NAME: NIKHIL ZODAPE

EMAIL ID.: NIKSZODAPE@GMAIL.COM

PROJECT NAME: OPERATION ANALYTICS AND INVESTIGATING METRIC SPIKE

USING: MYSQL WORKBENCH 8.0 CE

Purpose:

The Purpose of this project is to leverage advance SQL skills to perform Operational Analytics for a company, similar to Microsoft. Operational Analytics involves in-depth analysis of end-to-end operations to identify areas for improvement. As a Lead Data analyst, the goal is to collaborate with various departments, including operations, support and marketing, to extract valuable insights from the available datasets. A significant focus will be on investigating and explaining metric spikes, such as fluctuations in daily user engagement or sales.
The lead Data analyst aims to contribute to the continuous improvement of the company's operations and decision-making processes through data-driven insights.

Case Study1: Job Data Analysis

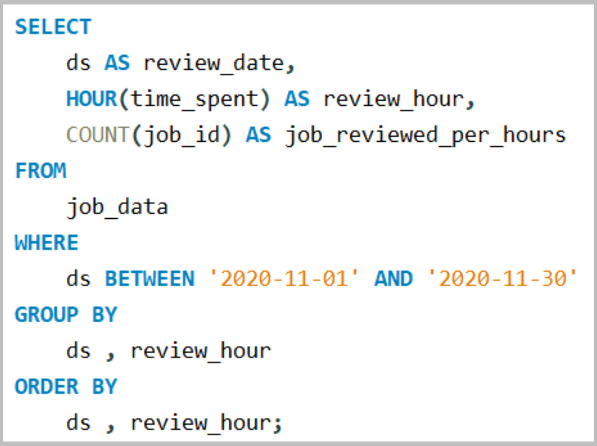
- **job_id:** Unique identifier of jobs
- **actor_id:** Unique identifier of actor
- **event:** The type of event (decision/skip/transfer).
- **language:** The Language of the content
- **time_spent:** Time spent to review the job in seconds.
- **org:** The Organization of the actor
- **ds:** The date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

Task:

A. Job reviewed over time:

Objective: Calculate the number of jobs reviewed per hour for each day in November 2020.

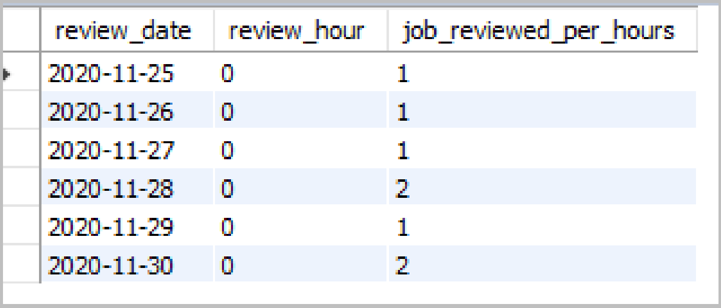
- SELECT the review data('ds'), extract the hour from the 'time_spent' column, and count the number of jobs reviewed in each hour.
- The 'WHERE' clause filters the data for the month on November 2020.
- The 'GROUP BY' clause groups the result by data and hours.
- The 'ORDER BY' clause ensures that the results are sorted by data and hours.



```
SELECT
    ds AS review_date,
    HOUR(time_spent) AS review_hour,
    COUNT(job_id) AS job_reviewed_per_hours
FROM
    job_data
WHERE
    ds BETWEEN '2020-11-01' AND '2020-11-30'
GROUP BY
    ds , review_hour
ORDER BY
    ds , review_hour;
```

OUTPUT →

We can say that the jobs reviewed per hours is minimum 1 and maximum 2 in the month of November.

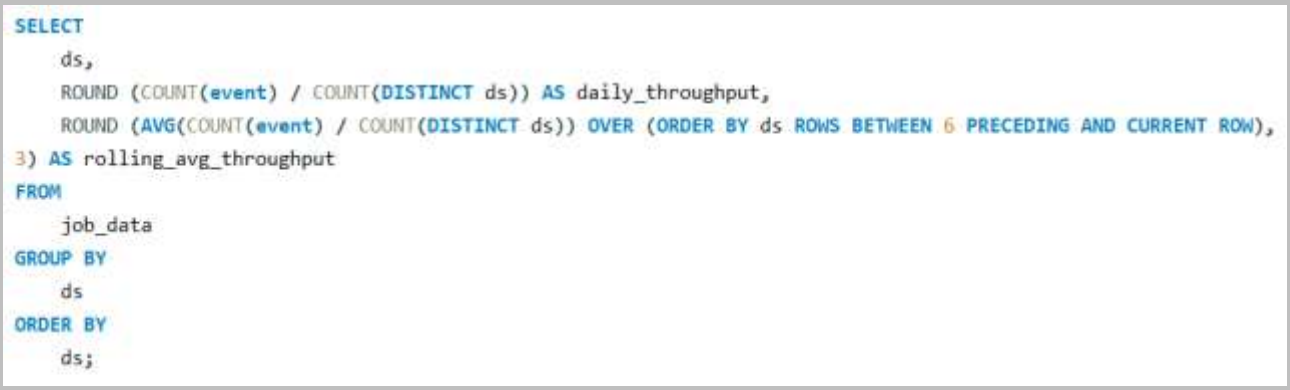


review_date	review_hour	job_reviewed_per_hours
2020-11-25	0	1
2020-11-26	0	1
2020-11-27	0	1
2020-11-28	0	2
2020-11-29	0	1
2020-11-30	0	2

B. Throughput Analysis:

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

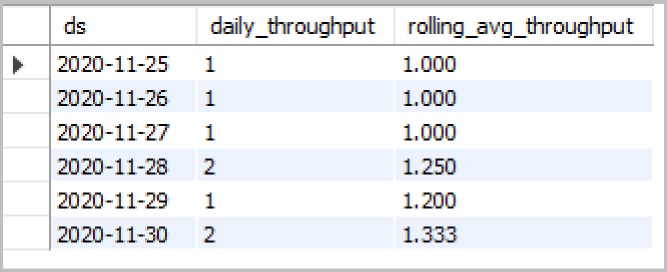
- SELECT the ds (date) column from the job_data
- Then calculated the daily throughput by dividing the count of events by the count of distinct dates, then rounding to the nearest whole number.
- Calculated the 7-day rolling average throughput using a windows function. The result is rounded to three decimal places.
- Then Groupe by and Order by the result using ds(date) column.



```
SELECT
    ds,
    ROUND (COUNT(event) / COUNT(DISTINCT ds)) AS daily_throughput,
    ROUND (AVG(COUNT(event) / COUNT(DISTINCT ds)) OVER (ORDER BY ds ROWS BETWEEN 6 PRECEDING AND CURRENT ROW),
3) AS rolling_avg_throughput
FROM
    job_data
GROUP BY
    ds
ORDER BY
    ds;
```

OUTPUT →

As for whether to use the daily metric or the 7-day rolling average, it depends on the context and special needs.



ds	daily_throughput	rolling_avg_throughput
2020-11-25	1	1.000
2020-11-26	1	1.000
2020-11-27	1	1.000
2020-11-28	2	1.250
2020-11-29	1	1.200
2020-11-30	2	1.333

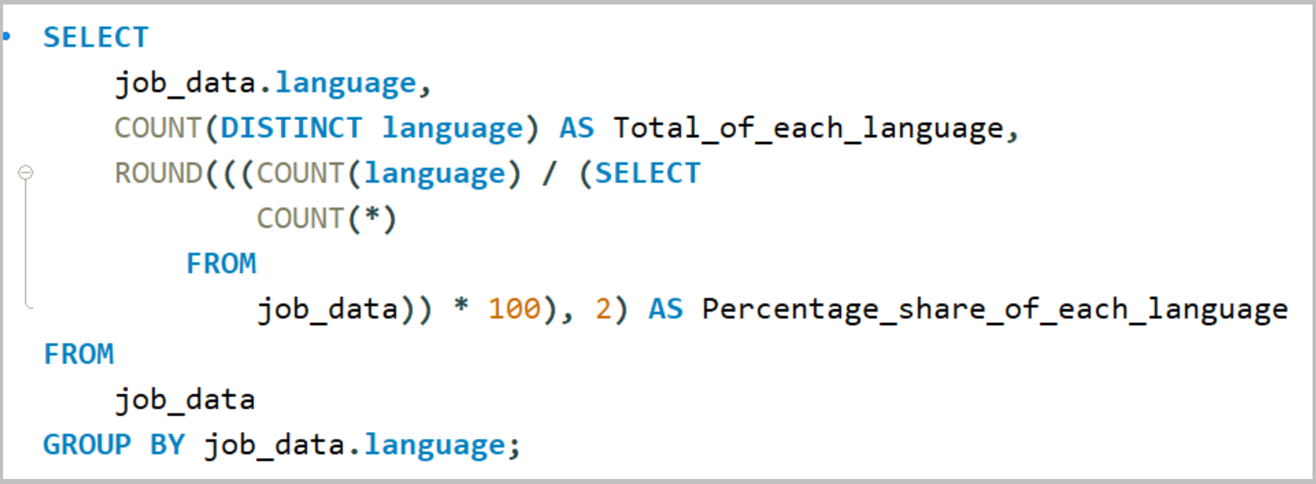
1. **Daily Metric:**
   - ➔ Advantages: It Gives a detailed look at throughput every day.
   - ➔ Disadvantages: Can be influenced by daily ups and downs, making it tricky to spot long-term patterns.
2. **7-Day rolling Average:**
   - ➔ Advantages: Evens out quick changes, showing a steady view of trends.
   - ➔ Disadvantages: Might not catch sudden changes quickly, as it considers data from the past 7 days.

If our focus is on short-term trends and we want to capture daily variations, we use the daily metric.
If we want the smoother representation that shows the long-term trends while minimizing daily functions, we will use the 7-day rolling average.

C. Language Share Analysis:

Objective: Calculate the percentage share of each language in the last 30 days.

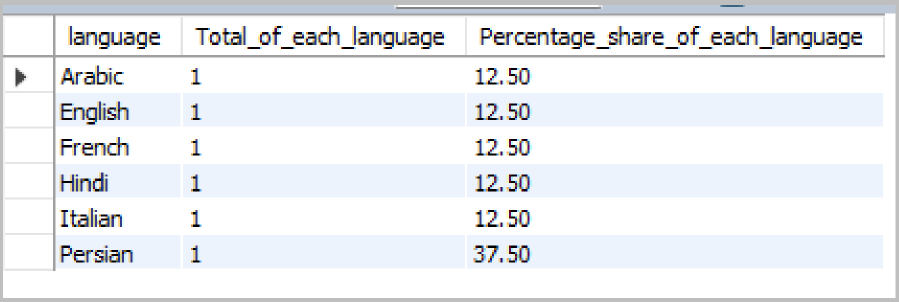
- SELECT the language column that we have to analyse.
- Counts the distinct occurrences of each language, giving the total count for each language.
- Calculates the percentage share of each language by dividing the count of each language by the total count of all language is the 'job_data' table. The result is rounded to two decimal places.
- From table "job_data".
- GROUP BY clause groups the result by the language, so the counts and percentages are calculated for each unique language.



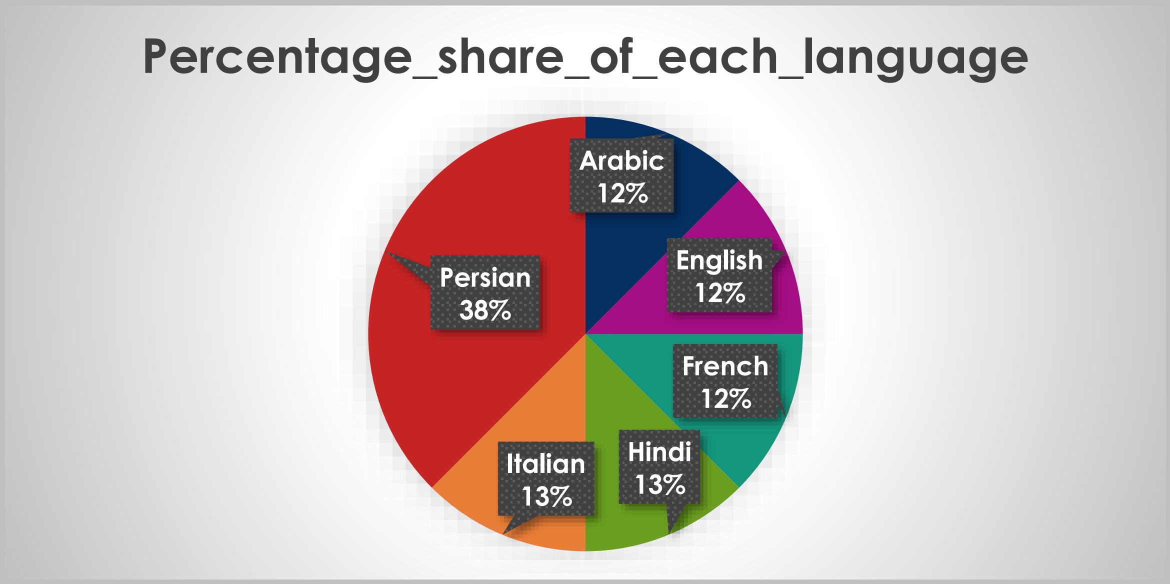
```
SELECT
    job_data.language,
    COUNT(DISTINCT language) AS Total_of_each_language,
    ROUND(((COUNT(language) / (SELECT
            COUNT(*)
        FROM
            job_data)) * 100), 2) AS Percentage_share_of_each_language
FROM
    job_data
GROUP BY job_data.language;
```

OUTPUT →

We can see that Persian language share are having more in past 30 days other than Arabic, English, French, Hindi and Italian.



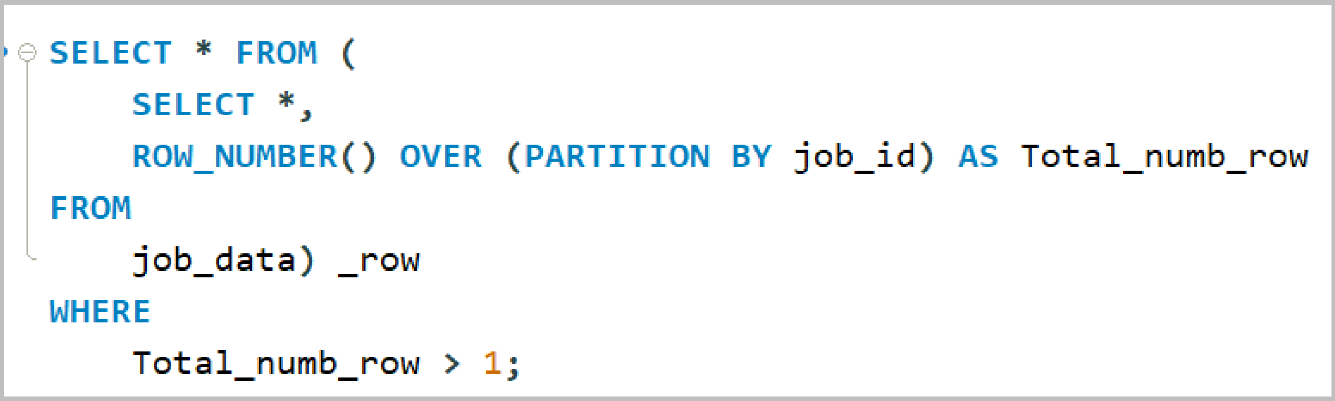
language	Total_of_each_language	Percentage_share_of_each_language
Arabic	1	12.50
English	1	12.50
French	1	12.50
Hindi	1	12.50
Italian	1	12.50
Persian	1	37.50



D. Duplicate Rows Detection:

Objective: Identify duplicate rows in the data.

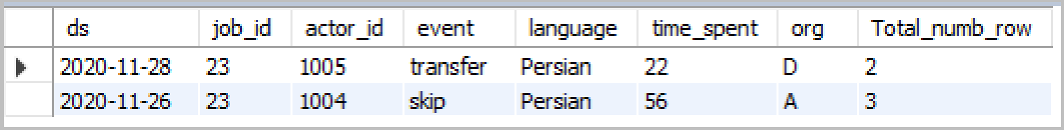
- First, we use the query of windows function ROW_NUMBER(), it assigns a unique row numbers to each row within the partition defined by 'job_id'.
- Selects all columns for rows derived from the inner query.
- Then we filter the result to include only rows where the row number is greater than 1.



```
SELECT * FROM (
    SELECT *,
    ROW_NUMBER() OVER (PARTITION BY job_id) AS Total_numb_row
FROM
    job_data) _row
WHERE
    Total_numb_row > 1;
```

Output →

We can see there are 2 duplicate rows in our data.



ds	job_id	actor_id	event	language	time_spent	org	Total_numb_row
2020-11-28	23	1005	transfer	Persian	22	D	2
2020-11-26	23	1004	skip	Persian	56	A	3

Case study 2: Investigating Metric Spike

We will be working with three tables:

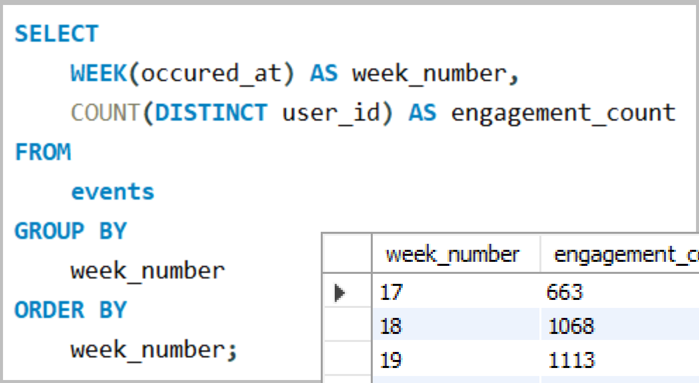
- **users**: Contains one row per user, with descriptive information about that user's account.
- **events**: Contains one row per event, where an event is an action that a user has taken (e.g., login, messaging, search)
- **email_events**: Contains events specific to the sending of emails.

Task:

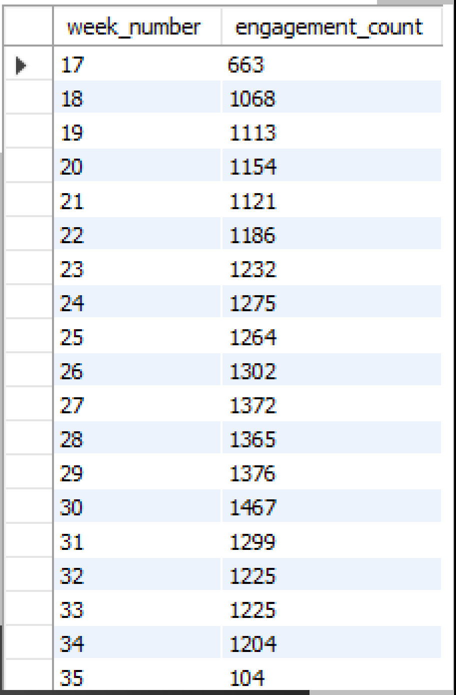
A. **Weekly User Engagement:**

Objective: Measure the activeness of users on a weekly basis.

- First, we extracted the week number from the 'occurred_at' timestamp in the 'events' table and gives the aliases as 'week_number'.
- Then counts the number of 'distinct user_id' in each week, representing the user engagement for that week.
- All the data was selected from the 'events' table.
- Then Grouped the results by the extracted week number. To ensures that the count of distinct user_id is calculated for each week separately.
- Lastly, we Ordered the 'week_number' result by week number in ascending order.



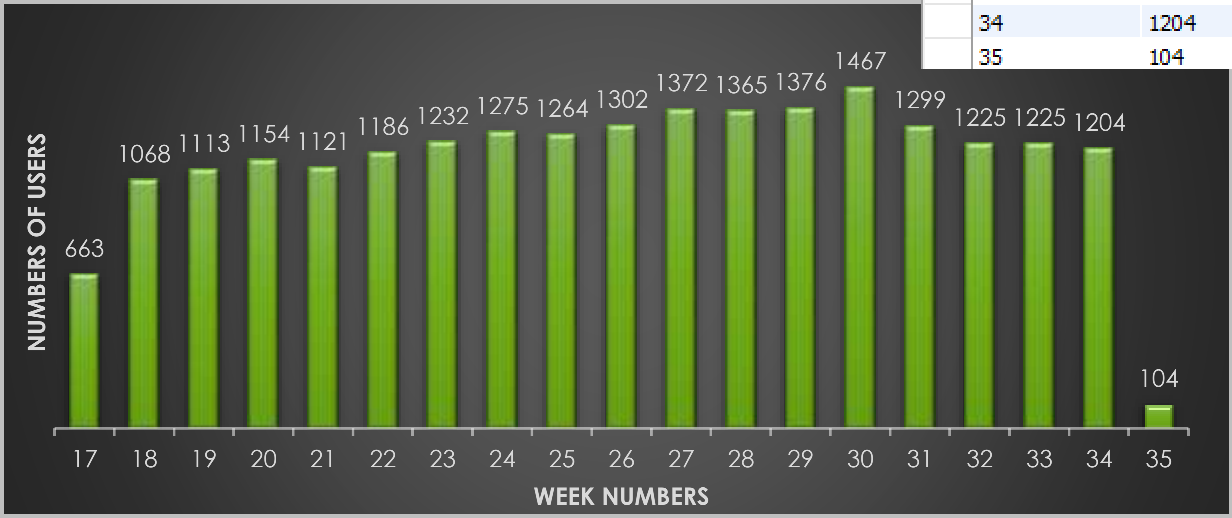
```
SELECT
    WEEK(occured_at) AS week_number,
    COUNT(DISTINCT user_id) AS engagement_count
FROM
    events
GROUP BY
    week_number
ORDER BY
    week_number;
```



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

Output →

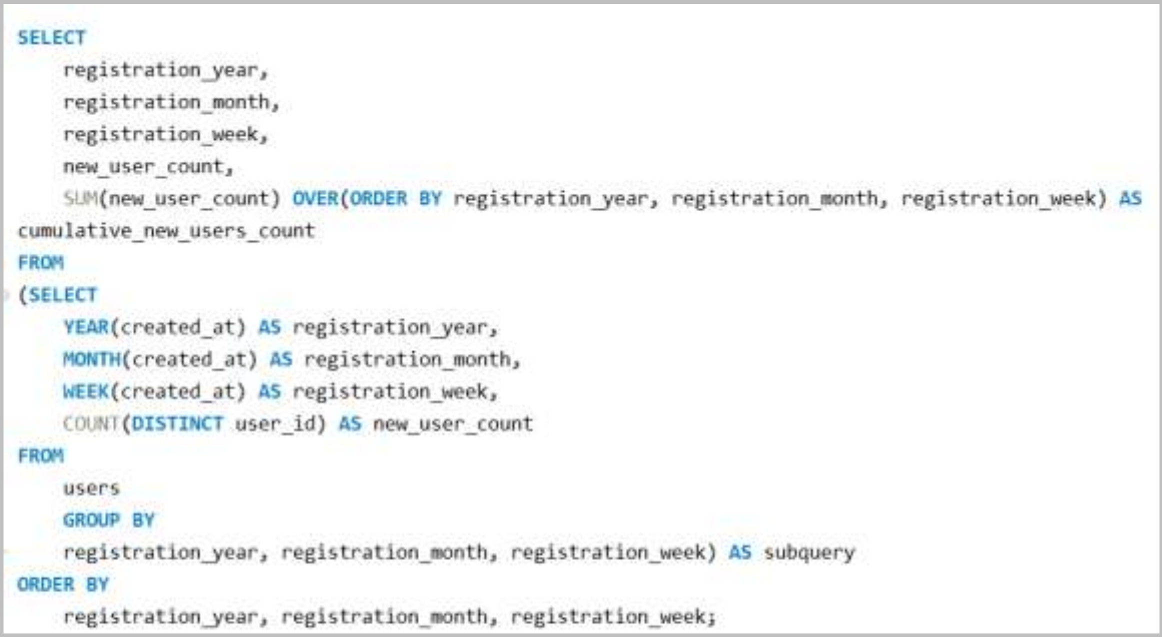
From the result we can say that on week 35 we are getting less user engagement and one week 30 we are getting highest numbers of user engagement.



B. User Growth Analysis:

Objective: Analyze the growth of users over time for a product.

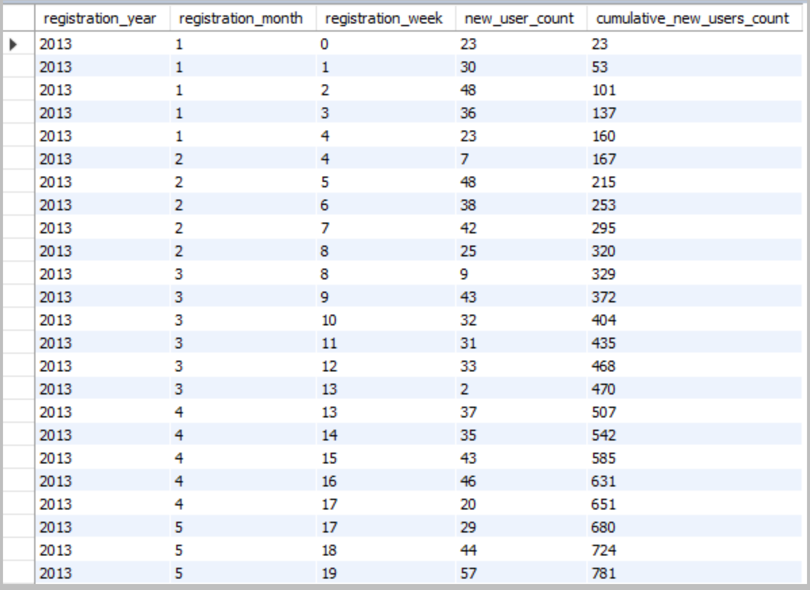
- This query analyze user registration over time, breaking down the data into weeks, months and year.
- It provide insights into the number or new users registering each weeks ('new_user_count') and also tracks the cumulative growth of users over time('cumulative_users').
- This information is valuable for understanding user acquisition patters and observing the overall growth trend in user registrations.



```
SELECT
    registration_year,
    registration_month,
    registration_week,
    new_user_count,
    SUM(new_user_count) OVER(ORDER BY registration_year, registration_month, registration_week) AS
cumulative_new_users_count
FROM
(SELECT
    YEAR(created_at) AS registration_year,
    MONTH(created_at) AS registration_month,
    WEEK(created_at) AS registration_week,
    COUNT(DISTINCT user_id) AS new_user_count
FROM
    users
    GROUP BY
    registration_year, registration_month, registration_week) AS subquery
ORDER BY
    registration_year, registration_month, registration_week;
```

Output →

As the result is big, I will provide the small screenshot of the result and to see the whole result click on below link.



registration_year	registration_month	registration_week	new_user_count	cumulative_new_users_count
2013	1	0	23	23
2013	1	1	30	53
2013	1	2	48	101
2013	1	3	36	137
2013	1	4	23	160
2013	2	4	7	167
2013	2	5	48	215
2013	2	6	38	253
2013	2	7	42	295
2013	2	8	25	320
2013	3	8	9	329
2013	3	9	43	372
2013	3	10	32	404
2013	3	11	31	435
2013	3	12	33	468
2013	3	13	2	470
2013	4	13	37	507
2013	4	14	35	542
2013	4	15	43	585
2013	4	16	46	631
2013	4	17	20	651
2013	5	17	29	680
2013	5	18	44	724
2013	5	19	57	781

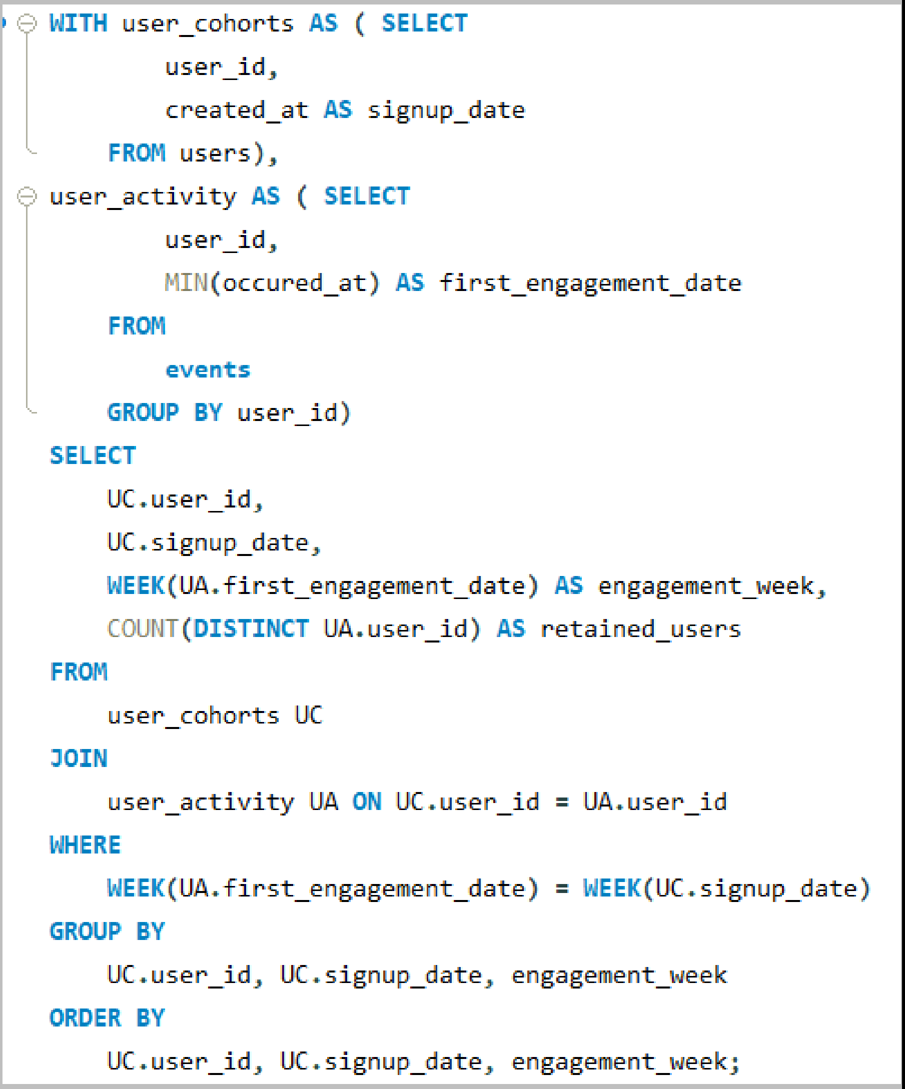
https://drive.google.com/file/d/11loefa8IRhpB9SBqOqmI1vuTjZPmQNwm/view?usp=drive_link

- The data spans across multiple years and months, providing a multiple view of the user registration trends.
- The cumulative count slowly increases, demonstrating the ongoing growth of users as weeks progress.
- The total growth of new users is 9381.

C. Weekly Retention Analysis:

Objective: Analyze the retention of users on a weekly basis after signing up for a product.

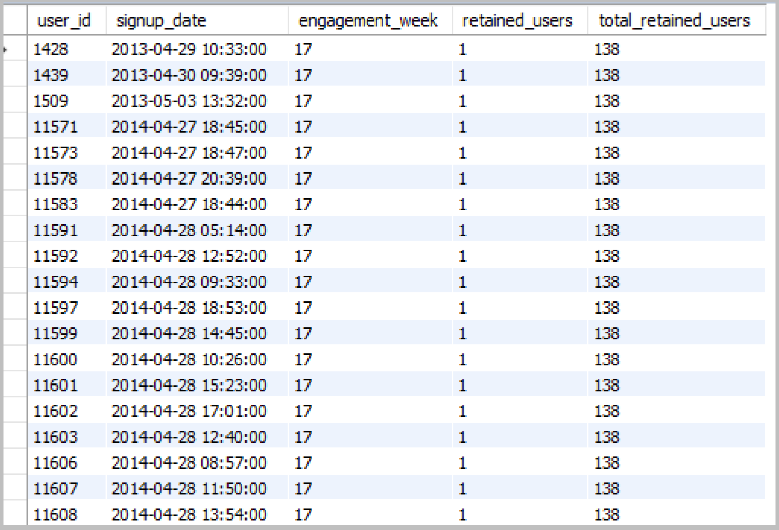
- ‘User_cohorts’ identifies each user’s sign-up by extracting the user_id and sign-up timestamp from the ‘users’ table.
- ‘User_activity’ determines the first engagement date for each users by finding the minimum occurrence timestamp from the ‘events’ table for each users.
- The main query joins the ‘user_chorrs’ and ‘user_activity’ based on the user id. It calculates the engagement week using the ‘WEEK()’ function on the first engagement table.
- The ‘COUNT(DISTINCT UA.user_id)’ counts the number of retained users for each week.
- The “WHERE’ clause ensures that only engagement occurring in the same week as the user’s sig-up date are considered. This filters the data to focus on weekly retention.



```
WITH user_cohorts AS ( SELECT
        user_id,
        created_at AS signup_date
    FROM users),
user_activity AS ( SELECT
        user_id,
        MIN(occured_at) AS first_engagement_date
    FROM
        events
    GROUP BY user_id)
SELECT
    UC.user_id,
    UC.signup_date,
    WEEK(UA.first_engagement_date) AS engagement_week,
    COUNT(DISTINCT UA.user_id) AS retained_users
FROM
    user_cohorts UC
JOIN
    user_activity UA ON UC.user_id = UA.user_id
WHERE
    WEEK(UA.first_engagement_date) = WEEK(UC.signup_date)
GROUP BY
    UC.user_id, UC.signup_date, engagement_week
ORDER BY
    UC.user_id, UC.signup_date, engagement_week;
```

Output →

As the result is big, I will provide the small screenshot of the result and to see the whole result click on below link.



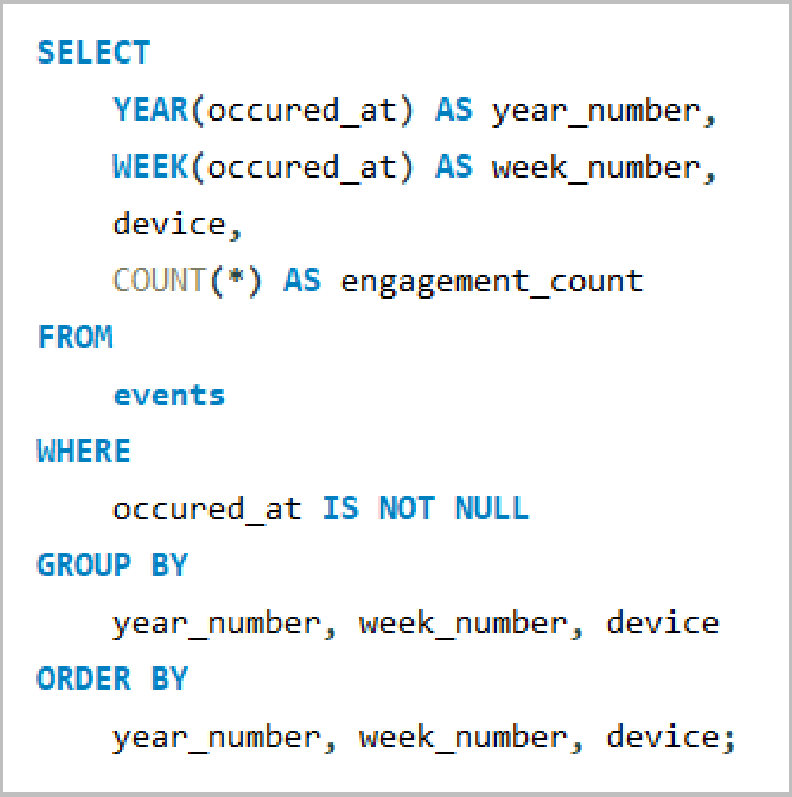
user_id	signup_date	engagement_week	retained_users	total_retained_users
1428	2013-04-29 10:33:00	17	1	138
1439	2013-04-30 09:39:00	17	1	138
1509	2013-05-03 13:32:00	17	1	138
11571	2014-04-27 18:45:00	17	1	138
11573	2014-04-27 18:47:00	17	1	138
11578	2014-04-27 20:39:00	17	1	138
11583	2014-04-27 18:44:00	17	1	138
11591	2014-04-28 05:14:00	17	1	138
11592	2014-04-28 12:52:00	17	1	138
11594	2014-04-28 09:33:00	17	1	138
11597	2014-04-28 18:53:00	17	1	138
11599	2014-04-28 14:45:00	17	1	138
11600	2014-04-28 10:26:00	17	1	138
11601	2014-04-28 15:23:00	17	1	138
11602	2014-04-28 17:01:00	17	1	138
11603	2014-04-28 12:40:00	17	1	138
11606	2014-04-28 08:57:00	17	1	138
11607	2014-04-28 11:50:00	17	1	138
11608	2014-04-28 13:54:00	17	1	138

https://drive.google.com/file/d/1-WoO9cpUavFEgSsd97YItZgpiO3Iyvck/view?usp=drive_link

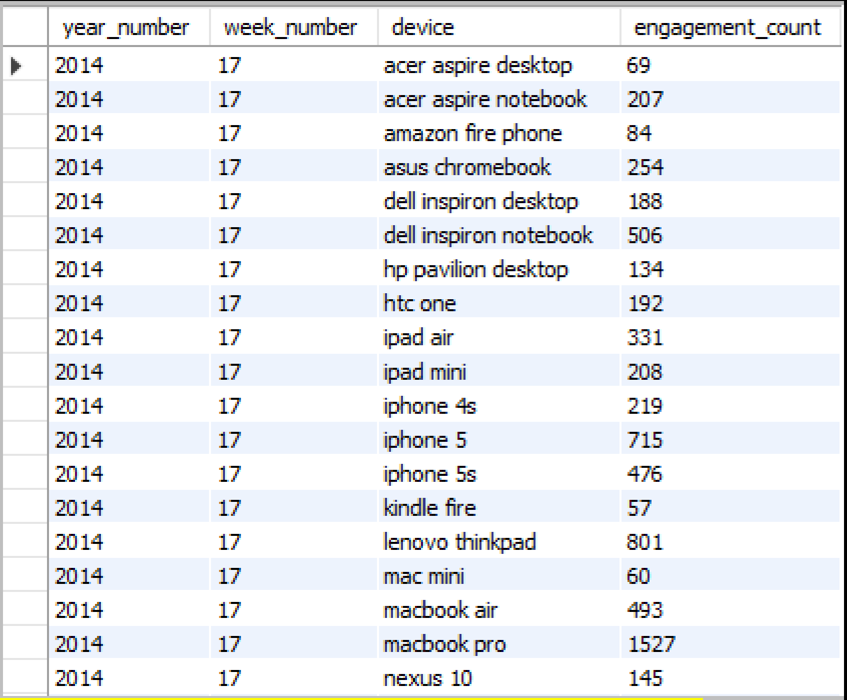
D. **Weekly Engagement Per Device:**

Objective: Measure the activeness of users on a weekly basis per device.

- 'YEAR(occured-at)' extracts the year number from the 'occured_at timestamp.
- 'WEEK(occured_at)' extracts the week number from the 'occured_at' timestamp.
- 'device' is the column representing the device used by the user during the event.
- 'COUNT(*)' calculates the numbers of events for each combination of week and device.
- The 'WHERE occured_at IS NOT NULL' condition filters out any rows where the 'occured_at' timestamp is null.
- The 'GROUP_BY' clause includes both 'year_number' and 'week_number' to group events by both year and week.
- The 'ORDER BY' clause is adjusted to order the result by year, week and device.



```
SELECT
    YEAR(occured_at) AS year_number,
    WEEK(occured_at) AS week_number,
    device,
    COUNT(*) AS engagement_count
FROM
    events
WHERE
    occured_at IS NOT NULL
GROUP BY
    year_number, week_number, device
ORDER BY
    year_number, week_number, device;
```



year_number	week_number	device	engagement_count
2014	17	acer aspire desktop	69
2014	17	acer aspire notebook	207
2014	17	amazon fire phone	84
2014	17	asus chromebook	254
2014	17	dell inspiron desktop	188
2014	17	dell inspiron notebook	506
2014	17	hp pavilion desktop	134
2014	17	htc one	192
2014	17	ipad air	331
2014	17	ipad mini	208
2014	17	iphone 4s	219
2014	17	iphone 5	715
2014	17	iphone 5s	476
2014	17	kindle fire	57
2014	17	lenovo thinkpad	801
2014	17	mac mini	60
2014	17	macbook air	493
2014	17	macbook pro	1527
2014	17	nexus 10	145

Output →

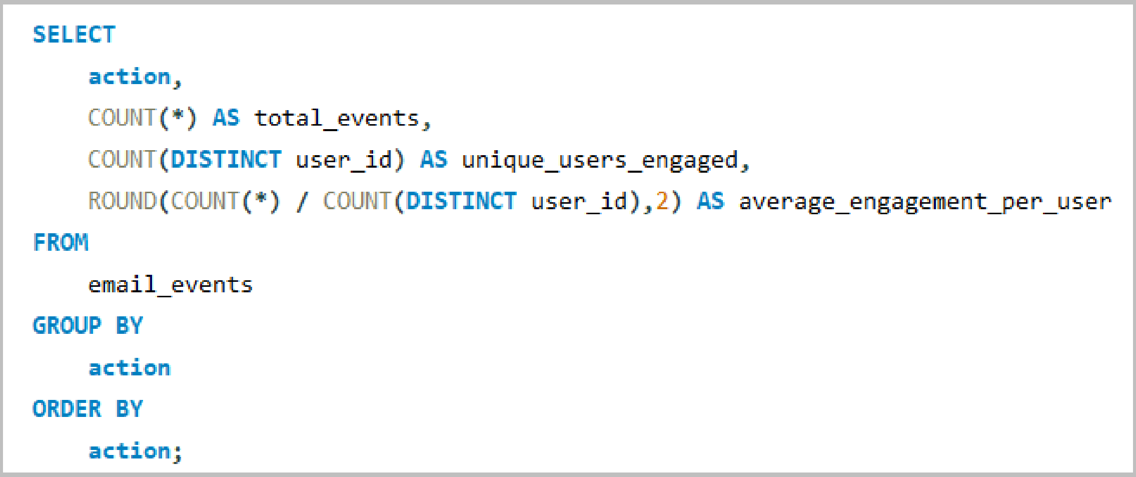
As the result is big, I will provide the small screenshot of the result and to see the whole result click on below link.

https://drive.google.com/file/d/1yAhNQTwOmFAUj4GBmMZwqbQUPP793OFE/view?usp=sharing

E. Email Engagement Analysis:

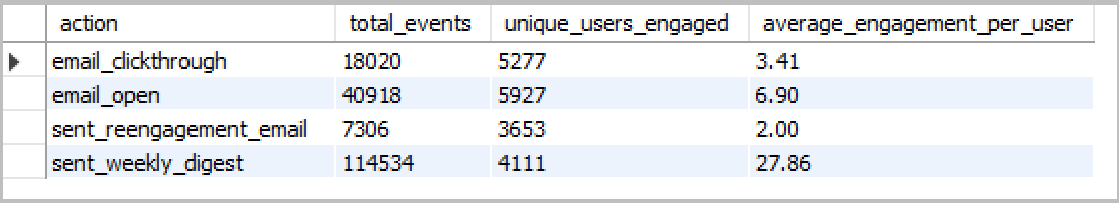
Objective: Analyze how users are engaging with the email service.

- 'action' is assumed to be the type of engagement event(e.g., open, click, etc.).
- 'total_events' represent the total count of email events for each action.
- 'unique_users_engaged' represents the count of unique users who engaged in each action.
- 'average_engagement_per_user' calculates the average engagement per user for each action.



```
SELECT
    action,
    COUNT(*) AS total_events,
    COUNT(DISTINCT user_id) AS unique_users_engaged,
    ROUND(COUNT(*) / COUNT(DISTINCT user_id),2) AS average_engagement_per_user
FROM
    email_events
GROUP BY
    action
ORDER BY
    action;
```

Output →



action	total_events	unique_users_engaged	average_engagement_per_user
email_clickthrough	18020	5277	3.41
email_open	40918	5927	6.90
sent_reengagement_email	7306	3653	2.00
sent_weekly_digest	114534	4111	27.86

Below are the links of SQL Files of CASE STUDY 1 and 2:

1. https://drive.google.com/file/d/1l3hvaIOWqxxNfd8ZBH-o8iKAXo_0aEhC/view?usp=sharing

2.https://drive.google.com/file/d/1vML6bQsg9d9ogVOP6tvqJnT4Jy4cwlhw/view?usp=sharing