	21	251	11.9524
2014	17	dell inspiron desktop	18	187	10.3889
2014	17	dell inspiron notebook	46	503	10.9348
2014	17	hp pavilion desktop	14	132	9.4286

Insights:

- It can be noted that highest number of active users are using macbook pro device.
- The device that is used least is amazon fire phone

5. Email Engagement Analysis

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

Approach:

- I implemented the case statements to count() the total emails sent, total emails opened and total emails clicked

SQL Query

```
SELECT

    COUNT(CASE WHEN ee.action IN ('sent_weekly_digest', 'sent_reengagement_email') THEN ee.user_id END) AS total_emails_sent,

    COUNT(CASE WHEN ee.action = 'email_open' THEN ee.user_id END) AS total_emails_opened,

    COUNT(CASE WHEN ee.action = 'email_clickthrough' THEN ee.user_id END) AS total_emails_clicked,

    ROUND(

        (COUNT(CASE WHEN ee.action = 'email_open' THEN ee.user_id END) * 100.0 /

        COUNT(CASE WHEN ee.action IN ('sent_weekly_digest', 'sent_reengagement_email') THEN ee.user_id END)), 2

    ) AS overall_open_rate,

    ROUND(

        (COUNT(CASE WHEN ee.action = 'email_clickthrough' THEN ee.user_id END) * 100.0 /

        COUNT(CASE WHEN ee.action LIKE 'sent_%' THEN ee.user_id END)), 2

    ) AS overall_click_rate

FROM

    email_events ee;
```

```
FROM email_events ee;
```

total_emails_sent	total_emails_opened	total_emails_clicked	overall_open_rate	overall_click_rate
60920	20459	9010	33.58	14.79

Insights:

- It can be observed that the one third of the email sent are opened.

Results and Discussion

The project successfully analyzed key operational metrics and provided valuable insights into job review patterns, throughput stability, language distribution, and duplicate entries for job data.

Additionally, user engagement trends, growth, retention, and email engagement were analyzed to understand user behaviour and identify areas for improvement.