Operation & Metric Analytics

- **Project Description:** This project aims to analyze user activity, engagement metrics, and operational data to extract valuable insights for performance optimization and strategic decision-making. Key tasks include calculating throughput, job review trends, language shares analysis, investigating metric spikes, and analyzing user growth, retention, and email engagement. The tasks were executed systematically using SQL queries to ensure data accuracy and generate actionable insights.

A) Job Data Analysis:

1. **Jobs Reviewed Over Time:** Write an SQL query to calculate the number of jobs reviewed per hour for each day in November 2020.

- **Approach:** To analyze the number of jobs reviewed per hour for each day in November 2020, the dataset is filtered so that only records within the specified date range (2020-11-01 to 2020-11-30) are included. By using SQL date and time are extracted from the ds column and then grouped by these time intervals. The query calculates the total jobs reviewed for each combination of date and hour, ensuring a chronological order for easy trend analysis.

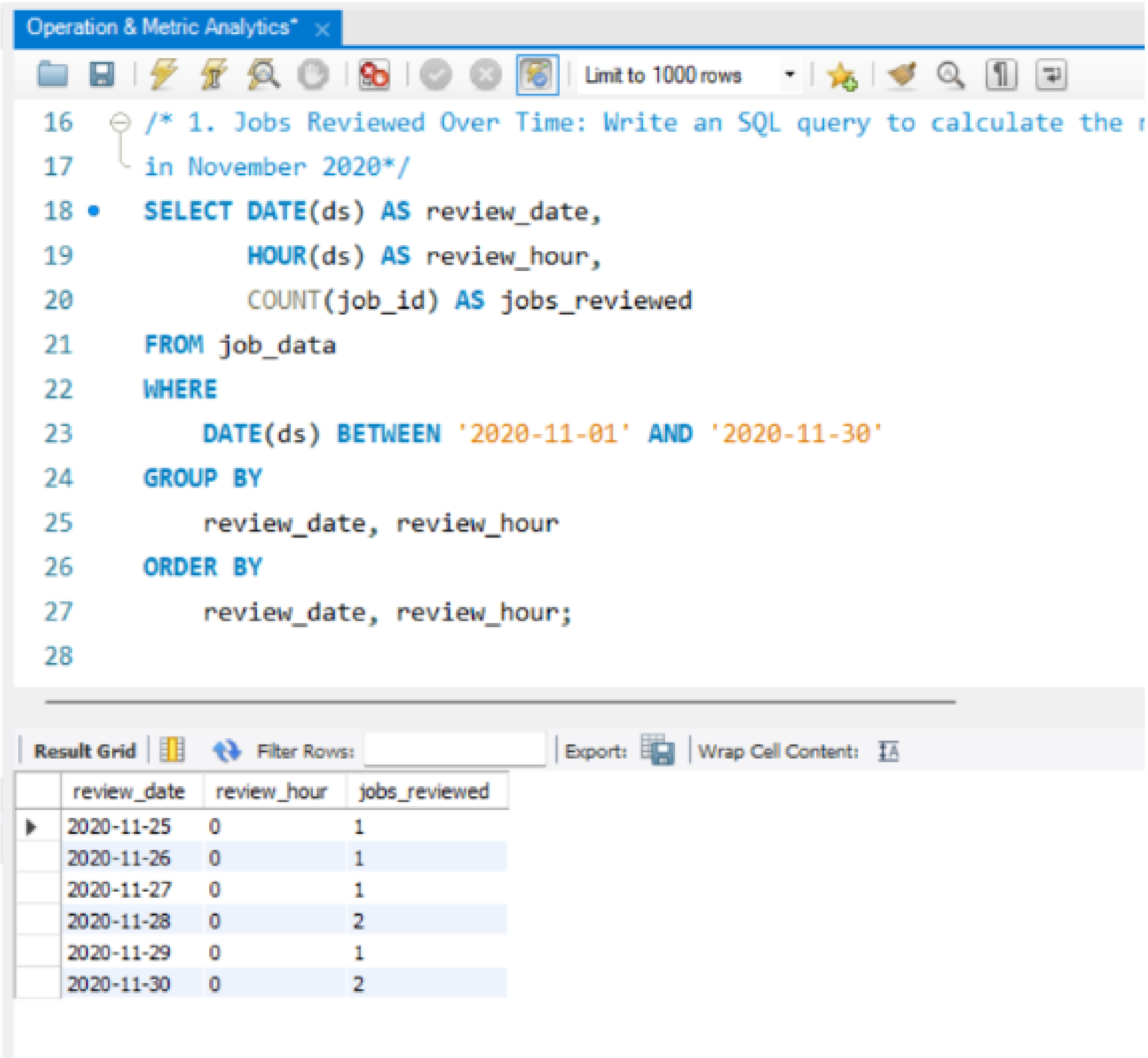
- **Insights:**

- **Daily Trends:** This query shows job review activity per hour, helping to identify peak working hours and low activity periods.

- **Operational Patterns:** Insights into hourly workloads can assist in resource allocation and process optimization.

- **Spike Detection:** It is possible to locate specific timeframes with unusual spikes in job reviews using hourly data, which can be further investigated.

- **Result:**



Operation & Metric Analytics*

Limit to 1000 rows

```sql
/* 1. Jobs Reviewed Over Time: Write an SQL query to calculate the r
in November 2020*/
SELECT DATE(ds) AS review_date,
       HOUR(ds) AS review_hour,
       COUNT(job_id) AS jobs_reviewed
FROM job_data
WHERE
    DATE(ds) BETWEEN '2020-11-01' AND '2020-11-30'
GROUP BY
    review_date, review_hour
ORDER BY
    review_date, review_hour;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

review_date	review_hour	jobs_reviewed
2020-11-25	0	1
2020-11-26	0	1
2020-11-27	0	1
2020-11-28	0	2
2020-11-29	0	1
2020-11-30	0	2

2. **Throughput Analysis:** 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.

- **Approach:**

  I used CTE to calculate the 7-day rolling average of throughput:

  i. The **daily_throughput** CTE computes the number of events per second for each day by dividing the total daily events by the total seconds in a day (24 * 60 * 60).

  ii. The **rolling_avg** CTE calculates the rolling average of events per second over a 7-day window using the AVG() window function.

- **Preference:**

I prefer the 7-day Rolling Average for throughput analysis as it provides a balanced view by smoothing short-term variations, making it easier to identify overall trends without being overly affected by daily noise. On the other hand, the daily metric might be more appropriate if decision-making depends on immediate daily variations.

- **Insights:**

- **7-Day Rolling Average:** Offers a smoother trend by mitigating the effect of daily fluctuations and outliers, making it better for long-term monitoring.

- **Daily Metric:** Useful for real-time monitoring and detecting abrupt spikes but can be noisy.

- **Conclusion:** Prefer 7-day rolling averages for stable insights and trend analysis.

- **Result:**

```sql
/* 2. Throughput Analysis: Write an SQL query to calculate the 7-day rolling average of
WITH daily_throughput AS (
    SELECT DATE(ds) AS event_date,
        COUNT(*) AS total_events,
        COUNT(*) * 1.0 / (24 * 60 * 60) AS events_per_second
    FROM job_data
    GROUP BY DATE(ds)),
rolling_avg AS (
    SELECT event_date,
        events_per_second,
        AVG(events_per_second)
        OVER (ORDER BY event_date
            ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS rolling_avg_throughput
    FROM daily_throughput)
SELECT
    event_date, events_per_second, rolling_avg_throughput
FROM rolling_avg
ORDER BY event_date;
```

event_date	events_per_second	rolling_avg_throughput
2020-11-25	0.00001	0.000010000
2020-11-26	0.00001	0.000010000
2020-11-27	0.00001	0.000010000
2020-11-28	0.00002	0.000012500
2020-11-29	0.00001	0.000012000
2020-11-30	0.00002	0.000013333

3. **Language Share Analysis:** Write an SQL query to calculate the percentage share of each language over the last 30 days.

- **Approach:**

I used CTE, to calculate the percentage share of each language in the last 30 days:

i. Calculate the number of job days for each language in the last 30 using the recent_jobs CTE.

ii. Calculate the overall job count using the total_jobs CTE.

iii. Calculate each language's percentage share by dividing its job count by the total count.

- **Insights:**

- The dominant languages have been identified in the past 30 days.

- **Persian** is the dominant language with a **37.50%** language share.

- Identifies trends in language preferences or helps allocate resources.

- **Result:**

```sql
/* 3. Language Share Analysis: Write an SQL query to calculate the percentage share of eac
WITH recent_jobs AS (
    SELECT language,
        COUNT(*) AS job_count
    FROM job_data
    WHERE DATE(ds) >= DATE_SUB('2020-12-01', INTERVAL 30 DAY)
    GROUP BY language
),
total_jobs AS (
    SELECT SUM(job_count) AS total_job_count
    FROM recent_jobs
)
SELECT
    language, job_count,
    ROUND((job_count * 100.0) / total_job_count, 2) AS language_share_percentage
FROM recent_jobs
CROSS JOIN total_jobs
ORDER BY language_share_percentage DESC;
```

language	job_count	language_share_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

4. **Duplicate Rows Detection:** Write an SQL query to display duplicate rows from the job_data table.

- **Approach:**

- To display duplicate rows from the job_data table, I used queries to identify exact duplicates across all columns and partial duplicates.

- For exact duplicates, I used the GROUP BY clause for all columns and filtered using HAVING COUNT(*) > 1.

- For partial duplicates based on certain columns (e.g. job_id and actor_id), I used the GROUP BY clause for specific columns and filtered similarly.

- **Insights:**

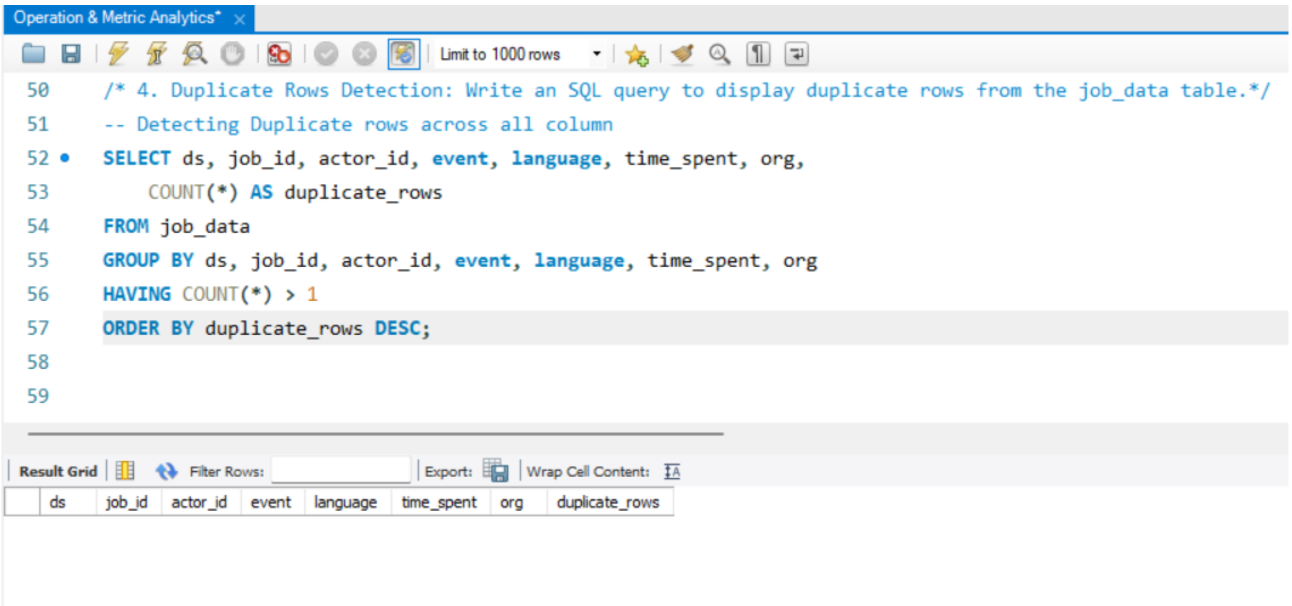
- There are no rows where all column values are identical, which means the dataset not contain any full duplicates.

- No duplicates rows are returned, the dataset does not have exact duplicates.

- **job_id 23** appears multiple times on different **dates** but with different **actor_id** or **event.**

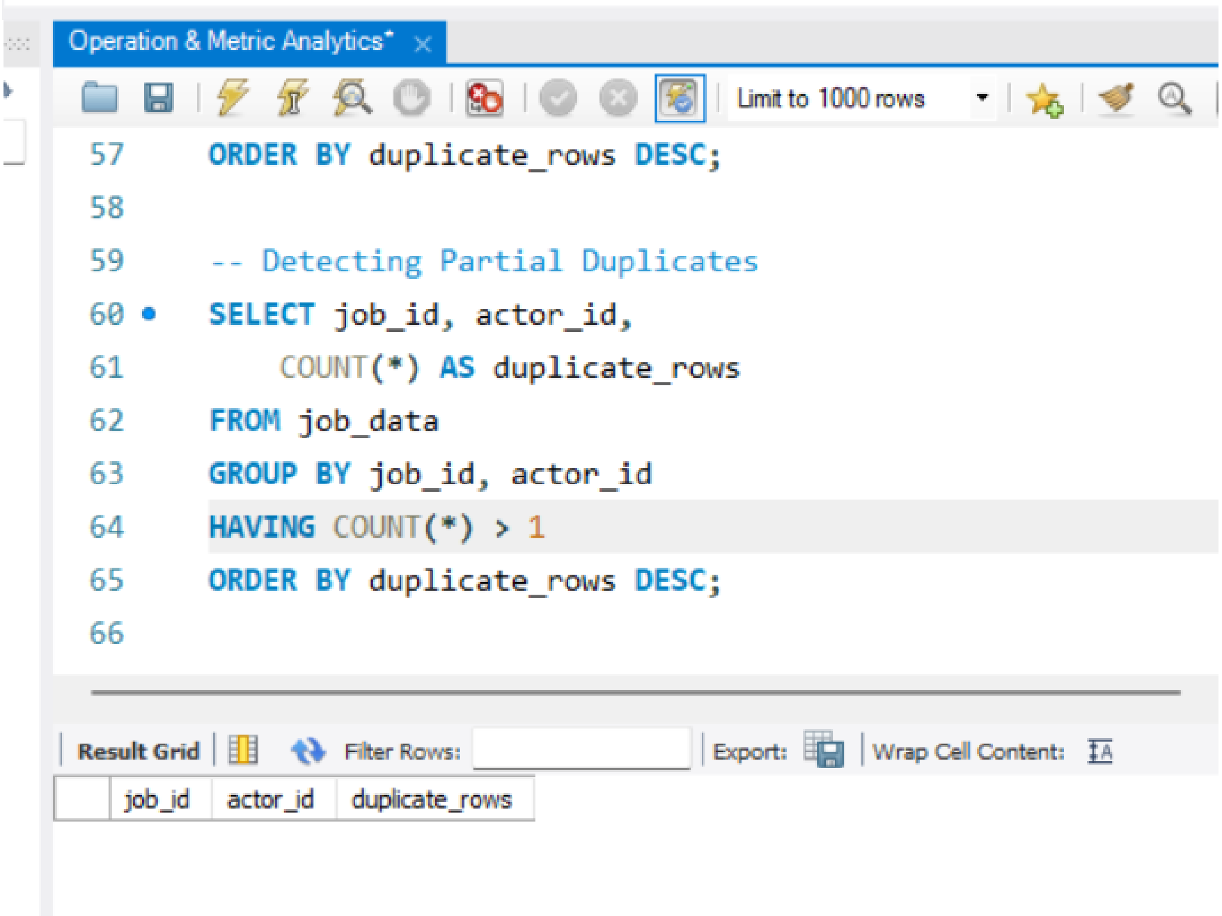
- **actor_id 1003** is also repeated on different **dates** with different **job_id** or **language.**

- **Result:**




```sql
/* 4. Duplicate Rows Detection: Write an SQL query to display duplicate rows from the job_data table.*/
-- Detecting Duplicate rows across all column
SELECT ds, job_id, actor_id, event, language, time_spent, org,
    COUNT(*) AS duplicate_rows
FROM job_data
GROUP BY ds, job_id, actor_id, event, language, time_spent, org
HAVING COUNT(*) > 1
ORDER BY duplicate_rows DESC;
```


	ds	job_id	actor_id	event	language	time_spent	org	duplicate_rows




```sql
ORDER BY duplicate_rows DESC;

-- Detecting Partial Duplicates
SELECT job_id, actor_id,
    COUNT(*) AS duplicate_rows
FROM job_data
GROUP BY job_id, actor_id
HAVING COUNT(*) > 1
ORDER BY duplicate_rows DESC;
```


	job_id	actor_id	duplicate_rows

B) Investigating Metric Spike:

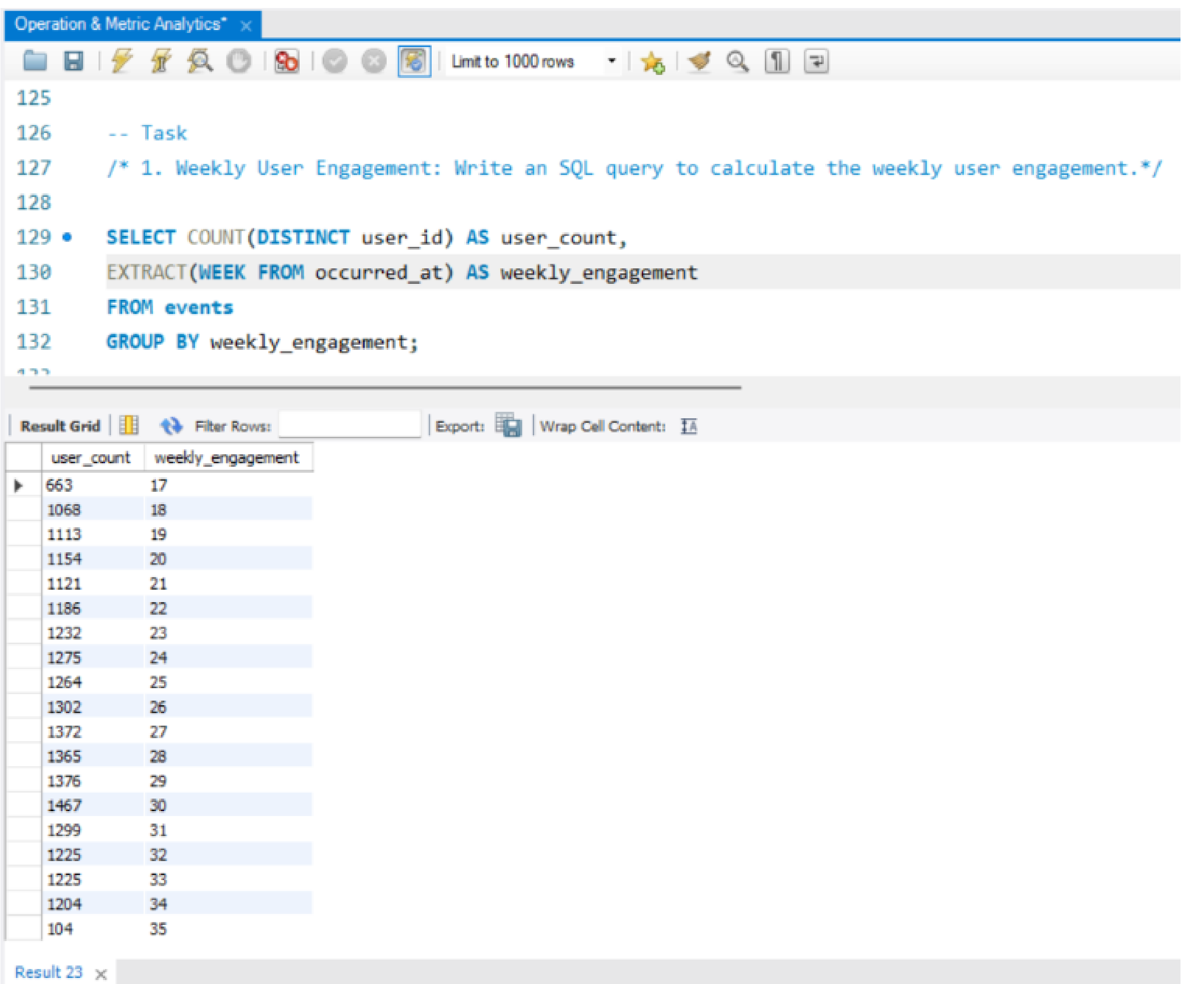
1. **Weekly User Engagement:** Write an SQL query to calculate the weekly user engagement.

- **Approach:**

- Calculate the number of unique users (COUNT(DISTINCT user_id)) for each week (EXTRACT(WEEK FROM occurred_at)).
- Group results by week to track weekly user engagement.

- **Insights:**
  Helps to monitor the number of active users each week and identify trends or anomalies.

- **Result:**




```sql
-- Task
/* 1. Weekly User Engagement: Write an SQL query to calculate the weekly user engagement.*/

SELECT COUNT(DISTINCT user_id) AS user_count,
EXTRACT(WEEK FROM occurred_at) AS weekly_engagement
FROM events
GROUP BY weekly_engagement;
```

user_count	weekly_engagement
663	17
1068	18
1113	19
1154	20
1121	21
1186	22
1232	23
1275	24
1264	25
1302	26
1372	27
1365	28
1376	29
1467	30
1299	31
1225	32
1225	33
1204	34
104	35

2. **User Growth Analysis:** Write an SQL query to calculate the user growth for the product.

- **Approach:**

- Compute the weekly number of active users by counting distinct user IDs (COUNT(DISTINCT user_id)) grouped by week and year of activation.

- Use a window function (SUM() with OVER) to calculate the cumulative sum of active users to track overall growth.

- **Insights:**
- Tracks user acquisition over time and reveals growth patterns.
- Helps assess the success of marketing or onboarding efforts.

- **Result:**

Operation & Metric Analytics*

Limit to 1000 rows

```sql

/* 2. User Growth Analysis:  Write an SQL query to calculate the user growth for the product.*/

SELECT year_num, weekly_engagement, active_user, SUM(active_user)
OVER( ORDER BY year_num, weekly_engagement ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cumulative_sum
FROM
(SELECT COUNT(DISTINCT user_id) AS active_user,
EXTRACT(WEEK FROM activated_at) AS weekly_engagement,
EXTRACT(YEAR FROM activated_at) AS year_num
FROM users
WHERE activated_at IS NOT NULL
GROUP BY weekly_engagement, year_num) AS user_growth;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

year_num	weekly_engagement	active_user	cumulative_sum
2013	0	23	23
2013	1	30	53
2013	2	48	101
2013	3	36	137
2013	4	30	167
2013	5	48	215
2013	6	38	253
2013	7	42	295
2013	8	34	329
2013	9	43	372
2013	10	32	404
2013	11	31	435

Result 16

3. **Weekly Retention Analysis**: Write an SQL query to calculate the weekly retention of users based on their sign-up cohort.

- ## Approach:

- Identify a cohort of users based on their signup week (cohort_year and cohort_week).

- Count the number of retained users (COUNT(DISTINCT wa.user_id)) for subsequent weeks by combining this cohort data with user activity.

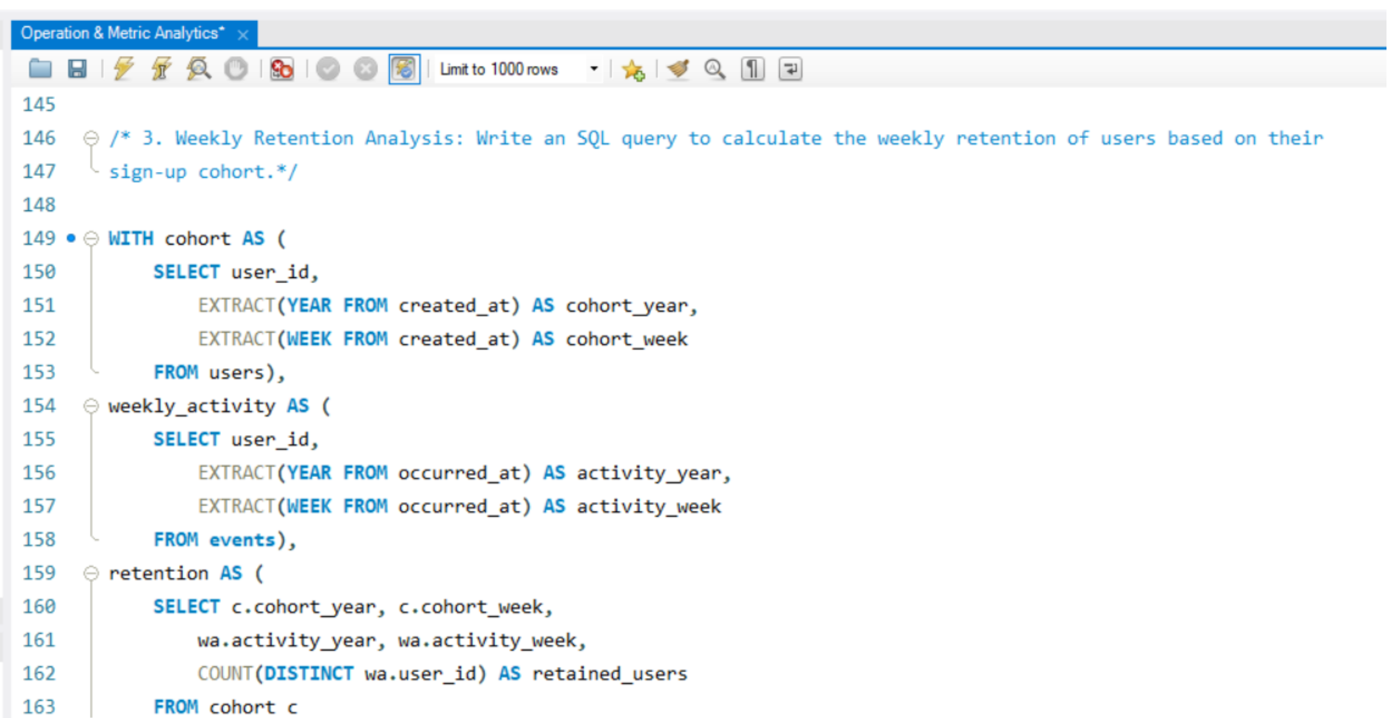
- Calculate the week number difference between signup and activity weeks.

- ## Insights:

- Analyzes retention trends by week, highlighting how well users are retained over time.

- This is useful for identifying issues with product engagement after signing up.

- ## Result:




```sql

/* 3. Weekly Retention Analysis: Write an SQL query to calculate the weekly retention of users based on their
sign-up cohort.*/

WITH cohort AS (
    SELECT user_id,
        EXTRACT(YEAR FROM created_at) AS cohort_year,
        EXTRACT(WEEK FROM created_at) AS cohort_week
    FROM users),
weekly_activity AS (
    SELECT user_id,
        EXTRACT(YEAR FROM occurred_at) AS activity_year,
        EXTRACT(WEEK FROM occurred_at) AS activity_week
    FROM events),
retention AS (
    SELECT c.cohort_year, c.cohort_week,
        wa.activity_year, wa.activity_week,
        COUNT(DISTINCT wa.user_id) AS retained_users
    FROM cohort c
```

Operation & Metric Analytics*

Limit to 1000 rows

```sql
    FROM cohort c
    JOIN weekly_activity wa
    ON c.user_id = wa.user_id
    GROUP BY c.cohort_year, c.cohort_week, wa.activity_year, wa.activity_week),
weekly_retention AS (
    SELECT
        cohort_year,cohort_week,
        activity_year,activity_week,
        retained_users,
        activity_week - cohort_week AS week_number
    FROM retention
    WHERE
        activity_year = cohort_year
        AND activity_week >= cohort_week)
SELECT
    cohort_year, cohort_week,
    week_number, retained_users
FROM weekly_retention
ORDER BY cohort_year, cohort_week, week_number;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Co

cohort_year	cohort_week	week_number	retained_users
2014	0	17	3
2014	0	18	8
2014	0	19	12
2014	0	20	9
2014	0	21	10
2014	0	22	10
2014	0	23	9
2014	0	24	13
2014	0	25	8
2014	0	26	10
2014	0	27	8
2014	0	28	7
2014	0	29	6
2014	0	30	7
2014	0	31	6
2014	0	32	5
2014	0	33	4
2014	0	34	3
2014	1	16	4
2014	1	17	10
2014	1	18	12
2014	1	19	12
2014	1	20	16
2014	1	21	8
2014	1	22	11

Result 32

4. **Weekly Engagement Per Device:** Write an SQL query to calculate the weekly engagement per device.

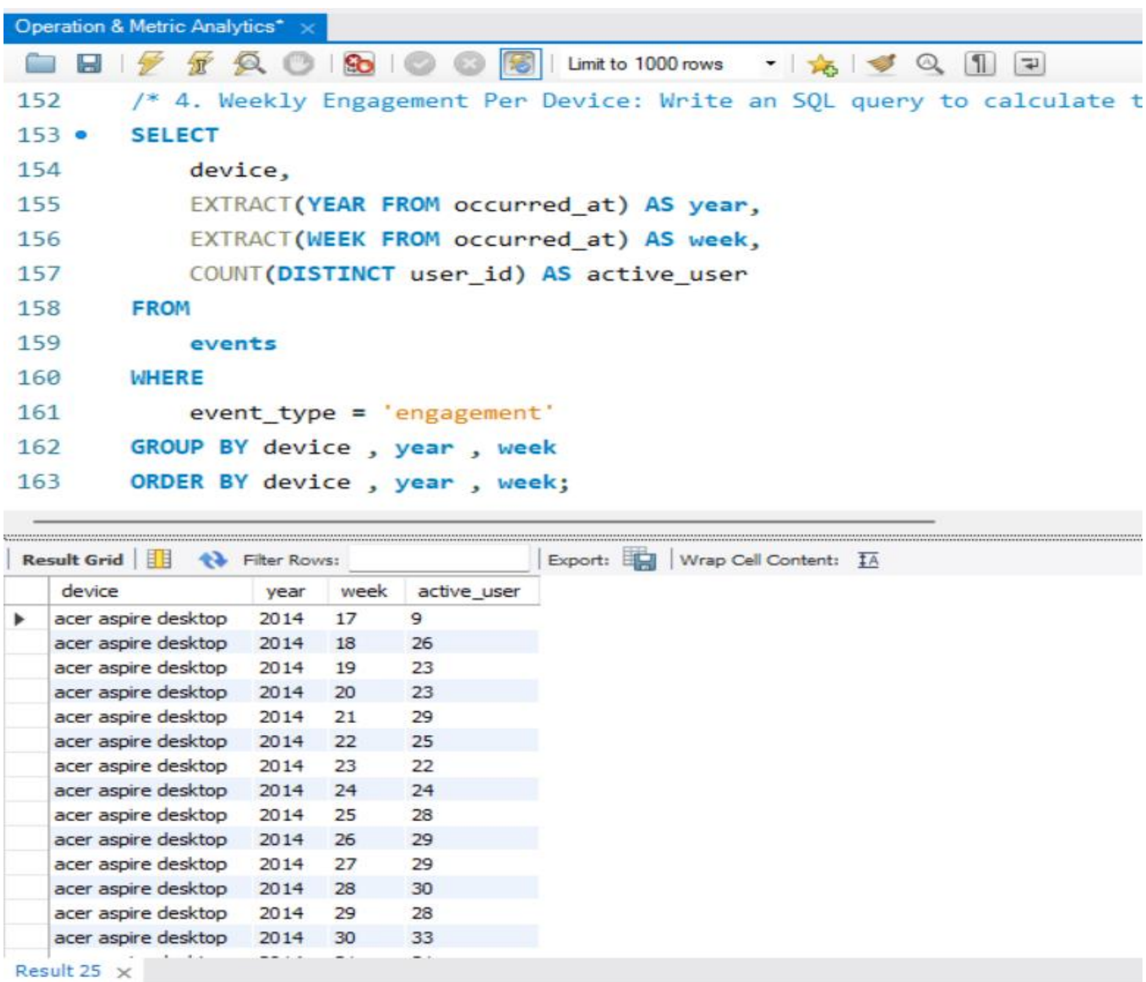
- **Approach:**

- Grouping user activity by device, year, and week to count the number of distinct active users (COUNT(DISTINCT user_id)).
- Filtered data where the event type is engagement.

- **Insights:**

- Identifies device-specific engagement trends.
- Helps optimize for devices with higher engagement rates.

- **Result:**




```sql
/* 4. Weekly Engagement Per Device: Write an SQL query to calculate t
SELECT
    device,
    EXTRACT(YEAR FROM occurred_at) AS year,
    EXTRACT(WEEK FROM occurred_at) AS week,
    COUNT(DISTINCT user_id) AS active_user
FROM
    events
WHERE
    event_type = 'engagement'
GROUP BY device , year , week
ORDER BY device , year , week;
```

device	year	week	active_user
acer aspire desktop	2014	17	9
acer aspire desktop	2014	18	26
acer aspire desktop	2014	19	23
acer aspire desktop	2014	20	23
acer aspire desktop	2014	21	29
acer aspire desktop	2014	22	25
acer aspire desktop	2014	23	22
acer aspire desktop	2014	24	24
acer aspire desktop	2014	25	28
acer aspire desktop	2014	26	29
acer aspire desktop	2014	27	29
acer aspire desktop	2014	28	30
acer aspire desktop	2014	29	28
acer aspire desktop	2014	30	33

5. **Email Engagement Analysis**: Write an SQL query to calculate the email engagement metrics.

- **Approach:**

Email engagement metrics are data points that provide information about how users interact with your emails such as Click-through rate (CTR) and Open rate. For calculating email engagement metrics.

1) Email Events Count:

a. Count the total number of events grouped by action.

2) Engagement Rates:

a. Classify events into categories (**email_sent, email_open, email_click**) using a **CASE statement.**

b. Calculate open and click rates as percentages based on the number of emails sent.

- **Insights:**

- Tracks email effectiveness through open and click rates.
- Helps refine email campaigns by identifying which emails perform well.

- **Result:**

Operation & Metric Analytics*

Limit to 1000 rows

```sql
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 email_events
) AS email_metrics;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

open_rate	click_rate
33.58339	14.78989

- **Tech-Stack Used**

  MySQL Workbench (Version 8.0): Used for writing and executing SQL queries due to its user-friendly interface and ability to visualize results.

- **Drive Link**

  SQL File:
  https://drive.google.com/file/d/1K0oGjcYRFgXAhWkBL_dMEU2xqMQ7hitu/view?usp=sharing

  Project Link:
  https://drive.google.com/drive/folders/1DxveRXxukxHPx2Vby1eIlH0CiPUfRT_N?usp=sharing