Operational Analytics and Investigating Metric Spike

Project Description

Operational Analytics is about analyzing business operations end-to-end. In this project, the focus was to explore daily activities, understand key metrics, and identify performance issues or unusual spikes. The project was divided into two case studies. As a Lead Data Analyst, the goal was to perform SQL-based analysis to help departments like operations, support, and marketing gain useful insights from data.

Approach

Case Study 1: Job Data Analysis

- The dataset was small and manageable, so I uploaded it manually into MySQL Workbench.
- I wrote SQL queries to answer the questions directly and added explanations below each one to describe the logic clearly.

Case Study 2: Investigating Metric Spike

- The dataset was much larger.
- I first explored it in Excel to check for missing values.
- After that, I created the required tables and imported the data into MySQL.
- Then, I wrote the queries and included detailed explanations for each.

Tech Stack Used

- **MySQL Workbench 8.0**: Used to create databases, write and run SQL queries, and analyze the data.
- **Microsoft Excel**: Used briefly for preliminary data exploration in Case Study 2.

Insights and Analysis

Case Study 1: Job Data Analysis

1. Jobs Reviewed Over Time

```
SELECT ds,
       COUNT(job_id) AS no_of_jobs,
       SUM(time_spent) AS total_seconds,
       SUM(time_spent) / 3600 AS total_hours,
       ROUND(COUNT(job_id) / NULLIF(GREATEST(SUM(time_spent) / 3600, 1),
0),2) AS jobs_per_hour
FROM job_data
WHERE ds BETWEEN "2020-11-01" AND "2020-11-30"
GROUP BY ds
ORDER BY ds;
```

Explanation:
Counted total jobs and total time spent per day. Converted time to hours, then divided jobs by hours to get jobs/hour. Used GREATEST(..., 1) to avoid inflated results from very low times and NULLIF(..., 0) to avoid division by zero.

ds	no_of_jobs	total_seconds	total_hours	jobs_per_hour
2020-11-25	1	45	0.0125	1.00
2020-11-26	1	56	0.0156	1.00
2020-11-27	1	104	0.0289	1.00
2020-11-28	2	33	0.0092	2.00
2020-11-29	1	20	0.0056	1.00
2020-11-30	2	40	0.0111	2.00

Insight:
Some days had exceptionally high job counts per hour, mainly due to very small time records that made the calculations look inflated. Using functions like GREATEST and NULLIF helped ensure the results were more accurate and not skewed by low values.

2. 7-Day Rolling Average of Throughput

```
SELECT
    ds,
    COUNT(job_id) AS total_events,
    SUM(time_spent) AS total_seconds,
    ROUND(COUNT(job_id) / NULLIF(SUM(time_spent), 0), 6) AS
throughput_per_second,
    ROUND(
        AVG(COUNT(job_id)) OVER (
            ORDER BY ds
            ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) /
        NULLIF(AVG(SUM(time_spent)) OVER (
            ORDER BY ds
            ROWS BETWEEN 6 PRECEDING AND CURRENT ROW), 0), 6
    ) AS rolling_avg_throughput_per_second
FROM job_data
WHERE ds BETWEEN '2020-11-01' AND '2020-11-30'
GROUP BY ds
ORDER BY ds;
```

Explanation:
Measured throughput (jobs per second) for each day and used window functions to get 7-day rolling sums for jobs and time, then divided them for rolling throughput. Rounded to 6 decimals.

ds	total_events	total_seconds	throughput_per_second	rolling_avg_throughput_per_second
2020-11-25	1	45	0.0222	0.022222
2020-11-26	1	56	0.0179	0.019802
2020-11-27	1	104	0.0096	0.014634
2020-11-28	2	33	0.0606	0.021008
2020-11-29	1	20	0.0500	0.023256
2020-11-30	2	40	0.0500	0.026845

Insight:
The rolling average gave us a clearer picture of throughput trends over time, helping to smooth out daily fluctuations. This approach is much more reliable when analyzing performance over a longer period.

3. Language Share Analysis

```
SELECT
    language,
    COUNT(*) AS total_jobs,
    ROUND((COUNT(*) * 100.0) / SUM(COUNT(*)) OVER (), 2) AS
percentage_share
FROM job_data
WHERE ds >= DATE_SUB((SELECT MAX(ds) FROM job_data), INTERVAL 30 DAY)
GROUP BY language
ORDER BY percentage_share DESC;
```

Explanation:
Filtered for last 30 days using ds >= DATE_SUB(...). Counted jobs per language, divided by total jobs using SUM(...) OVER (), multiplied by 100, and rounded to 2 decimals for percent share. Sorted in descending order.

language	total_jobs	percentage_share
Persian	3	37.50
English	1	12.50
Arabic	1	12.50
Hindi	1	12.50
French	1	12.50
Italian	1	12.50

Insight:
Persian came out on top, representing 37.5% of the data, while other languages had a more even distribution. This insight helps us understand language preferences and can guide decisions in language-specific projects.

4. Duplicate Rows Detection

```
SELECT ds, job_id, actor_id, event, language, time_spent, org, COUNT(*)
FROM job_data
GROUP BY ds, job_id, actor_id, event, language, time_spent, org
HAVING COUNT(*) > 1;
```

Explanation:
Checked for full-row duplicates in job_data by grouping on all columns. Used COUNT(*) to count occurrences of each row, and filtered with HAVING COUNT(*) > 1 to keep only duplicates. Ordered results by frequency of duplication.

Insight:
No duplicate rows were found, which means our data is clean and ready for deeper analysis.Duplicate rows can lead to incorrect metrics. It's essential to clean data before analysis to maintain accuracy.

Case Study 2: Investigating Metric Spike

1. Weekly User Engagement:

```
SELECT
    DATE_FORMAT(occurred_at - INTERVAL WEEKDAY(occurred_at) DAY, '%Y-%m-
%d') AS week_start,
    WEEK(occurred_at, 1) AS week_number,
    COUNT(DISTINCT user_id) AS active_users
FROM
    events
GROUP BY
    week_start, week_number
ORDER BY
    week_start;
```

Explanation:
We measured weekly engagement by counting distinct active users from the `events` table. Using `WEEK()` and `YEAR()`, events were grouped by week to track user activity trends over time.

week_start	week_number	new_users
2012-12-31	1	26
2013-01-07	2	29
2013-01-14	3	47
2013-01-21	4	36
2013-01-28	5	30
2013-02-04	6	48
2013-02-11	7	41
2013-02-18	8	39
2013-02-25	9	33
2013-03-04	10	43
2013-03-11	11	33
2013-03-18	12	32
2013-03-25	13	33
2013-04-01	14	40
2013-04-08	15	35
2013-04-15	16	42
2013-04-22	17	48
2013-04-29	18	48

Insight:
Weekly active users steadily increased from Week 18 to Week 31, peaking at Week 31 with 1,443 users. This upward trend suggests successful engagement or marketing efforts during this period before a slight drop in subsequent weeks.

2. User Growth Analysis:

```
SELECT
    DATE_FORMAT(created_at - INTERVAL WEEKDAY(created_at) DAY, '%Y-%m-%d')
AS week_start,
    WEEK(created_at, 1) AS week_number,
    COUNT(user_id) AS new_users
FROM
    users
GROUP BY
    week_start, week_number
ORDER BY
    week_start;
```

Explanation:
We analyzed user growth by counting how many users signed up each week. By grouping the `created_at` data using `WEEK()` and `YEAR()`, we observed weekly signup patterns and growth momentum.

week_start	week_number	new_users
2012-12-31	1	26
2013-01-07	2	29
2013-01-14	3	47
2013-01-21	4	36
2013-01-28	5	30
2013-02-04	6	48
2013-02-11	7	41
2013-02-18	8	39
2013-02-25	9	33
2013-03-04	10	43
2013-03-11	11	33
2013-03-18	12	32
2013-03-25	13	33
2013-04-01	14	40
2013-04-08	15	35
2013-04-15	16	42
2013-04-22	17	48
2013-04-29	18	48

Insight:
New user signups followed a steady weekly trend with occasional surges, reflecting typical growth patterns influenced by external or seasonal factors.

3. Weekly Retention Analysis:

```
SELECT
   signup_week AS 'Signup Week',
   SUM(CASE WHEN retention_week = 0 THEN 1 ELSE 0 END) AS 'Week 0',
   SUM(CASE WHEN retention_week = 1 THEN 1 ELSE 0 END) AS 'Week 1',
   SUM(CASE WHEN retention_week = 2 THEN 1 ELSE 0 END) AS 'Week 2',
```

```
    SUM(CASE WHEN retention_week = 3 THEN 1 ELSE 0 END) AS 'Week 3',
    SUM(CASE WHEN retention_week = 4 THEN 1 ELSE 0 END) AS 'Week 4',
    SUM(CASE WHEN retention_week = 5 THEN 1 ELSE 0 END) AS 'Week 5',
    SUM(CASE WHEN retention_week = 6 THEN 1 ELSE 0 END) AS 'Week 6',
    SUM(CASE WHEN retention_week = 7 THEN 1 ELSE 0 END) AS 'Week 7',
    SUM(CASE WHEN retention_week = 8 THEN 1 ELSE 0 END) AS 'Week 8',
    SUM(CASE WHEN retention_week = 9 THEN 1 ELSE 0 END) AS 'Week 9',
    SUM(CASE WHEN retention_week = 10 THEN 1 ELSE 0 END) AS 'Week 10',
    SUM(CASE WHEN retention_week = 11 THEN 1 ELSE 0 END) AS 'Week 11',
    SUM(CASE WHEN retention_week = 12 THEN 1 ELSE 0 END) AS 'Week 12',
    SUM(CASE WHEN retention_week = 13 THEN 1 ELSE 0 END) AS 'Week 13',
    SUM(CASE WHEN retention_week = 14 THEN 1 ELSE 0 END) AS 'Week 14',
    SUM(CASE WHEN retention_week = 15 THEN 1 ELSE 0 END) AS 'Week 15',
    SUM(CASE WHEN retention_week = 16 THEN 1 ELSE 0 END) AS 'Week 16',
    SUM(CASE WHEN retention_week = 17 THEN 1 ELSE 0 END) AS 'Week 17',
    SUM(CASE WHEN retention_week = 18 THEN 1 ELSE 0 END) AS 'Week 18'

FROM (
    SELECT
      e1.user_id,
      e1.activity_week,
      cohort.cohort_week AS signup_week,
      e1.activity_week - cohort.cohort_week AS retention_week
    FROM (
      SELECT
        user_id,
        MIN(EXTRACT(WEEK FROM occurred_at)) AS cohort_week
      FROM events
      GROUP BY user_id
    ) AS cohort
    JOIN (
      SELECT
        user_id,
        EXTRACT(WEEK FROM occurred_at) AS activity_week
      FROM events
      GROUP BY user_id, EXTRACT(WEEK FROM occurred_at)
    ) AS e1
    ON cohort.user_id = e1.user_id
) AS weekly_retention
GROUP BY signup_week
ORDER BY signup_week;
```

Explanation:

Identified each user's signup week, then calculated how many weeks later they returned. Used `CASE WHEN` inside `SUM` to count users by retention week, grouped by signup week.

Signup Week	Week 0	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15	Week 16	Week 17	Week 18
17	663	472	324	251	205	187	167	146	145	145	136	131	132	143	116	91	82	77	5
18	596	362	261	203	168	147	144	127	113	122	106	118	127	110	97	85	67	4	0
19	427	284	173	153	114	95	91	81	95	82	68	65	63	42	51	49	2	0	0
20	358	223	165	121	91	72	63	67	63	65	67	41	40	33	40	0	0	0	0
21	317	187	131	91	74	63	75	72	58	48	45	39	35	28	2	0	0	0	0
22	326	224	150	107	87	73	63	60	55	48	41	39	31	1	0	0	0	0	0
23	328	219	138	101	90	79	69	61	54	47	35	30	0	0	0	0	0	0	0
24	339	205	143	102	81	63	65	61	38	39	29	0	0	0	0	0	0	0	0
25	305	218	139	101	75	63	50	46	38	35	2	0	0	0	0	0	0	0	0
26	288	181	114	83	73	55	47	43	29	0	0	0	0	0	0	0	0	0	0
27	292	199	121	106	68	53	40	36	1	0	0	0	0	0	0	0	0	0	0
28	274	194	114	69	46	30	28	3	0	0	0	0	0	0	0	0	0	0	0
29	270	186	102	65	47	40	1	0	0	0	0	0	0	0	0	0	0	0	0
30	294	202	121	78	53	3	0	0	0	0	0	0	0	0	0	0	0	0	0
31	215	145	76	57	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	267	188	94	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	286	202	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	279	44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
35	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Insight:
Retention was strongest in Week 17, with users showing sustained activity up to 18 weeks later. This indicates a highly engaged cohort compared to later signup weeks (like Week 30–35) where retention drops off sharply after just a few weeks.

4. Weekly Engagement Per Device:

```
SELECT
    WEEK(occurred_at) AS week,
    device,
    COUNT(DISTINCT user_id) AS user_engagement
FROM
    events
GROUP BY
    device,
    WEEK(occurred_at)
ORDER BY
    WEEK(occurred_at);
```

Explanation:
Grouped events by week and device, then counted distinct users per combination using `COUNT(DISTINCT user_id)` to analyze engagement by device over time.

week	device	user_engagement
17	acer aspire desktop	9
17	acer aspire notebook	20
17	amazon fire phone	4
17	asus chromebook	21
17	dell inspiron desktop	18
17	dell inspiron notebook	46
17	hp pavilion desktop	14
17	htc one	16
17	ipad air	27
17	ipad mini	19
17	iphone 4s	21
17	iphone 5	65
17	iphone 5s	42
17	kindle fire	6
17	lenovo thinkpad	86

Insight:
Device engagement was spread across both mobile and desktop platforms. A few device types stood out with higher user activity, indicating that device type could influence how frequently users interact with the product.

5. Email Engagement Analysis:

```
SELECT
    action,
    COUNT(DISTINCT user_id) AS unique_users_count,
    COUNT(*) AS total_actions_count
```

```
FROM
    email_events
GROUP BY
    action
ORDER BY
    action;
```

Explanation:
Grouped by action to count both total actions (`COUNT(*)`) and unique users (`COUNT(DISTINCT user_id)`). Ordered results by action for easy review.

action	unique_users_count	total_actions_count
email_clickthrough	5277	9010
email_open	5927	20459
sent_reengagement_email	3653	3653
sent_weekly_digest	4111	57267

Insight:
"Email open" and "sent_weekly_digest" had the highest total actions, indicating users are most likely to engage with routine digest content. The high number of unique users also confirms that email remains a strong channel for reaching the user base.

Result

Through this project, I successfully analyzed two different types of datasets: a small operational dataset and a larger user activity dataset. I wrote optimized SQL queries to:

- Investigate review patterns and language share across operations.
- Track user activity, growth, retention, and email engagement.
- Apply advanced SQL concepts like window functions, date handling, grouping, and data cleaning.

This project helped me understand how to explore trends, handle metric spikes, and draw actionable insights using SQL — skills that are directly applicable in real-world data analyst roles.
