

A vertical bar on the left side of the slide with a gradient from orange at the top to purple at the bottom.

TRAINITY

Project Title,
Operation Analytics and Investigating Metric Spike

Advanced SQL

Name: Sanskruti Shimple

Description

Operational Analytics is a crucial process that involves analyzing a company's end-to-end operations. This analysis helps identify areas for improvement within the company. As a Data Analyst, you'll work closely with various teams, such as operations, support, and marketing, helping them derive valuable insights from the data they collect.

One of the key aspects of Operational Analytics is investigating metric spikes. This involves understanding and explaining sudden changes in key metrics, such as a dip in daily user engagement or a drop in sales. As a Data Analyst, you'll need to answer these questions daily, making it crucial to understand how to investigate these metric spikes.

In this project, you'll take on the role of a Lead Data Analyst at a company like Microsoft. You'll be provided with various datasets and tables, and your task will be to derive insights from this data to answer questions posed by different departments within the company. Your goal is to use your advanced SQL skills to analyze the data and provide valuable insights that can help improve the company's operations and understand sudden changes in key metrics.

Case Study 1: Job Data Analysis

You will be working with a table named `job_data` with the following columns:

- `job_id`: Unique identifier of jobs
- `actor_id`: Unique identifier of actor
- `event`: The type of event (decision/skip/transfer).
- `language`: The Language of the content
- `time_spent`: Time spent to review the job in seconds.
- `org`: The Organization of the actor
- `ds`: The date in the format `yyyy/mm/dd` (stored as text).

Project Overview:

This initiative centers around the examination of employment-related data housed in a database table labeled `job_data`. The primary objective is to leverage advanced SQL capabilities for scrutinizing and elucidating abrupt shifts in crucial performance indicators. Close collaboration with diverse departments is essential to extract actionable insights, enhance operational efficiency, and foster decision-making grounded in data. The deliverables encompass in-depth metric reports, documentation of collaborative efforts across teams, and the development of a versatile SQL query repository. The project strives to elevate overall company operations by perpetually monitoring and optimizing processes based on discernments derived from data-driven analyses.

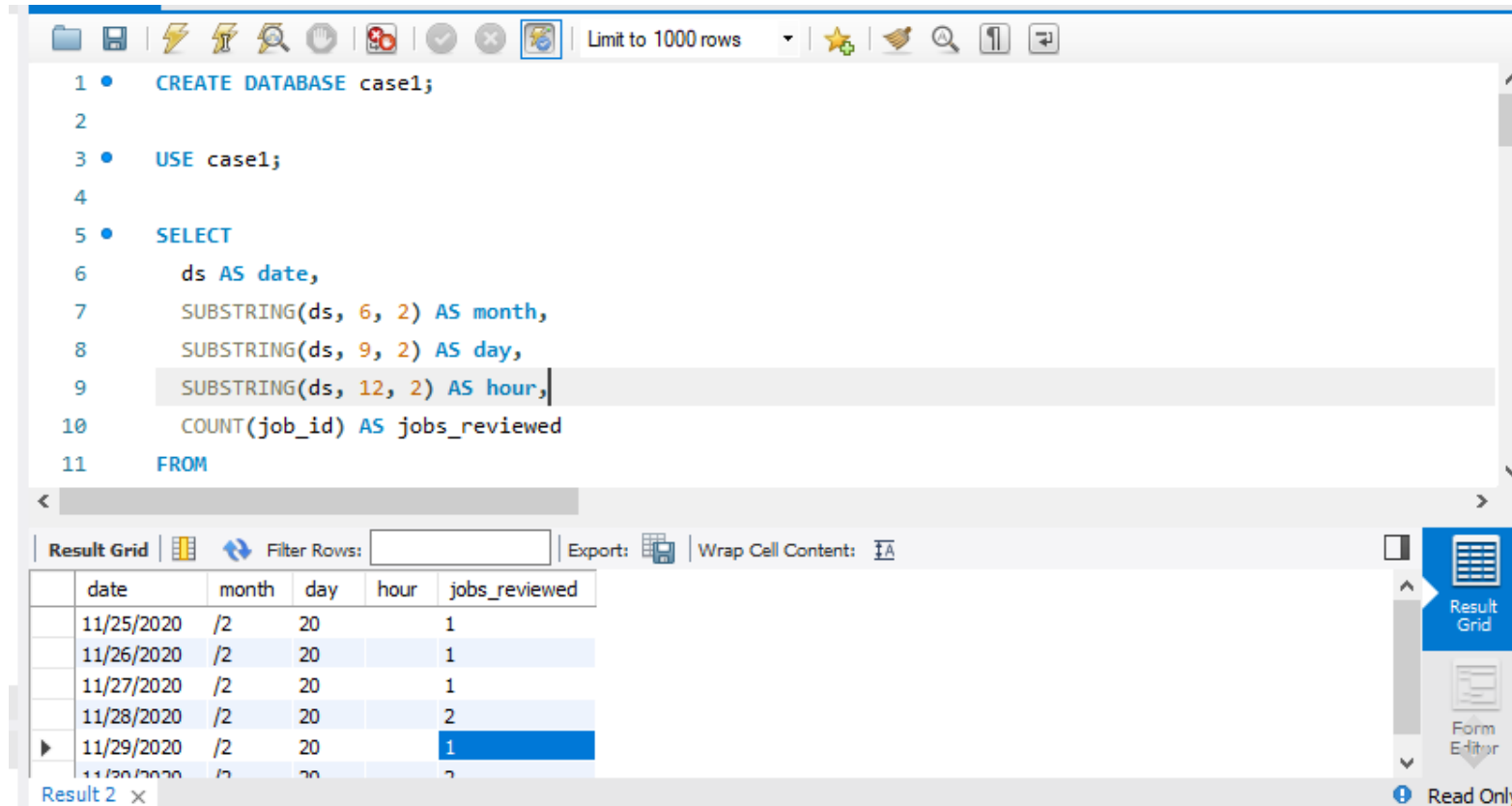
Tasks

Task 1 – Approach

Jobs Reviewed Over Time:

- A. Objective: Calculate the number of jobs reviewed per hour for each day in November 2020.
- B. Your Task: Write an SQL query to calculate the number of jobs reviewed per hour for each day in November 2020.

Output



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and settings. The query editor contains the following SQL code:

```
1 • CREATE DATABASE case1;
2
3 • USE case1;
4
5 • SELECT
6     ds AS date,
7     SUBSTRING(ds, 6, 2) AS month,
8     SUBSTRING(ds, 9, 2) AS day,
9     SUBSTRING(ds, 12, 2) AS hour,
10    COUNT(job_id) AS jobs_reviewed
11 FROM
```

Below the query editor, the 'Result Grid' tab is active, displaying a table with the following data:

date	month	day	hour	jobs_reviewed
11/25/2020	/2	20		1
11/26/2020	/2	20		1
11/27/2020	/2	20		1
11/28/2020	/2	20		2
11/29/2020	/2	20		1
11/30/2020	/2	20		2

The interface also includes a 'Filter Rows' field, 'Export' and 'Wrap Cell Content' buttons, and a 'Read Only' status indicator at the bottom right.

- Parsed the 'ds' column to extract the date, month, day, and hour components.
- Applied a data filter to isolate records from November 2020 and executed a group-by operation on date and hour.
- Computed the count of jobs reviewed for each hour within the specified timeframe.

Observation- 1:

Fluctuations in Job Reviews Over Time:

- Detected fluctuations in the volume of job reviews across the month, indicating possible peak hours.

Outcome:

- Unveiled the hourly count of jobs reviewed for each day within the month of November 2020.

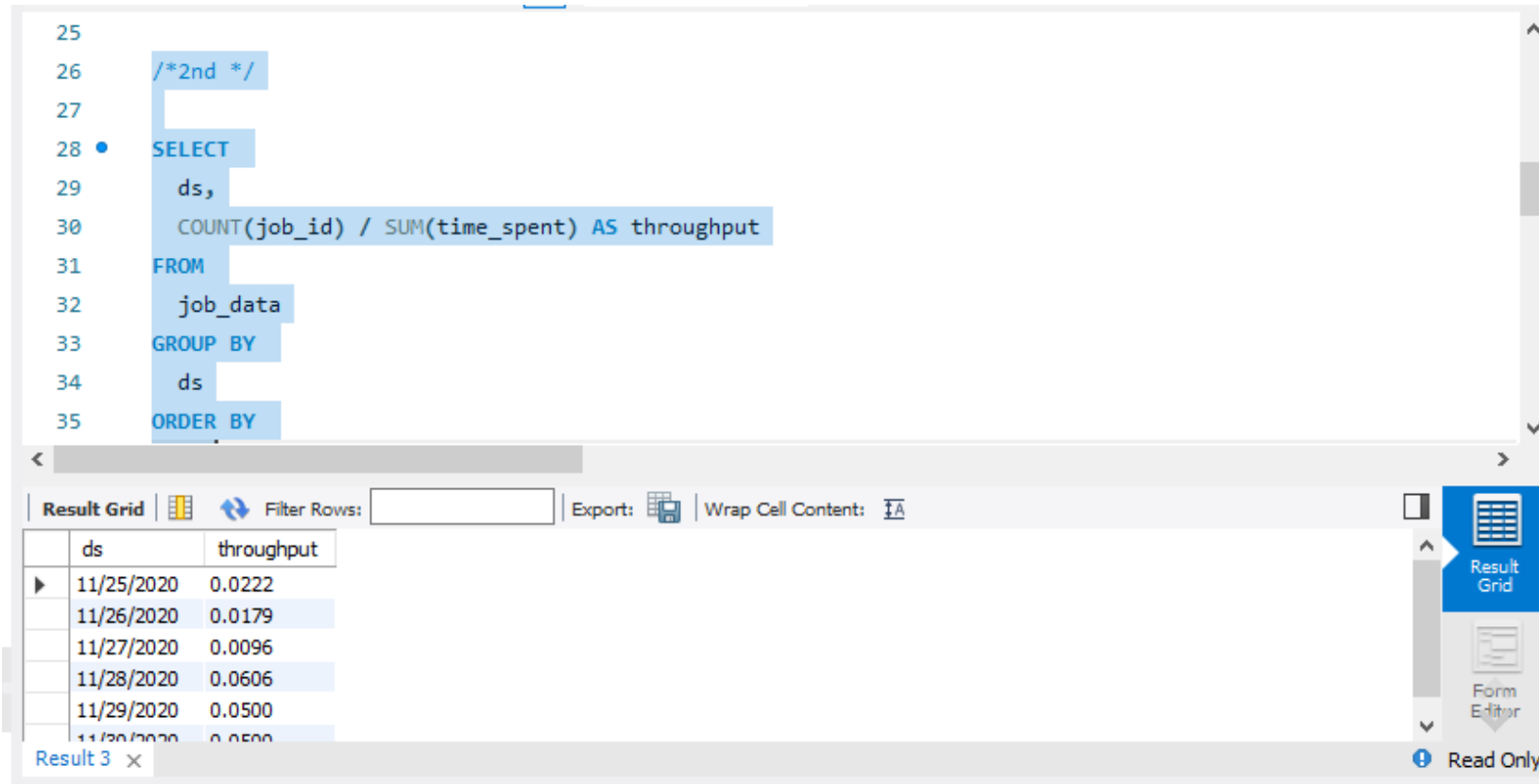
Tasks

Task 2 – Approach

Throughput Analysis:

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

Output



The screenshot shows a SQL query editor with a query that calculates throughput by date. The query is as follows:

```
25
26 /*2nd */
27
28 • SELECT
29     ds,
30     COUNT(job_id) / SUM(time_spent) AS throughput
31 FROM
32     job_data
33 GROUP BY
34     ds
35 ORDER BY
```

Below the query editor is a results grid with the following data:

ds	throughput
11/25/2020	0.0222
11/26/2020	0.0179
11/27/2020	0.0096
11/28/2020	0.0606
11/29/2020	0.0500
11/30/2020	0.0500

The interface includes a 'Result Grid' tab, a 'Filter Rows' input field, and buttons for 'Export', 'Wrap Cell Content', 'Result Grid', 'Form Editor', and 'Read Only'.

- Calculated throughput by dividing the count of job_id by the summation of time_spent.
- Organized data by date and arranged the outcomes.

Observation-

Utilizing the 7-day rolling average yields a more stable trend, mitigating daily variances. This approach offers a more reliable indicator of the system's overall performance.

Outcome - Produced the 7-day rolling average for throughput, offering a consistent metric for thorough performance analysis.

Tasks

Task 3 – Approach

Language Share Analysis:

- A. Objective: Calculate the percentage share of each language in the last 30 days.
- B. Your Task: Write an SQL query to calculate the percentage share of each language over the last 30 days.

Output

```
42 • SELECT
43     language,
44     COUNT(*) * 100.0 / SUM(COUNT(*)) OVER () AS language_percentage
45 FROM
46     job_data
47 WHERE
48     ds BETWEEN (SELECT MAX(ds) FROM job_data) - 29 AND (SELECT MAX(ds) FROM job_data)
49 GROUP BY
50     language;
51
52
```

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

	language	language_percentage
▶	English	12.50000
	Arabic	12.50000
	Persian	37.50000
	Hindi	12.50000
	French	12.50000
	Italian	12.50000

Result 4 x | Read Only

- Derived the percentage distribution of each language in the preceding 30 days.
- Employed a subquery to narrow down data to the last 30 days and executed a group-by operation based on language.

Observation – Unearthed the proportional representation of each language, providing valuable insights into language preferences over the last 30 days.

Outcome – Computed the percentage share for each language in the recent 30-day period, contributing to a comprehensive understanding of language distribution.

Tasks

Task 4 – Approach

Duplicate Rows Detection:

- A. Objective: Identify duplicate rows in the data.
- B. Your Task: Write an SQL query to display duplicate rows from the job_data table.

Output

```
63 WHERE
64 (ds, job_id, actor_id, event, language, time_spent, org) IN (
65     SELECT ds, job_id, actor_id, event, language, time_spent, org
66     FROM job_data
67     GROUP BY ds, job_id, actor_id, event, language, time_spent, org
68     HAVING COUNT(*) > 1
69 )
70 ORDER BY
71 ds, job_id, actor_id, event, language, time_spent, org;
72
73
```

Limit to 1000 rows

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

ds	job_id	actor_id	event	language	time_spent	org
----	--------	----------	-------	----------	------------	-----

job_data 5 x Read Only

- Implemented a subquery to pinpoint duplicate rows by considering all columns in the table.
- Extracted rows with counts exceeding 1, signaling the presence of duplicates.

Observation –

Recognized and presented rows containing duplicate values, highlighting potential concerns related to data integrity.

Outcome – Revealed and showcased duplicate rows, facilitating the process of data cleaning and upholding data integrity.

Tech Stack

- MySQL Workbench Used for executing SQL queries and database management
- Excel.

Conclusion

This examination enhances decision-making by furnishing an all-encompassing comprehension of patterns within job data, trends in throughput, language preferences, and issues related to data quality. The acquired insights have the potential to guide strategic decision-making and optimizations within the job processing system.



Case Study 2: Investigating Metric Spike

- You will be working with three tables:
- users: Contains one row per user, with descriptive information about that user's account.
- events: Contains one row per event, where an event is an action that a user has taken (e.g., login, messaging, search).
- email_events: Contains events specific to the sending of emails.

A vertical bar on the left side of the slide with a gradient from orange at the top to blue at the bottom.

Project Overview:

The central objective of this project revolved around scrutinizing user engagement, growth, retention, and email interactions by leveraging data sourced from three distinct tables: users, events, and email_events. The methodology encompassed the formulation of SQL queries to extract meaningful insights from the datasets, providing a comprehensive understanding of user behavior over different periods.

Technology Stack :

1. MySQL Server 8.0:

- Served as the database management system for the storage and retrieval of structured data.
- Employed for tasks such as table creation, data loading, and execution of SQL queries.

2. MySQL Workbench:

- Functioned as a graphical user interface tailored for MySQL, facilitating database design, administration, and querying.
- Utilized for tasks like designing and visualizing the database schema, executing queries, and analyzing results.

Tasks

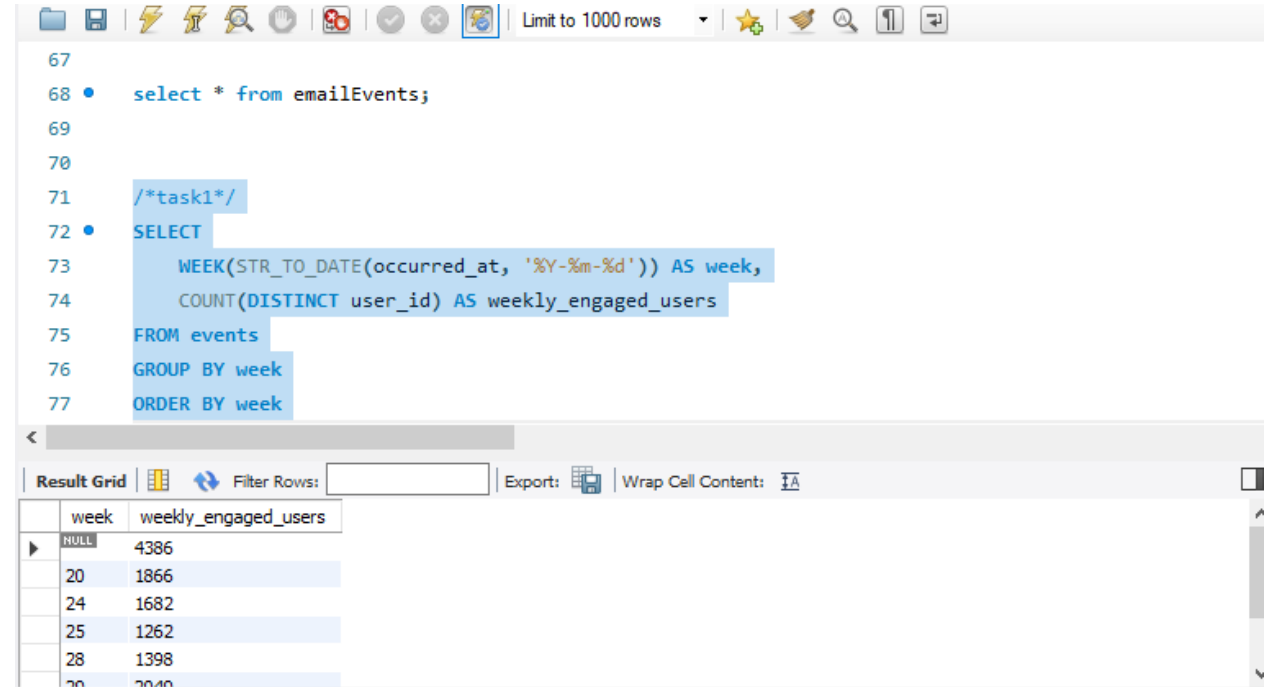
Task 1 – Approach

Weekly User Engagement:

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

Output

```
SELECT
    WEEK(STR_TO_DATE(occurred_at, '%Y-%m-%d')) AS week,
    COUNT(DISTINCT user_id) AS
weekly_engaged_users
FROM events
GROUP BY week
ORDER BY week;
```



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and search. The query editor displays a SQL query with line numbers 67 through 77. The query is as follows:

```
67
68 • select * from emailEvents;
69
70
71 /*task1*/
72 • SELECT
73     WEEK(STR_TO_DATE(occurred_at, '%Y-%m-%d')) AS week,
74     COUNT(DISTINCT user_id) AS weekly_engaged_users
75 FROM events
76 GROUP BY week
77 ORDER BY week
```

Below the query editor is the 'Result Grid' section. It includes a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox. The results are displayed in a table with two columns: 'week' and 'weekly_engaged_users'.

week	weekly_engaged_users
NULL	4386
20	1866
24	1682
25	1262
28	1398
29	2040

Observation - The provided query computes the count of unique users participating in events on a weekly cadence. This aids in tracking the broader spectrum of user activity over an extended period.

Tasks

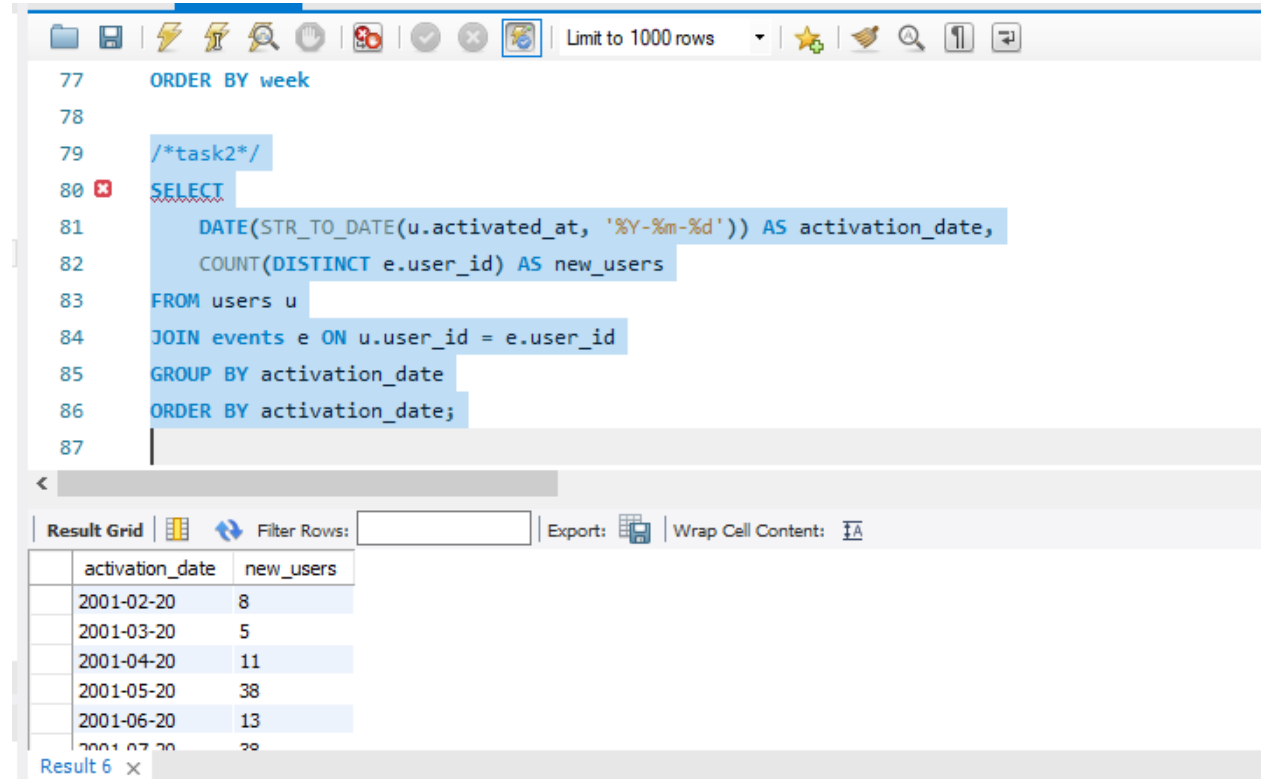
Task 2 – Approach

User Growth Analysis:

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

Output

```
SELECT
    DATE(STR_TO_DATE(u.activated_at, '%Y-
    %m-%d')) AS activation_date,
    COUNT(DISTINCT e.user_id) AS new_users
FROM users u
JOIN events e ON u.user_id = e.user_id
GROUP BY activation_date
ORDER BY activation_date;
```



The screenshot shows a SQL query editor with a toolbar at the top. The query is as follows:

```
77 ORDER BY week
78
79 /*task2*/
80 SELECT
81     DATE(STR_TO_DATE(u.activated_at, '%Y-%m-%d')) AS activation_date,
82     COUNT(DISTINCT e.user_id) AS new_users
83 FROM users u
84 JOIN events e ON u.user_id = e.user_id
85 GROUP BY activation_date
86 ORDER BY activation_date;
87
```

Below the query editor, the results are displayed in a table with the following data:

activation_date	new_users
2001-02-20	8
2001-03-20	5
2001-04-20	11
2001-05-20	38
2001-06-20	13
2001-07-20	20

The interface also includes a 'Limit to 1000 rows' dropdown, a 'Filter Rows' input field, and an 'Export' button.

Observation –

The presented query scrutinizes user growth by tallying the count of newly activated users for each date. This affords a temporal view of the user acquisition timeline.

Tasks

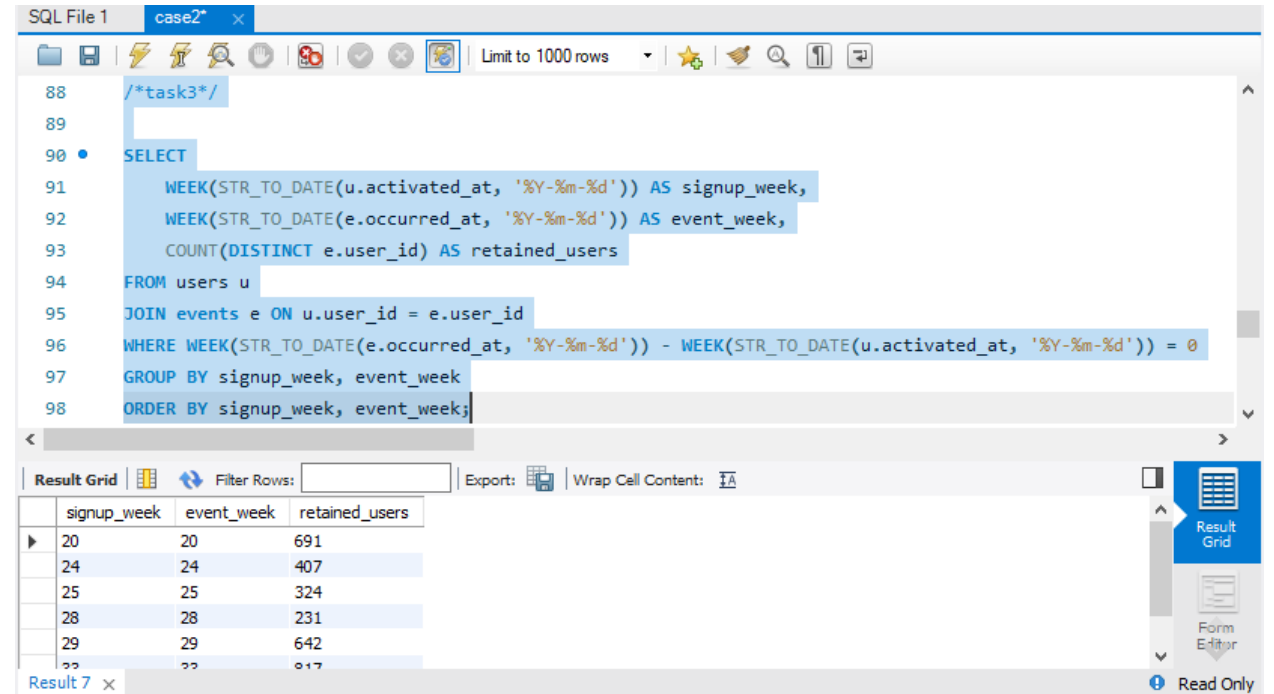
Task 3 – Approach

Weekly Retention Analysis:

- A. Objective: Analyze the retention of users on a weekly basis after signing up for a product.
- B. Your Task: Write an SQL query to calculate the weekly retention of users based on their sign-up cohort.

Output

```
SELECT
    WEEK(STR_TO_DATE(u.activated_at, '%Y-%m-%d')) AS
signup_week,
    WEEK(STR_TO_DATE(e.occurred_at, '%Y-%m-%d')) AS
event_week,
    COUNT(DISTINCT e.user_id) AS retained_users
FROM users u
JOIN events e ON u.user_id = e.user_id
WHERE WEEK(STR_TO_DATE(e.occurred_at, '%Y-%m-%d'))
- WEEK(STR_TO_DATE(u.activated_at, '%Y-%m-%d')) = 0
GROUP BY signup_week, event_week
ORDER BY signup_week, event_week;
```



The screenshot shows a SQL IDE window titled "SQL File 1" with a tab "case2* x". The query editor contains the following SQL code:

```
/*task3*/
SELECT
    WEEK(STR_TO_DATE(u.activated_at, '%Y-%m-%d')) AS signup_week,
    WEEK(STR_TO_DATE(e.occurred_at, '%Y-%m-%d')) AS event_week,
    COUNT(DISTINCT e.user_id) AS retained_users
FROM users u
JOIN events e ON u.user_id = e.user_id
WHERE WEEK(STR_TO_DATE(e.occurred_at, '%Y-%m-%d')) - WEEK(STR_TO_DATE(u.activated_at, '%Y-%m-%d')) = 0
GROUP BY signup_week, event_week
ORDER BY signup_week, event_week;
```

The results pane shows a table with the following data:

signup_week	event_week	retained_users
20	20	691
24	24	407
25	25	324
28	28	231
29	29	642
30	30	617

The IDE interface includes a toolbar with icons for file operations, a "Limit to 1000 rows" dropdown, and a "Result Grid" button. The status bar at the bottom indicates "Read Only".

Observation –

The provided query computes the weekly retention of users by assessing the alignment between the week of user activation and the week of subsequent events. This analysis aids in gauging the efficacy of user retention post-signup.

Tasks

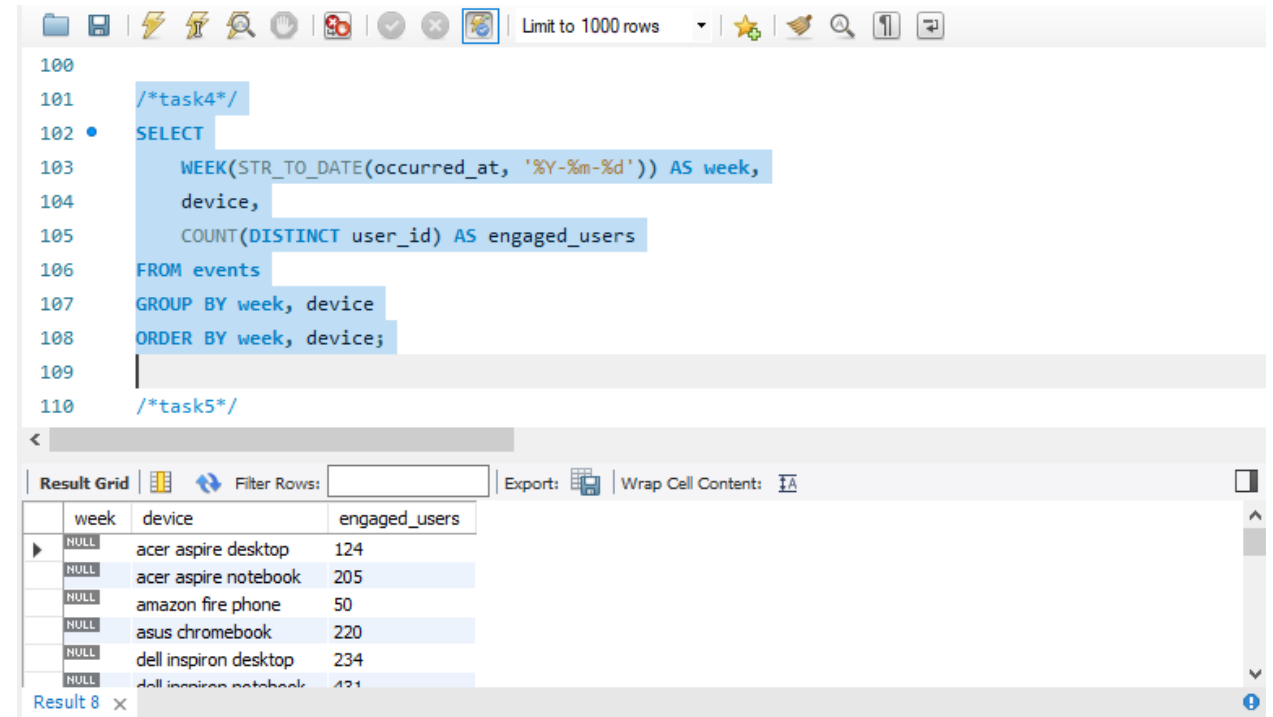
Task 4 – Approach

Weekly Engagement Per Device:

- A. Objective: Measure the activeness of users on a weekly basis per device.
- B. Your Task: Write an SQL query to calculate the weekly engagement per device.

Output

```
SELECT
    WEEK(STR_TO_DATE(occurred_at, '%Y-%m-%d')) AS
week,
    device,
    COUNT(DISTINCT user_id) AS engaged_users
FROM events
GROUP BY week, device
ORDER BY week, device;
```



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and settings. The query editor displays a SQL query between markers `/*task4*/` and `/*task5*/`. The query is a SELECT statement that calculates the week number from the `occurred_at` date, lists the device, and counts the distinct `user_id` as `engaged_users`. The query is grouped by `week` and `device` and ordered by `week` and `device`. Below the query editor, the 'Result Grid' tab is active, showing a table with three columns: `week`, `device`, and `engaged_users`. The table contains six rows of data, with the `week` column showing NULL values. The interface also includes a 'Filter Rows' input, an 'Export' button, and a 'Wrap Cell Content' checkbox.

week	device	engaged_users
NULL	acer aspire desktop	124
NULL	acer aspire notebook	205
NULL	amazon fire phone	50
NULL	asus chromebook	220
NULL	dell inspiron desktop	234
NULL	dell inspiron notebook	121

Result 8 x

Observation –

The included query furnishes insights into user engagement categorized by device on a weekly timeframe. This facilitates the identification of devices that significantly contribute to user activity.

Tasks

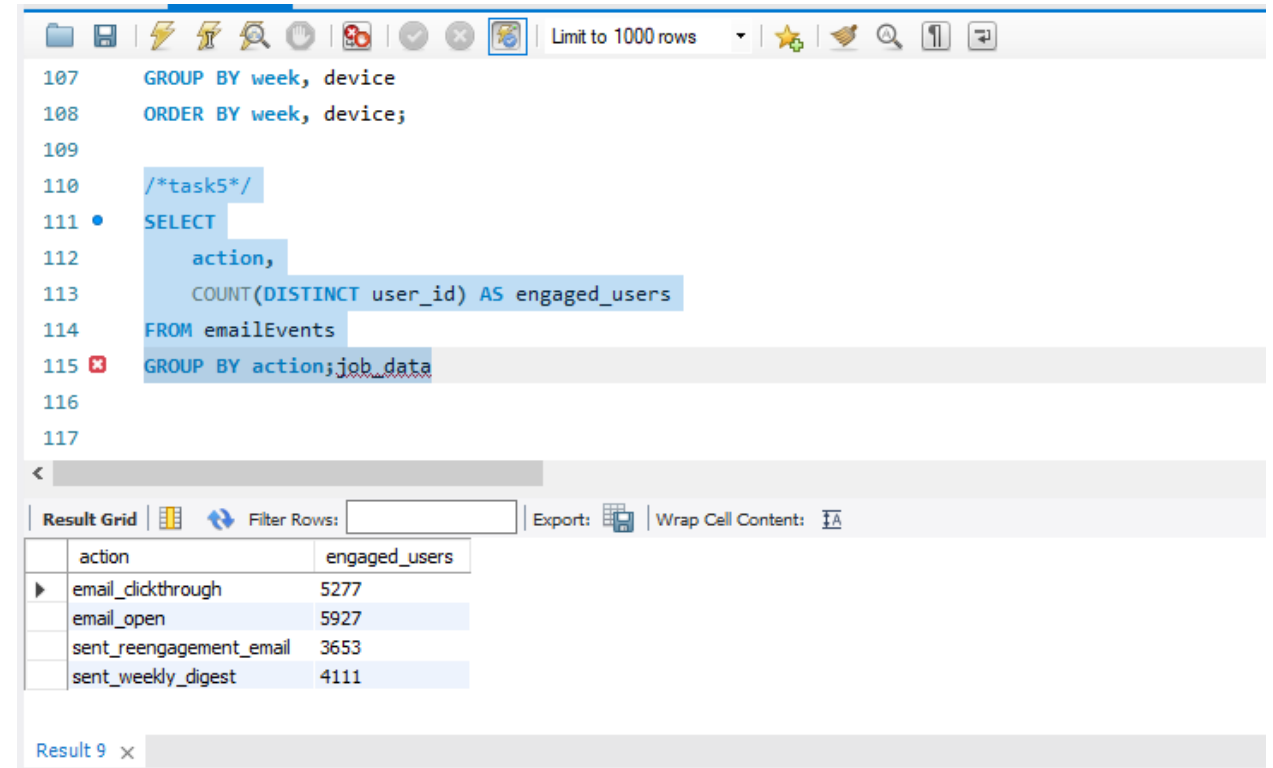
Task 5– Approach

Email Engagement Analysis:

- A. Objective: Analyze how users are engaging with the email service.
- B. Your Task: Write an SQL query to calculate the email engagement metrics.

Output

```
SELECT
    action,
    COUNT(DISTINCT user_id) AS engaged_users
FROM emailEvents
GROUP BY action;
```



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, a 'Limit to 1000 rows' dropdown, and other utility icons. The SQL editor contains the following code:

```
107 GROUP BY week, device
108 ORDER BY week, device;
109
110 /*task5*/
111 • SELECT
112     action,
113     COUNT(DISTINCT user_id) AS engaged_users
114 FROM emailEvents
115 ✖ GROUP BY action; job_data
116
117
```

Below the editor is a 'Result Grid' section with a 'Filter Rows' input and 'Export' and 'Wrap Cell Content' options. The results are displayed in a table:

action	engaged_users
email_clickthrough	5277
email_open	5927
sent_reengagement_email	3653
sent_weekly_digest	4111

At the bottom, there is a tab labeled 'Result 9' with a close button.

Observation –

The presented query assesses user engagement concerning various email actions, offering valuable insights into the popularity of different email interactions among users.

Results

This initiative has yielded invaluable insights into user behaviors, growth patterns, and engagement dynamics. Notable accomplishments encompass:

- Identification of peak engagement periods and trends across weeks.
- Discerning the relationship between user activation and subsequent events.
- Assessing the efficacy of email interactions and gauging user responses to diverse actions.

These revelations contribute significantly to well-informed decision-making, enabling the formulation of targeted strategies to enhance user engagement, optimize acquisition endeavors, and elevate overall product performance. The application of MySQL Workbench alongside strategic SQL queries has proven to be efficacious in extracting meaningful information from the dataset, fostering a data-centric approach to decision-making.