



OPERATION ANALYTICS AND INVESTIGATING METRIC SPIKES

PROJECT DESCRIPTION---

Operational Analytics is a crucial process that involves analysing a company's end-to-end operations. This analysis helps identify areas for improvement within the company and shared 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. These questions must be answered daily, making it crucial to understand how to investigate these metric spikes.

APPROACH ---

DATABASE CREATION

Created and inserted the values in the database using DDL & DML SQL queries provided by the Product Manager in the MySQL database using MySQL Workbench.

EXTRACTION OF INSIGHTS

After creating the database required insights are generated from the database table by running SQL queries in MySQL Workbench.

TECH-STACKED USED ---

Used MySQL Community Server which is a free and open source relational database management system that uses SQL.

INSIGHTS ---

While working in the data I came to know that the number of user were increasing in the engagement event with the increase the number of week , this indicates the growth .

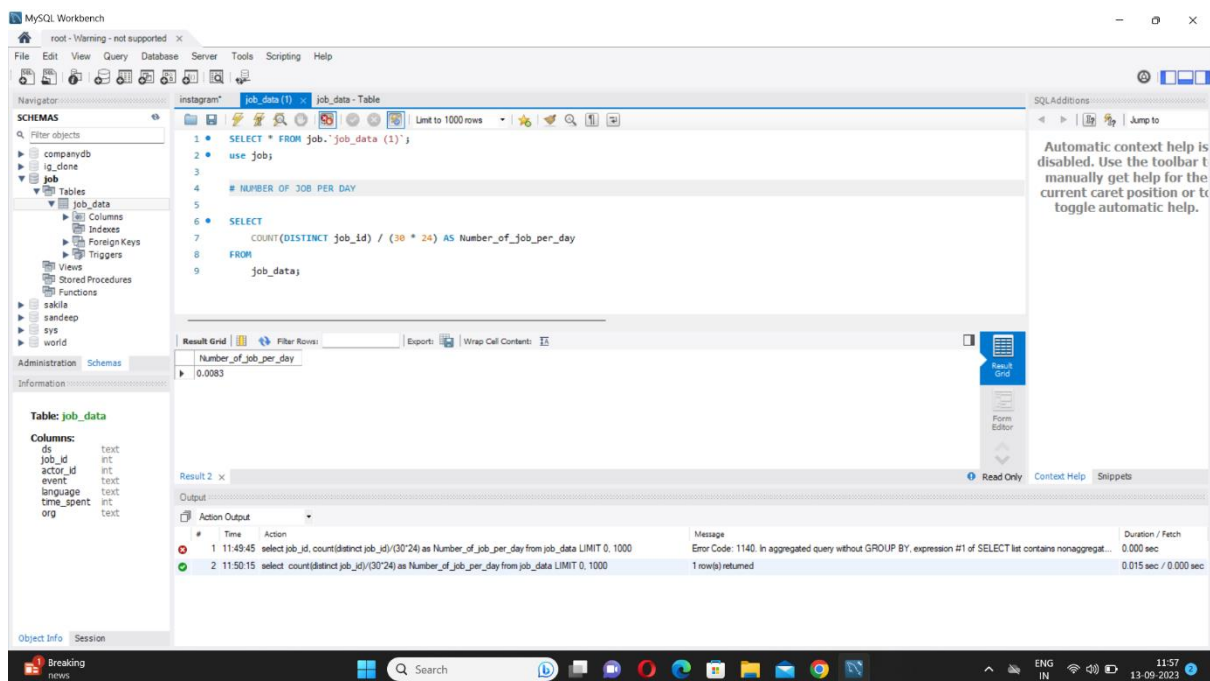
NUMBER OF JOB PER DAY

SELECT

```
COUNT(DISTINCT job_id) / (30 * 24) AS Number_of_job_per_day
```

FROM

```
job_data;
```



7 DAY ROLLING AVERAGE OF THROUGHPUT

with rolling_average as (

select ds, count(job_id) as Number_of_job, sum(time_spent) as Total_time

from job_data

where event in ('transfer','decision')

and ds between '11/01/2020' and '11/30/2020'

group by ds)

select ds, sum(Number_of_job)

over(order by ds rows between 6 preceding and current row) / sum(Total_time)

over(order by ds rows between 6 preceding and current row) as throughput_7days

from rolling_average;

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
with rolling_average as (  
  select ds, count(job_id) as Number_of_job, sum(time_spent) as Total_time  
  from job_data  
  where event in ('transfer','decision')  
  and ds between '11/01/2020' and '11/30/2020'  
  group by ds)  
  
select ds, sum(Number_of_job)  
over(order by ds rows between 6 preceding and current row) / sum(Total_time)  
over(order by ds rows between 6 preceding and current row) as throughput_7days  
from rolling_average;
```

The Results Grid shows the following data:

ds	throughput_7days
11/25/2020	0.0222
11/27/2020	0.0134
11/28/2020	0.0220
11/29/2020	0.0248
11/30/2020	0.0264

The Action Output shows the following message:

```
1 13:35:38 with rolling_average as ( select ds, count(job_id) as Number_of_job, sum(time_spent) as Total_time from job_data where event in ('transfer','decision') and ds between '11/01/2020' and '11/30/2020' group by ds) select ds, sum(Number_of_job) over(order by ds rows between 6 preceding and current row) / sum(Total_time) over(order by ds rows between 6 preceding and current row) as throughput_7days from rolling_average;
```

PERCENTAGE SHARE OF EACH OF LANGUAGE

Select language , count(language) as number_of_language ,

count(language)*100/sum(count(language))

over () as percentage

from job_data

group by language;

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
21 over(order by ds rows between 6 preceding and current row) / sum(Total_time)
22 over(order by ds rows between 6 preceding and current row) as throughput_7days
23 from rolling_average;
24
25 # PERCENTAGE SHARE OF EACH LANGUAGE
26
27 • Select language , count(language) as number_of_language ,
28   count(language)*100/sum(count(language))
29   over () as percentage
30   from job_data
31   group by language;
```

The Results Grid shows the following data:

language	number_of_language	percentage
English	1	12.5000
Arabic	1	12.5000
Persian	3	37.5000
Hindi	1	12.5000
French	1	12.5000
Italian	1	12.5000

The Action Output shows the execution of the query:

```
1 16:34:12 Select language , count(language) as number_of_language , count(language)*100/sum(count(language)) o... 6 row(s) returned
```

DUPLICATE ROWS FROM THE TABLE

WITH duplicate as (

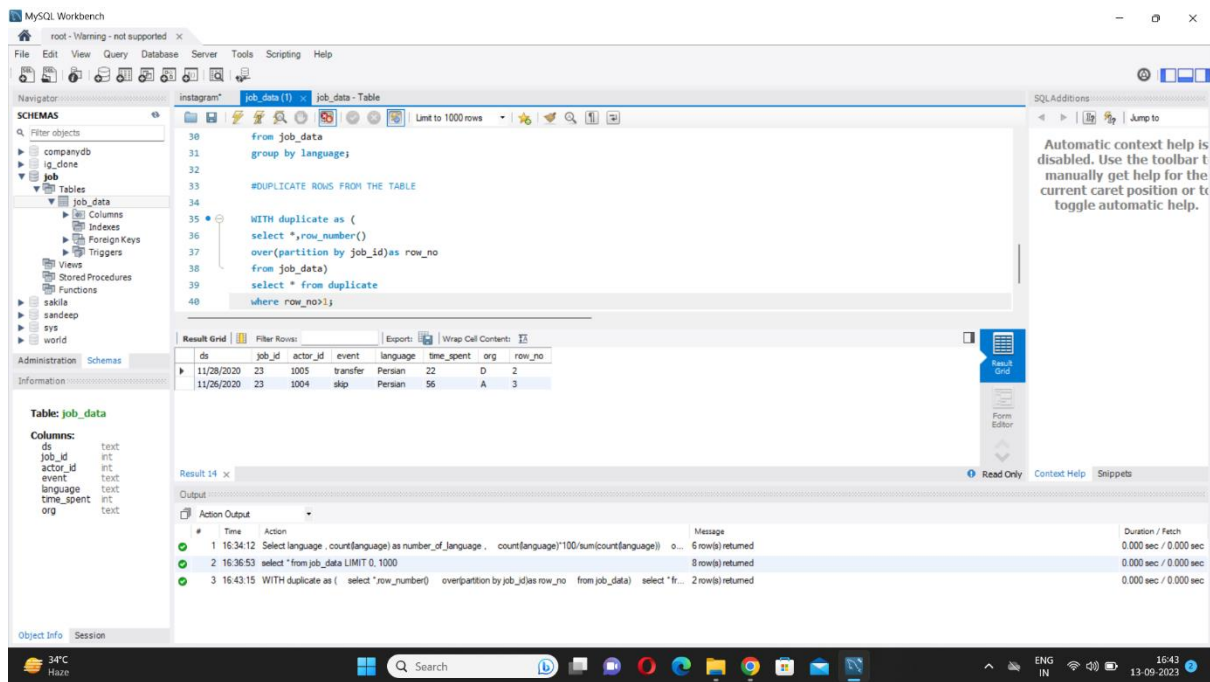
select *,row_number()

over(partition by job_id)as row_no

from job_data)

select * from duplicate

where row_no>1;



INVESTIGATING METRIC SPIKE

WEEKLY USER ENGAGEMENT

SELECT

COUNT(DISTINCT user_id) AS number_of_user,

EXTRACT(WEEK FROM occurred_at) AS number_of_week

FROM events

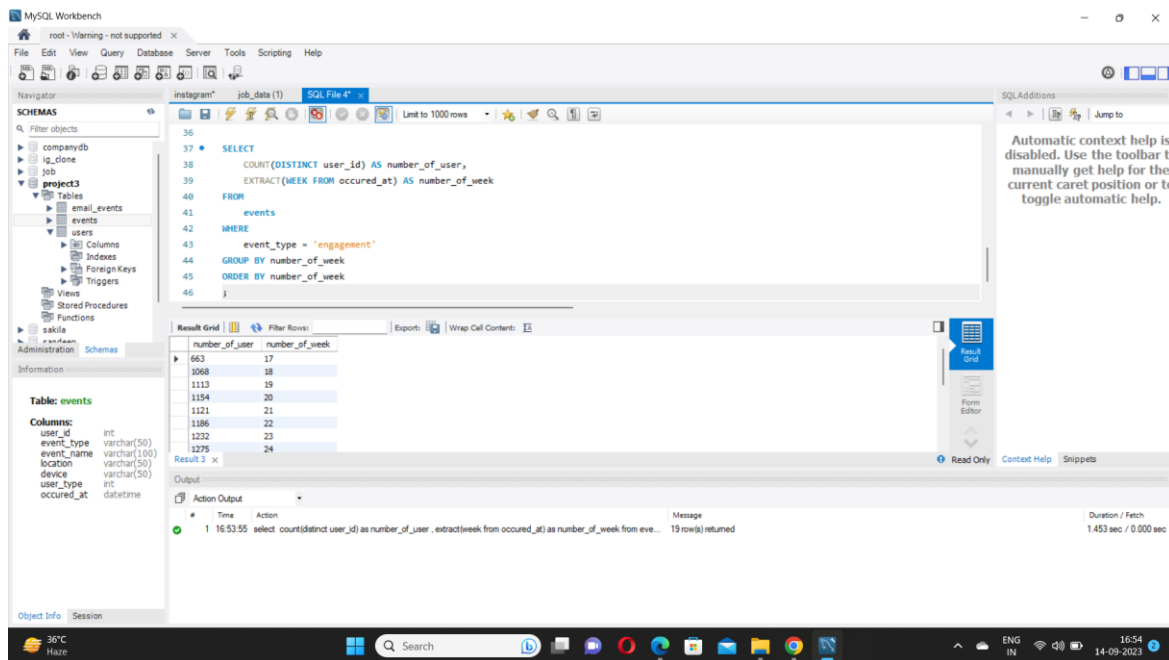
WHERE

event_type = 'engagement'

GROUP BY number_of_week

ORDER BY number_of_week

;

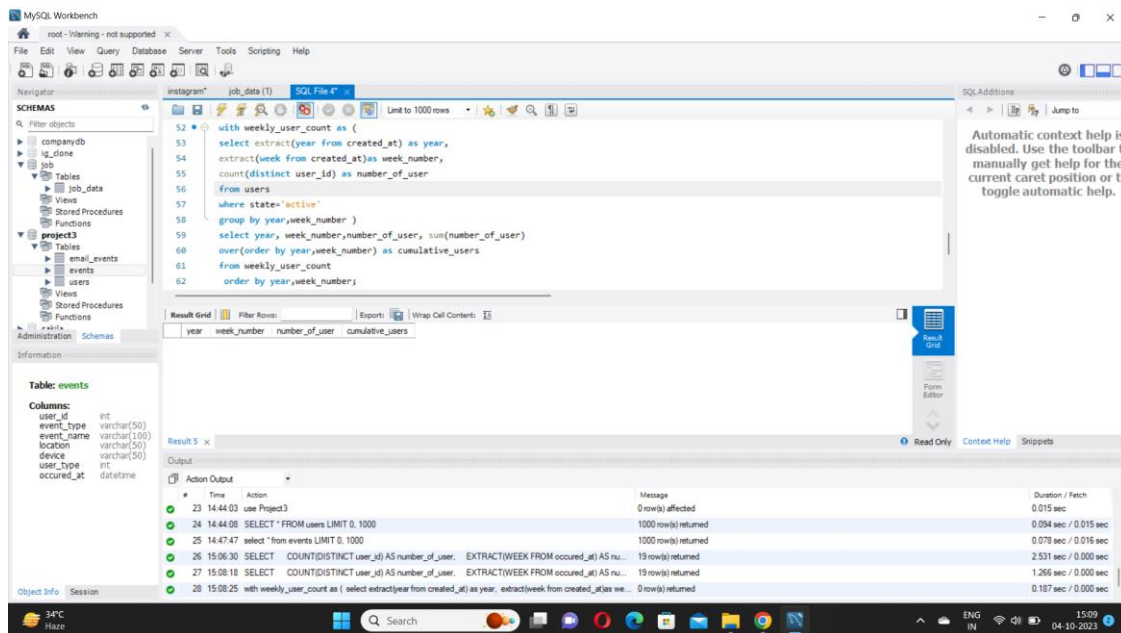


USER GROWTH FOR THE PRODUCT

```

with weekly_user_count as (
select extract(year from created_at) as year,
extract(week from created_at) as week_number,
count(distinct user_id) as number_of_user
from users
where state='active'
group by year, week_number )
select year, week_number, number_of_user, sum(number_of_user)
over(order by year, week_number) as cumulative_users
from weekly_user_count
order by year , week_number ;

```



WEEKLY RETENTION ANALYSIS BASED ON SIGNUP

SELECT

COUNT(user_id) AS total_users,

SUM(CASE

WHEN retention_week = 1 THEN 1

ELSE 0

END) AS per_week_retention

FROM

(SELECT

a.user_id,

a.sign_up_week,

b.engagement_week,

b.engagement_week - a.sign_up_week AS retention_week

FROM

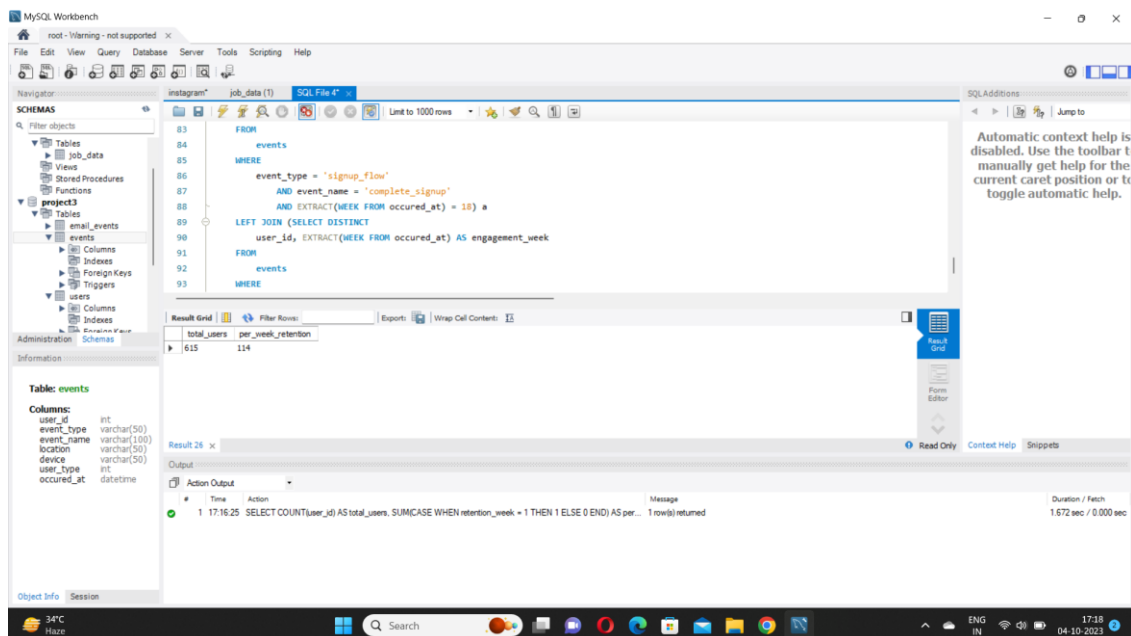
((SELECT DISTINCT

user_id, EXTRACT(WEEK FROM occurred_at) AS sign_up_week

```

FROM
    events
WHERE
    event_type = 'signup_flow'
    AND event_name = 'complete_signup'
    AND EXTRACT(WEEK FROM occurred_at) = 18) a
LEFT JOIN (SELECT DISTINCT
    user_id, EXTRACT(WEEK FROM occurred_at) AS engagement_week
FROM
    events
WHERE
    event_type = 'engagement') b ON a.user_id = b.user_id)
GROUP BY a.user_id , a.sign_up_week , b.engagement_week
ORDER BY a.user_id , a.sign_up_week) subquery
;

```



WEEKLY ENGAGEMENT PER DEVICE

SELECT

EXTRACT(WEEK FROM occurred_at) AS week,

EXTRACT(YEAR FROM occurred_at) AS year,

device,

COUNT(DISTINCT user_id) AS user_count

FROM

events

WHERE

event_type = 'engagement'

GROUP BY week , year , device

ORDER BY week , year , device;

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
SELECT extract(WEEK from occurred_at) as week ,
extract(year from occurred_at) as year,
device , count(distinct user_id) as user_count
from events
where event_type='engagement'
group by week,year,device
order by week, year, device ;
```

The Results tab shows the following data:

week	year	device	user_count
17	2014	acer aspire desktop	9
17	2014	acer aspire notebook	20
17	2014	amazon fire phone	4
17	2014	asus chromebook	21
17	2014	dell inspiron desktop	18
17	2014	dell inspiron notebook	46
17	2014	hp pavilion desktop	14
17	2014	htc one	16

The Output tab shows the execution log:

Time	Action	Message	Duration / Feels
1 17:16:25	SELECT COUNT(user_id) AS total_users, SUM(CASE WHEN retention_week = 1 THEN 1 ELSE 0 END) AS per...	1 row(s) returned	1.672 sec / 0.000 sec
2 17:21:06	select * from events LIMIT 0, 1000	1000 row(s) returned	0.016 sec / 0.000 sec
3 17:29:30	SELECT extract(WEEK from occurred_at) as week , extract(year from occurred_at) as year, device, count(distinc...	491 row(s) returned	0.640 sec / 0.000 sec
4 17:30:10	SELECT extract(WEEK from occurred_at) as week , extract(year from occurred_at) as year, device, count(distinc...	491 row(s) returned	0.641 sec / 0.000 sec

EMAIL ENGAGEMENT ANALYSIS

SELECT

SUM(action = 'email_open') / SUM(action = 'sent_weekly_digest') AS email_opening_rate,

```

SUM(action = 'email_clickthrough') / SUM(action = 'sent_weekly_digest') AS email_clicking_rate
FROM
email_events;

```

The screenshot shows the MySQL Workbench interface. The central pane displays a SQL query for email engagement analysis. The query is as follows:

```

112 GROUP BY week, year, device;
113 ORDER BY week, year, device;
114
115 # EMAIL ENGAGEMENT ANALYSIS
116
117
118 SELECT
119     SUM(action = 'email_open') / SUM(action = 'sent_weekly_digest') AS email_opening_rate,
120     SUM(action = 'email_clickthrough') / SUM(action = 'sent_weekly_digest') AS email_clicking_rate
121 FROM
122     email_events;

```

The Results pane shows the output of the query, which is a single row with two columns: `email_opening_rate` and `email_clicking_rate`. The values are 0.3573 and 0.1573 respectively.

email_opening_rate	email_clicking_rate
0.3573	0.1573

The left sidebar shows the database schema, including tables like `email_events`, `job_data`, and `users`. The bottom status bar shows the system time as 17:45 on 04-10-2023.