Operation Analytics and Investigating Metric Spike

Project Description

The project requires us to analyse the data set given, i.e. job_data, users, events and email_events and use SQL to answer the questions posed by different teams of the company. Also, it is required to give meaningful insights after the analysis.

Approach

First, I will go through the datasets given and note down the important key features of the columns, such as primary keys, unique columns, meaning of columns, datatypes, etc. Then, I will load the datasets in MySQL Workbench and change the datatypes accordingly. For e.g, the date column is stored in text format, so I'll update it to the date format.

Tech-Stack Used

MySQL Workbench and online SQL editor.

Case Study 1: Job Data Analysis

Tasks:

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.

```
SELECT
  ds as dates,
  round((COUNT(job_id)/sum(time_spent))*3600) AS jobs_reviewed
FROM
  job_data
  WHERE ds BETWEEN '2020-11-01' and '2020-11-30'
GROUP BY dates;
```

RESULT:

	dates	jobs_reviewed
1	2020-11-25 00:00:00	80
2	2020-11-26 00:00:00	64
3	2020-11-27 00:00:00	35
4	2020-11-28 00:00:00	218
5	2020-11-29 00:00:00	180
6	2020-11-30 00:00:00	180

On 28th Nov 2020, maximum number of jobs reviewed were : **218**

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.

```
WITH daily_throughput AS (
  SELECT
    ds,
    COUNT(*) AS events,
    COUNT(*) / 86400.0 AS throughput
  FROM
    job_data
  GROUP BY
    ds
),
rolling_average AS (
  SELECT
    ds,
    throughput,
    AVG(throughput) OVER (
      ORDER BY ds
      ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
```

```
    ) AS seven_day_rolling_avg
  FROM
    daily_throughput
)
SELECT
  ds,
  throughput,
  ROUND(seven_day_rolling_avg, 2) AS rolling_avg_rounded
FROM
  rolling_average;
```

RESULT:

	ds	throughput	rolling_avg_rounded
1	2020-11-25 00:00:00	0.00001157	0
2	2020-11-26 00:00:00	0.00001157	0
3	2020-11-27 00:00:00	0.00001157	0
4	2020-11-28 00:00:00	0.00002315	0
5	2020-11-29 00:00:00	0.00001157	0
6	2020-11-30 00:00:00	0.00002315	0

Using a daily metric means measuring and analysing throughput on a day-to-day basis. This approach can provide a more granular view of changes and trends over short time periods. Whereas, a 7-day rolling average smooths out daily fluctuations by calculating the average throughput over a 7-day period, updating the average daily. This approach provides a more stable representation of trends and patterns over time, helping to eliminate noise caused by daily variations. So, it really depends on the context of the project and stakeholders' expectations. Personally, I would choose a 7-day period as it is more stable and beneficial in the longer periods of time.

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.

```
SELECT
 language,
 count(*) AS total_lang,
 round(count(*) * 100 / sum(count(*)) over(), 2) AS percentage
FROM
 job_data
WHERE
 ds >= DATE_SUB(NOW(), INTERVAL 30 DAY)
GROUP BY
 language
ORDER BY
 total_lang DESC;
```

RESULT:

language	total_lang	percentage
Persian	3	37.50
English	1	12.50
Arabic	1	12.50
Hindi	1	12.50
French	1	12.50
Italian	1	12.50

Persian language has the highest share percentage of **37.50**

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.

```
SELECT
   actor_id, COUNT(*) AS duplicates
FROM
   job_data
GROUP BY actor_id
HAVING COUNT(*) > 1;
```

RESULT:

	actor_id	duplicates
1	1003	2

actor_id **1003** has duplicate rows.

Case Study 2: Investigating Metric Spike

Tasks:

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.

```
SELECT
   EXTRACT(WEEK from activated_at) AS week_number,
   EXTRACT(YEAR from activated_at) AS year,
   COUNT(DISTINCT user_id) AS weekly_engaged_users
FROM
   users
GROUP BY
   week_number, year
ORDER BY
   year, week_number;
```

RESULT:

	week_number	year	weekly_engaged_users
1	1	2013	67
2	2	2013	29
3	3	2013	47
4	4	2013	36
5	5	2013	30
6	6	2013	48
7	7	2013	41
8	8	2013	39

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.

```
SELECT
 week_number,
 year_number,
 active_users,
 sum(active_users) over (
  order by year_number,
  week_number ROWS BETWEEN unbounded preceding
  AND current ROW
 ) AS cumulative_sum_growth
FROM
(
  SELECT
   extract(
    week
    FROM
     activated_at
   ) AS week_number,
   extract(
```

```
    year
    FROM
      activated_at
  ) AS year_number,
  count(DISTINCT user_id) AS active_users
FROM
  users
WHERE
  state = 'active'
GROUP BY
  year_number,
  week_number
ORDER BY
  year_number,
  week_number
) time_frame
```

RESULT:

	week_number	year_number	active_users	cumulative_sum_growth
1	1	2013	67	67
2	2	2013	29	96
3	3	2013	47	143
4	4	2013	36	179
5	5	2013	30	209
6	6	2013	48	257
7	7	2013	41	298
8	8	2013	39	337

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.

```
SELECT
  extract(
    week
    FROM
      u.created_at
  ) AS signup_week,
  extract (
    week
    FROM
      a.activated_at
  ) AS retention_week,
  COUNT(DISTINCT u.user_id) AS active_users,
  ROUND(
    COUNT(DISTINCT u.user_id) * 100.0 / FIRST_VALUE(COUNT(DISTINCT u.user_id)) OVER (
      PARTITION BY extract(
        week
        FROM
          u.created_at
      )
    ),
    2
  ) AS retention_rate
FROM
  users u
  LEFT JOIN users a ON u.user_id = a.user_id
  AND a.state = 'active'
  AND a.activated_at >= u.created_at
GROUP BY
  signup_week,
  retention_week
```

```
ORDER BY
 signup_week,
 retention_week;
```

RESULT:

	signup_week	retention_week	active_users	retention_rate
1	1	1	158	100
2	1		131	82.91
3	2	2	151	100
4	2		148	98.01
5	3	3	159	100
6	3		142	89.31
7	4	4	149	100
8	4		177	118.79

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.

```
SELECT
 extract(
  week
  FROM
   occurred_at
 ) AS week,
 extract(
  year
  FROM
   occurred_at
 ) AS year,
 device,
 count(DISTINCT user_id) AS count
```

```
FROM
  events
WHERE
  event_type = 'engagement'
GROUP BY
  week,
  year,
  device
ORDER BY
  week,
  year,
  device;
```

RESULT:

	week	year	device	count
1	18	2014	acer aspire desktop	10
2	18	2014	acer aspire notebook	21
3	18	2014	amazon fire phone	4
4	18	2014	asus chromebook	23
5	18	2014	dell inspiron desktop	21
6	18	2014	dell inspiron notebook	49
7	18	2014	hp pavilion desktop	15
8	18	2014	htc one	16

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.

```
SELECT
  action,
  count(action) AS action_count
FROM
  emails
```

```
GROUP BY
  action;
```

RESULT:

	action	action_count
1	email_open	20459
2	email_clickthrough	9010
3	sent_weekly_digest	57267
4	sent_reengagement_em...	3653

```
SELECT
  100.0 * sum(
    CASE
      WHEN email_category = 'email_open' THEN 1
      ELSE 0
    END
  ) / sum(
    CASE
      WHEN email_category = 'email_sent' THEN 1
      ELSE 0
    END
  ) AS open_rate,
  100.0 * sum(
    CASE
      WHEN email_category = 'email_click' THEN 1
      ELSE 0
    END
  ) / sum(
    CASE
      WHEN email_category = 'email_sent' THEN 1
      ELSE 0
```

```
    END
  ) AS click_rate
FROM
  (
    SELECT
      *,
      CASE
        WHEN ACTION IN ('sent_weekly_digest', 'sent_reengagement_email') THEN 'email_sent'
        WHEN ACTION IN ('email_open') THEN 'email_open'
        WHEN ACTION IN ('email_clickthrough') THEN 'email_click'
      END AS email_category
    FROM
      emails
  ) mail_ case
```

RESULT:

	open_rate	click_rate
1	33.5834	14.7899

Result

This project helped me understand the importance of operational metrics analysis that various companies use to analyse the KPIs. Also, this project challenged my Advanced SQL skills and made me think beyond the horizon. A lot of trials and errors and understanding the datasets, the project helped me grow in terms of SQL skills.

I learnt new things such as loading data in MySQL workbench and came across errors while doing so. Changed the data type of date, which was stored as text, to date type and continued with the process.

Conclusion

Operational analytics is crucial for modern businesses aiming to stay competitive and agile in today's data-driven landscape. By leveraging data and insights, organizations can enhance their operational efficiency, customer satisfaction, and strategic decision-making processes.