Task-3: Operation Analytics and Investigating Metric Spike

SQL Tasks:

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

1. Jobs Reviewed Over Time
2. Throughput Analysis
3. Language Share Analysis
4. Duplicate Rows Detection

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

1. Weekly User Engagement
2. User Growth Analysis
3. Weekly Retention Analysis
4. Weekly Engagement Per Device
5. Email Engagement Analysis

- **Software Used: MySQL Workbench 8.0 CE**

CASE STUDY-1: JOB DATA ANALYSIS

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.

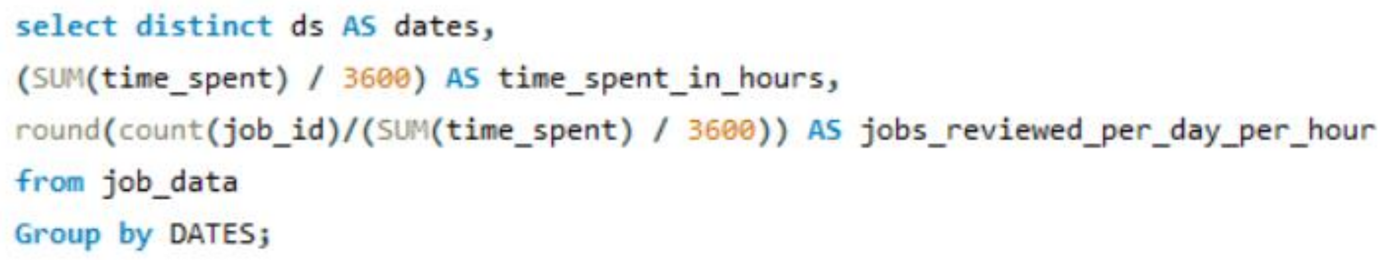
- Firstly, we **select distinct ds** values, which represent **dates** from the **job_data table**.
- **time_spent** calculates the total time spent (in hours) using the **SUM(time_spent)** function and converts it to hours by dividing by **3600** (seconds in an hour).
- **jobs_reviewed_per_day_per_hour** calculates the average number of jobs reviewed per day per hour by dividing the **count** of **job_id** values by the time spent in hours.
- Finally, The data is grouped by **ds** (dates) using the **GROUP BY clause**.

CASE STUDY-1: JOB DATA ANALYSIS

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.

Query:



```
select distinct ds AS dates,
(SUM(time_spent) / 3600) AS time_spent_in_hours,
round(count(job_id)/(SUM(time_spent) / 3600)) AS jobs_reviewed_per_day_per_hour
from job_data
Group by DATES;
```

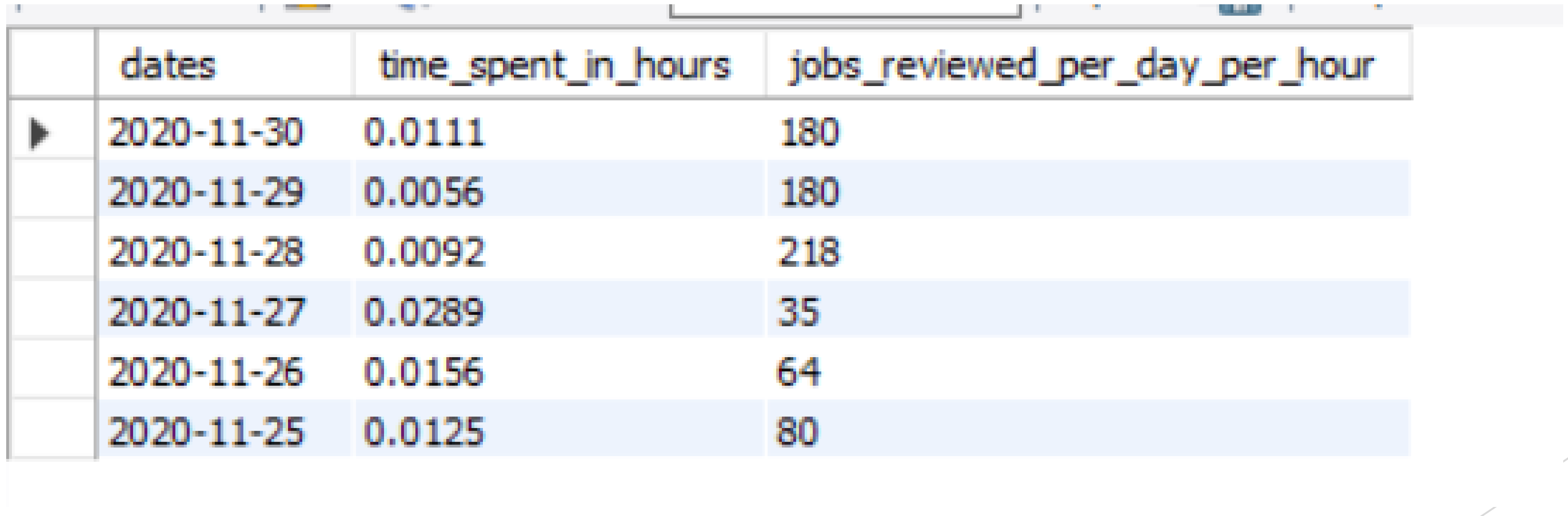
```
select distinct ds as dates,
(sum(time_spent)/3600) as time_spent_in_hours,
round(count(job_id)/(sum(time_spent)/3600)) as
jobs_reviewed_per_day_per_hour  from job_data
group by dates;
```

CASE STUDY-1: JOB DATA ANALYSIS

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.

Result/Output:



dates	time_spent_in_hours	jobs_reviewed_per_day_per_hour
2020-11-30	0.0111	180
2020-11-29	0.0056	180
2020-11-28	0.0092	218
2020-11-27	0.0289	35
2020-11-26	0.0156	64
2020-11-25	0.0125	80

CASE STUDY-1: JOB DATA ANALYSIS

2) 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.

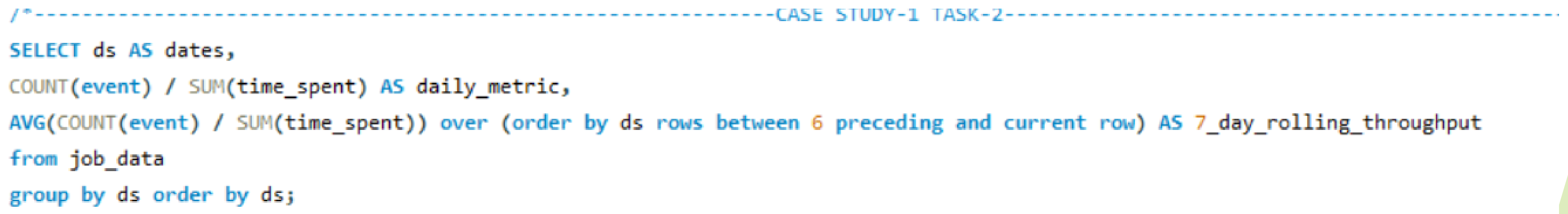
- We **select** the **ds** column as **dates** and calculates the daily metric as the ratio of COUNT(event) to **SUM(time_spent)** for each date.
- **7_day_rolling_throughput** calculates the 7-day rolling average of the daily metric using the AVG function with a window frame **(rows between 6 preceding and current row).**
- The data is grouped by **ds** (dates) using the **GROUP BY clause**.
- The results are ordered chronologically by **ds** using the **ORDER BY clause**.

CASE STUDY-1: JOB DATA ANALYSIS

2) 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.

Query:



```
/*-----------------------------------------------------------CASE STUDY-1 TASK-2--------------------------------------------
SELECT ds AS dates,
COUNT(event) / SUM(time_spent) AS daily_metric,
AVG(COUNT(event) / SUM(time_spent)) over (order by ds rows between 6 preceding and current row) AS 7_day_rolling_throughput
from job_data
group by ds order by ds;
```

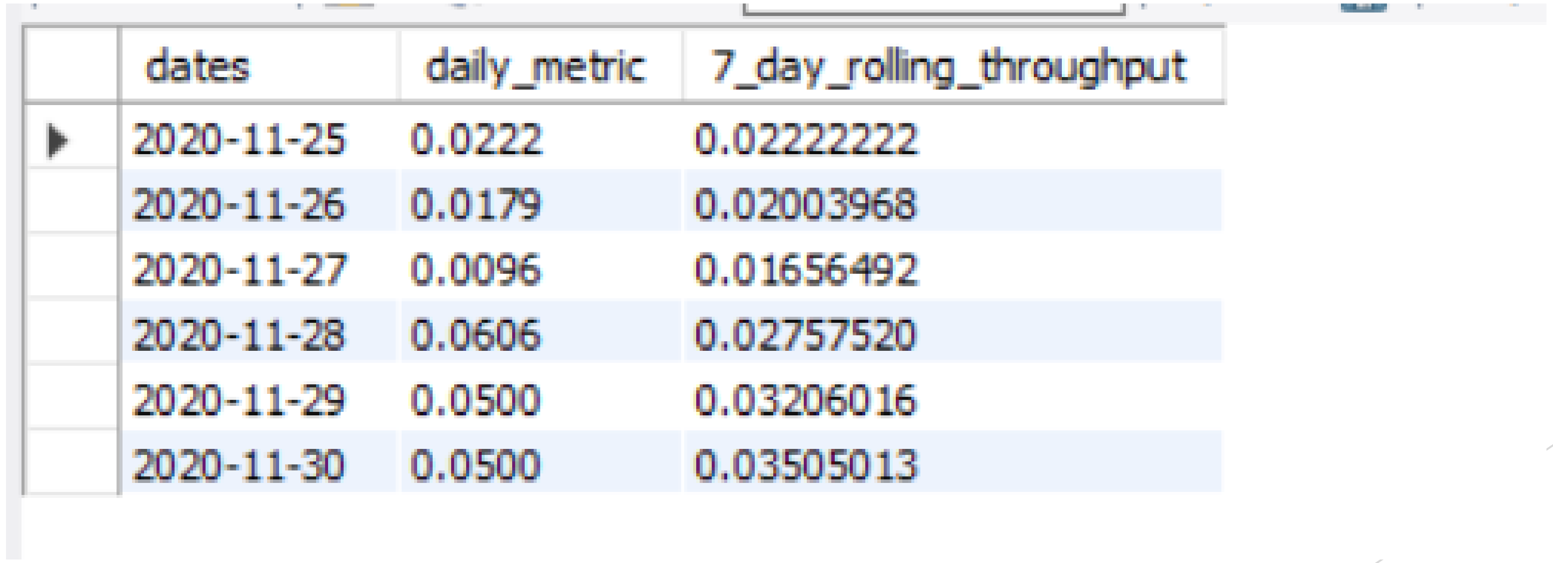
SELECT ds AS dates,COUNT(event) / SUM(time_spent) AS daily_metric,AVG(COUNT(event) / SUM(time_spent)) over (order by ds rows between 6 preceding and current row) AS 7_day_rolling_throughputfrom job_data group by ds order by ds;

CASE STUDY-1: JOB DATA ANALYSIS

2) 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.

Result/Output:



dates	daily_metric	7_day_rolling_throughput
2020-11-25	0.0222	0.02222222
2020-11-26	0.0179	0.02003968
2020-11-27	0.0096	0.01656492
2020-11-28	0.0606	0.02757520
2020-11-29	0.0500	0.03206016
2020-11-30	0.0500	0.03505013

CASE STUDY-1: JOB DATA ANALYSIS

3) 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.

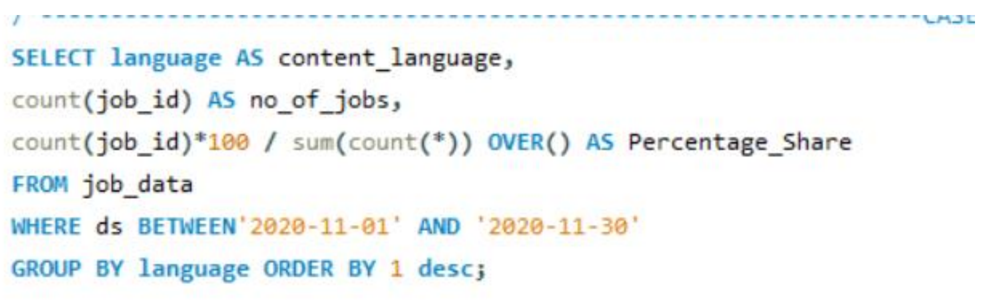
- We **select** the language column as **content_language** and calculate the count of **job_id** values as the number of jobs for each language.
- **Percentage_share** calculates the percentage of each language's job count by dividing the count of jobs for that language by the total count of all rows in the dataset.
- This is done using a window function **(sum(count(*)) OVER())** that calculates the sum of all counts.
- The code is filtered only rows with **ds** (date) values between '**2020-11-01**' and '**2020-11-30**'.
- The data is grouped by the language column using the **GROUP BY** clause. The results are ordered in **descending order** based on the language column.

CASE STUDY-1: JOB DATA ANALYSIS

3) 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.

Query:



```
SELECT language AS content_language,
count(job_id) AS no_of_jobs,
count(job_id)*100 / sum(count(*)) OVER() AS Percentage_Share
FROM job_data
WHERE ds BETWEEN'2020-11-01' AND '2020-11-30'
GROUP BY language ORDER BY 1 desc;
```

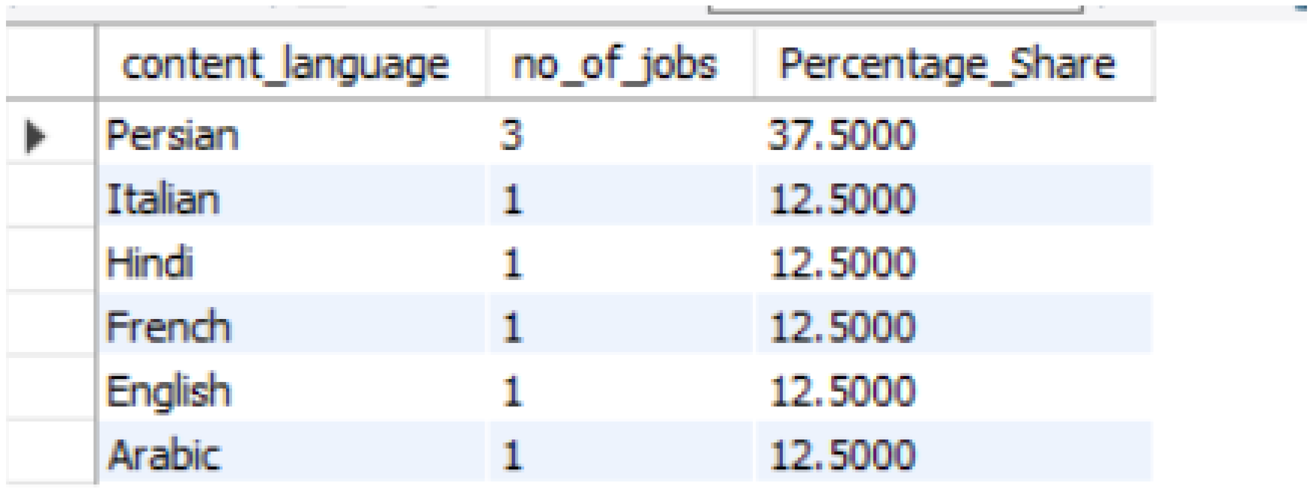
SELECT language AS content_language,count(job_id) AS no_of_jobs,
count(job_id)*100 / sum(count(*)) OVER() AS Percentage_Share
FROM job_data
WHERE ds BETWEEN'2020-11-01' AND '2020-11-30‘
GROUP BY language ORDER BY 1 desc;

CASE STUDY-1: JOB DATA ANALYSIS

3) 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.

Result/Output:



content_language	no_of_jobs	Percentage_Share
Persian	3	37.5000
Italian	1	12.5000
Hindi	1	12.5000
French	1	12.5000
English	1	12.5000
Arabic	1	12.5000

CASE STUDY-1: JOB DATA ANALYSIS

4) Duplicate Rows Detection

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

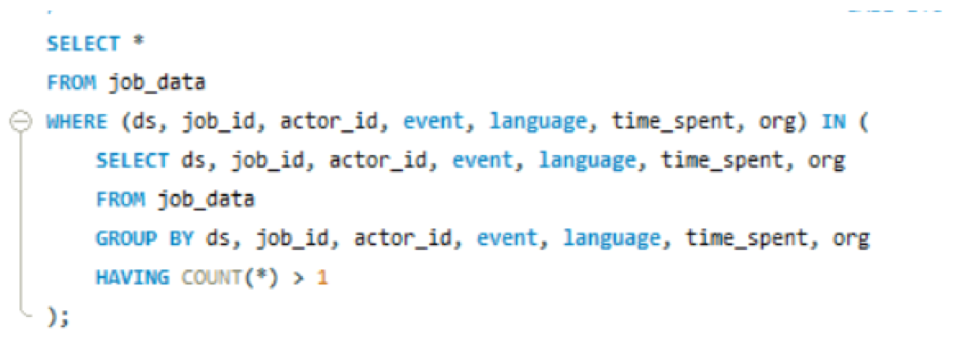
- Firstly, we **select** all columns **(*)** from the **job_data table**.
- We use a subquery to identify duplicate rows in the **job_data table**. The subquery selects columns **ds**, **job_id**, **actor_id**, **event**, **language**, **time_spent**, and **org** while grouping by these columns.
- It filters this grouped data using the **HAVING clause**, ensuring that only groups with a **count greater than 1** (duplicate entries) are considered.
- The main query then uses the **WHERE clause** to filter the original **job_data table** by rows that match the values identified as duplicates in the subquery..

CASE STUDY-1: JOB DATA ANALYSIS

4) Duplicate Rows Detection

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

Query:


```
SELECT *
FROM job_data
WHERE (ds, job_id, actor_id, event, language, time_spent, org) IN (
    SELECT ds, job_id, actor_id, event, language, time_spent, org
    FROM job_data
    GROUP BY ds, job_id, actor_id, event, language, time_spent, org
    HAVING COUNT(*) > 1
);
```

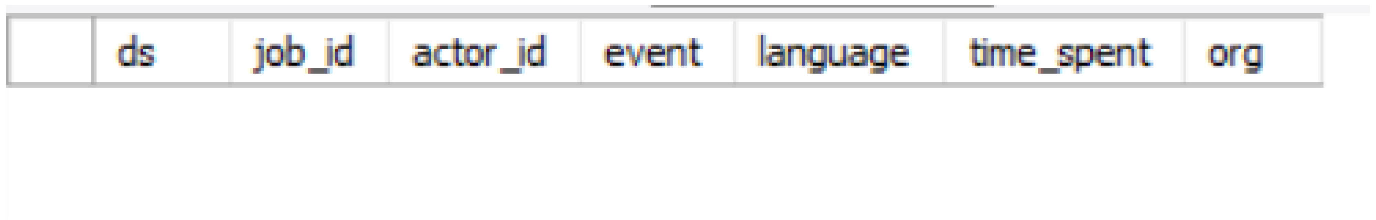
SELECT *FROM job_dataWHERE (ds, job_id, actor_id, event, language, time_spent, org) IN (SELECT ds, job_id, actor_id, event, language, time_spent, org   FROM job_data   GROUP BY ds, job_id, actor_id, event, language, time_spent, org HAVING COUNT(*) > 1);

CASE STUDY-1: JOB DATA ANALYSIS

4) Duplicate Rows Detection

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

Result/Output:



	ds	job_id	actor_id	event	language	time_spent	org

There are **no duplicate rows** in job_data. Hence, no rows are returned.

CASE STUDY-2: INVESTIGATING METRIC SPIKE

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.

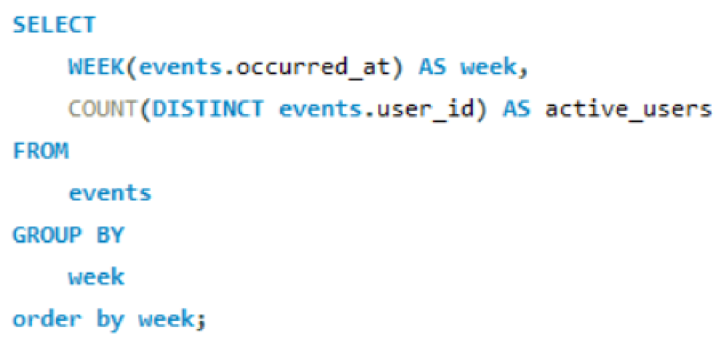
- Firstly, we **select** the week number based on the **occurred_at** column in the **events table.**
- **active_users** counts the **distinct user_id** values in the **events table** to calculate the number of **active_users** for each week.
- The data is grouped by week using the **GROUP BY clause**. This arranges the results by week, aggregating the user counts for each week.
- Finally, The results are ordered chronologically by week using the **ORDER BY clause**.

CASE STUDY-2: INVESTIGATING METRIC SPIKE

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.

Query:



```
SELECT
    WEEK(events.occurred_at) AS week,
    COUNT(DISTINCT events.user_id) AS active_users
FROM
    events
GROUP BY
    week
order by week;
```

SELECT WEEK(events.occurred_at) AS week, COUNT(DISTINCT events.user_id) AS active_users FROM events GROUP BY week order by week;

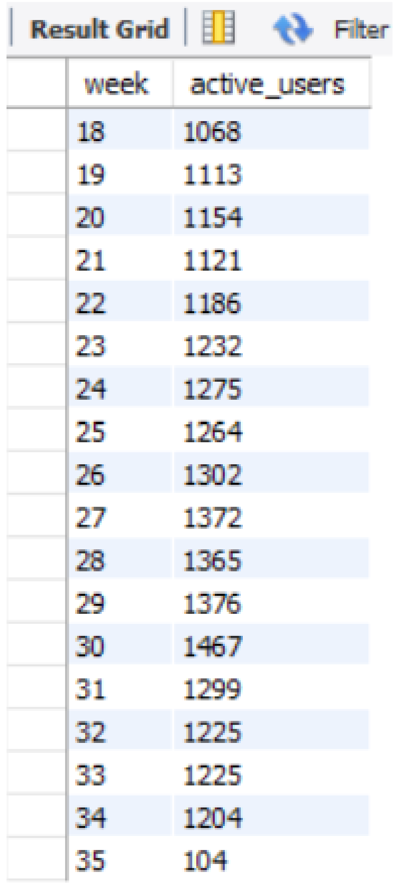
CASE STUDY-2: INVESTIGATING METRIC SPIKE

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.

Result/Output:

So, the most active users are on **week 30**. week 30 has almost **1467** active users



week	active_users
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

CASE STUDY-2: INVESTIGATING METRIC SPIKE

2) User Growth Analysis

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

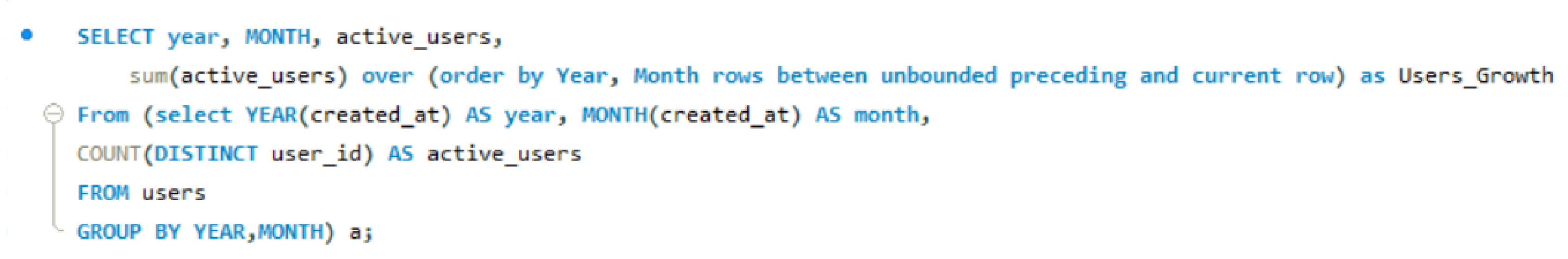
- Firstly, we **select** the **year** and **month** from the **created_at** column of the **users table**.
- Inside a subquery, it counts the **distinct user_id** values to calculate the number of active users for each year-month combination. This subquery groups the data by year and month.
- The outer query employs a window function **(sum(active_users) over (order by Year, Month rows between unbounded preceding and current row))** to compute the cumulative growth of active users over time.
- The window frame ranges from the beginning of the result set up to the current row, summing up active users.
- The final result showcases the **year**, **month**, **active users** count, and the cumulative growth of users up to the present month.

CASE STUDY-2: INVESTIGATING METRIC SPIKE

2) User Growth Analysis

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

Query:



```
SELECT year, MONTH, active_users,
    sum(active_users) over (order by Year, Month rows between unbounded preceding and current row) as Users_Growth
From (select YEAR(created_at) AS year, MONTH(created_at) AS month,
COUNT(DISTINCT user_id) AS active_users
FROM users
GROUP BY YEAR,MONTH) a;
```

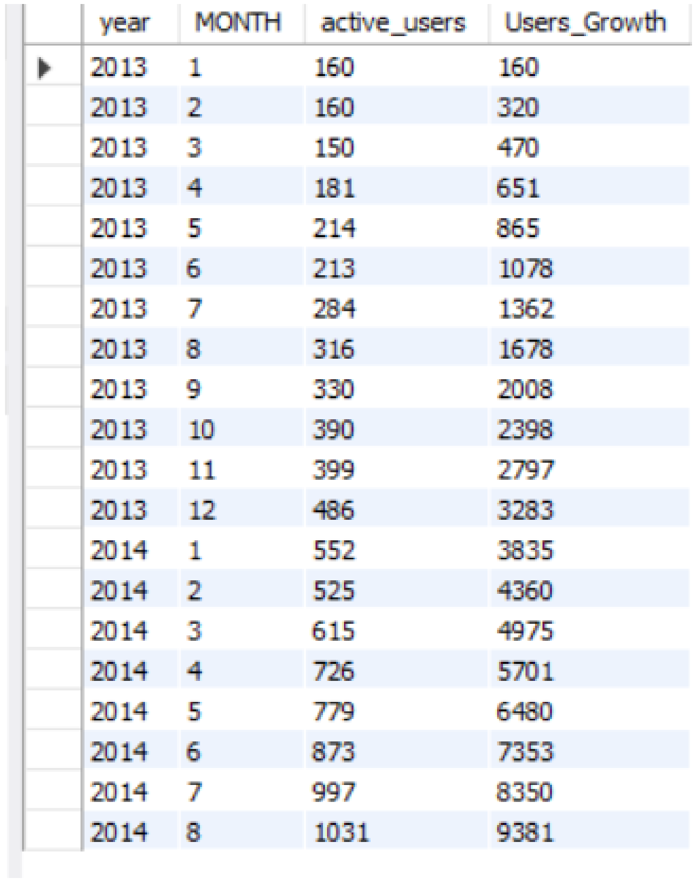
SELECT year, MONTH, active_users, sum(active_users) over (order by Year, Month rows between unbounded preceding and current row) as Users_Growth From (select YEAR(created_at) AS year, MONTH(created_at) AS month, COUNT(DISTINCT user_id) AS active_users FROM users GROUP BY YEAR,MONTH) a;

CASE STUDY-2: INVESTIGATING METRIC SPIKE

2) User Growth Analysis

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

Result/Output:



year	MONTH	active_users	Users_Growth
2013	1	160	160
2013	2	160	320
2013	3	150	470
2013	4	181	651
2013	5	214	865
2013	6	213	1078
2013	7	284	1362
2013	8	316	1678
2013	9	330	2008
2013	10	390	2398
2013	11	399	2797
2013	12	486	3283
2014	1	552	3835
2014	2	525	4360
2014	3	615	4975
2014	4	726	5701
2014	5	779	6480
2014	6	873	7353
2014	7	997	8350
2014	8	1031	9381

CASE STUDY-2: INVESTIGATING METRIC SPIKE

3) 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.

- Firstly, we select the week number from **events.occurred_at** and calculates the count of distinct **user_id** values from the **users table**, which represents the cohort size.
- It counts **distinct user_id** values from **events** that match the cohort's creation week, using a **CASE statement**. This indicates the number of active users during the cohort's inception week.
- The code counts **distinct user_id** values from **events** that occurred in or after the cohort's creation week. This quantifies the number of **users retained** from the cohort.
- The **users** and **events tables** are combined using a **left join** based on the **user_id**. This links user information to their associated events.
- Finally, **active_week** is grouped by using **GROUP BY**.

CASE STUDY-2: INVESTIGATING METRIC SPIKE

3) 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.

Query:

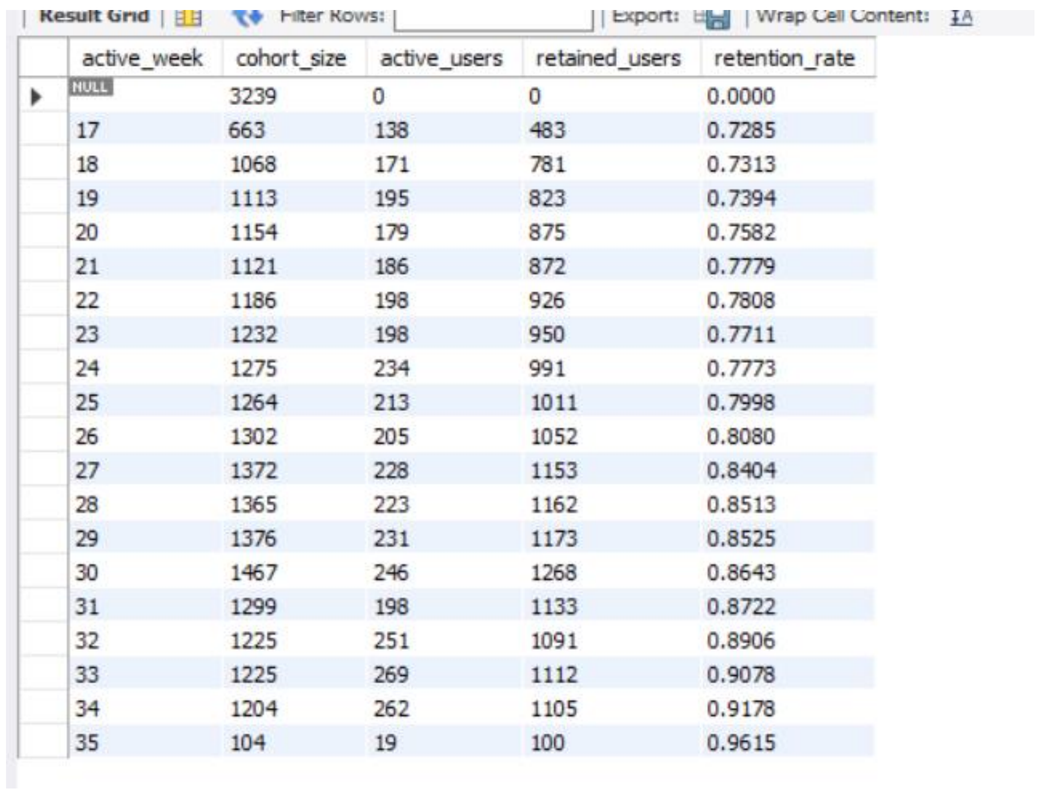
```
SELECT  WEEK(events.occurred_at) AS active_week,
COUNT(DISTINCT users.user_id) AS cohort_size,
COUNT(DISTINCT CASE WHEN WEEK(events.occurred_at) = WEEK(users.created_at)
THEN events.user_id END) AS active_users, COUNT(DISTINCT CASE WHEN
WEEK(events.occurred_at) >=  WEEK(users.created_at) THEN events.user_id END)
AS retained_users, COUNT(DISTINCT CASE WHEN WEEK(events.occurred_at) >=
WEEK(users.created_at) THEN events.user_id END) / COUNT(DISTINCT users.user_id)
AS retention_rateFROM    users LEFT JOIN    events ON users.user_id = events.user_id
GROUP BY    active_weekORDER BY    active_week;
```

CASE STUDY-2: INVESTIGATING METRIC SPIKE

3) 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.

Result/Output:



active_week	cohort_size	active_users	retained_users	retention_rate
NULL	3239	0	0	0.0000
17	663	138	483	0.7285
18	1068	171	781	0.7313
19	1113	195	823	0.7394
20	1154	179	875	0.7582
21	1121	186	872	0.7779
22	1186	198	926	0.7808
23	1232	198	950	0.7711
24	1275	234	991	0.7773
25	1264	213	1011	0.7998
26	1302	205	1052	0.8080
27	1372	228	1153	0.8404
28	1365	223	1162	0.8513
29	1376	231	1173	0.8525
30	1467	246	1268	0.8643
31	1299	198	1133	0.8722
32	1225	251	1091	0.8906
33	1225	269	1112	0.9078
34	1204	262	1105	0.9178
35	104	19	100	0.9615

CASE STUDY-2: INVESTIGATING METRIC SPIKE

4) 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.

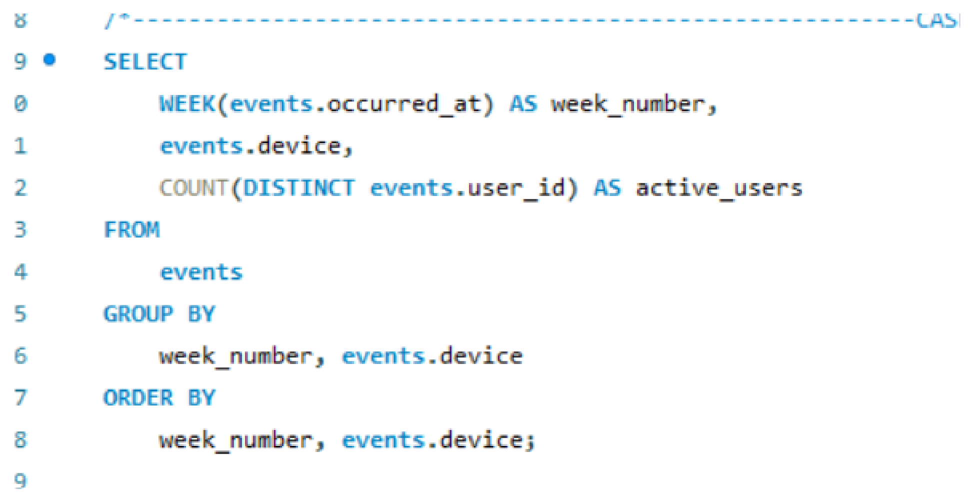
- Firstly, We start by selecting the week number **(calculated from events.occurred_at)** and the device used **(events.device)** from the **events table**.
- Then, Using the **COUNT(DISTINCT events.user_id)** function, we calculate the count of distinct user IDs to determine the number of active users for each specific week and device combination.
- To organize the data for analysis, we apply the **GROUP BY clause**. The results are grouped based on two criteria: **week_number** and **events.device**.
- This groups the data into distinct combinations of weeks and devices, enabling us to analyze user activity patterns more effectively.
- Then, the **week_number** and **events.device** are then ordered using the **ORDER BY** clause.

CASE STUDY-2: INVESTIGATING METRIC SPIKE

4) 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.

Query:



```
/*----------------------------------------------------------CAS
SELECT
    WEEK(events.occurred_at) AS week_number,
    events.device,
    COUNT(DISTINCT events.user_id) AS active_users
FROM
    events
GROUP BY
    week_number, events.device
ORDER BY
    week_number, events.device;
```

SELECT WEEK(events.occurred_at) AS week_number,
events.device, COUNT(DISTINCT events.user_id) AS active_users
FROM events GROUP BY week_number, events.device
ORDER BY week_number, events.device;

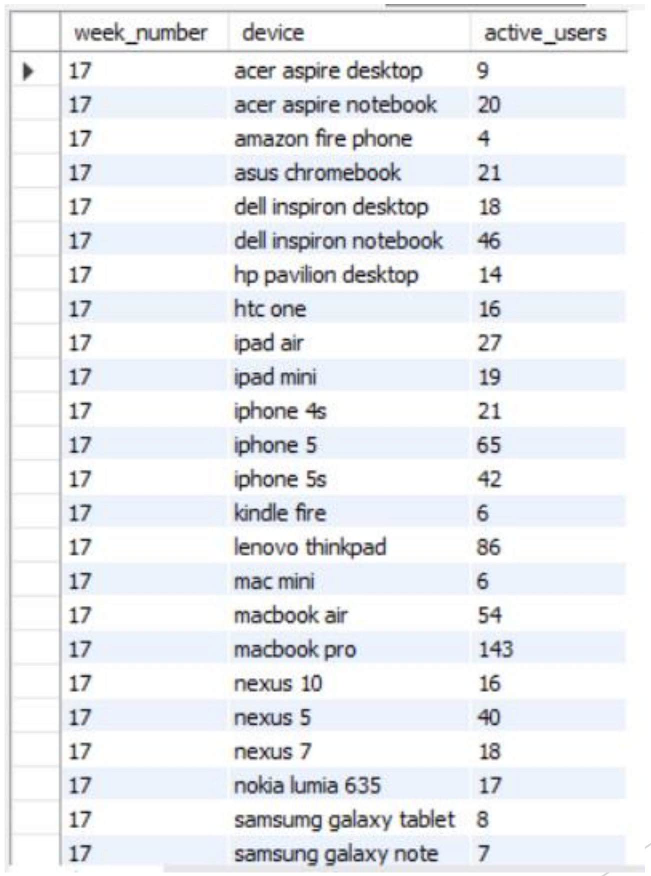
CASE STUDY-2: INVESTIGATING METRIC SPIKE

4) 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.

Result/Output:

So, these first few results. The full results and SQL code are uploaded along with the report. Here, we can observe **macbook pro** device has the highest weekly engagement.



week_number	device	active_users
17	acer aspire desktop	9
17	acer aspire notebook	20
17	amazon fire phone	4
17	asus chromebook	21
17	dell inspiron desktop	18
17	dell inspiron notebook	46
17	hp pavilion desktop	14
17	htc one	16
17	ipad air	27
17	ipad mini	19
17	iphone 4s	21
17	iphone 5	65
17	iphone 5s	42
17	kindle fire	6
17	lenovo thinkpad	86
17	mac mini	6
17	macbook air	54
17	macbook pro	143
17	nexus 10	16
17	nexus 5	40
17	nexus 7	18
17	nokia lumia 635	17
17	samsumg galaxy tablet	8
17	samsung galaxy note	7

CASE STUDY-2: INVESTIGATING METRIC SPIKE

5) Email Engagement Analysis

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

- Firstly, we should extract data from the **email_events table** to analyze user engagement with emails.
- Then, The data is aggregated on a weekly basis using the **WEEK(occurred_at)** function. This groups the **email events** by the week in which they occurred.
- Then, The code utilizes three different types of email actions: **'sent_weekly_digest'**, **'email_open'**, and **'email_clickthrough'**. It counts the distinct number of users for each of these actions within each week.
- This is achieved using conditional statements **(CASE expressions)** to filter and count users based on their **actions**. Using the **group** by function we group the desired results table on the basis of **likes.user_id.**
- The **results** are then **grouped by week** using the **GROUP BY clause** and ordered by week using the **ORDER BY clause**.

CASE STUDY-2: INVESTIGATING METRIC SPIKE

5) Email Engagement Analysis

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

Query:

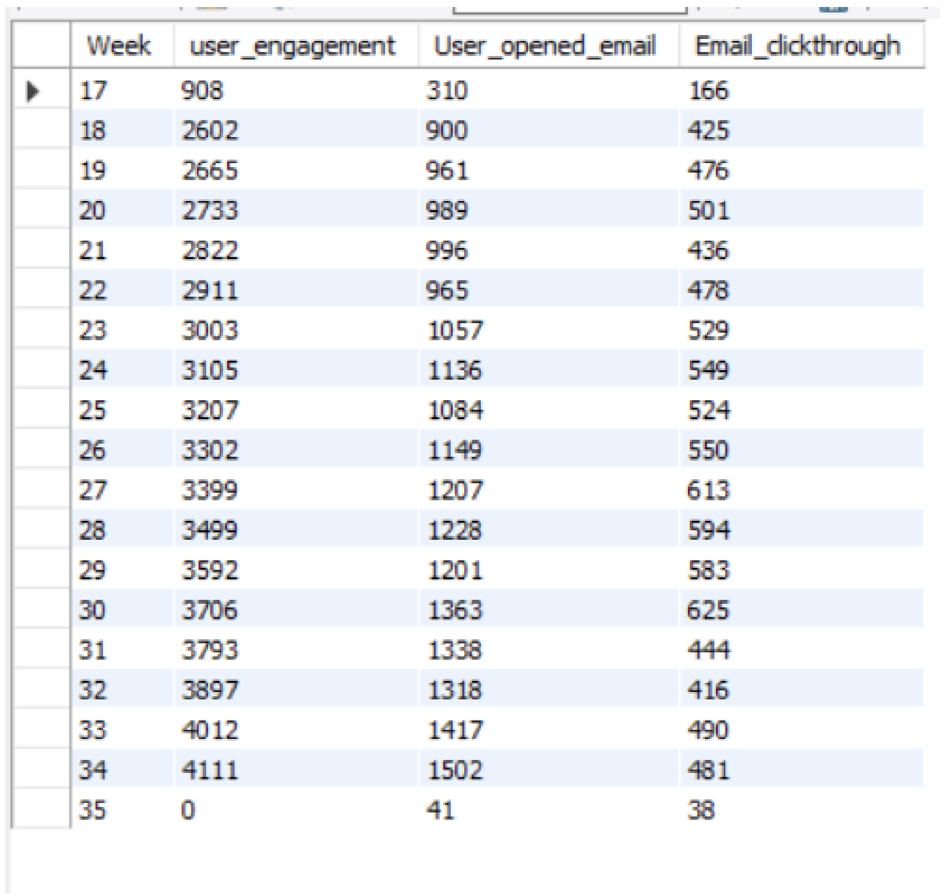
```
SELECT  WEEK(occurred_at) AS Week, COUNT(DISTINCT (CASE
WHEN action = 'sent_weekly_digest' THEN user_id END)) AS
user_engagement, COUNT(DISTINCT (CASE  WHEN action =
'email_open' THEN user_id END)) AS User_opened_email,
COUNT(DISTINCT (CASE WHEN action = 'email_clickthrough'
THEN user_id END)) AS Email_clickthrough FROM  email_events
GROUP BY WEEK(occurred_at)ORDER BY Week;
```

CASE STUDY-2: INVESTIGATING METRIC SPIKE

5) Email Engagement Analysis

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

Result/Output:



Week	user_engagement	User_opened_email	Email_clickthrough
17	908	310	166
18	2602	900	425
19	2665	961	476
20	2733	989	501
21	2822	996	436
22	2911	965	478
23	3003	1057	529
24	3105	1136	549
25	3207	1084	524
26	3302	1149	550
27	3399	1207	613
28	3499	1228	594
29	3592	1201	583
30	3706	1363	625
31	3793	1338	444
32	3897	1318	416
33	4012	1417	490
34	4111	1502	481
35	0	41	38

Conclusion:

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

1. On average 126 jobs, with highest of 218 jobs and lowest of 35 jobs are viewed.
2. I used 7 Day rolling Throughput Analysis as we get the average of all days from the start.
3. The percentage share of Persian language is 37.5% over the last 30 days.
4. There are no Duplicate Rows Detected.

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

1. The most Weekly User Engagement is on 3oth week. It has almost 1467 active users.
2. The User Growth is more in the month of August 2014.
3. The Average Retention rate is 72.85% in 17^{th} week.
4. Macbook Pro has the highest Weekly Engagement Per Device.
5. On average 1061 users opens the Email and only 469 users clicks on the links.

Results:

- With the help of all the tasks which are given in Operation Analytics and Investigating Metric Spike project, I have clearly understood MySQL.
- For completing these tasks, I have used MySQL workbench 8.0
- All the Images, Codes of the results are also provided.

Thank You