

A PROJECT ON OPERATION & METRIC ANALYTICS



By – Vansh

Mail ID – aroravansh11@gmail.com

SQL TASKS

Case Study 1: Job Data Analysis

Tasks:

A) Jobs Reviewed Over Time:

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

Your Task: Write an SQL query to calculate the number of jobs reviewed per hour for each day in November 2020.

Solution:

Query:

```
# Write an SQL query to calculate the number of jobs reviewed per hour for each  
# day in November 2020.
```

```
• select  
    date(ds) as review_date,  
    hour(from_unixtime(unix_timestamp(ds) + time_spent)) as review_hour,  
    count(*) as jobs_reviewed  
from  
    job_data  
where  
    ds between '2020/11/01' and '2020/11/30'  
    and event = 'decision'  
group by  
    review_date,  
    review_hour  
order by  
    review_date,  
    review_hour;
```

Output:

	review_date	review_hour	jobs_reviewed
▶	2020-11-27	0	1
	2020-11-28	0	1
	2020-11-29	0	1

Explanation:

Here,

- In this task firstly I extracted date part from “**ds**” as “**review_date**”
- Then using hour function I extracted the hour part from this datetime which tells the hour when the job review was completed and i used “**unix_timestamp**” for converting the date “**ds**” to a **unix_timestamp** , another function that is “**from_unixtime**” which I have used for converting this result timestamp back to a datetime format
- After that “**count**” function I have used for counting the total number of jobs reviews
- “**Job_data**” table is used
- “**Where clause**” is used for mentioning the conditions like to filter the records for the November month which is accomplished using **between** clause, which specified a range for a month and for counting only job reviews **event** is set to **decision**
- Then **group by** is used for the count of job reviewed is calculated for each hour of each day and **order by** is used

ordering the result set into ascending or descending format

→ So, finally we got the “**jobs_reviewed**” per hour for each day in November 2020 in the result set.

And **review_hour** contains 0 value in every row which means computed hour is midnight according to “from_unixtime” and “unix_timestamp”

B) Throughput Analysis:

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

Your 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.

Solution:

Query:

Write an SQL query to calculate the 7-day rolling average of throughput.

```
SELECT j1.ds AS ds,  
       COUNT(*) AS events,  
       COUNT(*) / SUM(j1.time_spent) AS avg_throughput  
FROM job_data j1  
INNER JOIN job_data j2 ON j1.ds >= DATE_SUB(j2.ds, INTERVAL 6 DAY)  
WHERE j1.ds = j2.ds  
GROUP BY j1.ds  
ORDER BY j1.ds;
```

Output:

	ds	events	avg_throughput
►	2020/11/25	1	0.0222
	2020/11/26	1	0.0179
	2020/11/27	1	0.0096
	2020/11/28	4	0.0606
	2020/11/29	1	0.0500
	2020/11/30	4	0.0500

Explanation:

Here,

- Firstly, I selected “**ds**” column and used **count** function for counting the number of events for each day and then used **sum** function for calculating the total time spent reviewing the jobs for the 7-day window.
- Data is extracted **from job_data** table which is given a name i.e. “**j1**”
- After that I used **Inner Join** for self-joining two tables from a single table i.e. “**j1**” and “**j2**”. This is done on a specific condition which includes
 - Main j1 table
 - **Date_Sub** function which I used for subtracting 6 days from the **ds** value in the j2 table.
 - **>=** greater than equal to operator is used for ensuring **j1.ds** falls within the 7-day window created by **Date_sub**.

- **Where** clause I just used for additional clarity
- **Group by clause** is used for grouping the results by **ds** column from main table **j1**.
- **Order by clause** used for arranging the results by ascending order.

Additionally, explain whether you prefer using the daily metric or the 7-day rolling average for throughput, and why.

From my point of view, we should not go for 1 method for every single task, as in some cases the other method can be the most efficient one to use.

So, I will recommend using these both methods according to Problems/Tasks that what actually we want to extract from data.

- Like for clearer picture of overall throughput trends over time and for weekly variations in throughput we can use **7-day Rolling average method**
- **Daily Metric** can be used for potential fluctuations throughout the day and for identifying sudden spikes or dips in throughput.

C) Language Share Analysis:

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

Your Task: Write an SQL query to calculate the percentage share of each language over the last 30 days.

Solution:

Query:

```
# Calculate the percentage share of each language in the last 30 days.
```

```
• SELECT language,
      ROUND(100.0 * COUNT(*) / total_count, 2) AS percentage_share
FROM (
    SELECT language, COUNT(*) AS job_count
    FROM job_data
    WHERE ds >= DATE_SUB(CURDATE(), INTERVAL 30 DAY)
    GROUP BY language
) AS job_data_30days
CROSS JOIN (
    SELECT COUNT(*) AS total_count
    FROM job_data
    WHERE ds >= DATE_SUB(CURDATE(), INTERVAL 30 DAY)
) AS total_jobs
GROUP BY language, total_count
ORDER BY percentage_share DESC
LIMIT 0, 1000;
```

Output:

Result Grid			Filter Rows:
	language	percentage_share	

Explanation:

Here,

- In the First Subquery i.e. “**job_data_30days**” I Selected the **language** column and **count** function is used for counting the number of jobs for each unique language value.
 - Data is extracted from “**job_data**” table
 - Then **where** clause is used for specifying conditions in which **ds >= Date_sub(..)**, “**Date_sub**” function is used for calculating a date 30 days before the current date and **>=** is used for ensuring that ds come under last 30 days relative to the current date.
 - **Group by** clause is used for grouping the result based on **language** column
- After 1st Subquery I used **Cross join** for getting cartesian product by combining two tables or we can say two virtual tables which are created for calculations i.e. “**job_data_30days**” and “**total_jobs**”

- **Then 2nd Subquery** I used for calculating total number of jobs reviewed in the last 30 days i.e. “**total_jobs**”
- **Now for Percentage Share Calculation:**
 - I used **select** clause outside the subqueries in which I used **round** function for rounding the percentage value to 2 decimal places, **count** function is used for counting the number of jobs for each unique language value
 - **Total_count** I used for representing the total number of jobs reviewed in the last 30 days

→**So, after applying multiplication and division** we come to a point that, there is no **percentage share of each language over the last 30 days. As we didn't get any value in the result set.**

(P.T.O)

D) Duplicate Rows Detection:

Objective: Identify Duplicate Rows in the data

Your Task: Write an SQL query to display duplicate rows from the `job_data` table.

Solution:


Query:


```
# Write an SQL query to display duplicate rows from the job_data table.
```

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

Output:


Result Grid





Filter Rows:

Export:



Wrap

	ds	job_id	actor_id	event	language	time_spent	org
--	----	--------	----------	-------	----------	------------	-----

Explanation:

Here,

- Firstly, I used **Select clause** for selecting all column **from** the “**job_data**” table
- Then I used **Group by** for grouping the rows based on all columns in the table and this ensures that rows are

considered duplicates only if they have same values in every column

- After Group by I used **Having clause** and with that **count function** for identifying groups containing duplicate rows.

→**So**, I got No result in the result set that means there is not even a single duplicate row in the “**job_data**” table.

Case Study 2: Investigating Metric Spike

Tasks:

A) Weekly User Engagement:

Objective: Measure the activeness of users on a weekly basis

Your Task: Write an SQL query to calculate the user growth for the product.

Solution:

(P.T.O)



Query:

Write an SQL query to calculate the weekly user engagement.

```
WITH weekly_active_users AS (  
    SELECT  
        DATE_SUB(DATE(STR_TO_DATE(e.occurred_at, '%d-%m-%Y %H:%i')),  
            INTERVAL WEEKDAY(STR_TO_DATE(e.occurred_at, '%d-%m-%Y %H:%i')) DAY) AS week_start,  
        COUNT(DISTINCT e.user_id) AS active_users  
    FROM  
        events e  
    GROUP BY  
        week_start  
) ,  
total_users AS (  
    SELECT  
        COUNT(*) AS user_count  
    FROM  
        users  
)  
SELECT  
    w.week_start,  
    w.active_users,  
    ROUND((w.active_users / tu.user_count) * 100.0, 2) AS engagement_percentage  
FROM  
    weekly_active_users w  
INNER JOIN  
    total_users tu  
ORDER BY  
    w.week_start DESC;
```

(P.T.O)

Output:

Result Grid  Filter Rows: <input type="text"/> Export: 			
	week_start	active_users	engagement_percentage
▶	2014-08-25	1194	12.73
	2014-08-18	1203	12.82
	2014-08-11	1215	12.95
	2014-08-04	1266	13.50
	2014-07-28	1443	15.38
	2014-07-21	1363	14.53
	2014-07-14	1345	14.34
	2014-07-07	1355	14.44
	2014-06-30	1271	13.55
	2014-06-23	1249	13.31
	2014-06-16	1263	13.46
	2014-06-09	1219	12.99
	2014-06-02	1173	12.50
	2014-05-26	1113	11.86
	2014-05-19	1147	12.23
	2014-05-12	1094	11.66
	2014-05-05	1054	11.24
	2014-04-28	701	7.47

Explanation:

Here,

- For calculating weekly active users, I used **CTE** (Common Table Expression)
 - Firstly, I converted “**occurred_at**” from to “**datetime**” using “**str_to_date**” function
 - Then extracted the date portion using “**date**”
 - After that for adjusting the date to the start of the week that is **Monday** I used “**date_sub**” and “**weekday**”, then I gave it alias “**week_start**”.

- Now for counting the number of distinct users who were active in that week I used **count** function
- Then I used **group by** for grouping the results by calculated week start date to gather the active users by week
- Again, I used one more **CTE** for calculating the total number of users.
 - In this I used **count** function for counting all the users in the “**users**” table
- Now I used a final **select** statement for combining the weekly active users with the total users to calculate the engagement percentage and present the results
 - In this I first selected “**week_start**” from the “**weekly_active_users**” CTE.
 - Then “**active_users**” which tells the count of active users for each week from the “**weekly_active_users**” CTE.
 - After that for calculating the **engagement percentage** I divided the number of active users in the week by the total number of users and multiplied the result **by** 100 to get a percentage, then I used **round** function for rounding the result to 2 decimal places.
 - Used the results **from** the “**weekly_active_users**” CTE
 - Now I joined “**weekly_active_users**” with the “**total_users**” CTE using **inner join** as “**total users**” returns a single row with the total user count, so it spread this value to each row of “**weekly_active_users**”

- Now at last I ordered the result by the week start date in descending order, so the most recent weeks appear first using **order by** clause.

→ So, we got the weekly user engagement with the week start date, active users, and the engagement percentage

B) 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.

Solution:



Query:

Write an SQL query to calculate the user growth for the product.

```
• SELECT
  DATE_FORMAT(STR_TO_DATE(created_at, '%d-%m-%Y %H:%i'), '%Y-%m') AS month,
  COUNT(user_id) AS new_users,
  SUM(COUNT(user_id)) OVER (ORDER BY DATE_FORMAT(STR_TO_DATE(created_at, '%d-%m-%Y %H:%i'), '%Y-%m'))
  AS additive_users
FROM
  users
GROUP BY
  DATE_FORMAT(STR_TO_DATE(created_at, '%d-%m-%Y %H:%i'), '%Y-%m')
ORDER BY
  month;
```

(P.T.O)

Output:

Result Grid   Filter Rows: <input type="text"/>			
	month	new_users	additive_users
▶	2013-01	160	160
	2013-02	160	320
	2013-03	150	470
	2013-04	181	651
	2013-05	214	865
	2013-06	213	1078
	2013-07	284	1362
	2013-08	316	1678
	2013-09	330	2008
	2013-10	390	2398
	2013-11	399	2797
	2013-12	486	3283
	2014-01	552	3835
	2014-02	525	4360
	2014-03	615	4975
	2014-04	726	5701
	2014-05	779	6480
	2014-06	873	7353
	2014-07	997	8350
	2014-08	1031	9381

Explanation:

Here,

- I selected **“date_format(str_to_date(..)) as month”** which converts the **“created_at”** string to a **“datetime”** format and then formats it to **“yyyy-mm”** for grouping users by month
- Then I used **count** function for counting the number of new users created in each month

- After that I used “**sum(count(..) over(((..))) as additive_users**” this calculates the additive sum of new users over the months. The “**sum() over(order by ...)**” window function is used for computing the running total of new users up to the current month.
- **Group by** clause I used for grouping the results by each month.
- **Order by** clause I used for ordering the results by month to get the data in same order exactly as it is calculated .
→ So at the end I got **result** which contains the monthly breakdown of new users and their **additive total** which allows me to track **user growth over time**.

(P.T.O)

C) Weekly Engagement Per Device:

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

Your Task: Write an SQL query to calculate the weekly retention of users based on their sign-up cohort.

Solution:

Query:



```
WITH signup_weeks AS (  
    SELECT  
        user_id,  
        YEARWEEK(STR_TO_DATE(created_at, '%d-%m-%Y %H:%i'), 1)  
        AS signup_week  
    FROM users  
)  
  
weekly_active_users AS (  
    SELECT  
        w.user_id,  
        YEARWEEK(STR_TO_DATE(eu.occurred_at, '%d-%m-%Y %H:%i'), 1)  
        AS event_week  
    FROM signup_weeks w  
    INNER JOIN events eu ON w.user_id = eu.user_id  
    GROUP BY w.user_id, YEARWEEK(STR_TO_DATE(eu.occurred_at, '%d-%m-%Y %H:%i'), 1)  
)
```

```

SELECT
    su.signup_week,
    COUNT(DISTINCT su.user_id) AS total_signed_up,
    COUNT(DISTINCT w.user_id) AS retained_users,
    ROUND((COUNT(DISTINCT w.user_id) / COUNT(DISTINCT su.user_id)) * 100.0, 2)
    AS retention_rate
FROM signup_weeks su
LEFT JOIN weekly_active_users w ON su.user_id = w.user_id
    AND su.signup_week = w.event_week - 1
GROUP BY su.signup_week
ORDER BY su.signup_week ASC;

```

Output:

Result Grid  Filter Rows: <input type="text"/> Export:  Wrap Cell Center				
	signup_week	total_signed_up	retained_users	retention_rate
▶	201301	26	0	0.00
	201302	29	0	0.00
	201303	47	0	0.00
	201304	36	0	0.00
	201305	30	0	0.00
	201306	48	0	0.00
	201307	41	0	0.00
	201308	39	0	0.00
	201309	33	0	0.00
	201310	43	0	0.00
	201311	33	0	0.00
	201312	32	0	0.00
	201313	33	0	0.00
	201314	40	0	0.00
	201315	35	0	0.00
	201316	42	0	0.00
	201317	48	0	0.00
	201318	48	0	0.00
	201319	45	0	0.00
	201320	55	0	0.00
	201321	41	0	0.00
	201322	49	0	0.00
	201323	51	0	0.00
	201324	51	0	0.00
	201325	46	0	0.00
	201326	57	0	0.00
	201327	57	0	0.00

	signup_week	total_signed_up	retained_users	retention_rate
	201328	52	0	0.00
	201329	71	0	0.00
	201330	66	0	0.00
	201331	69	0	0.00
	201332	66	0	0.00
	201333	73	0	0.00
	201334	71	0	0.00
	201335	79	0	0.00
	201336	65	0	0.00
	201337	71	0	0.00
	201338	84	0	0.00
	201339	92	0	0.00
	201340	81	0	0.00
	201341	88	0	0.00
	201342	74	0	0.00
	201343	97	0	0.00
	201344	92	0	0.00
	201345	97	0	0.00
	201346	94	0	0.00
	201347	82	0	0.00
	201348	103	0	0.00
	201349	96	0	0.00
	201350	117	0	0.00
	201351	123	0	0.00
	201352	104	0	0.00
	201401	132	0	0.00
	201402	122	0	0.00

	signup_week	total_signed_up	retained_users	retention_rate
	201403	112	0	0.00
	201404	113	0	0.00
	201405	130	0	0.00
	201406	132	0	0.00
	201407	135	0	0.00
	201408	127	0	0.00
	201409	127	0	0.00
	201410	135	0	0.00
	201411	152	0	0.00
	201412	132	0	0.00
	201413	151	0	0.00
	201414	161	0	0.00
	201415	166	0	0.00
	201416	165	0	0.00
	201417	176	56	31.82
	201418	172	118	68.60
	201419	160	104	65.00
	201420	186	144	77.42
	201421	177	121	68.36
	201422	186	117	62.90
	201423	197	133	67.51
	201424	198	146	73.74

201425	222	135	60.81
201426	210	151	71.90
201427	199	130	65.33
201428	223	152	68.16
201429	215	144	66.98
201430	228	156	68.42
201431	234	154	65.81
201432	189	126	66.67
201433	250	163	65.20
201434	259	173	66.80
201435	266	0	0.00

Explanation:

Here,

- Firstly, I used **CTE “signup_weeks”** for calculating the sign-up week for each user.
 - In this I used **“str_to_date”** function for converting the **“created_at”** string into a **“datetime”** format.
 - Then I used **“Yearweek(...,1)”** which extracts the year and week number from the **“datetime”** value.
 - **“Signup_week”** I have used for representing the year and week number when the user signed up.
- Now I used another CTE **“weekly_active_users”** for calculating the event week for each user to determine when they were active.
 - In this I used **“str_to_date”** for converting the **“occurred_at”** string into a **“datetime”** format
 - **“Yearweek(...,1)”** function I used for extracting the year and week number from the **“datetime”** value
 - **“event_week”** I used for representing the year and week number when the event occurred.

- Then I used **“Inner join”** between **“signup_weeks”** and **“events”** which links users sign-up data with their event data.
- **Group by** I used for grouping the data by user ID and event week.
- Now this **Final Query** is used for calculating the retention rate of users
 - **“Signup_week”** tells the week in which users signed up
 - **Count (Distinct su.user_id)** is used for counting the total number of distinct users who signed up in each week
 - **Count (Distinct w.user_id) as retained_users** is used for counting the number of distinct users who were active in the week following their sign-up week.
 - Then I used **Round(...)** function which is used for calculating the retention rate as a percentage and for rounding to 2 decimal places.
 - After that I used **“Left join”** which joins **“signup_weeks”** with **“ weekly_active_users”** to find users who were active in the week following their sign-up
 - Group By I used for grouping the results by sign-up week.
 - Order By is used for ordering the results by sign-up week in ascending order.

→ So in the result we got

- sign-up week for each user.
- Total users signed up
- Retained users
- And the Retention Rate

D) Weekly Engagement Per Device:

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

Your Task: Write an SQL query to calculate the weekly engagement per device

Solution:

Query:

Write an SQL query to calculate the weekly engagement per device.

```
WITH weekly_device_engagement AS (  
    SELECT  
        YEARWEEK(STR_TO_DATE(occurred_at, '%d-%m-%Y %H:%i:%s'), 1) AS week,  
        device,  
        COUNT(DISTINCT user_id) AS active_users  
    FROM events  
    GROUP BY week, device  
)  
SELECT  
    week,  
    device,  
    active_users  
FROM  
    weekly_device_engagement  
ORDER BY  
    week, device;
```

	week	device	active_users
▶	201418	acer aspire desktop	10
	201418	acer aspire notebook	21
	201418	amazon fire phone	4
	201418	asus chromebook	23
	201418	dell inspiron desktop	21
	201418	dell inspiron notebook	49
	201418	hp pavilion desktop	15
	201418	htc one	16
	201418	ipad air	30
	201418	ipad mini	21
	201418	iphone 4s	21
	201418	iphone 5	70
	201418	iphone 5s	45
	201418	kindle fire	6
	201418	lenovo thinkpad	90
	201418	mac mini	8
	201418	macbook air	57
	201418	macbook pro	154
	201418	nexus 10	16
	201418	nexus 5	43

	week	device	active_users
	201434	windows surface	14
	201435	acer aspire desktop	30
	201435	acer aspire notebook	63
	201435	amazon fire phone	11
	201435	asus chromebook	48
	201435	dell inspiron desktop	48
	201435	dell inspiron notebook	104
	201435	hp pavilion desktop	37
	201435	htc one	26
	201435	ipad air	36
	201435	ipad mini	25
	201435	iphone 4s	50
	201435	iphone 5	100
	201435	iphone 5s	68
	201435	kindle fire	15
	201435	lenovo thinkpad	186
	201435	mac mini	29
	201435	macbook air	135
	201435	macbook pro	290
	201435	nexus 10	23
	201435	nexus 5	67
	201435	nexus 7	31
	201435	nokia lumia 635	17
	201435	samsung galaxy tablet	13
	201435	samsung galaxy note	13
	201435	samsung galaxy s4	89
	201435	windows surface	20

Here I have pasted only the starting of the output and the ending of the output Because there is large result set.

Explanation:

Here,

- I started by “**str_to_date**” for converting the “**occurred_at**” string into a “**datetime**” format.
- Then I used “**Yearweek(...,1)**” for extracting the year and week number from the “**datetime**” value. This gives us the week of event.
- After that I used **CTE** “**weekly_device_engagement**” in which I selected yearweek as week, device ,count(distinct user_id) as active_users here,

- **Week** is used for the week of the event
- **Device** is used for telling device used for the event
- **Count** function is used for counting the number of unique users who were active on each device during each week
- **group by** is used for grouping the data by **week** and **device** to calculate the number of active users per device for each week
- **Finally Selecting the “week”, “device”, “active_users” from CTE “weekly_device_engagement”.**
- **And ordering the result by week and device using order by clause**

→ **So Now In the result set we got “weekly engagement per device” by counting the number of unique users active on each device for each week.**

(P.T.O)

E) Email Engagement Analysis:

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

Your Task: Write an SQL query to calculate the email engagement metrics.

Solution:

Query:

```
# Write an SQL query to calculate the email engagement metrics.
```

```
• SELECT
    COUNT(*) AS total_emails_sent,
    COUNT(CASE WHEN ee.action = 'email_open' THEN 1 END) AS total_emails_opened,
    COUNT(CASE WHEN ee.action = 'email_clickthrough' THEN 1 END) AS total_email_clicks,
    ROUND(COUNT(CASE WHEN ee.action = 'email_open' THEN 1 END) / COUNT(*) * 100, 2) AS open_rate,
    ROUND(COUNT(CASE WHEN ee.action = 'email_clickthrough' THEN 1 END) / COUNT(*) * 100, 2) AS click_rate
FROM
    email_events ee;
```

Output:

Result Grid					
Filter Rows: <input type="text"/>					
Export: <input type="button" value="Export"/>					
Wrap Cell Content: <input type="button" value="Wrap"/>					
	total_emails_sent	total_emails_opened	total_email_clicks	open_rate	click_rate
▶	90389	20459	9010	22.63	9.97

(P.T.O)

Explanation:

Here,

- Firstly, I used **“count(*) as total_email_sent”** for counting the total number of email events, which represents the total emails sent.
- Then I used **“Count(case when ee.action = “email_open” then 1 end) as total_emails_opened”** for counting the number of emails events where the event type is **“email_open”**.
- After that I used **“Count (case when ee.action = “email_clickthrough” then 1 end) as total_email_clicks** for counting the number of email events where the event type is **“email_clickthrough”**.
- Next I used **round(.....action = “email_open”.....)** for calculating the percentage of emails opened out of the total emails sent, rounded to two decimal places.
- Finally I used **round(.....action = “email_clickthrough”.....)** for calculating the percentage of emails clicked out of the total emails sent, rounded to two decimal places.
- Data is extracted from **“email_events ee”**.

→ So in the result set I got calculations of the email engagement metrics:

- Total emails sent
- Total emails opened
- Total emails clicked
- Email opened rate
- Email clicked rate

PROJECT DESCRIPTION

In this Project I assumed me as a Lead Data Analyst working for the Microsoft Company. My role involves analyzing end to end operations to uncovers areas for improvement. I had to work closely with various teams – operations, support and marketing to provide valuable insights from the collected data.

The purpose of this project is to gain some practical knowledge of end to end operations and to enhance operational efficiency, decision - making process of the company.

The approach I followed is first I analyzed and derived insights from the database provided and then using that data I solved the problems.

APPROACH

The approach is very simple

- Firstly I checked the database that what data or what type of data I have to clean or analyze
- Then after observing the database carefully I analyzed the data and derived some valuable insights
- After analyzing the database I started solving Problems/Tasks and at the time of solving problems:
 - At the first step, I understand the question's sense that exactly what I have to find
 - Then after understanding the question's sense i solved the problems with my skills and provided a brief explanation.

TECH STACK USED

❖ MYSQL WORKBENCH 8.0 CE

PURPOSE OF MYSQL WORKBENCH 8.0 CE IN THE ANALYSIS

I used MySQL Workbench for this analysis because I learnt sql on “**MYSQL WORKBENCH**” software, So for me this software is best for analysis.

INSIGHTS

- In this project I
 - Analyzed end to end operations
 - Collaborated and worked with various teams
 - Investigated and explained sudden changes in key metrics
 - Have done Daily quick and accurate data analysis
 - Have used advanced SQL skills to analyze various datasets and tables.

So By Doing this project I gained so much knowledge for the role of data analyst in Operational Analytics which requires a combination of both technical skills and effective communication with various departments for providing operational improvements and sudden changes in key metrics.

RESULT

So at last I would like to say that this project really gave me a picture of advanced sql concepts as I seriously felt like I am actually working as a data analyst at microsoft because this project includes tasks which are not for beginners as tasks requires critical thinking for understanding and analyzing the query.

This Project really helped me to enhance my skills as I have gathered a great set of insights from the datasets provided and also this project prepared me for solving real-world business challenges.

Now I am ready to do more advanced projects and keep myself in the growth category.