PROJECT III

OPERATION ANALYTICS AND INVESTIGATING METRIC SPIKE

PROJECT DESCRIPTION

The project has two case studies that deal with analyzing two key aspects of operation analytics and investigating metric spikes. This process is very crucial for businesses, using these analyses they can improve their strategies and performance level. As a data analyst, one has to closely monitor the ups and downs in the metrics, derive valuable insights, understand the reason behind them, and report them to the team. Therefore the key purpose of this project is to help these companies so that they can work in fields where they lack improvement and accelerate their growth.

I'm planning to use my advanced SQL skills to perform analysis based on the questions raised by various departments and provide valuable insights that can help the company's operations.

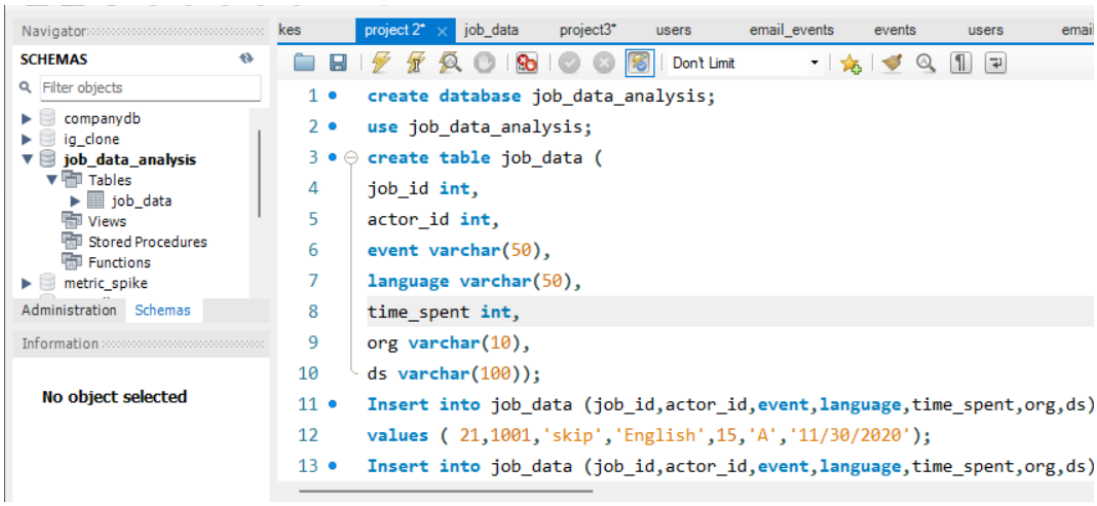
APPROACH

There are two case studies in this project.

Case 1 deals with Job data analysis **Case 2 deals with Investigating Metric Spike**

Case Study 1: In Job data analysis there is only one job data table which involves analyzing job data to improve the efficiency in the operations field. Four tasks are to be investigated in the job data table. The tasks are to monitor how many jobs are reviewed, analyze the throughput, and percentage share of languages, and identify whether there are any duplicate rows. These are the tasks to be carried out for the analysis of job_data.

The first thing to be done before running the queries is to create a database. I created a 'job_data_analysis' database and a 'job_data' table.

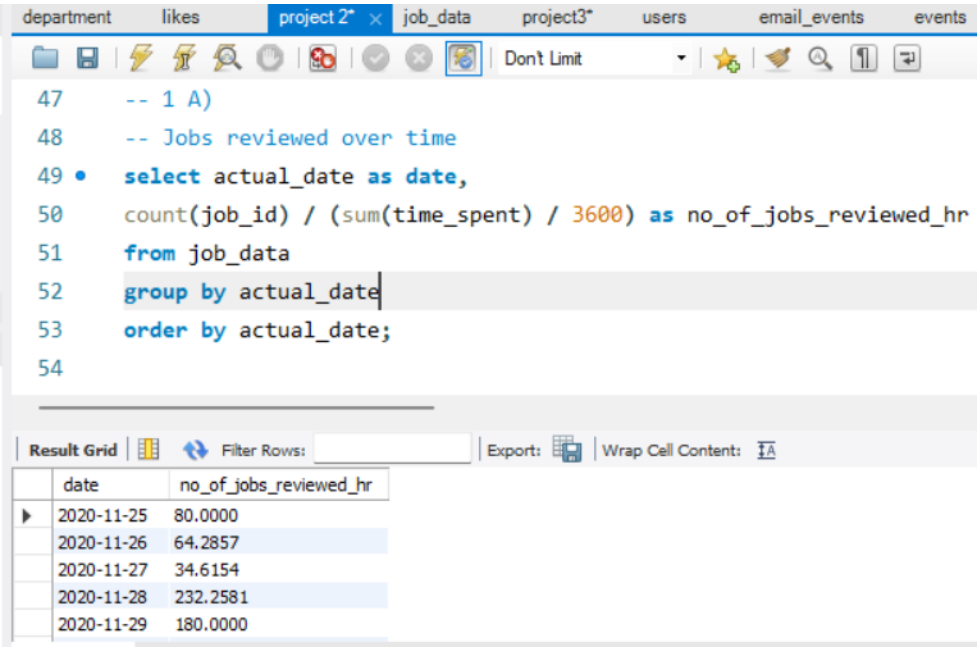


1 A) Jobs Reviewed Over Time

The first task was to calculate the number of jobs reviewed per hour for each day in November 2020.

In the data given the time_spent was given in seconds. I used a mathematical equation To convert it into hours as mentioned in the task. After running the queries I found that some days in November more jobs are reviewed and some days very few are viewed per hour.

I have attached the SQL queries below:



In this picture, the most number of jobs were viewed on 28th November and only 34 jobs were viewed on 11th November.

1 B) Throughput Analysis:

The second task was to calculate the 7-day rolling average for throughput and explain whether I prefer daily metric or the 7-day rolling average throughput and why.

To perform the second task one must know what a 7-day rolling average is. It is an average taken over the last 7 days i.e. for a given day calculate the average of that particular day and the preceding six days. Using this approach one can observe any trend that occurs over time because it smooths out short-term fluctuations.

First I used a subquery to find the throughput value, it calculates the number of jobs per time for each day. In the main query to find the 7-day rolling average I used a Window function 'average () over()'.The condition was passed in the query and further grouped the results by the 'actual_date column'.

In the daily metric i.e. throughput, it provides a direct day-to-day metric that includes short-term fluctuations. Whereas the 7-day-average-throughput provides a week-long trend of the daily metric. In the end, it depends on the objective, If one wants to analyze immediate trends then the daily-metric approach is the better fit if one wants to observe the trend for a

long time then the rolling average method is the best because it smooths out the short-term fluctuations. I prefer the 7-day rolling average method for this task because it provides more informed decisions and stabilization.

The following queries were used to calculate the throughput and 7-day Rolling average of the throughput.

```
order by actual_date;

 -- 1 B)
 -- Throughput Analysis
 -- 7day rolling avg
SELECT a.actual_date ,
a.throughput,
avg(a.throughput) over ( order by a.actual_date  rows between 6 preceding and current row ) as
7_day_rolling_avg
from (
select actual_date, count(job_id) / sum(time_spent) as throughput
 from job_data group by actual_date ) as a;

-- 1 C)
-- Language Share Analysis
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

actual_date	throughput	7_day_rolling_avg
2020-11-25	0.0222	0.02220000
2020-11-26	0.0179	0.02005000
2020-11-27	0.0096	0.01656667
2020-11-28	0.0645	0.02855000
2020-11-29	0.0500	0.03284000
2020-11-30	0.0500	0.03570000

The throughput and & 7-day rolling average throughput are calculated as required by the department.

One can see the difference between throughput and the 7-day rolling average.

1 C) Language Share Analysis

The third task of this project was to calculate the percentage share of each language over the last 30 days.

To run this task I used the language column from the job_data table. To find the percentage share divide the count of each language by the total count and multiply by 100. I used the where clause which considers only the last 30 days and then used the group by and order by clauses to sort the query.

After running the queries it was observed that the Persian language had the highest percentage share compared to the other languages

The query below calculates the percentage share of each language in the job_data table.

```
group by a.actual_date;

-- 1 C)
-- Language Share Analysis
 select language, count(language) as language_count ,
    round(100*count(*)/(select count(*) from job_data),2) as percentage_share
from job_data
where actual_date between '2020-11-01' and '2020-11-30'
group by language
order by language desc;

-- alternate
```

language	language_count	percentage_share
Persian	3	37.50
Italian	1	12.50
Hindi	1	12.50
French	1	12.50
English	1	12.50
Arabic	1	12.50

One can observe that the Persian language has the highest percentage share of 37.50% and other languages are equally balanced out with 12.50%

1D) Duplicate Rows Identification

The final task of this case study was to display duplicate rows from the job_data table.

It was mentioned in the attribute description that job_id and actor_id uniquely identifies jobs and actor. Therefore I did this task in three parts, First I only considered the actor_id attribute, to find whether there were any duplicate actors with the same actor_id.

This is what I observed when I ran the query to find the duplicates in actor_id.

```
-- 1 D)
-- Duplicate Row Detection
-- If the actor_id attribute alone is considered then there are two duplicates
select * from job_data
where (actor_id) in (
    select actor_id
    from job_data
    group by actor_id
    having count(*) > 1
)
order by actor_id;
-- If the job_id attribute alone is considered then there are three duplicates
```

job_id	actor_id	event	language	time_spent	org	actual_date
20	1003	transfer	Italian	45	C	2020-11-25
23	1003	decision	Persian	20	C	2020-11-29
NULL	NULL	NULL	NULL	NULL	NULL	NULL

One can observe that there are two duplicate rows in the actor_id. The duplicate actor_id is 1003.

Then I considered only the job_id attribute since that is also unique. In the job_id attribute also there are duplicates. I used a subquery to find if there are any duplicate rows in the job_id column.

```
)
order by actor_id;
-- If the job_id attribute alone is considered t
SELECT *from job_data
where (job_id) in (
    select job_id
    from job_data
    group by job_id
    having COUNT(*) > 1
)
order by job_id;
```

job_id	actor_id	event	language	time_spent	org	actual_date
23	1003	decision	Persian	20	C	2020-11-29
23	1004	skip	Persian	56	A	2020-11-26
23	1005	transfer	Persian	20	C	2020-11-28

There were duplicate rows in the job_id attribute. Three actors had the same job_id (23).

Finally, I considered all the attributes at once to find if there were any duplicates.

There were no duplicate rows when the entire row was considered.

```
    from job_data
    group by job_id
    having COUNT(*) > 1
)
order by job_id;

-- If the entire row is considered then there are no duplicates
select * from job_data
group by job_id, actor_id, event, language, time_spent, org, actual
having count(*) > 1;
```

job_id	actor_id	event	language	time_spent	org	actual_date

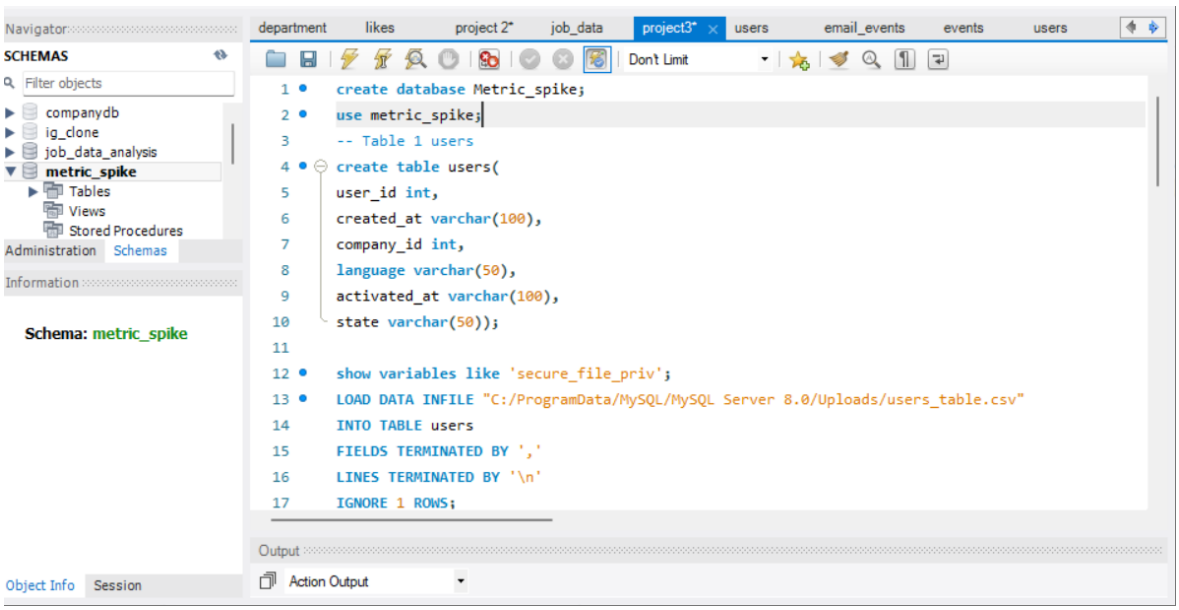
There are no duplicate rows when the entire row is considered.

To conclude, using various SQL queries the completion of the above tasks was carried out successfully. These insights can be useful for the company to improve its growth by rectifying the changes in fields where it particularly needs to grow.

Case study 2: In investigating metric spike there were three tables users, events, and email_events table. In case study 2 there were 5 tasks to be investigated. The tasks included checking the activeness of users, analyzing the growth of users over time, cohort-based analysis, and email engagement analysis. The aim is to analyze the data carefully, draw valuable insights, and understand the sudden change in metrics. These insights are crucial in improving the company's operations.

A database was created to import the metrics data. Then three tables were also created in the databases. Since the dataset was huge "Load Data Infile" command was used to import the data from Excel to the database.

These were the commands used to create the Metric_spike database and three tables.



2A) Weekly User Engagement:

The first task to perform in the case study was to calculate the weekly user engagement.

To perform this task I made use of the events table in the metric_spike database. Before that, the date in the 'occurred_at' attribute was in text format. So after importing the dataset I changed the string formatted text into date format and carried out my analysis. Since the task was to calculate user engagement every week, I made use of the 'week' command to extract the week from the occurred_at attribute. Therefore occurred_at and user_id attributes were retrieved from the events table to analyze the weekly user engagement and further sorted them using group by and order by clauses.

The following queries were used to find the weekly user engagement

```
alter table email_events change column temp_occured_at  occured_at Datet

-- Investigating Metric Spike
 -- 2 A)
-- Weekly User Management
select * from events;
select
week(occured_at) AS week_no,count(distinct user_id) as user_engagement
from
    events
group by week_no
order by week_no ;
-- 2 B)
-- User Growth Analysis
```

week_no	user_engagement
17	663
18	1068
19	1113
20	1154
21	1121
22	1186

After running the queries it was observed that Week 30 had the highest count of user engagement of 1467, and the lowest count was 105 which was observed in Week 35.

2B) User Growth Analysis

The second task was to calculate the user growth for the product over time.

In this task, we need to find the growth of users for a product over time. Since it's not mentioned to calculate it weekly or monthly or daily. I decided to calculate on a daily as well as on a monthly basis. To calculate it daily a subquery using the CTE (Common Table Expression) was created. In the CTE date was extracted from the 'created_at' attribute, using the count function no of distinct users were also calculated, and then a window function called 'lead' was used to calculate the users of the next day. In that way, we can analyze the user growth daily. Then in the main query, we calculated the user growth by the difference of users on the current day and the next day, and % of user growth was also calculated.

The following queries were used to perform the second task.

```
-- 2 B)
-- User Growth Analysis
-- Here we considered the user growth daily
with DailyUsers as (
select
    date(created_at) as day,
    count(distinct user_id) as daily_new_users,
    lead(count(distinct user_id)) over (order by  date(created_a
from
    users
group by
    date(created_at))
```

day	daily_new_users	next_day_users	user_growth	growth_percentage
2013-01-01	7	7	0	0.0000
2013-01-02	7	6	-1	-14.2857
2013-01-03	6	1	-5	-83.3333
2013-01-04	1	2	1	100.0000
2013-01-05	2	3	1	50.0000
2013-01-06	3	4	1	33.3333

The highest user growth was 41 and it was observed on 2014-08-24.

Similarly, if we want to calculate user growth every month then follow the steps except use the month function to extract the month from the created_at attribute.

```
order by
    day;
    -- 2B
    -- Here we considered the user growth monthly

    with monthlyUsers as (
    select
        month(created_at) as month,
        count(distinct user_id) as monthly_new_users,
        lead(count(distinct user_id)) over (order by  month(
    from
        users
```

month	monthly_new_users	next_monthly_users	user_growth	growth_percentage
1	712	685	-27	-3.7921
2	685	765	80	11.6788
3	765	907	142	18.5621
4	907	993	86	9.4818
5	993	1086	93	9.3656
6	1086	1281	195	17.9558

The highest user growth was observed in the 6th month where the user growth spiked to 195.

The lowest growth was observed in the 8th month where user growth plummeted to 1017.

We can also consider it on a weekly basis then the 31st week shows the highest user growth.

2 C) Weekly Retention Analysis

The third task is to calculate the weekly retention of users based on their sign-up cohort.

To perform the above task one has to break it down into bits. I used a subquery to store the weekly activity of events. In this query, we can find how each event is associated with the week. In the weekly activity subquery, I used a few window functions such as partition by and case to correctly identify the user's signup on the specific event. A min function was used because if all more events were associated with the users then only one sign-up activity would be considered. I joined the events and the users table using the join clause. Another subquery was used for the signup cohort. The main use of this CTE is to find the size of each weekly sign_up_cohort.In the main query, we use the retention formula to find the retention rate and we also join the CTE to find the weekly retention analysis and further it grouped by signup_week.

The formula used to calculate the retention rate is =

(No of users in the end - No of users during the event)/(No of users in the starting period of the event))*100

The following queries were used to perform the third task.

```
select
    wa.signup_week,
    wa.activity_week,
    count(distinct wa.user_id) as active_users,
    sc.users_in_cohort,
    (count(distinct wa.user_id)  / sc.users_in_cohort) as rete
    (count(distinct wa.user_id) * 100.0 / sc.users_in_cohort)
from WeeklyActivity wa
join SignupCohorts sc on wa.signup_week = sc.signup_week
```

signup_week	activity_week	active_users	users_in_cohort	retention	retention_rate
17	17	138	138	1.0000	100.00000
17	18	103	138	0.7464	74.63768
17	19	59	138	0.4275	42.75362
17	20	38	138	0.2754	27.53623
17	21	26	138	0.1884	18.84058

One can observe the weekly user retention for each group of users.

One can observe various ranges of retention rates in the subsequent weeks. In some weeks the rates are really low and in some weeks it's high. This can be due to various factors.

2 D) Weekly Engagement per Device

The fourth task was to calculate the weekly engagement per device.

To find the weekly engagement per device, I used the events table. I extracted the week from the occurred_at attribute and found the distinct users and devices from the events table. I used the 'where ' condition to select only the events with "Engagement" and then sorted them by group by and order by clauses.

The following queries were used to perform the fourth task.

```
where wa.signup_week is not null
group by wa.signup_week, wa.activity_week, sc.
order by wa.signup_week, wa.activity_week;
-- 2D)
-- Weekly Engagement per device
Select week(occured_at) as week,device,
count(distinct user_id) as users
 from events
 where event_type='Engagement'
 group by week,device
 order by week,device;
 -- 2E)
 -- Email Engagement Analysis
select
```

Result Grid | Filter Rows: | Export: | Wrap

week	device	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

Mac Book Pro had the highest weekly engagement per device with 252 users using it on the 18Th week.

2 E) Email Engagement per Device

The final task of this case study is to calculate the email engagement metrics.

To perform this task I used the email_events table. In the email_events table, the action attribute has four categories. These four actions' weekly engagement can be calculated using the case function and then grouped by using the group by and order by clauses.

The following queries were used to find the email engagement action per device

```
count(distinct user_id) as users
 from events
 where event_type='Engagement'
 group by week,device
 order by week,device;
 -- 2E)
 -- Email Engagement Analysis
select
    WEEK(occured_at) as week,
    count(distinct case when action = "sent_weekly_digest" then user_id end) as Weekly_dige
    count(distinct case when action = "sent_reengagement_email" then user_id end) as reenga
    count(distinct case when action = "email_open" then user_id end) as opened_email,
    count(distinct case when action = "email_clickthrough" then user_id end) as email_click
from email_events
```

week	Weekly_digest	reengagement_mail	opened_email	email_clickthrough
17	908	73	310	166
18	2602	157	900	425
19	2665	173	961	476
20	2733	191	989	501
21	2822	164	996	436
22	2911	192	965	478

This query gives the Weekly engagement actions per device every week.

```
group by week,device
order by week,device;
-- 2E)
-- Email Engagement Analysis
Select avg(weekly_digest) as AVG_WEEKLY_DIGEST,avg(reengagement_email) as AVG_REENGAGEMENT_EM
(select
    WEEK(occured_at) as week,
    count(distinct case when action = "sent_weekly_digest" then user_id end) as Weekly_digest,
    count(distinct case when action = "sent_reengagement_email" then user_id end) as reengagem
```

AVG_WEEKLY_DIGEST	AVG_REENGAGEMENT_EMAIL	AVG_OPENED_MAIL	AVG_EMAIL_CLICK_THROUGH
3014.0526	192.2632	1061.1579	469.3684

From this query, one can notice that Weekly Digest has the highest weekly engagement with an average of 3014

TECH-STACK USED

I used MySQL workbench version 8.0.34 build 3263449 CE(64 bits) for this project. The reason I used MySQL workbench for my project is because I was more comfortable using this software. I practiced my SQL queries mostly in this software. MySQL workbench is user-friendly and is also open-source.

INSIGHTS

This project is the most challenging one I have done so far. To do this project one should possess a strong knowledge of SQL concepts. Once you dive into the data and get to

understand more about it, then it gets more and more interesting. Each task was challenging and nerve-racking in its way. I'm glad for this opportunity wherein I got to apply my SQL knowledge in a Live project.

The key insights I found while working on projects are listed below

In Case Study 1

- The average number of jobs reviewed per hour per day in November alone is 128 and most number of jobs were reviewed on 28th Nov.
- The 7-day rolling average of throughput is better than the daily metric if you want to observe any long-term trend.
- Persian language is the most used language with a share of 37.5%
- If we consider the overall rows to find duplicates then there are none, but there are duplicate rows in the actor_id and job_id attribute if we consider them individually.

In Case Study 2

- There's a spike in the initial weeks but towards the end, it takes a dip. There are many possible reasons for such occurrences. One such reason can be due to the lack of satisfaction with the product.
- The sixth month has the highest user growth
- The weekly retention varies each week
- Most of the users use MacBook Pro on a weekly basis
- Weekly Digest emails have the highest weekly engagement.

RESULT

It was indeed a great experience for me as I got hands-on experience in working with the Live projects. This is my second project working in SQL, I had to work extra on so many concepts to tackle the challenges faced during this project. This in turn helped me to gain more in-depth knowledge of SQL concepts and I believe this project helped me to hone my SQL skills to a great extent. I gained a lot of knowledge and understanding while working on this Operation analytic and investigating metric spike project

END

Submitted by,

S Nandana.