**TRAINITY PROJECT 3: DATA ANALYTICS PROCESS**

##### TOPIC: Operation Analytics and Investigating Metric Spike

**PROJECT 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, one needs to 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, one needs to answer these questions daily, making it crucial to understand how to investigate these metric spikes. In this project, with various datasets and tables, task is to derive insights from this data to answer questions posed by different departments within the company. Here advanced SQL skills is used to analyze the data and provide valuable insights that can help improve the company's operations and understand sudden changes in key metrics.

**APPROACH:**

##### Create a Database: The database file was provided in the attachments. The necessay tables were created using the provided table structures and links. The csv file was imported into MySQL Workbench.

**TECH- STACK USED:**

The software I am using to do the analysis is MySQL Workbench, Version: 8.0.34 (MySQL Community Server – GPL) and Microsoft Excel to prepare data. I used these two softwares for the project as it is easy to use and freely available.

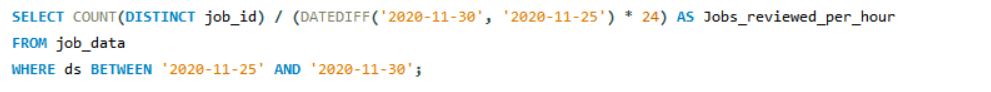
**INSIGHTS:**

###### **Case Study 1: Job Data Analysis**

###### **A table named job\_data with the following columns was created:**

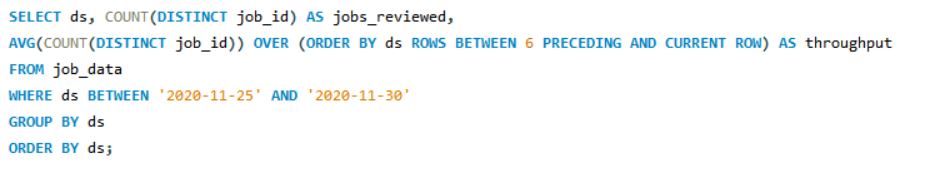
* **job\_id:**Unique identifier of jobs
* **actor\_id:**Unique identifier of actor
* **event:**The type of event (decision/skip/transfer).
* **language:**The Language of the content
* **time\_spent:**Time spent to review the job in seconds.
* **org:**The Organization of the actor
* **ds:**The date in the format yyyy/mm/dd (stored as text).

**Tasks:**

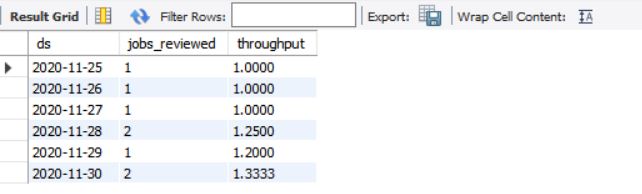
1. **Jobs Reviewed Over Time:**
   * Objective: Calculate the number of jobs reviewed per hour for each day in November 2020.
   * Task: Write an SQL query to calculate the number of jobs reviewed per hour for each day in November 2020.

The SQL Query calculates the number of jobs reviewed per day per hour from date 25-11-2020 to 30-11-2020 from the table job\_data.

Only 0.5 jobs were reviewed per day per hour from date 25-11-2020 to 30-11-2020

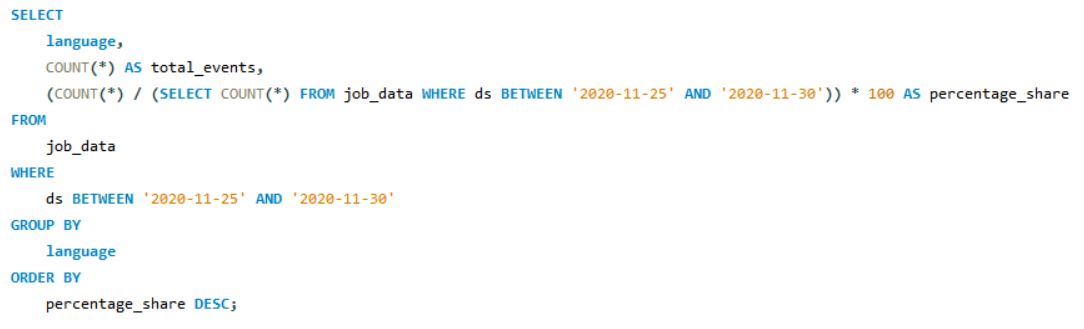
1. **Throughput Analysis:**
   * Objective: Calculate the 7-day rolling average of throughput (number of events per second).
   * Task: Write an SQL query to calculate the 7-day rolling average of throughput. Additionally, explain whether you prefer using the daily metric or the 7-day rolling average for throughput, and why.

We directly calculate the count of distinct job IDs reviewed (**jobs\_reviewed**) and use it within the window function **AVG()** to calculate the 7-day rolling average. We utilize **ORDER BY ds** to ensure the rows are ordered by date. The **ROWS BETWEEN 6 PRECEDING AND CURRENT ROW** clause specifies the window size as the current row plus the previous 6 rows, enabling the calculation of the 7-day rolling average. The **GROUP BY ds** groups the data by date. The **ORDER BY ds** orders the results by date.



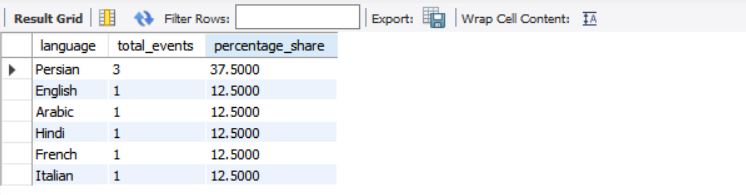
The 7-day rolling average of throughput is maximum on 30-11-2020.

Also preference either to use the daily metric or the 7-day rolling average depends on granularity needed. Daily metrics are for short-term insights; 7-day rolling average is for smoother, long-term trends. It helps to highlight longer-term trends and patterns while reducing the impact of daily variations.

1. **Language Share Analysis:**
   * Objective: Calculate the percentage share of each language in the last 30 days.
   * Task: Write an SQL query to calculate the percentage share of each language over the last 30 days.

We use COUNT (\*) to calculate the total number of events for each language, which is consistent throughout the query. The subquery (SELECT COUNT(\*) FROM job\_data WHERE ds BETWEEN ‘2020-11-25’ AND ‘2020-11-30’)) \* 100 retrieves the total count of events which is then used to calculate the percentage share.

The percentage share calculation is accurate, as it divides the count of events for each language by the total count of events.



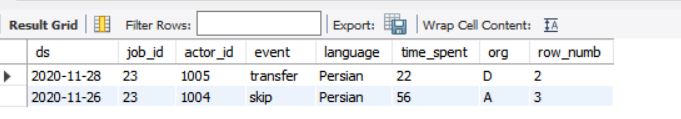
So, the percentage share of Persian is more as compared to other languages in the given time frame.

1. **Duplicate Rows Detection:**
   * Objective: Identify duplicate rows in the data.
   * Task: Write an SQL query to display duplicate rows from the job\_data table.

The WITH clause is used to define a Common Table Expression (CTE) named CTE. Inside the CTE, we select all columns (\*) from the job\_data table. We use the **ROW\_NUMBER()** window function combined with the OVER clause to assign a unique row number to each row within each partition defined by the job\_id column. The **PARTITION BY** job\_id clause partitions the rows of the result set into groups based on the job\_id column. This means that the row number is reset for each distinct job\_id. The assigned row number is aliased as **row\_numb** in the result set.

Selecting Duplicate Rows from the CTE:

In this part, we select all columns (\*) from the CTE named CTE. We filter the rows by specifying WHERE row\_numb > 1. This condition ensures that only rows with a row number greater than 1 are selected. Rows with a row number greater than 1 indicate duplicates because they have more than one occurrence within the partition defined by the job\_id.



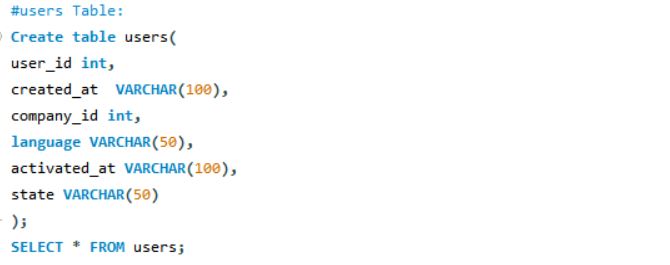
There were two duplicate rows of job\_id as depicted above in the executed query.

###### **Case Study 2: Investigating Metric Spike**

**There are three tables:**

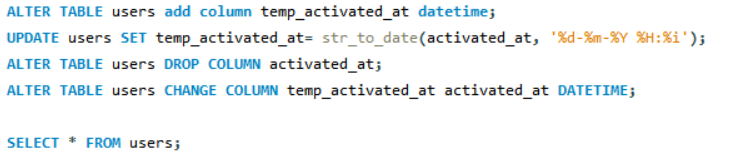
* **users**: Contains one row per user, with descriptive information about that user’s account.
* **events**: Contains one row per event, where an event is an action that a user has taken (e.g., login, messaging, search).
* **email\_events**: Contains events specific to the sending of emails.

Before moving on to do the analysis, we first load the data from excel to MySQL Workbench. Though data contains millions of records, importing through “Table Data Import Wizard” will take hours to import. For quick importing the data, we use “LOAD DATA INFILE” command which helps in loading large datasets within seconds.

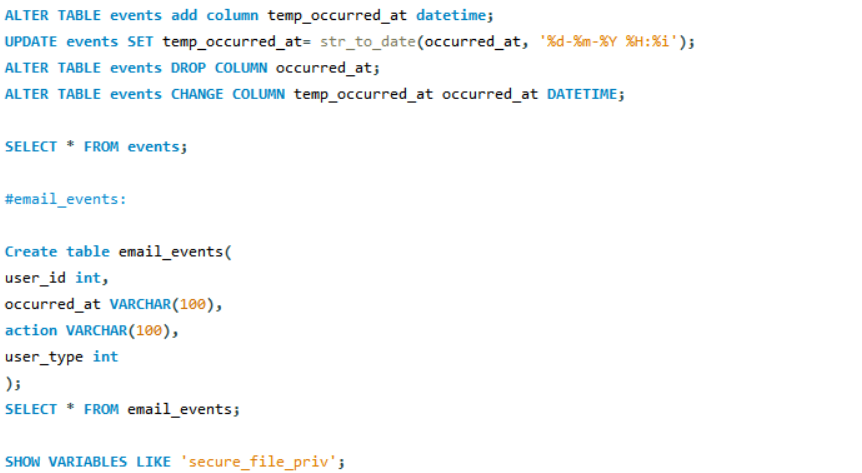
First we create a Table users:

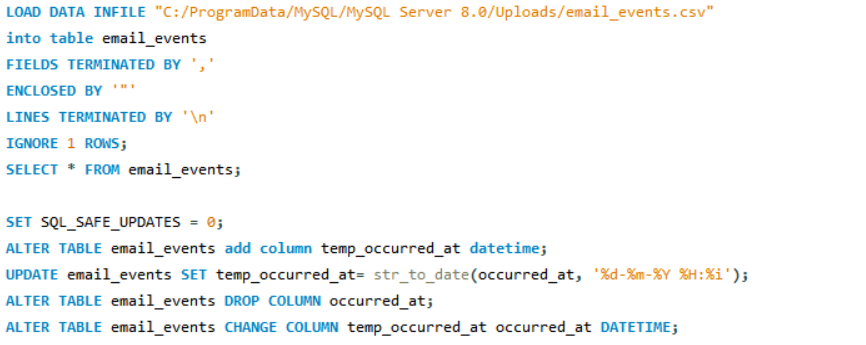
****Executing this command will show the path where the datasets that need to imported to be kept.

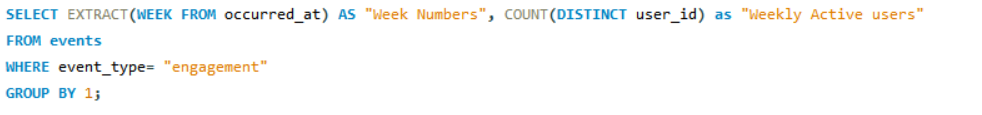
****

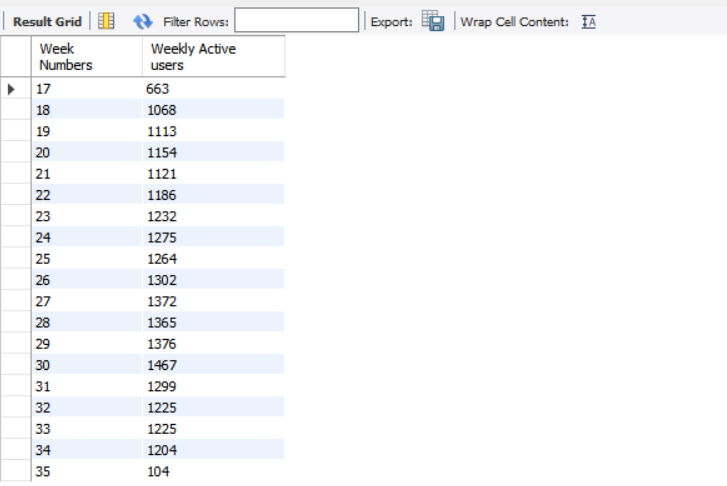
****In the dataset, the date column was in STRING format. For effctive execution of queries, we need to convert it into date type format using **str\_to\_date.**

Similarly, events table and email\_events table were created and data were imported.



**Tasks:**

1. **Weekly User Engagement:**
   * Objective: Measure the activeness of users on a weekly basis.
   * Task: Write an SQL query to calculate the weekly user engagement.

We use the EXTRACT (WEEK FROM occurred\_at) function to extract the week number from the occurred\_at column. The COUNT (DISTINCT user\_id) function calculates the number of distinct users for each week. We filter the events based on the event\_type being 'engagement'. Finally, we group the results by the week number. This query will give you the count of weekly active users based on the 'engagement' events.

1. **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.

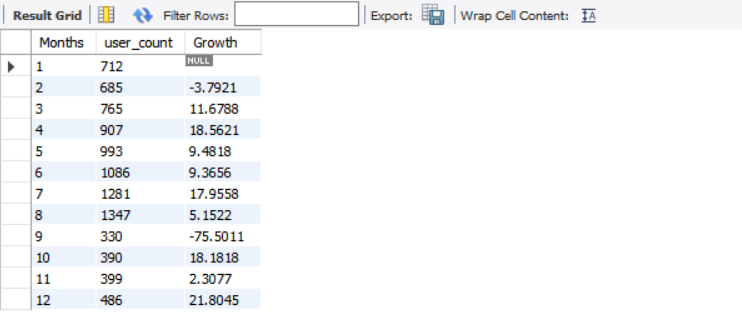
***Subquery to Calculate User Count by Month:***

This subquery retrieves data from the USERS table. It calculates the month component from the created\_at timestamp column using the **EXTRACT()** function and aliases it as Months. It counts the number of users **(COUNT(\*))** in each month and aliases it as **user\_count**. The WHERE clause filters out rows where the activated\_at column is not null, indicating activated users. Results are grouped by the Months column and ordered by Months.

***Main Query to Calculate User Growth:***

In the main query, we select columns from the subquery. We select Months and user\_count directly from the subquery. For calculating growth, we use the **LAG()** window function to retrieve the user count from the previous month. **LAG(user\_count, 1) OVER (ORDER BY Months)** retrieves the user count from the previous row, ordered by Months. We calculate **growth as ((user\_count / PreviousMonthUserCount) - 1) \* 100.**

The results are presented as Months, user\_count, and Growth.



1. **Weekly Retention Analysis:**
   * Objective: Analyze the retention of users on a weekly basis after signing up for a product.
   * Task: Write an SQL query to calculate the weekly retention of users based on their sign-up cohort.

**Subquery to Calculate Weekly Logins and First Week:**

This subquery selects the user\_id, login week, and the first week of login for each user.

In the first part of the subquery (A), it selects the user\_id and the week number of logins (LOGIN\_WEEK) from the events table using the EXTRACT(WEEK FROM OCCURRED\_AT) function. It groups the results by user\_id and LOGIN\_WEEK.

In the second part of the subquery (B), it selects the user\_id and calculates the minimum week number of login (FIRST) for each user. The subquery calculates the difference between the login week and the first week (LOGIN\_WEEK - FIRST) to determine the week number for each login event.

It then filters the results to include only the rows where the user\_id from subquery A matches the user\_id from subquery B.

**Main Query to Pivot Weekly Retention:**

The main query pivots the data to calculate weekly retention for each cohort.

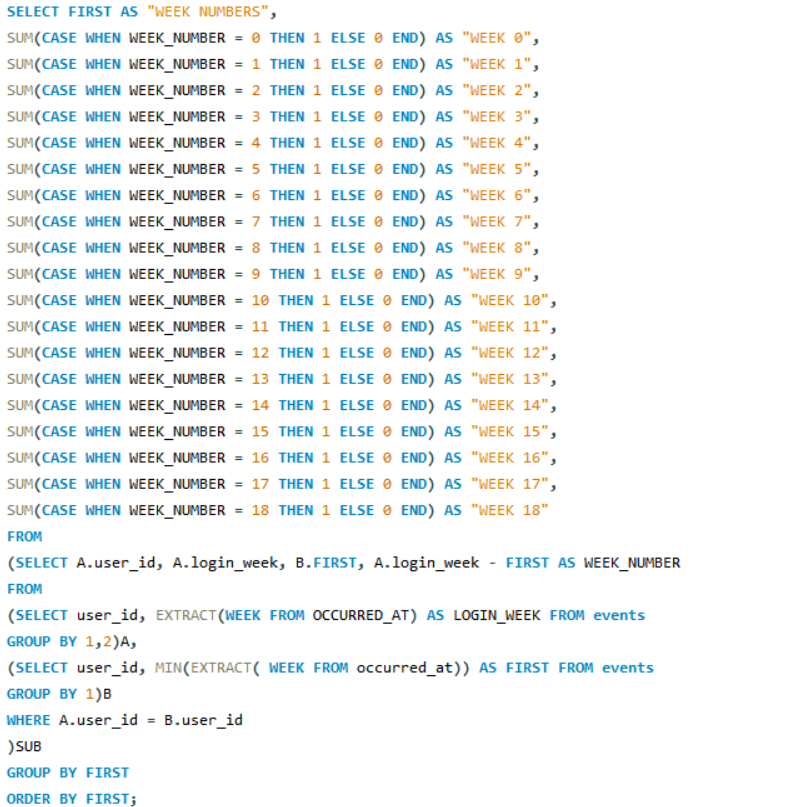
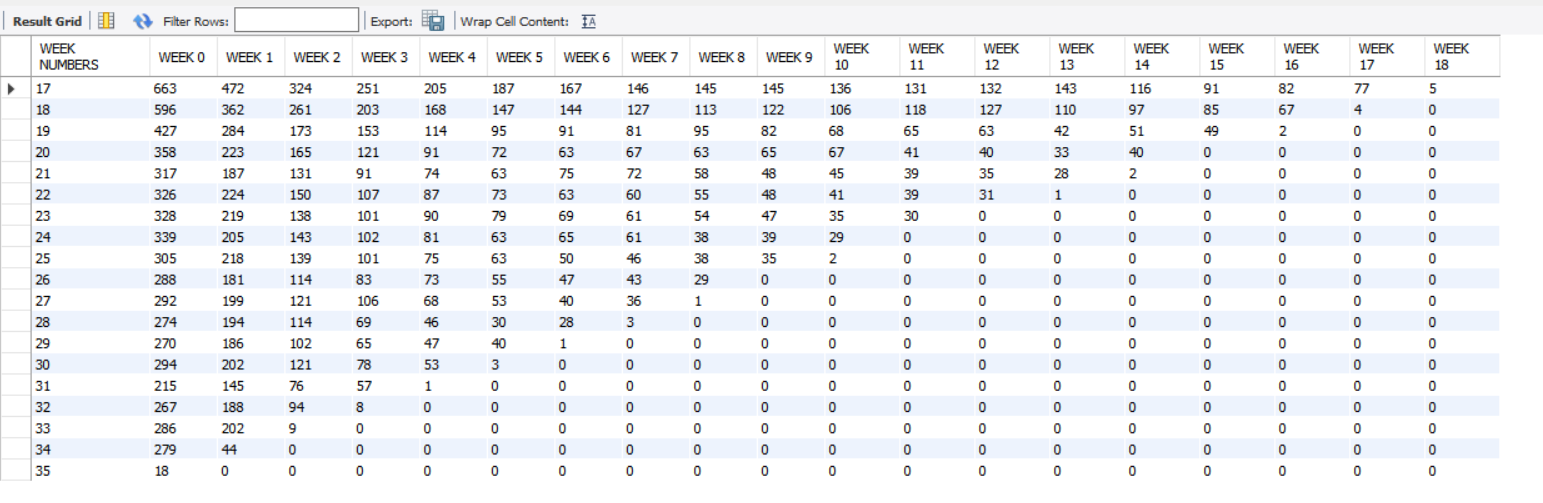
It uses conditional aggregation with SUM (CASE ...) statements to count the number of users who logged in during each week (WEEK 0, WEEK 1, ..., WEEK 18).

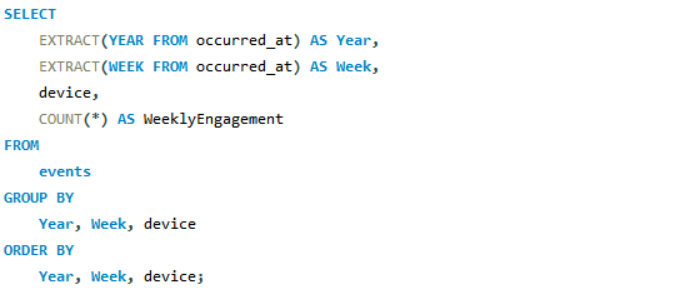
The CASE statements check if the week number equals the respective week (WEEK\_NUMBER = 0, WEEK\_NUMBER = 1, etc.), and if so, it adds 1 to the sum; otherwise, it adds 0.

The data is grouped by the first login week (FIRST), representing the user cohort.

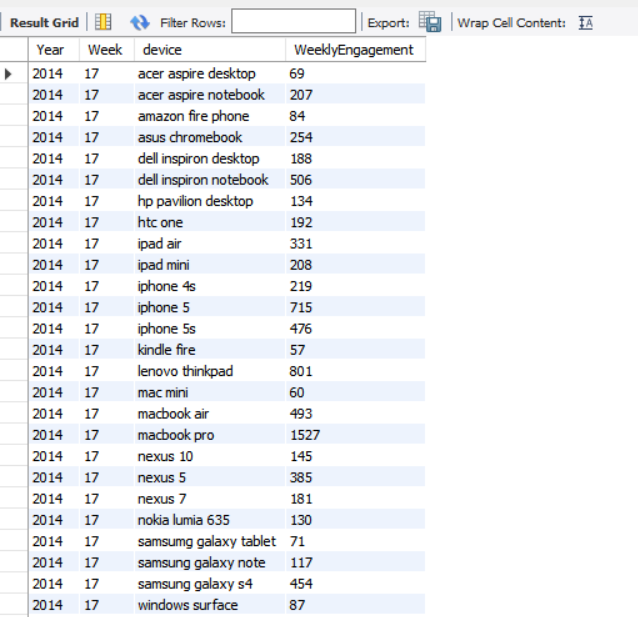
The results are ordered by the first login week.

This query effectively calculates the weekly retention of users based on their sign-up cohort. Adjustments may be needed based on your specific database schema and requirements.

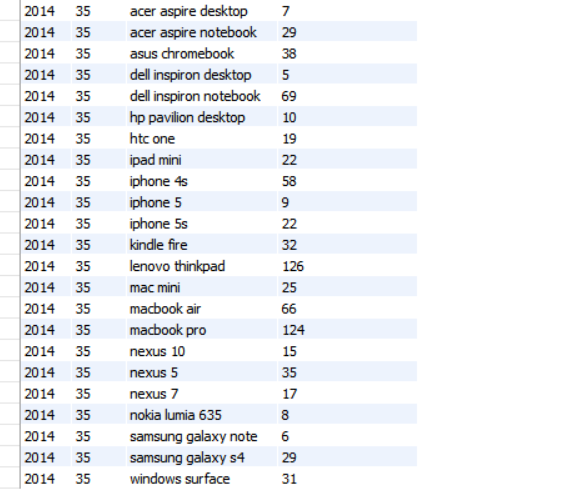


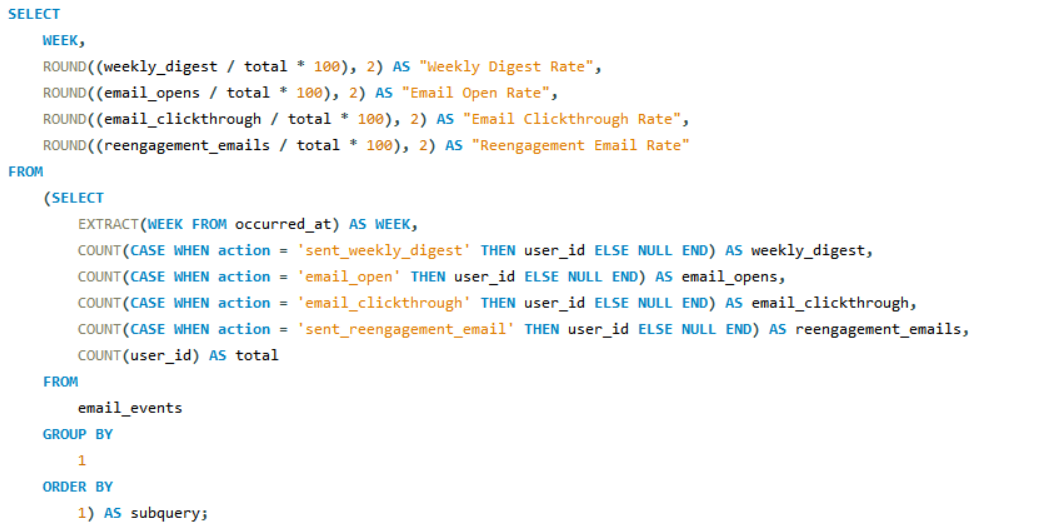
1. **Weekly Engagement Per Device:**
   * Objective: Measure the activeness of users on a weekly basis per device.
   * Task: Write an SQL query to calculate the weekly engagement per device.

* EXTRACT(YEAR FROM occurred\_at) AS Year: Extracts the year component from the occurred\_at timestamp column.
* EXTRACT(WEEK FROM occurred\_at) AS Week: Extracts the week number component from the occurred\_at timestamp column.
* device: Specifies the device used for the engagement.
* COUNT(\*) AS WeeklyEngagement: Counts the number of engagements per week for each device.
* GROUP BY Year, Week, device: Groups the data by year, week, and device to calculate engagement statistics for each device per week.
* ORDER BY Year, Week, device: Orders the results by year, week, and device for better readability.



It goes till 35weeks.



1. **Email Engagement Analysis:**
   * Objective: Analyze how users are engaging with the email service.
   * Task: Write an SQL query to calculate the email engagement metrics.

**Main Query:**

The main query selects columns for the week number (WEEK) and various email engagement rates. These rates include the percentage of users who engaged in specific email actions (such as opening an email or clicking through a link) relative to the total number of users who received emails. The ROUND function is used to round the calculated rates to two decimal places for better readability.

**Subquery:**

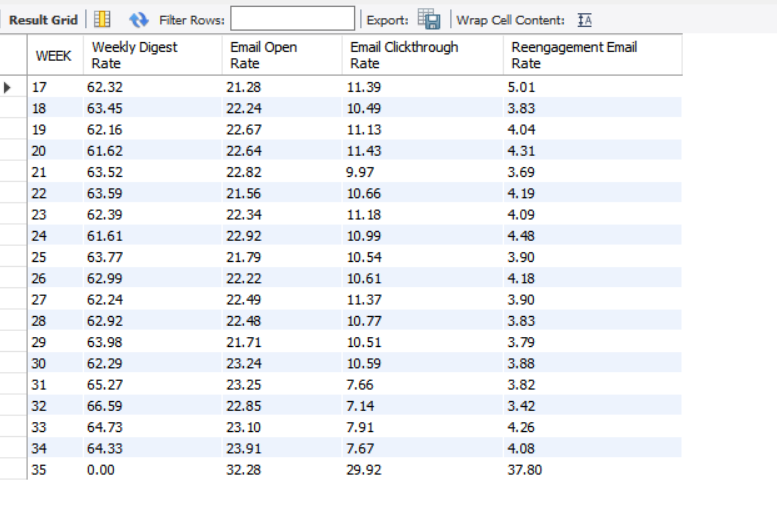
The subquery is enclosed within parentheses and aliased as subquery. It retrieves data from the email\_events table. It calculates the counts of different email actions (sent\_weekly\_digest, email\_open, email\_clickthrough, sent\_reengagement\_email) for each week using COUNT and CASE statements. Additionally, it calculates the total number of email events for each week by counting all occurrences of user\_id. The data is grouped by week number using GROUP BY 1, where 1 refers to the first column in the SELECT list (EXTRACT(WEEK FROM occurred\_at)).

The results are ordered by the week number using ORDER BY 1.

**Joining the Main Query and Subquery:**

The main query selects data from the subquery. It calculates the email engagement rates by dividing the counts of specific email actions by the total count of email events for each week. The resulting rates are rounded using the ROUND function.

Finally, the results are returned.



**RESULTS:**

This project aimed to analyze user engagement and activity on Instagram using SQL queries. By examining metrics such as likes, comments, and follows, insights were derived to inform decision-making for product enhancements and marketing strategies. Through data-driven analysis, trends in user behavior were uncovered, enabling the identification of key areas for improvement and growth. The utilization of SQL queries allowed for the extraction of meaningful insights from Instagram's vast dataset, aiding in understanding user interactions and preferences. Overall, this project demonstrated the power of SQL in extracting actionable insights from large-scale social media data, contributing to informed decision-making and the continuous improvement of the Instagram platform.

The insights and the key findings I derived in these projects:

* Users exhibit high engagement, with significant likes, comments, and follows.
* Growth trends indicate a steady increase in user activity over time.
* Certain users exhibit suspicious behavior, possibly indicating bot activity.
* Throughput analysis reveals peak activity periods, aiding in resource allocation.
* Language analysis highlights dominant languages among users.
* Retention analysis identifies patterns in user longevity on the platform.
* Email engagement metrics provide insights into user interactions with email services.
* Rolling averages smooth out short-term fluctuations in throughput data.
* Weekly engagement per device showcases platform usage across different devices.
* Duplicate row detection helps maintain data integrity and accuracy.