PROJECT 3- Operation Analytics and Investigating Metric Spike

Yashi Gupta

PROJECT DESCRIPTION

This project aims analyzing user engagement and other activities using the provided dataset to gain meaningful insights and identify trends in the product. The data contains several tables for analysis. This project aims to answer key questions about how users interact with the product, how they engage through email, and what their behavior is like after sign-up. The goal is to measure the growth of users on a monthly, weekly, or yearly basis.

Operational analytics is a important process One of the key aspects is investigating metric spikes, which includes understanding sudden drops in user engagement, increases in sales, spikes in user engagement, and changes in the number of sign-ups or weekly retention. In this project, we leveraged SQL to answer these questions and gain better insights from the data.

Approach

To approach this project, I followed a structured methodology:

Data Exploration: First, I studied the provided database and identified the key tables: users, photos, comments, likes, follows, and tags. From those tables, I identified primary keys, foreign keys, and composite keys for better understanding.

SQL Queries: I executed SQL queries for the given tasks

Concepts used for querying: I used aggregation techniques, such as COUNT, GROUP BY, and HAVING, to calculate metrics like the most common days of the week users engage on Instagram and the top-performing users.

JOINS.UNIONS were also crutial requiremensts.

For every question, I tried thinking of different approaches to improve my concept understanding.

.For any given question, the main tasks are to: 1] Identify the tables involved 2] The required output 3] The function which will help in finding the required output

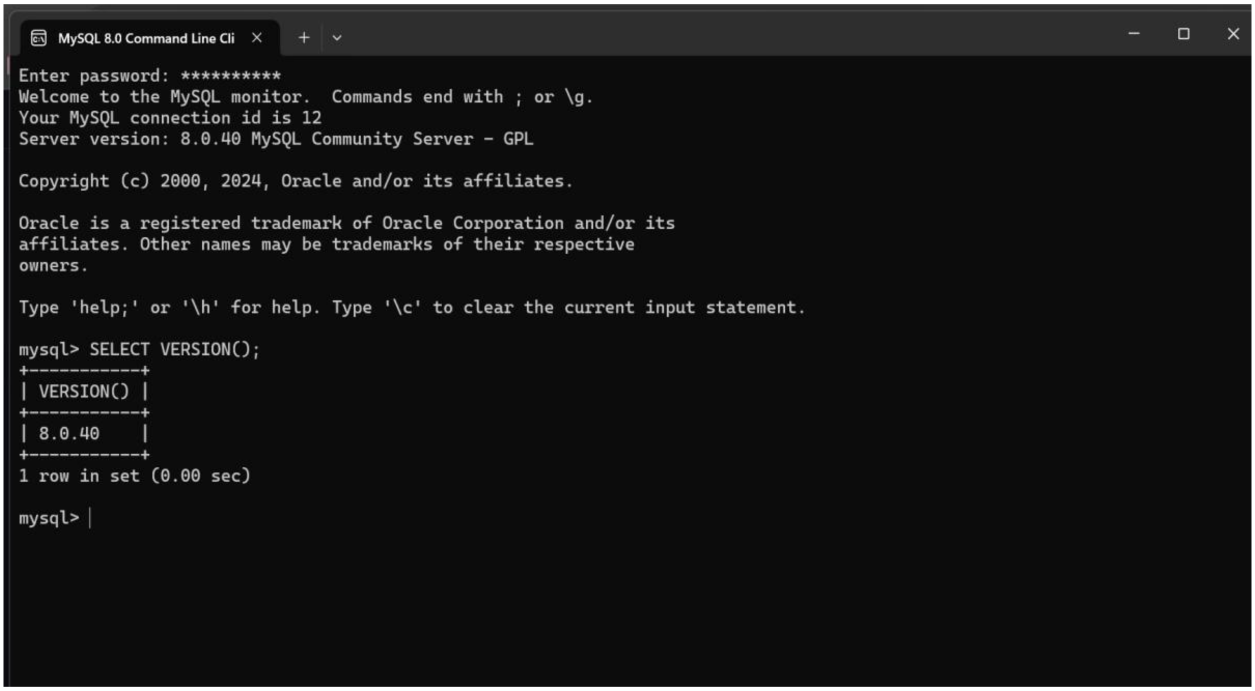
Tech-Stack Used

1. MySQL Workbench
2. MySQL
3. Microsoft Word.

MySQL Workbench: This integrated development environment (IDE) was used for SQL development, allowing for efficient database management, query writing.

MySQL: The relational database management system (RDBMS) used for storing and manipulating the data. It was ideal for querying and analyzing large datasets.

Microsoft Word: Used for documenting the project. For versions-



```
Enter password: **********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 8.0.40 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SELECT VERSION();
+-----------+
| VERSION() |
+-----------+
| 8.0.40    |
+-----------+
1 row in set (0.00 sec)

mysql>
```

Insights

The following insights were derived while working on this project.

At first, monthly and weekly engagement metrics revealed how many active or passive users there were, the period in which the interaction was highest, and the types of events that users were most attracted to and engaged with the most. We used sign-up data to have a clear view of identifying the users who stayed and visited one week after signing up. Different types of devices were used for interacting with the service, and we also determined which types of devices were used the most. The behavior after first signing up helped assess how good the product was and identified the product users over time.

Result and Achievements

After working on this project I have successfully achieved the following –

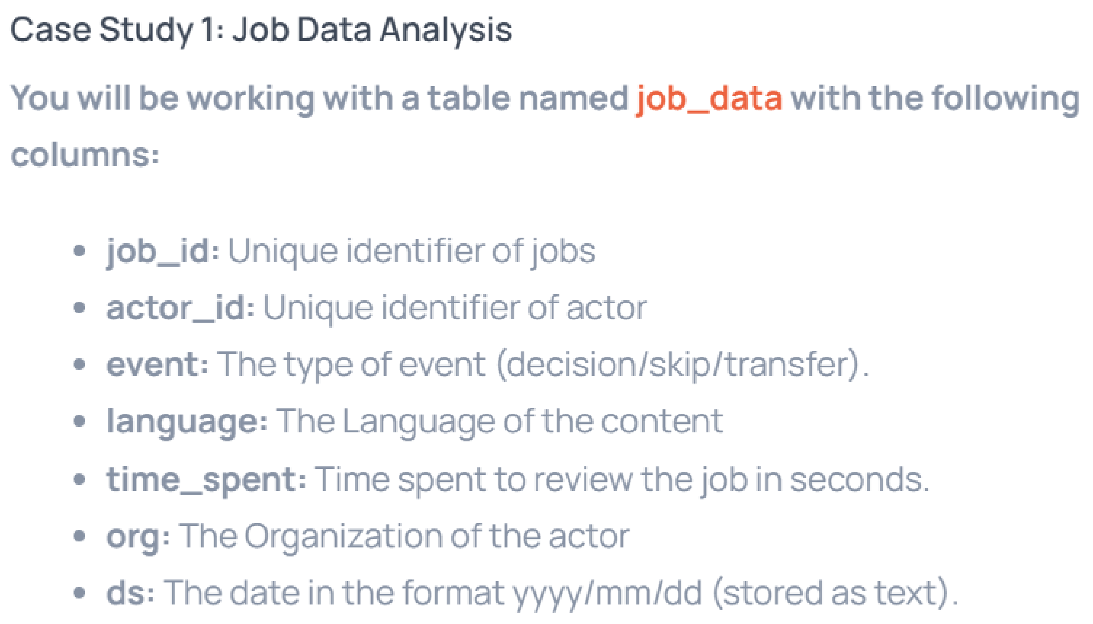
This project involved working with a large dataset and solving complex querying problems, which required a deeper understanding of advanced SQL

Now understand the meaning of operational analytics and the investigation of metric spikes, as well as how they affect the user base and a company. I learned what valuable insights can be collected from data in the context of operational analytics. By leveraging SQL,

I was able to analyze large datasets and gain advanced understanding of users, their retention patterns, growth trends, and acquisition strategies.

Additionally, I examined user activity and engagement on a weekly, monthly, and yearly basis

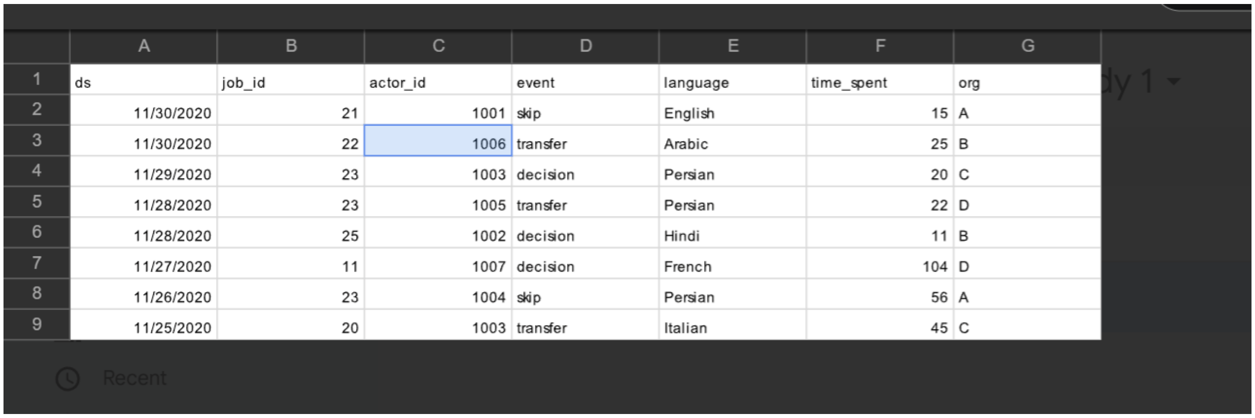
Screenshots and explanation



Case Study 1: Job Data Analysis

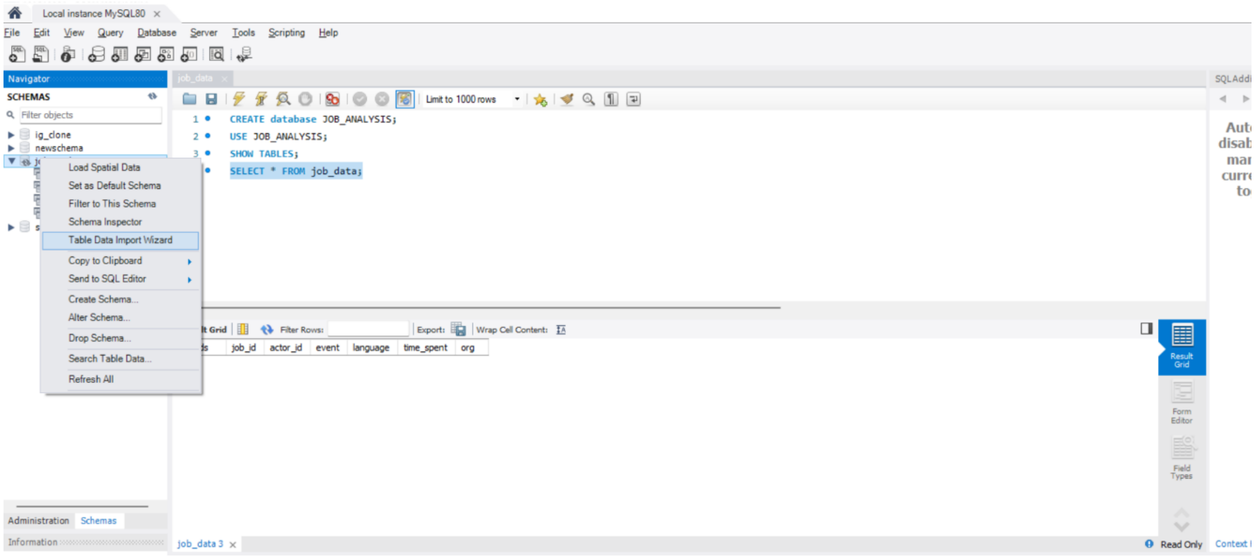
You will be working with a table named job_data with the following columns:

- **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).



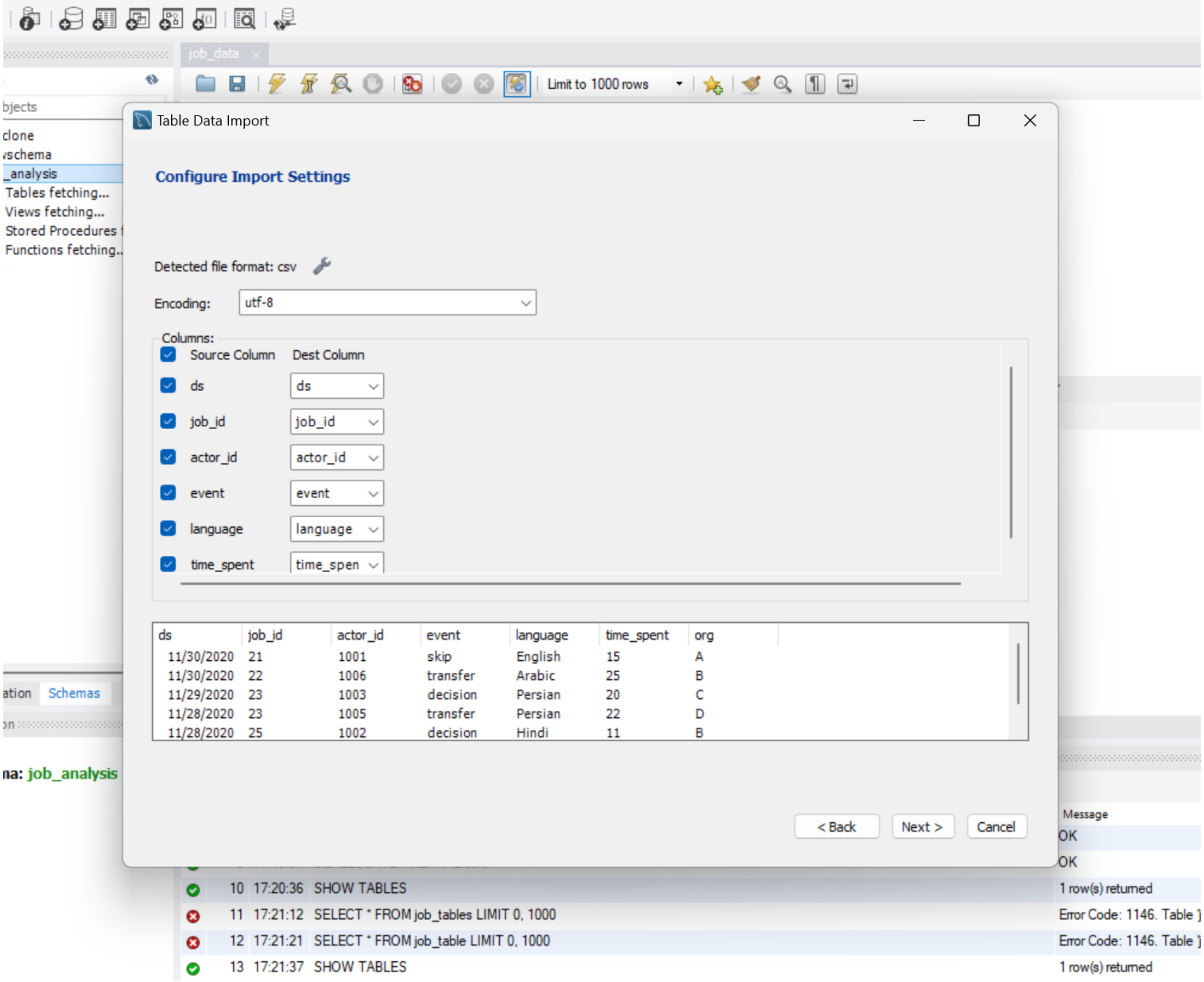
	A	B	C	D	E	F	G
1	ds	job_id	actor_id	event	language	time_spent	org
2	11/30/2020	21	1001	skip	English	15	A
3	11/30/2020	22	1006	transfer	Arabic	25	B
4	11/29/2020	23	1003	decision	Persian	20	C
5	11/28/2020	23	1005	transfer	Persian	22	D
6	11/28/2020	25	1002	decision	Hindi	11	B
7	11/27/2020	11	1007	decision	French	104	D
8	11/26/2020	23	1004	skip	Persian	56	A
9	11/25/2020	20	1003	transfer	Italian	45	C

.csv file provided as dataset for the project.



```
CREATE database JOB_ANALYSIS;
USE JOB_ANALYSIS;
SHOW TABLES;
SELECT * FROM job_data;
```

ds	job_id	actor_id	event	language	time_spent	org



Configure Import Settings

Detected file format: csv

Encoding: utf-8

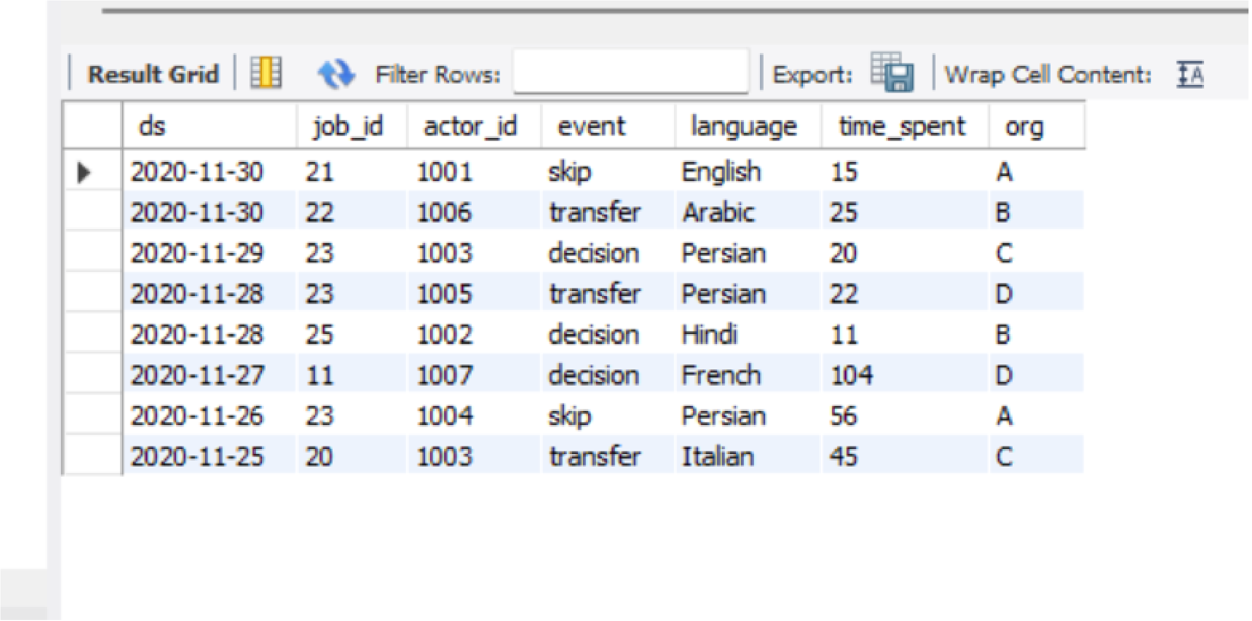
Columns:

☑ Source Column	Dest Column
☑ ds	ds
☑ job_id	job_id
☑ actor_id	actor_id
☑ event	event
☑ language	language
☑ time_spent	time_spen

ds	job_id	actor_id	event	language	time_spent	org
11/30/2020	21	1001	skip	English	15	A
11/30/2020	22	1006	transfer	Arabic	25	B
11/29/2020	23	1003	decision	Persian	20	C
11/28/2020	23	1005	transfer	Persian	22	D
11/28/2020	25	1002	decision	Hindi	11	B

```
                                                          OK
                                                          OK
10  17:20:36  SHOW TABLES                                 1 row(s) returned
11  17:21:12  SELECT * FROM job_tables LIMIT 0, 1000      Error Code: 1146. Table
12  17:21:21  SELECT * FROM job_table LIMIT 0, 1000       Error Code: 1146. Table
13  17:21:37  SHOW TABLES                                 1 row(s) returned
```

1. Create database as JOB_ANALYSIS.
2. Import the csv file use import wizard

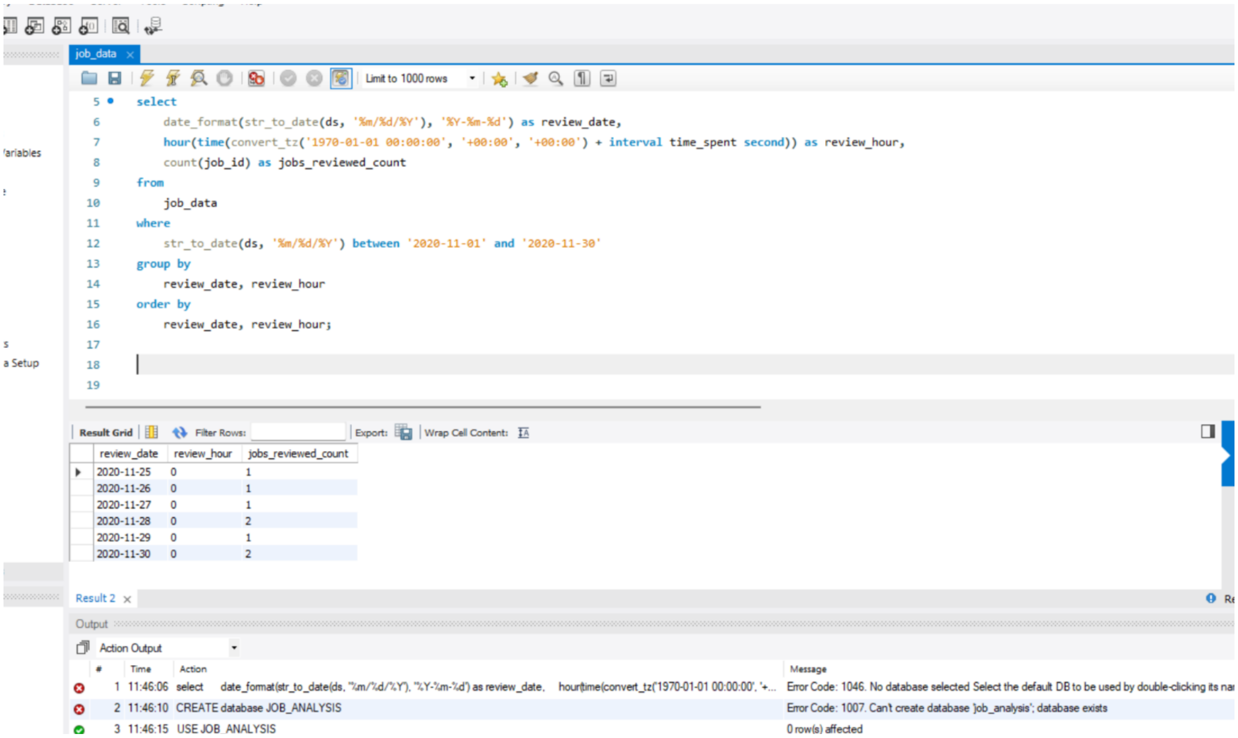


ds	job_id	actor_id	event	language	time_spent	org
2020-11-30	21	1001	skip	English	15	A
2020-11-30	22	1006	transfer	Arabic	25	B
2020-11-29	23	1003	decision	Persian	20	C
2020-11-28	23	1005	transfer	Persian	22	D
2020-11-28	25	1002	decision	Hindi	11	B
2020-11-27	11	1007	decision	French	104	D
2020-11-26	23	1004	skip	Persian	56	A
2020-11-25	20	1003	transfer	Italian	45	C

All the required data is filled and now we will move on towards the 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.



```sql
select
    date_format(str_to_date(ds, '%m/%d/%Y'), '%Y-%m-%d') as review_date,
    hour(time(convert_tz('1970-01-01 00:00:00', '+00:00', '+00:00') + interval time_spent second)) as review_hour,
    count(job_id) as jobs_reviewed_count
from
    job_data
where
    str_to_date(ds, '%m/%d/%Y') between '2020-11-01' and '2020-11-30'
group by
    review_date, review_hour
order by
    review_date, review_hour;
```

review_date	review_hour	jobs_reviewed_count
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

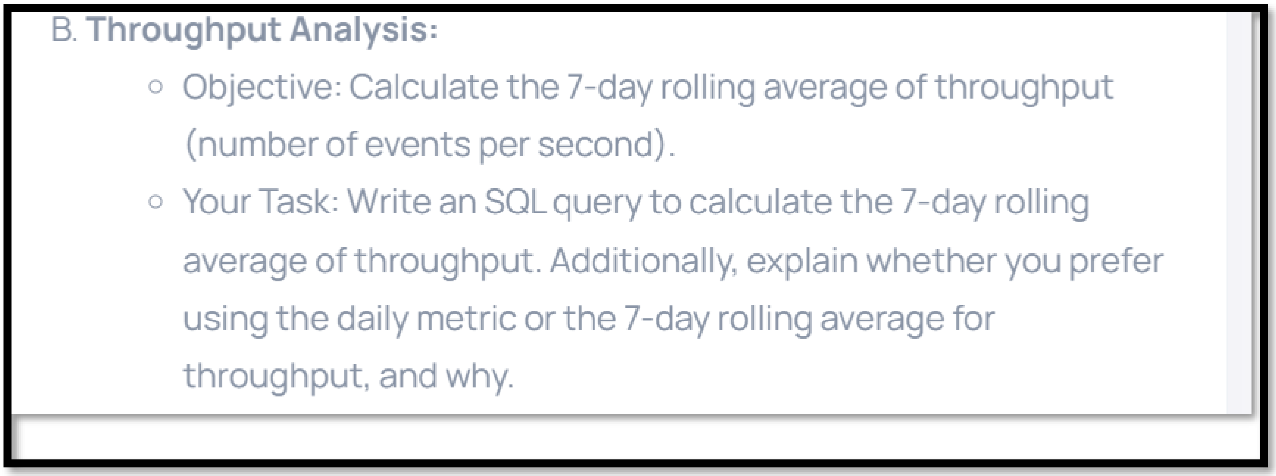
```
#  Time      Action                                                                                                       Message
1  11:46:06  select    date_format(str_to_date(ds, '%m/%d/%Y'), '%Y-%m-%d') as review_date,    hour(time(convert_tz('1970-01-01 00:00:00', '+...   Error Code: 1046. No database selected Select the default DB to be used by double-clicking its na
2  11:46:10  CREATE database JOB_ANALYSIS                                                                                 Error Code: 1007. Can't create database 'job_analysis'; database exists
3  11:46:15  USE JOB_ANALYSIS                                                                                             0 row(s) affected
```

So lets understand this question and query step by step-

The purpose of this query is to calculate the **number of jobs reviewed per hour for each day in November 2020**.

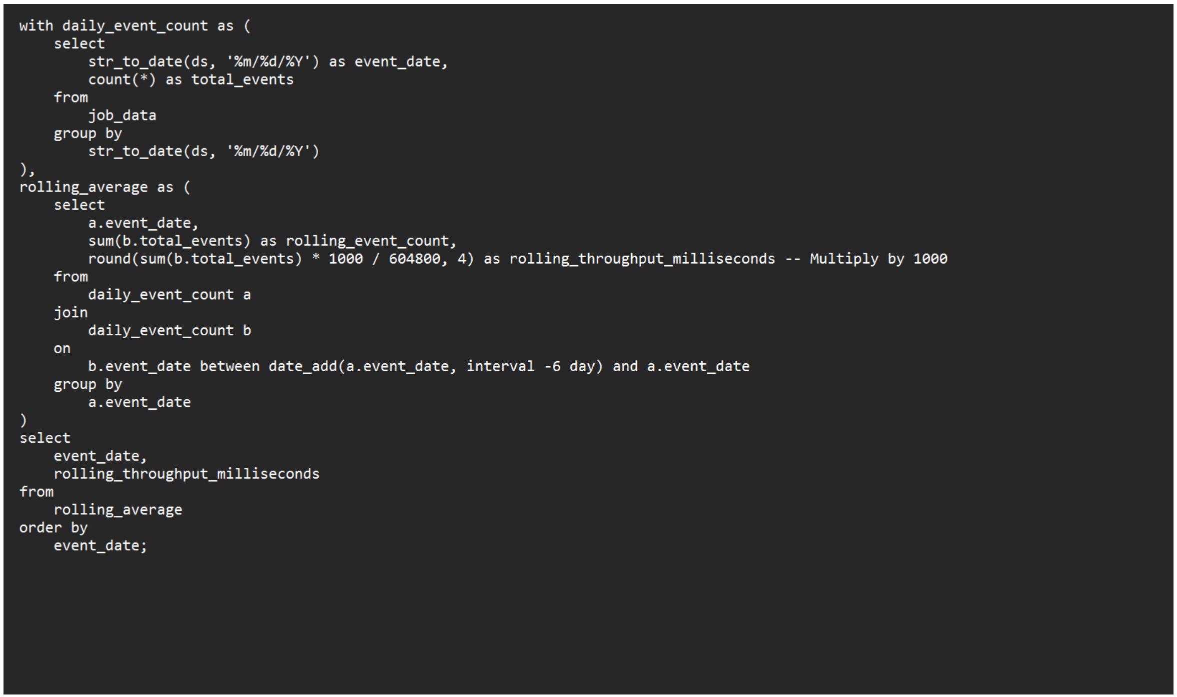
1. It uses the ds column to group data by day and derives the hour from the time_spent column.
2. The ds column, which contains dates in the format MM/DD/YYYY, is converted into a proper date format using str_to_date().This ensures the date can be used for grouping and filtering .
3. The time_spent column represents the time spent reviewing a job in seconds. This value is added as an interval to a base timestamp
4. The query counts the number of job_id entries for each unique combination of review_date and review_hour.
5. The group by clause aggregates data by review_date and review_hour.
6. The order by clause ensures the output is sorted chronologically.

We can observe that the review_hour in the output is consistently 0 because the dataset does not include a specific time for the job reviews. The ds column only provides the date, and since there is no explicit hour information in the dataset, the query defaults to midnight when calculating the review hour.

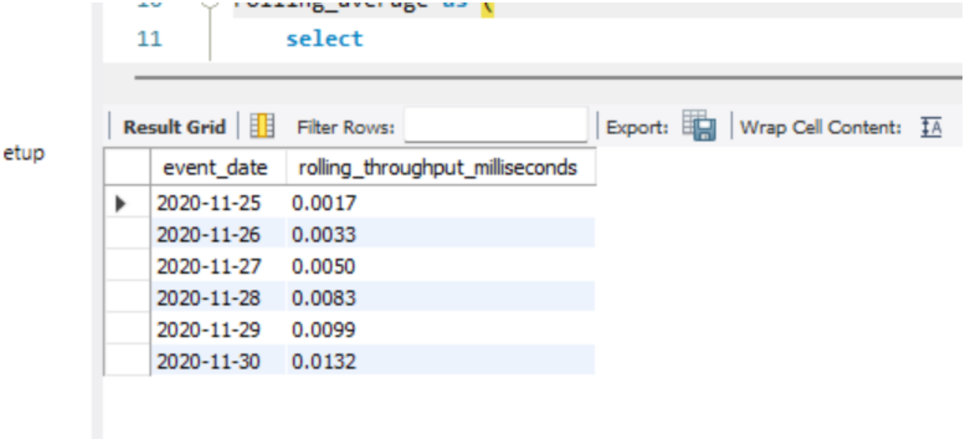


B. **Throughput Analysis:**

- 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_event_count as (
    select
        str_to_date(ds, '%m/%d/%Y') as event_date,
        count(*) as total_events
    from
        job_data
    group by
        str_to_date(ds, '%m/%d/%Y')
),
rolling_average as (
    select
        a.event_date,
        sum(b.total_events) as rolling_event_count,
        round(sum(b.total_events) * 1000 / 604800, 4) as rolling_throughput_milliseconds -- Multiply by 1000
    from
        daily_event_count a
    join
        daily_event_count b
    on
        b.event_date between date_add(a.event_date, interval -6 day) and a.event_date
    group by
        a.event_date
)
select
    event_date,
    rolling_throughput_milliseconds
from
    rolling_average
order by
    event_date;
```

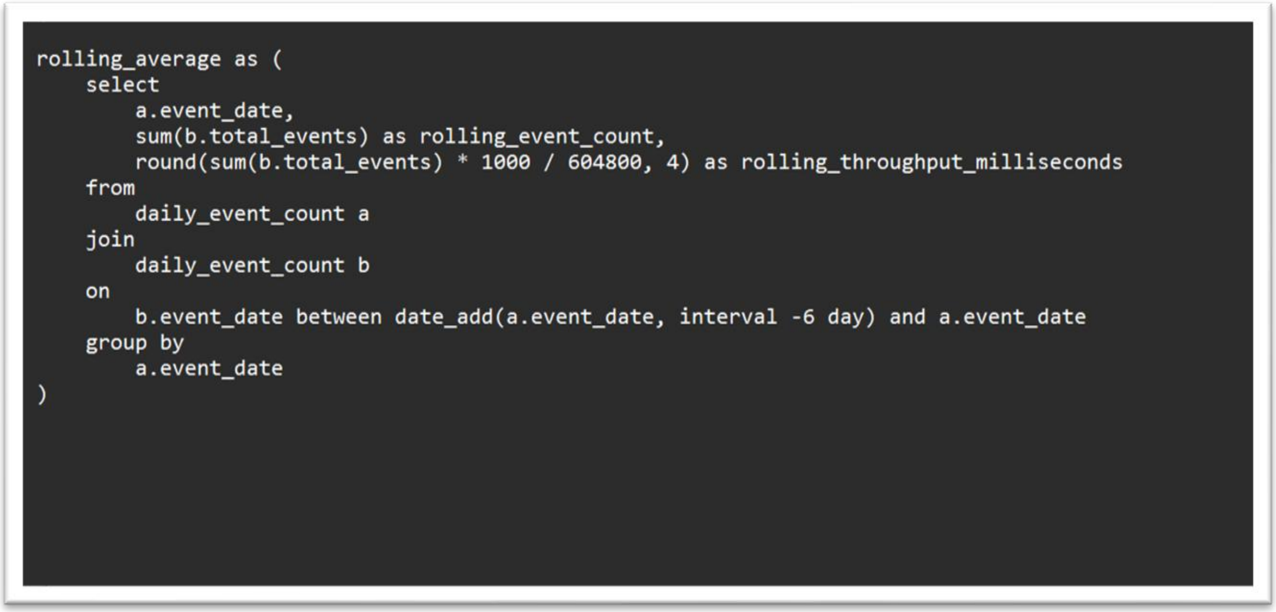


```
    select
```

event_date	rolling_throughput_milliseconds
2020-11-25	0.0017
2020-11-26	0.0033
2020-11-27	0.0050
2020-11-28	0.0083
2020-11-29	0.0099
2020-11-30	0.0132

The query calculates the **7-day rolling average** of throughput, which is the number of events per second over a 7-day window.

1. The str_to_date(ds, '%m/%d/%Y') function converts the ds column (which stores the date as a string) into a DATE format.
2. The count(*) function counts the number of events (job_id occurrences) for each day.
3. This part calculates the 7-day rolling sum of events (rolling_event_count) and the throughput per second over a 7-day window.



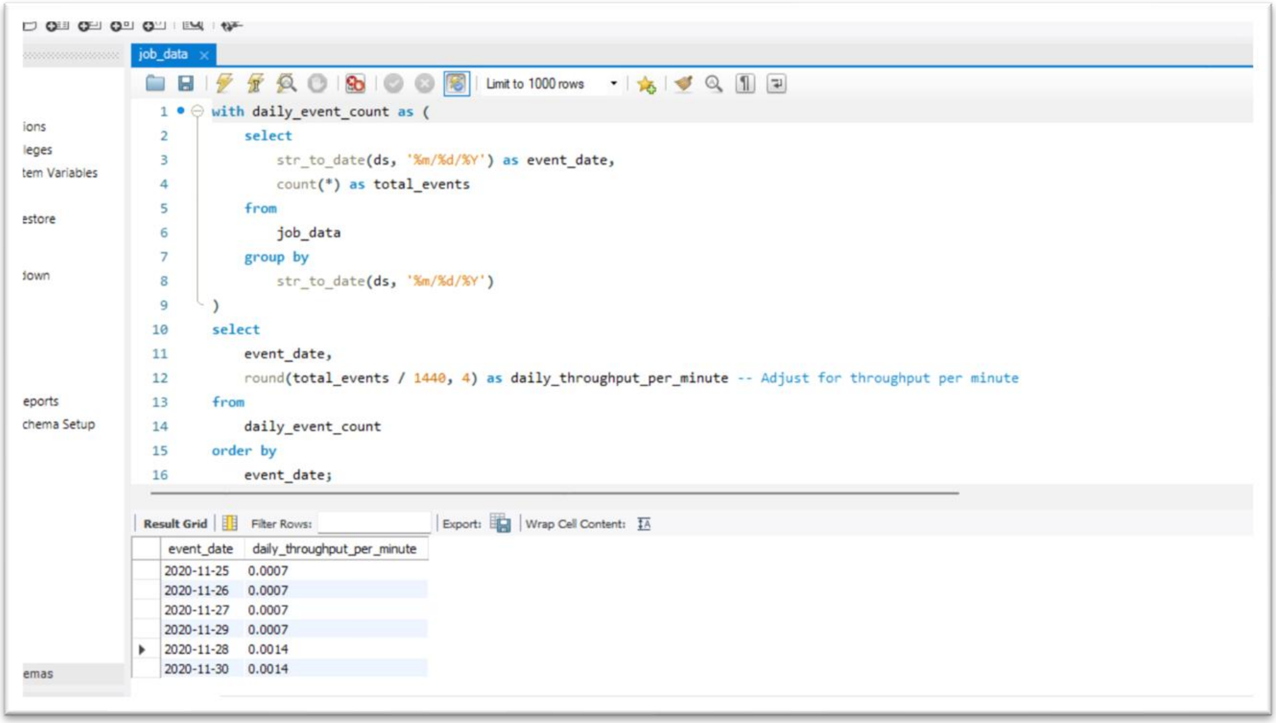
```
rolling_average as (
    select
        a.event_date,
        sum(b.total_events) as rolling_event_count,
        round(sum(b.total_events) * 1000 / 604800, 4) as rolling_throughput_milliseconds
    from
        daily_event_count a
    join
        daily_event_count b
    on
        b.event_date between date_add(a.event_date, interval -6 day) and a.event_date
    group by
        a.event_date
)
```

a.The query is comparing two sets of data: one for the current day (a) and one for the previous 6 days (b).

b.For each day in a, it looks at the total events that happened from 6 days before that day to the current day (this is the 7-day window).

Now to compare **7-day rolling average throughput** and **daily throughput**

We will perform the same task with daily throughput as well



```
with daily_event_count as (
    select
        str_to_date(ds, '%m/%d/%Y') as event_date,
        count(*) as total_events
    from
        job_data
    group by
        str_to_date(ds, '%m/%d/%Y')
)
select
    event_date,
    round(total_events / 1440, 4) as daily_throughput_per_minute -- Adjust for throughput per minute
from
    daily_event_count
order by
    event_date;
```

event_date	daily_throughput_per_minute
2020-11-25	0.0007
2020-11-26	0.0007
2020-11-27	0.0007
2020-11-29	0.0007
2020-11-28	0.0014
2020-11-30	0.0014

1. **Calculation**: For daily throughput, the total number of events for a given day is divided by the number of minutes in that day (1440 minutes).
2. **Result**: The daily throughput gives you the **events per minute for each day**, based only on that day's data.
3. For example:
   On 2020-11-25, if there were 1 event, the calculation would be:
   Daily Throughput=> 1/1440=0.0007 events per minute
   This calculation is repeated for each day independently.

This was for daily avg.

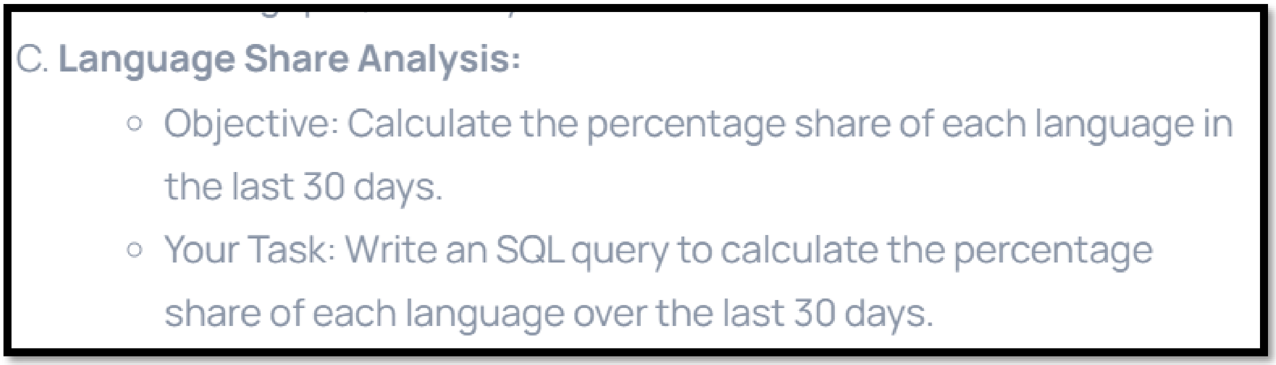
On **2020-11-30**, the 7-day rolling average will include the total events from **2020-11-24** to **2020-11-30**, and then the throughput will be calculated as:

Rolling Throughput= Total Events from Nov 24 to Nov 30/ 604800
seconds in 7 days

This results in a smoother, averaged value because it includes more data points.

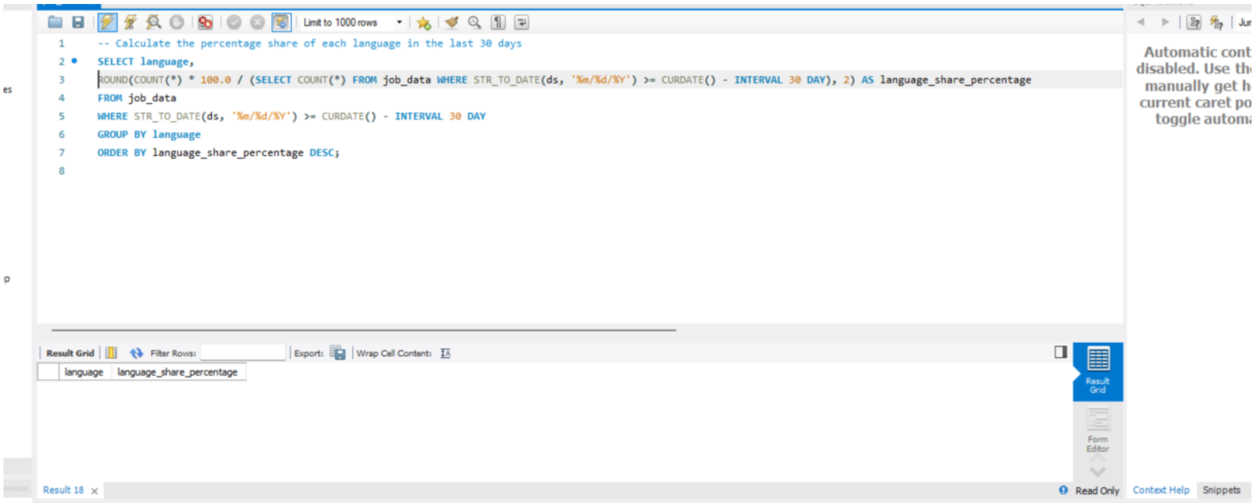
Conclusion-

1. **The key difference is that the daily throughput is a snapshot of a single day's events, while the 7-day rolling average throughput averages out fluctuations over a 7-day period to show trends**
2. **For long-term trend analysis I would choose 7-day rolling average throughput because it smooths out fluctuations and gives a clearer understanding of the trend.**
3. **For real-time analysis or detecting specific issues on a particular day, daily throughput would be more suitable.**
4. **We have only >10 entries in the given dataset, I would recommend using the daily throughput metric. Because-**
5. **With >10 entries, a 7-day rolling average would not provide meaningful insights because the rolling average requires a larger dataset to smooth out fluctuations and show trends effectively.**
6. **With a small dataset, simplicity is key. The daily throughput metric will be easy to interpret and will directly answer any questions**



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.

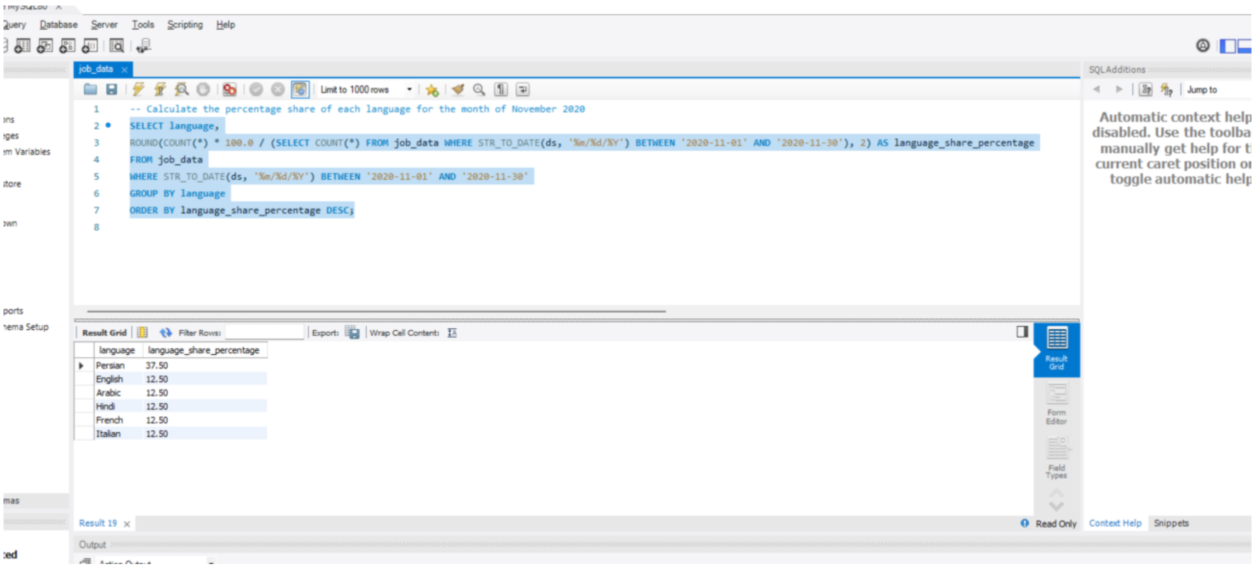


```
-- Calculate the percentage share of each language in the last 30 days
SELECT language,
ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM job_data WHERE STR_TO_DATE(ds, '%m/%d/%Y') >= CURDATE() - INTERVAL 30 DAY), 2) AS language_share_percentage
FROM job_data
WHERE STR_TO_DATE(ds, '%m/%d/%Y') >= CURDATE() - INTERVAL 30 DAY
GROUP BY language
ORDER BY language_share_percentage DESC;
```

language	language_share_percentage

When we did the calculation over the last 30 days ,as we can see the query has not returned any rows that’s because its calculated for the last 30 days over the current timestamp.

To calculate it for the month of November ----



```
-- Calculate the percentage share of each language for the month of November 2020
SELECT language,
ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM job_data WHERE STR_TO_DATE(ds, '%m/%d/%Y') BETWEEN '2020-11-01' AND '2020-11-30'), 2) AS language_share_percentage
FROM job_data
WHERE STR_TO_DATE(ds, '%m/%d/%Y') BETWEEN '2020-11-01' AND '2020-11-30'
GROUP BY language
ORDER BY language_share_percentage DESC;
```

language	language_share_percentage
Persian	37.50
English	12.50
Arabic	12.50
Hindi	12.50
French	12.50
Italian	12.50

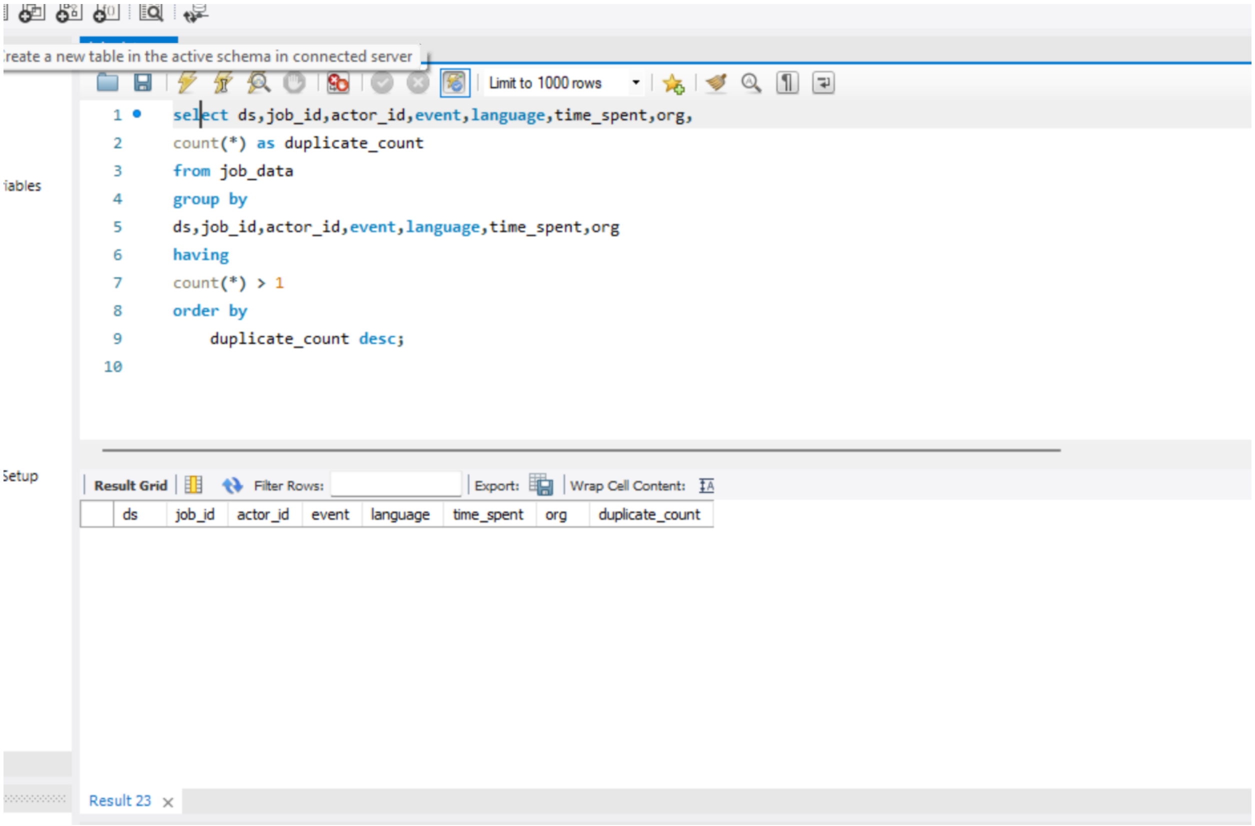
Query Database Server Tools Scripting Help
job_data
Limit to 1000 rows
SQLAdditions
Jump to
Automatic context help disabled. Use the toolba manually get help for t current caret position o toggle automatic help
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
Form Editor
Field Types
Result 19
Read Only
Context Help
Snippets
Output

This query is more designed for the given dataset ,which is of nov 2020

This query returns the lang percentage of each language.

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 ds,job_id,actor_id,event,language,time_spent,org,
count(*) as duplicate_count
from job_data
group by
ds,job_id,actor_id,event,language,time_spent,org
having
count(*) > 1
order by
    duplicate_count desc;
```

ds	job_id	actor_id	event	language	time_spent	org	duplicate_count

1. Group rows by the combination of columns: ds, job_id, actor_id, event, language, time_spent, and org.
2. Count the number of occurrences of each combination of values in these columns.
3. **count(*) > 1** filters out all groups that only have one occurrence (i.e., non-duplicates).
4. Filter out groups that have only one occurrence (i.e., non-duplicates).
5. Display the duplicate rows, sorted by how many times they appear in the dataset.

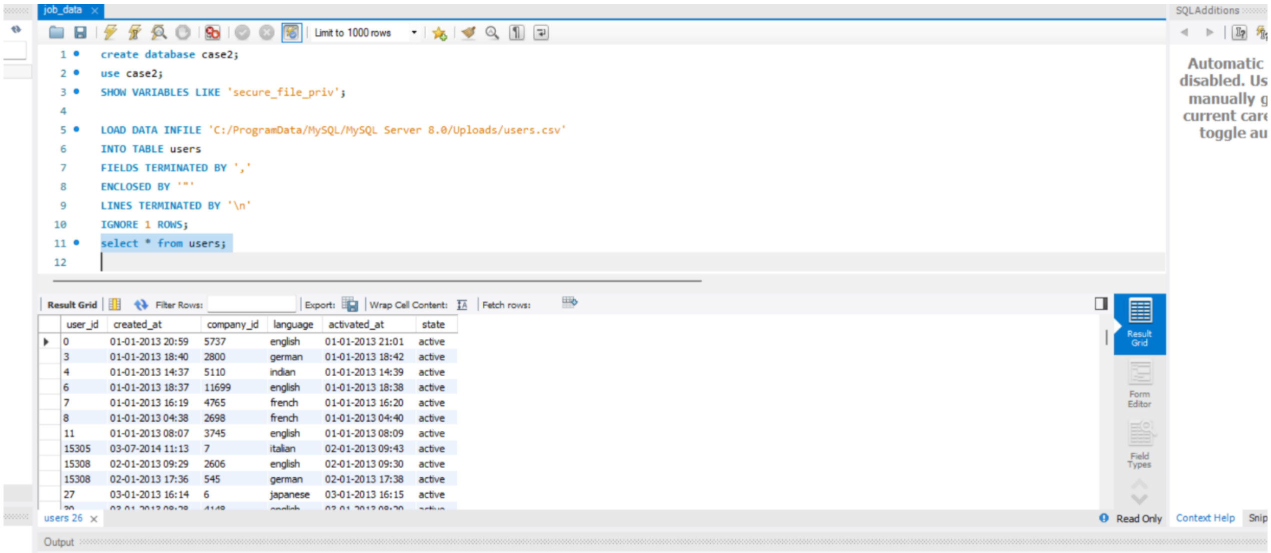
Case Study 2: Investigating Metric Spike

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.

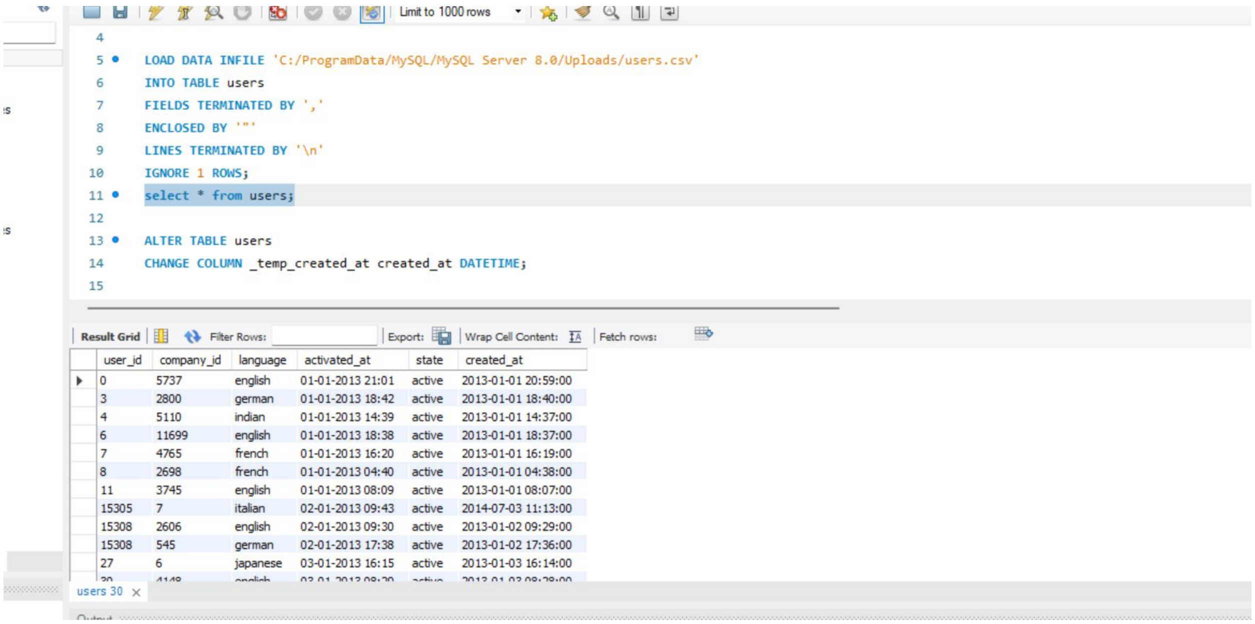
To move to the question part we will import the data to workbench as per instyructed in the guide video—

Screenshots for the same-



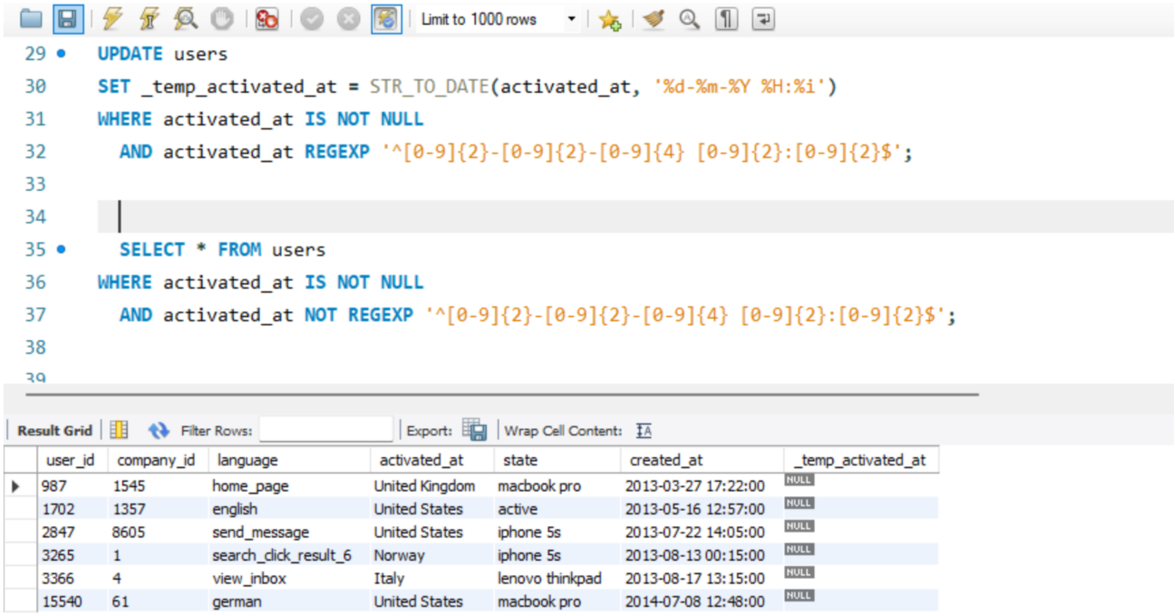
```
create database case2;
use case2;
SHOW VARIABLES LIKE 'secure_file_priv';

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users.csv'
INTO TABLE users
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
select * from users;
```

user_id	created_at	company_id	language	activated_at	state
0	01-01-2013 20:59	5737	english	01-01-2013 21:01	active
3	01-01-2013 18:40	2800	german	01-01-2013 18:42	active
4	01-01-2013 14:37	5110	indian	01-01-2013 14:39	active
6	01-01-2013 18:37	11699	english	01-01-2013 18:38	active
7	01-01-2013 16:19	4765	french	01-01-2013 16:20	active
8	01-01-2013 04:38	2698	french	01-01-2013 04:40	active
11	01-01-2013 08:07	3745	english	01-01-2013 08:09	active
15305	03-07-2014 11:13	7	italian	02-01-2013 09:43	active
15308	02-01-2013 09:29	2606	english	02-01-2013 09:30	active
15308	02-01-2013 17:36	545	german	02-01-2013 17:38	active
27	03-01-2013 16:14	6	japanese	03-01-2013 16:15	active
30	03-01-2013 08:28	4148	english	03-01-2013 08:30	active



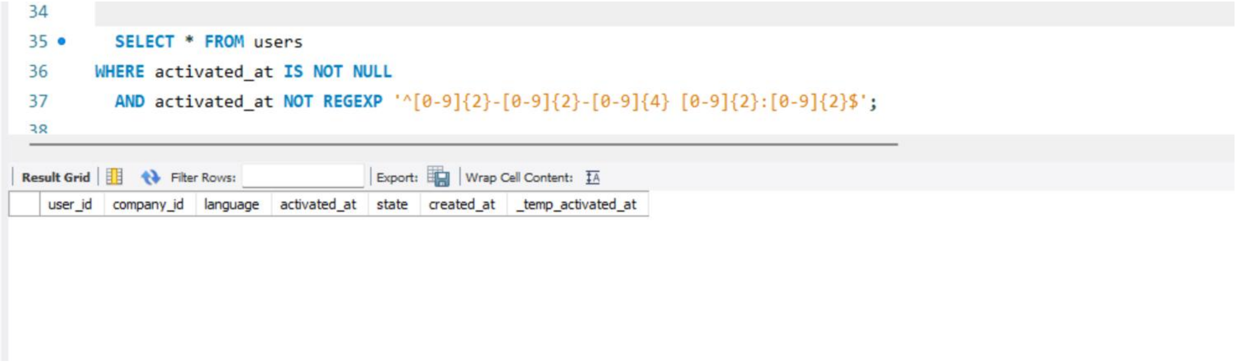
```sql
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users.csv'
INTO TABLE users
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
select * from users;

ALTER TABLE users
CHANGE COLUMN _temp_created_at created_at DATETIME;
```

user_id	company_id	language	activated_at	state	created_at
0	5737	english	01-01-2013 21:01	active	2013-01-01 20:59:00
3	2800	german	01-01-2013 18:42	active	2013-01-01 18:40:00
4	5110	indian	01-01-2013 14:39	active	2013-01-01 14:37:00
6	11699	english	01-01-2013 18:38	active	2013-01-01 18:37:00
7	4765	french	01-01-2013 16:20	active	2013-01-01 16:19:00
8	2698	french	01-01-2013 04:40	active	2013-01-01 04:38:00
11	3745	english	01-01-2013 08:09	active	2013-01-01 08:07:00
15305	7	italian	02-01-2013 09:43	active	2014-07-03 11:13:00
15308	2606	english	02-01-2013 09:30	active	2013-01-02 09:29:00
15308	545	german	02-01-2013 17:38	active	2013-01-02 17:36:00
27	6	japanese	03-01-2013 16:15	active	2013-01-03 16:14:00



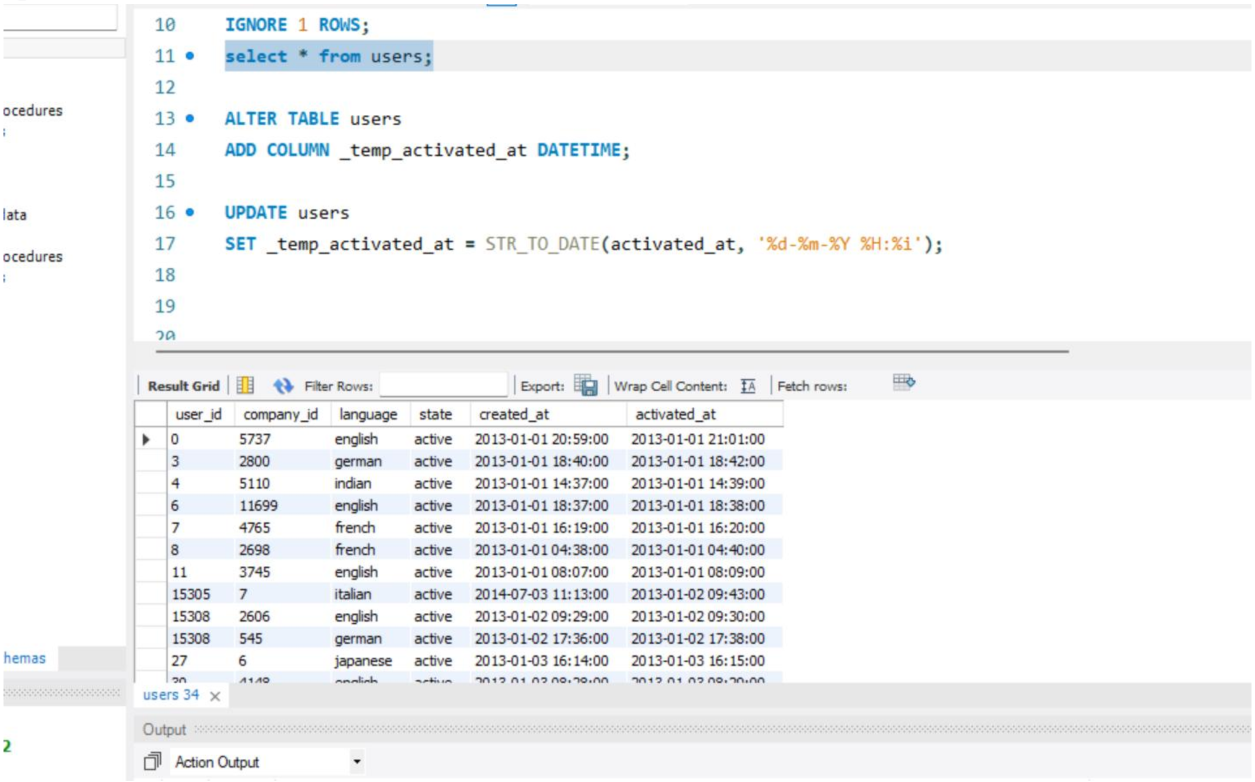
```sql
UPDATE users
SET _temp_activated_at = STR_TO_DATE(activated_at, '%d-%m-%Y %H:%i')
WHERE activated_at IS NOT NULL
  AND activated_at REGEXP '^[0-9]{2}-[0-9]{2}-[0-9]{4} [0-9]{2}:[0-9]{2}$';


  SELECT * FROM users
WHERE activated_at IS NOT NULL
  AND activated_at NOT REGEXP '^[0-9]{2}-[0-9]{2}-[0-9]{4} [0-9]{2}:[0-9]{2}$';
```

user_id	company_id	language	activated_at	state	created_at	_temp_activated_at
987	1545	home_page	United Kingdom	macbook pro	2013-03-27 17:22:00	NULL
1702	1357	english	United States	active	2013-05-16 12:57:00	NULL
2847	8605	send_message	United States	iphone 5s	2013-07-22 14:05:00	NULL
3265	1	search_click_result_6	Norway	iphone 5s	2013-08-13 00:15:00	NULL
3366	4	view_inbox	Italy	lenovo thinkpad	2013-08-17 13:15:00	NULL
15540	61	german	United States	macbook pro	2014-07-08 12:48:00	NULL



```sql
  SELECT * FROM users
WHERE activated_at IS NOT NULL
  AND activated_at NOT REGEXP '^[0-9]{2}-[0-9]{2}-[0-9]{4} [0-9]{2}:[0-9]{2}$';
```

	user_id	company_id	language	activated_at	state	created_at	_temp_activated_at



```sql
IGNORE 1 ROWS;
select * from users;

ALTER TABLE users
ADD COLUMN _temp_activated_at DATETIME;

UPDATE users
SET _temp_activated_at = STR_TO_DATE(activated_at, '%d-%m-%Y %H:%i');
```

user_id	company_id	language	state	created_at	activated_at
0	5737	english	active	2013-01-01 20:59:00	2013-01-01 21:01:00
3	2800	german	active	2013-01-01 18:40:00	2013-01-01 18:42:00
4	5110	indian	active	2013-01-01 14:37:00	2013-01-01 14:39:00
6	11699	english	active	2013-01-01 18:37:00	2013-01-01 18:38:00
7	4765	french	active	2013-01-01 16:19:00	2013-01-01 16:20:00
8	2698	french	active	2013-01-01 04:38:00	2013-01-01 04:40:00
11	3745	english	active	2013-01-01 08:07:00	2013-01-01 08:09:00
15305	7	italian	active	2014-07-03 11:13:00	2013-01-02 09:43:00
15308	2606	english	active	2013-01-02 09:29:00	2013-01-02 09:30:00
15308	545	german	active	2013-01-02 17:36:00	2013-01-02 17:38:00
27	6	japanese	active	2013-01-03 16:14:00	2013-01-03 16:15:00

1. users table:

- user_id (INT) - Unique identifier for each user.
- created_at (DATETIME) - The date and time when the user was created (converted from VARCHAR).
- company_id (INT) - The ID of the company the user is associated with.
- language (VARCHAR(50)) - The language preference of the user.
- activated_at (DATETIME) - The date and time when the user was activated (converted from VARCHAR).
- state (VARCHAR(50)) - The state or status of the user.

2. email_events table:

- user_id (INT) - Unique identifier for each user (foreign key).
- occurred_at (DATETIME) - The date and time when the email event occurred (converted from VARCHAR).
- action (VARCHAR(50)) - The type of action performed (e.g., sent, opened, clicked).
- user_type (VARCHAR(50)) - The type of user (e.g., admin, regular).

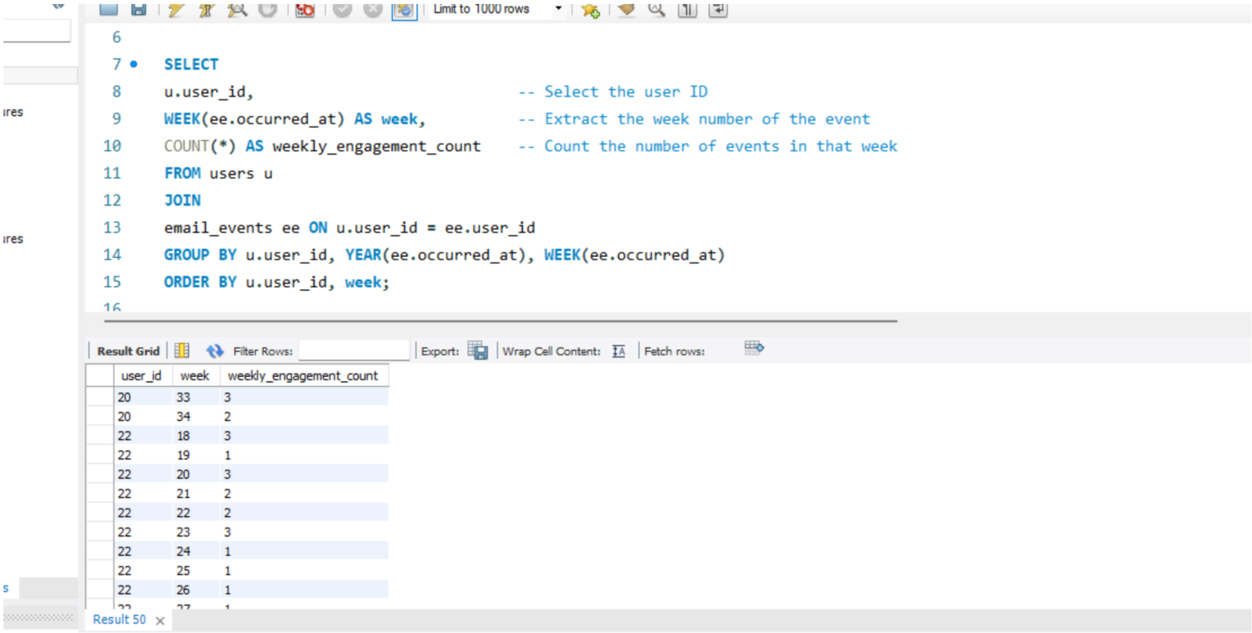
3. events table:

- user_id (INT) - Unique identifier for each user (foreign key).
- occurred_at (DATETIME) - The date and time when the event occurred (converted from VARCHAR).
- event_type (VARCHAR(50)) - The type of the event (e.g., login, purchase).
- event_name (VARCHAR(100)) - The specific name of the event.
- location (VARCHAR(100)) - The location where the event occurred.
- device (VARCHAR(50)) - The device used during the event.
- user_type (VARCHAR(50)) - The type of user performing the event.

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.

We are asked to calculate the weekly user engagement. This means we need to determine how active each user was on a weekly basis, based on the data available.



```sql

SELECT
u.user_id,                          -- Select the user ID
WEEK(ee.occurred_at) AS week,       -- Extract the week number of the event
COUNT(*) AS weekly_engagement_count -- Count the number of events in that week
FROM users u
JOIN
email_events ee ON u.user_id = ee.user_id
GROUP BY u.user_id, YEAR(ee.occurred_at), WEEK(ee.occurred_at)
ORDER BY u.user_id, week;
```

Result Grid

user_id	week	weekly_engagement_count
20	33	3
20	34	2
22	18	3
22	19	1
22	20	3
22	21	2
22	22	2
22	23	3
22	24	1
22	25	1
22	26	1
22	27	1

For this-

- We need to join the users table and the email_events table to combine user information with their respective events.
- Extract the year and week from the occurred_at timestamp in email_events (because we want the engagement per week).
- Count how many events the user performed in each week.

- Finally, display the result for each user, by week.

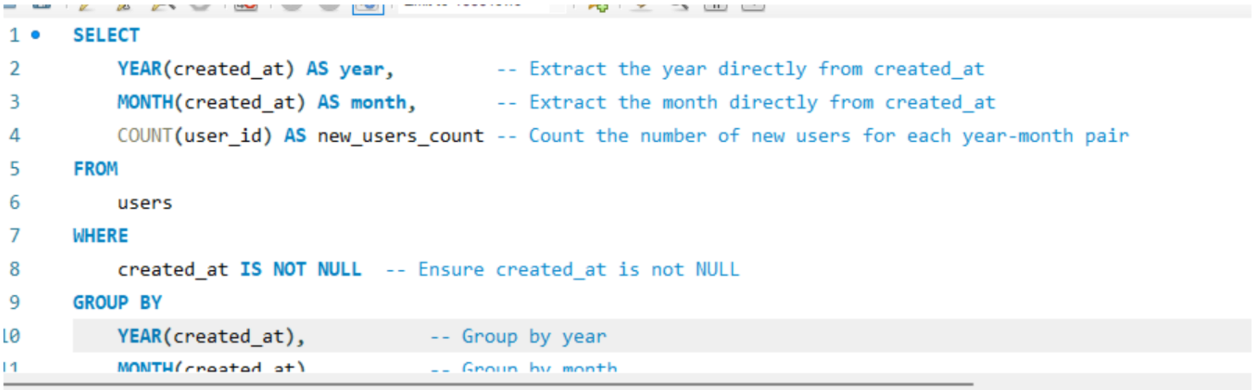
JOIN email_events ee ON u.user_id = ee.user_id: This combines the users table and email_events table by matching the user_id column from both tables.

The query gives you the **weekly engagement count** for each user, based on the activities they performed during the week.

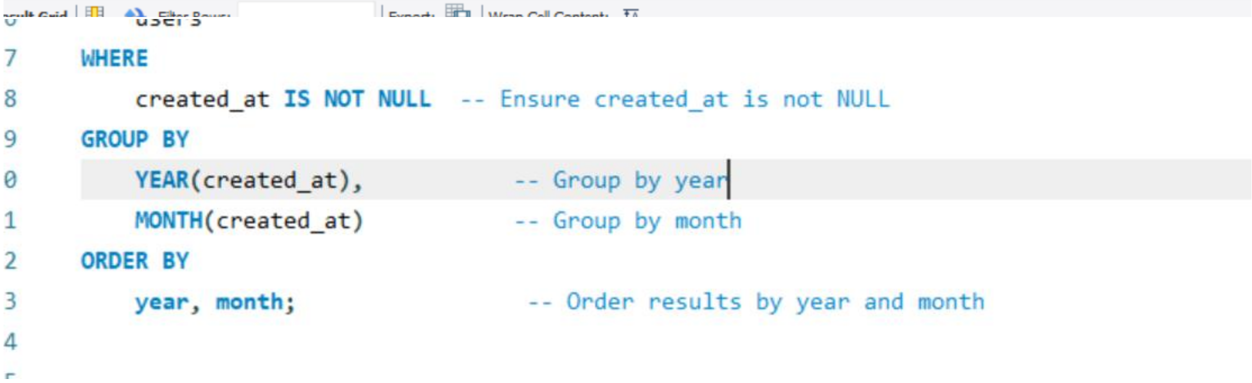
B. **User Growth Analysis:**

- 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.

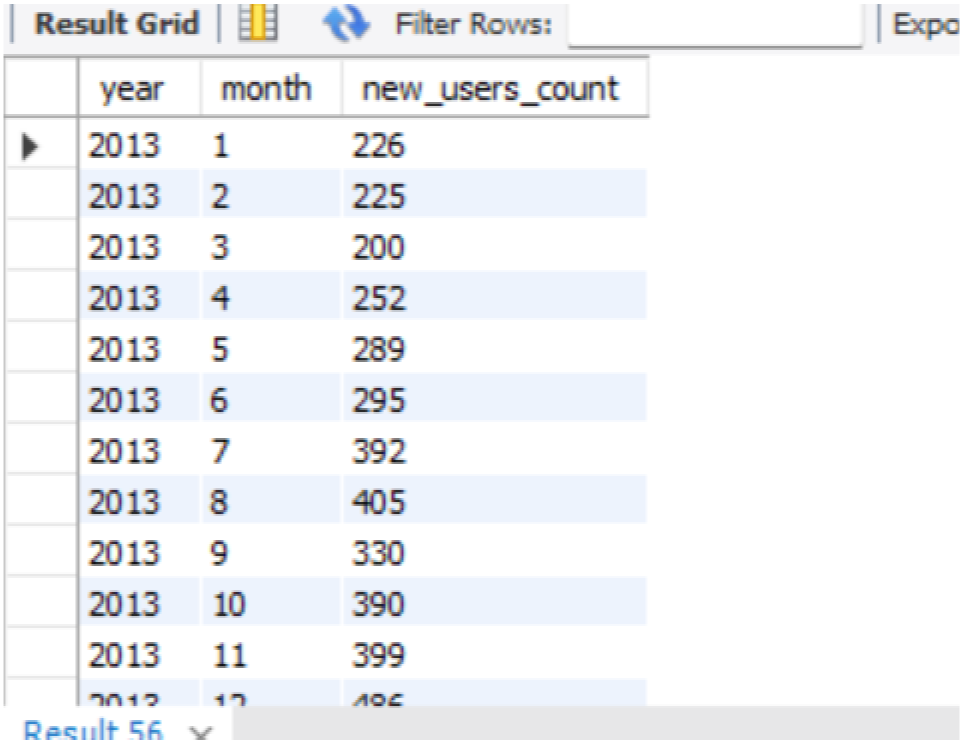
To calculate user growth over time for a product, we need to analyze how the number of users has increased over time (such as by day, week, or month). For this task, we can track how many users were created over a period of time and see how the user base has grown.’



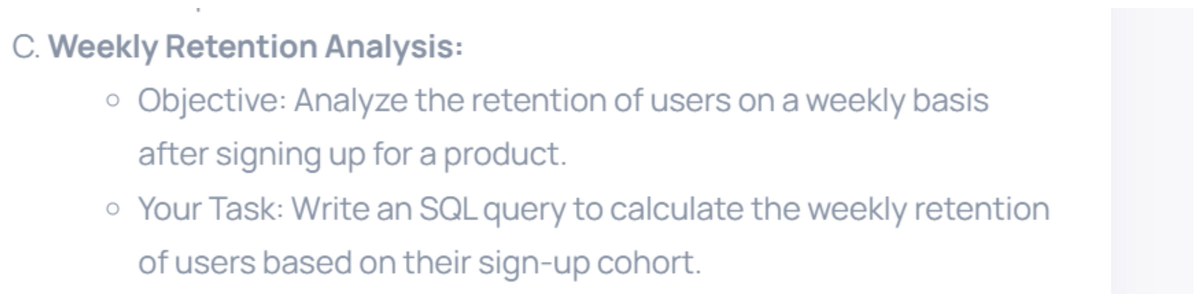
```sql
SELECT
    YEAR(created_at) AS year,          -- Extract the year directly from created_at
    MONTH(created_at) AS month,        -- Extract the month directly from created_at
    COUNT(user_id) AS new_users_count -- Count the number of new users for each year-month pair
FROM
    users
WHERE
    created_at IS NOT NULL  -- Ensure created_at is not NULL
GROUP BY
    YEAR(created_at),             -- Group by year
    MONTH(created_at)             -- Group by month
```



```sql
    users
WHERE
    created_at IS NOT NULL  -- Ensure created_at is not NULL
GROUP BY
    YEAR(created_at),             -- Group by year
    MONTH(created_at)             -- Group by month
ORDER BY
    year, month;                   -- Order results by year and month
```



year	month	new_users_count
2013	1	226
2013	2	225
2013	3	200
2013	4	252
2013	5	289
2013	6	295
2013	7	392
2013	8	405
2013	9	330
2013	10	390
2013	11	399
2013	12	486



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.

We need to calculate how many users return to the product each week after they first sign up.

STEPS-

Each user has a **sign-up date**, which is when they first created an account. This is the **start point** for measuring their retention.

Retention refers to how many of the users who signed up in a particular week **return** in the subsequent weeks. For example:

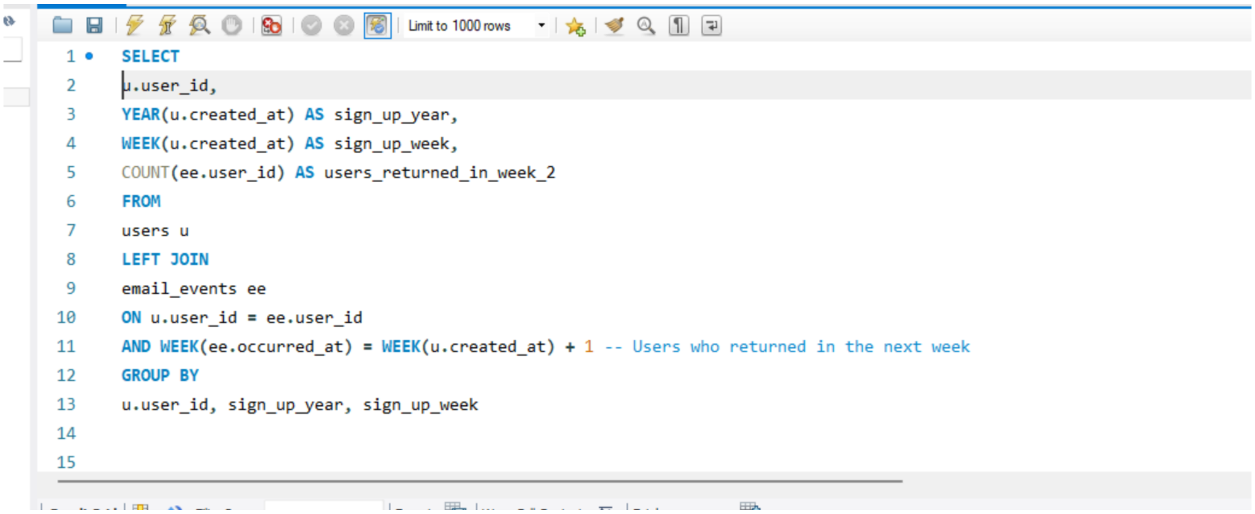
- If 100 users signed up in the first week of January, and 60 of them returned in the second week, the retention for that group of users in the second week is 60%.

We need to measure how many users who signed up in **each week** are still active in the **following weeks**.

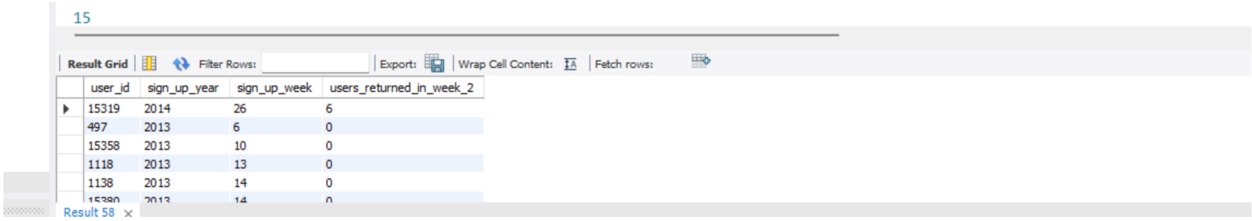
For example:

- **Week 1**: 100 users signed up.
- **Week 2**: 60 of those 100 users returned.
- **Week 3**: 50 of the 100 users returned.

1. **Identify when users signed up (the "sign-up date").**
2. **Track if they came back in subsequent weeks after their sign-up.**
3. **Count how many users are active each week after they signed up.**



```
SELECT
u.user_id,
YEAR(u.created_at) AS sign_up_year,
WEEK(u.created_at) AS sign_up_week,
COUNT(ee.user_id) AS users_returned_in_week_2
FROM
users u
LEFT JOIN
email_events ee
ON u.user_id = ee.user_id
AND WEEK(ee.occurred_at) = WEEK(u.created_at) + 1 -- Users who returned in the next week
GROUP BY
u.user_id, sign_up_year, sign_up_week
```

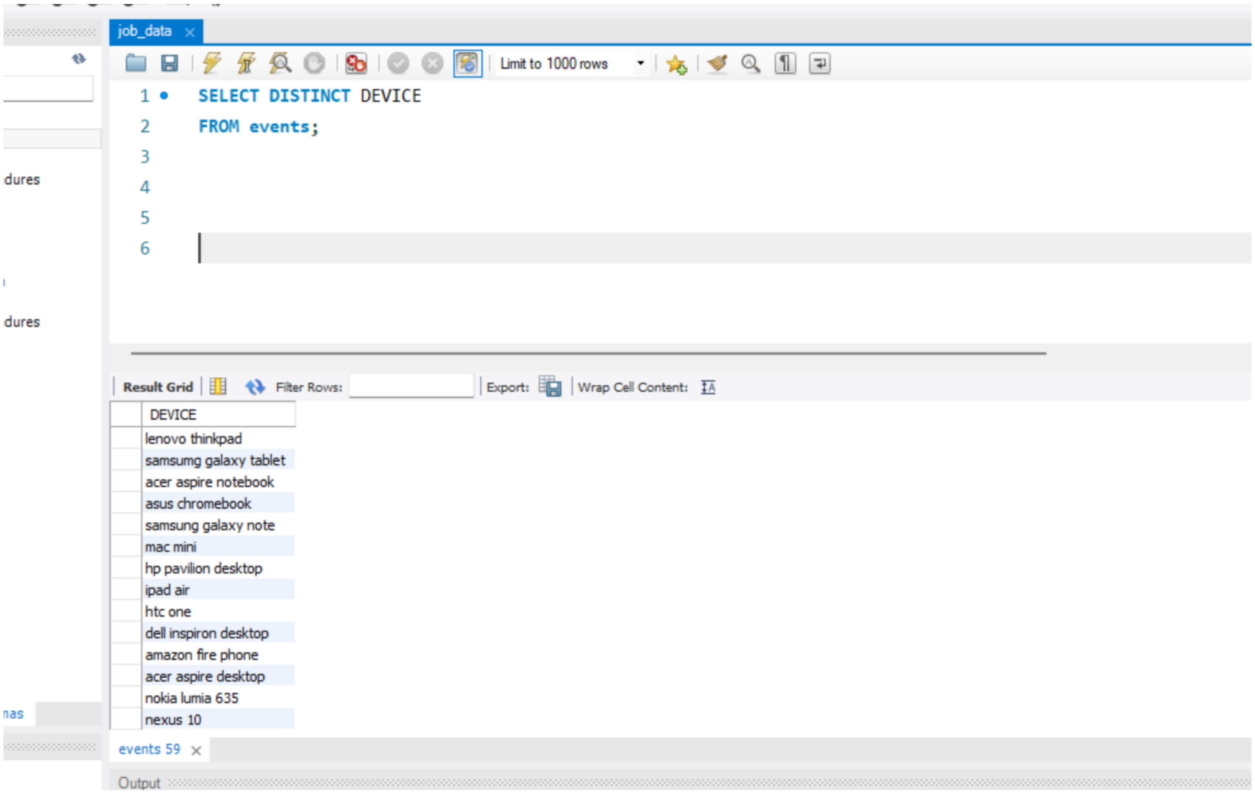


user_id	sign_up_year	sign_up_week	users_returned_in_week_2
15319	2014	26	6
497	2013	6	0
15358	2013	10	0
1118	2013	13	0
1138	2013	14	0
15380	2013	14	0

- YEAR(u.created_at) and WEEK(u.created_at): We extract the year and week from the user's sign-up date.
- We join the email_events table to check if the user was active in the subsequent weeks.
- AND WEEK(ee.occurred_at) = WEEK(u.created_at) + 1: We focus on users who signed up in a specific week and check if they were active in the next week (i.e., Week 2).

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.

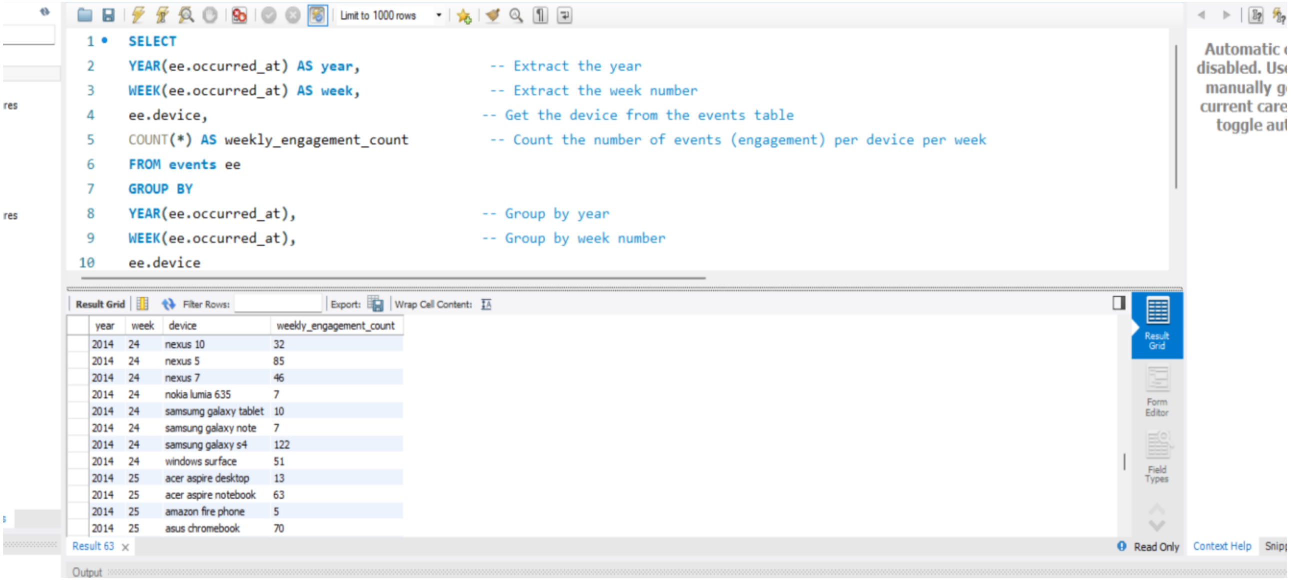


```
SELECT DISTINCT DEVICE
FROM events;
```

DEVICE
lenovo thinkpad
samsumg galaxy tablet
acer aspire notebook
asus chromebook
samsung galaxy note
mac mini
hp pavilion desktop
ipad air
htc one
dell inspiron desktop
amazon fire phone
acer aspire desktop
nokia lumia 635
nexus 10

We use YEAR() in the query to extract the year from the occurred_at timestamp to ensure that the week number (WEEK()) is correctly mapped within the context of a specific year.

1. Week 1 of 2023 and Week 1 of 2024 will both have the same week number (1), but they belong to different years.
2. By using YEAR(), we make sure that the engagement data is grouped and analyzed correctly for each year.

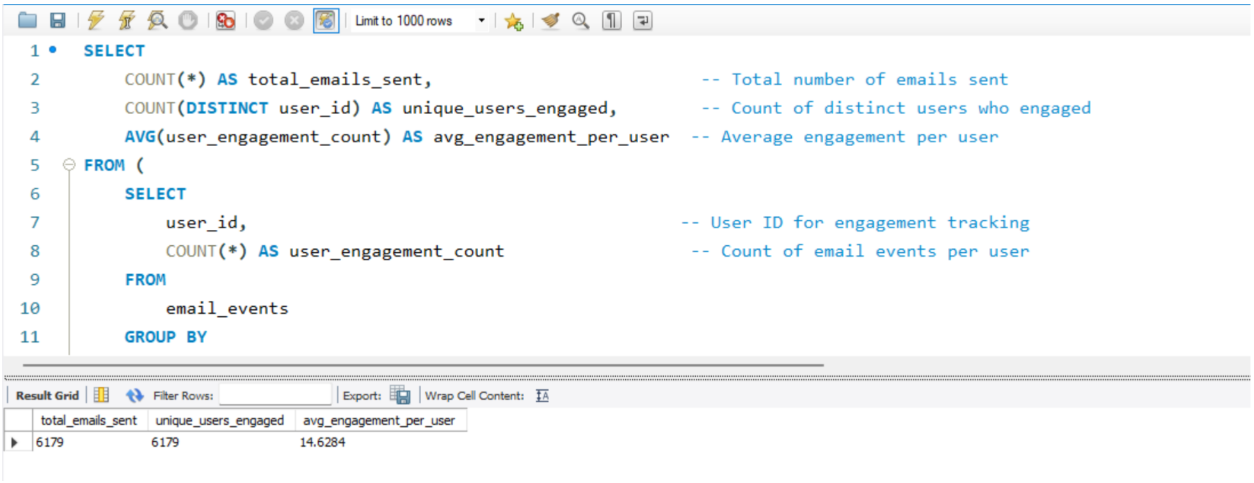


```
SELECT
YEAR(ee.occurred_at) AS year,             -- Extract the year
WEEK(ee.occurred_at) AS week,             -- Extract the week number
ee.device,                               -- Get the device from the events table
COUNT(*) AS weekly_engagement_count       -- Count the number of events (engagement) per device per week
FROM events ee
GROUP BY
YEAR(ee.occurred_at),                    -- Group by year
WEEK(ee.occurred_at),                    -- Group by week number
ee.device
```

year	week	device	weekly_engagement_count
2014	24	nexus 10	32
2014	24	nexus 5	85
2014	24	nexus 7	46
2014	24	nokia lumia 635	7
2014	24	samsumg galaxy tablet	10
2014	24	samsung galaxy note	7
2014	24	samsung galaxy s4	122
2014	24	windows surface	51
2014	25	acer aspire desktop	13
2014	25	acer aspire notebook	63
2014	25	amazon fire phone	5
2014	25	asus chromebook	70

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.



```sql
SELECT
    COUNT(*) AS total_emails_sent,                          -- Total number of emails sent
    COUNT(DISTINCT user_id) AS unique_users_engaged,        -- Count of distinct users who engaged
    AVG(user_engagement_count) AS avg_engagement_per_user  -- Average engagement per user
FROM (
    SELECT
        user_id,                                          -- User ID for engagement tracking
        COUNT(*) AS user_engagement_count                  -- Count of email events per user
    FROM
        email_events
    GROUP BY
```

total_emails_sent	unique_users_engaged	avg_engagement_per_user
6179	6179	14.6284