

Operation Analytics and Investigating Metric Spike

Description :- This project focuses on Operational Analytics which is a critical process used to analyse and improve a company's daily operations. This is done by deriving insights from data. My role as a Lead Data Analyst will involve collaborating with cross-functional teams to answer questions which will help identify areas of improvement and understand sudden metric spikes, such as dips in user engagement or sales. I will be working with advanced SQL queries to analyse the given datasets to gain valuable insights about user activity, retention, throughput, and engagement.

Approach :- My approach will include data cleaning and validation. I will be using powerful SQL features like window functions and cross joins to build sophisticated queries. In the final step I will communicate the insights derived, helping the company make data-driven decisions which will help the company gain operational efficiency.

Tech-Stack used :- My SQL Workbench 8.0 is being used.

Case Study 1: Job Data Analysis

Task 1 :- **Jobs Reviewed Over Time:**

- **Objective**: Calculate the number of jobs reviewed per hour for each day in November 2020.
- **My Task**: Write an SQL query to calculate the number of jobs reviewed per hour for each day in November 2020.
- **Code Written**:

```
use assignment3;

SELECT
    ds AS date,
    ROUND((COUNT(job_id) / (SUM(time_spent) / 3600)),
        2) AS jobs_reviewed_per_hr_per_day
FROM
    job_data
GROUP BY ds
ORDER BY jobs_reviewed_per_hr_per_day DESC ;
```

MySQL Workbench

ASSIGNMENT 3

File Edit View Query Database Server Tools Scripting Help

Navigator: Filter objects

assignment3

Tables

job_data

Views

Stored Procedures

Functions

ig_clone

Tables

comments

follows

likes

Administration Schemas

Information

Table: job_data

Columns:

job_id int

actor_id int

event varchar(50)

language varchar(50)

time_spent int

org varchar(1)

ds date

Query Editor:

```

1 /*Objective: Calculate the number of jobs reviewed per hour for each day in November 2020.
2 My Task: Write an SQL query to calculate the number of jobs reviewed per hour for each day in November 2020*/
3
4 use assignment3;
5
6 SELECT
7     ds AS date,
8     ROUND((COUNT(job_id) / (SUM(time_spent) / 3600)),
9           2) AS jobs_reviewed_per_hr_per_day
10 FROM
11     job_data
12 GROUP BY ds
13 ORDER BY jobs_reviewed_per_hr_per_day DESC
14

```

Result Grid:

date	jobs_reviewed_per_hr_per_day
2020-11-28	218.18
2020-11-30	180.00
2020-11-29	180.00
2020-11-25	80.00
2020-11-26	64.29
2020-11-27	34.62

Object Info Session Result 3

Read Only

○ Output:-

Result Grid | Filter Rows:

	date	jobs_reviewed_per_hr_per_day
▶	2020-11-28	218.18
	2020-11-30	180.00
	2020-11-29	180.00
	2020-11-25	80.00
	2020-11-26	64.29
	2020-11-27	34.62

○ Insights derived:-

- The jobs reviewed per hour per day was the highest on 28th November, 2020
- The jobs reviewed per hour per day was the lowest on 27th November, 2020
- The jobs reviewed per hour per day was the same on 30th and 29th of the aforesaid month

Task 2 :- Throughput Analysis:

- Objective: Calculate the 7-day rolling average of throughput (number of events per second).
- My task: Write an SQL query to calculate the 7-day rolling average of throughput. Additionally, explain whether I prefer using the daily metric or the 7-day rolling average for throughput, and why.
- Code Written:

use assignment3;

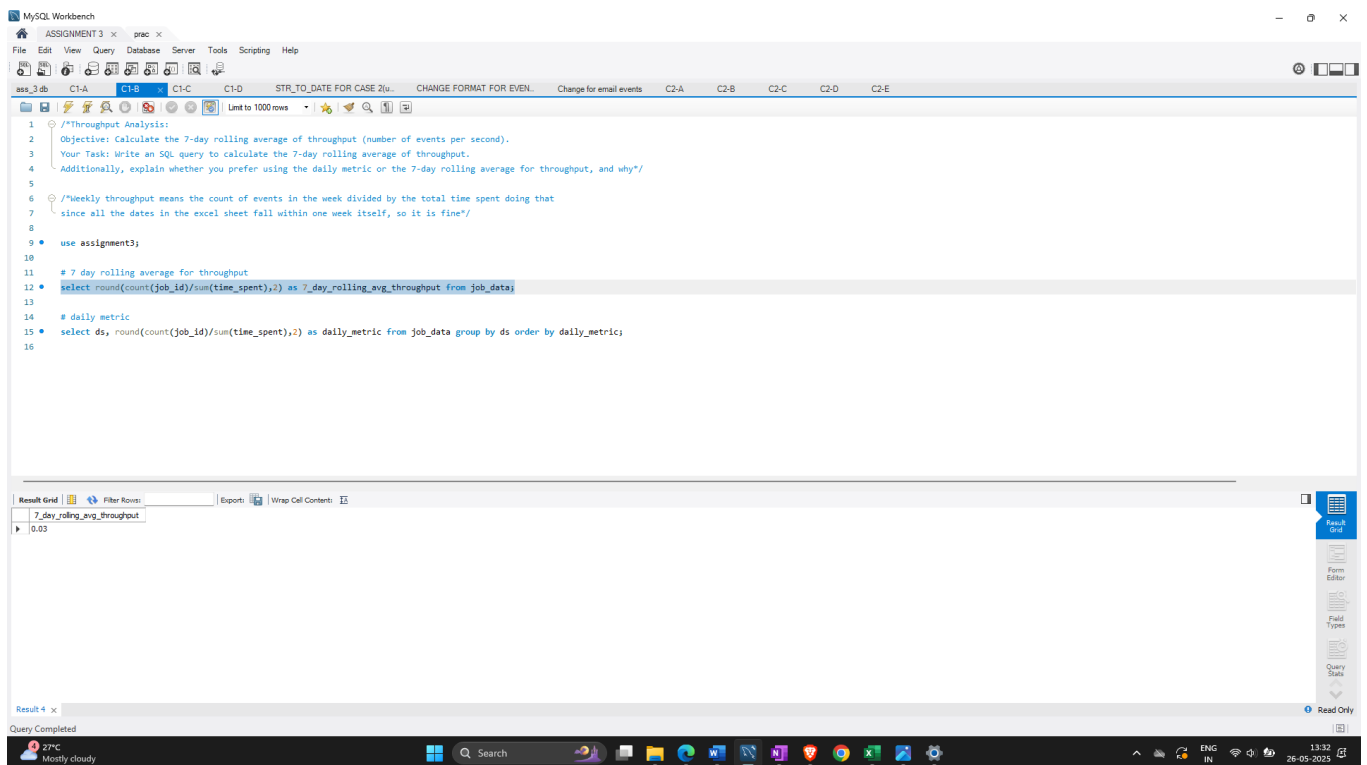
7 day rolling average for throughput

```
select round(count(job_id)/sum(time_spent),2) as 7_day_rolling_avg_throughput from job_data;
```

daily metric

```
select ds, round(count(job_id)/sum(time_spent),2) as daily_metric from job_data group by ds order by daily_metric;
```

For 7 day rolling average:-

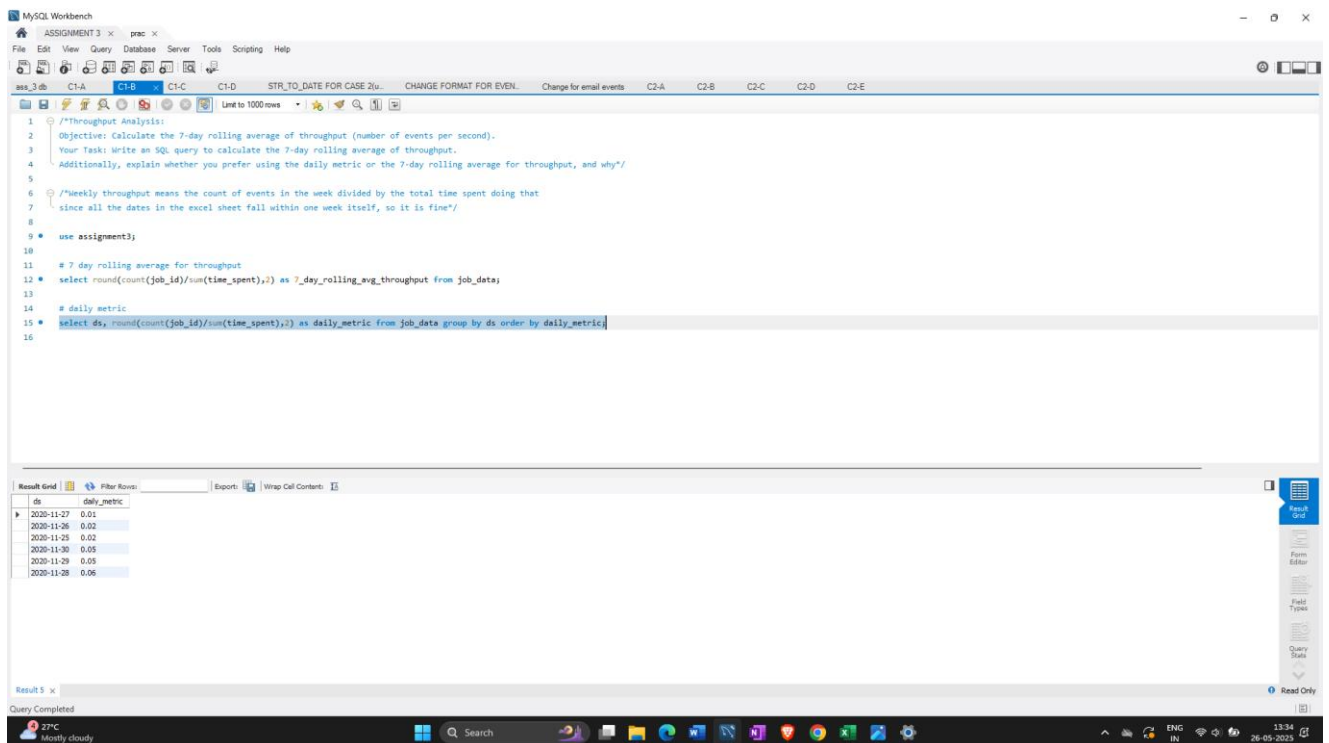


Output:-



7_day_rolling_avg_throughput
0.03

For Daily Metric:-



MySQL Workbench

ASSIGNMENT 3

File Edit View Query Database Server Tools Scripting Help

Query: STR_TO_DATE FOR CASE 2(u... CHANGE FORMAT FOR EVEN... Change for email events C2-A C2-B C2-C C2-D C2-E

Limit to 1000 rows

```
1  /*Throughput Analysis:
2  Objective: Calculate the 7-day rolling average of throughput (number of events per second).
3  Your Task: Write an SQL query to calculate the 7-day rolling average of throughput.
4  Additionally, explain whether you prefer using the daily metric or the 7-day rolling average for throughput, and why*/
5
6  /*Weekly throughput means the count of events in the week divided by the total time spent doing that
7  since all the dates in the excel sheet fall within one week itself, so it is fine*/
8
9  use assignment3;
10
11  # 7 day rolling average for throughput
12  select round(count(job_id)/sum(time_spent),2) as 7_day_rolling_avg_throughput from job_data;
13
14  # daily metric
15  select ds, round(count(job_id)/sum(time_spent),2) as daily_metric from job_data group by ds order by daily_metric;
16
```

Result Grid

ds	daily_metric
2020-11-27	0.01
2020-11-26	0.02
2020-11-25	0.02
2020-11-30	0.05
2020-11-29	0.05
2020-11-28	0.06

Query Completed

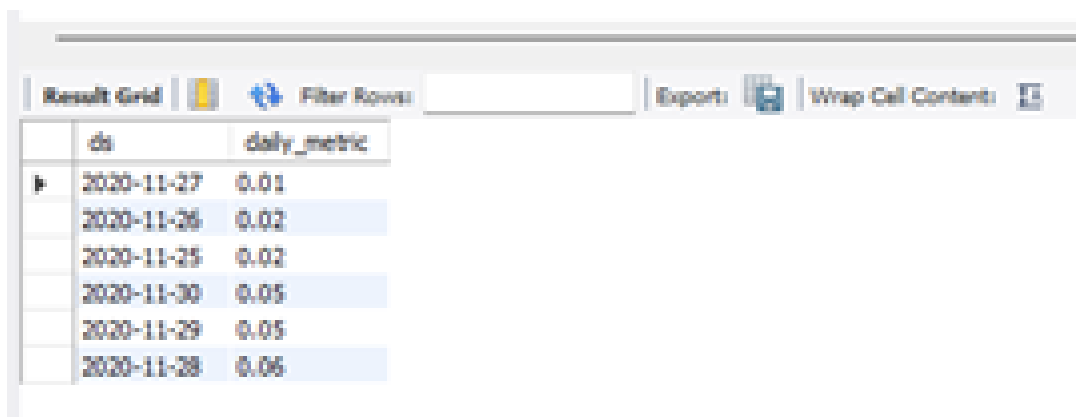
27°C Mostly cloudy

Search

ENG IN

13:34 26-05-2023

Output:-



ds	daily_metric
2020-11-27	0.01
2020-11-26	0.02
2020-11-25	0.02
2020-11-30	0.05
2020-11-29	0.05
2020-11-28	0.06

Whether I prefer using the daily metric or the 7-day rolling average for throughput?

For the throughput analysis, I will prefer the 7-day rolling average because it gives the average for all the days right from day 1 to day 7. On the other hand, daily metric gives us the average for only that day itself.

Task 3:- 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.
- Code Written:-

```
SELECT
    language,
    (individual_occ_lang * 100 / total_occurence_languages) AS
percentage_distribution
FROM
    (SELECT
        COUNT(language) AS total_occurence_languages
    FROM
        job_data) table1
    CROSS JOIN
    (SELECT
        language, COUNT(language) AS individual_occ_lang
    FROM
        job_data
    GROUP BY language) table2;
```

MySQL Workbench

ASSIGNMENT 3 x prac x unconnected x

File Edit View Query Database Server Tools Scripting Help

Navigator: ass_3_db C1-A C1-B C1-C

SCHMAS

Filter objects

assignment3

Tables

job_data

Views

Stored Procedures

Functions

Information

Table: job_data

Columns:

job_id int

actor_id int

event varchar(50)

language varchar(50)

time_spent int

org varchar(1)

ds date

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

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

3 USE assignment3;

4 /* WHY CROSS JOIN? COZ WE HAVE TO CONNECT THE TOTAL COUNT OF LANG WITH EACH LANG'S TOTAL CROSS BY CROSS

5 SELECT

6 language,

7 (individual_occ_lang * 100 / total_occurrence_languages) AS percentage_distribution

8 FROM

9 (SELECT

10 COUNT(language) AS total_occurrence_languages

11 FROM

12 job_data) table1

13 CROSS JOIN

14 (SELECT

15 language, COUNT(language) AS individual_occ_lang

16 FROM

17 job_data

18 GROUP BY language) table2;

Result Grid

language percentage_distribution

English 12.5000

Arabic 12.5000

Persian 37.5000

Hindi 12.5000

French 12.5000

Italian 12.5000

Object Info Session Result 6 x

SQL script saved to 'C:\Users\abhi\Downloads\C1-C.sql'

Output:-

Result Grid

Filter Rows:

	language	percentage_distribution
▶	English	12.5000
	Arabic	12.5000
	Persian	37.5000
	Hindi	12.5000
	French	12.5000
	Italian	12.5000

Result 6 x

Derived Insights:-

- It was observed that the Persian language had the highest percentage distribution amongst all i.e. 37.5%
- Whereas all other languages had an equal percentage distribution of 12.5%.

Task 4:- Duplicate Rows Detection:

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

#This is based on actor_id

```
SELECT actor_id, COUNT(*) AS duplicate FROM job_data GROUP BY actor_id  
HAVING COUNT(*) > 1;
```

#OR

```
SELECT * FROM job_data a JOIN job_data b ON a.actor_id = b.actor_id WHERE  
a.ds > b.ds;
```

#OR

```
SELECT a.* FROM job_data a JOIN job_data b ON a.actor_id = b.actor_id WHERE  
a.ds > b.ds;
```

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'assignment3' selected, containing 'Tables', 'Views', 'Stored Procedures', and 'Functions'. The 'job_data' table is highlighted under 'Tables'. The 'Information' tab for 'job_data' shows columns: job_id (int), actor_id (int), event (varchar(50)), language (varchar(50)), time_spent (int), org (varchar(1)), and ds (date). The main editor window shows a SQL script for duplicate row detection. The script includes comments and three queries. The first query uses GROUP BY and HAVING. The second and third queries use self-joins. The bottom panel shows the 'Result Grid' with one row of data.

```
1  /*Duplicate Rows Detection:
2  Objective: Identify duplicate rows in the data.
3  My Task: Write an SQL query to display duplicate rows from the job_data table*/
4
5  #This is based on actor_id
6
7  • SELECT
8      actor_id, COUNT(*) AS duplicate
9  FROM
10     job_data
11  GROUP BY actor_id
12  HAVING COUNT(*) > 1;
13  #OR
14  • SELECT * FROM job_data a JOIN job_data b ON a.actor_id = b.actor_id WHERE a.ds > b.ds;
15  #OR
16  • SELECT a.* FROM job_data a JOIN job_data b ON a.actor_id = b.actor_id WHERE a.ds > b.ds;
17
```

job_id	actor_id	event	language	time_spent	org	ds
23	1003	decision	Persian	20	C	2020-11-29

Output:-

	job_id	actor_id	event	language	time_spent	org	ds
▶	23	1003	decision	Persian	20	C	2020-11-29

Insights Derived:-

- There was one duplicate row having the job-id 23.

Case Study 2: Investigating Metric Spike

Before proceeding we assign the proper format to the given datasets.

Users table format Change:-

We use the STR_TO_DATE function for this. Converting the String format to DateTime format for columns 'Created_at' and 'occurred_at'.

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'Schemas' tree with 'assignment3' expanded, showing tables like 'email_events', 'events', 'job_data', and 'users'. The main editor window contains the following SQL script:

```
5 use assignment3;
6
7 create table users (
8   user_id int not null,
9   created_at varchar(50) not null,
10  company_id int not null,
11  language varchar(50) not null,
12  activated_at varchar(50) not null,
13  state varchar(50) not null);
14
15 select * from users;
16
17 #where the create_at and activated_at columns are in string/varchar, so we have to convert it into date_time format
18
19 Alter table users add column temp_created_at datetime;
20 update users set temp_created_at = STR_TO_DATE(created_at, '%d-%m-%Y %H:%i'); /*this line shows error because %y means YY year format
21 but we have the YYYY format which is denoted by %Y, hence the error*/
22 update users set temp_created_at = STR_TO_DATE(created_at, '%d-%m-%Y %H:%i');
23 alter table users drop column created_at;
24 alter table users change column temp_created_at created_at datetime;
25
26 #CONVERTING 'ACTIVATED_AT' STRING column to Datetime format
27
28 alter table users add column temp_activated_at datetime;
29 update users set temp_activated_at = str_to_date(activated_at, '%d-%m-%Y %H:%i');
30 alter table users drop column activated_at;
31 alter table users change column temp_activated_at activated_at datetime;
```

Below the script, the 'Result Grid' shows the data for the 'users' table (13 rows):

	user_id	company_id	language	state	created_at	activated_at
▶	0	5737	english	active	2013-01-01 20:59:00	2013-01-01 21:01:00
	3	2800	german	active	2013-01-01 18:40:00	2013-01-01 18:42:00
	4	5110	indian	active	2013-01-01 14:37:00	2013-01-01 14:39:00
	6	11699	english	active	2013-01-01 18:37:00	2013-01-01 18:38:00
	7	4765	french	active	2013-01-01 16:19:00	2013-01-01 16:20:00
	8	2698	french	active	2013-01-01 04:38:00	2013-01-01 04:40:00
	11	3745	english	active	2013-01-01 08:07:00	2013-01-01 08:09:00
	13	4025	english	active	2013-01-02 12:27:00	2013-01-02 12:29:00
	15	4259	english	active	2013-01-02 15:39:00	2013-01-02 15:41:00
	17	5025	japanese	active	2013-01-02 10:56:00	2013-01-02 10:57:00
	19	326	english	active	2013-01-02 09:54:00	2013-01-02 09:55:00

The bottom status bar indicates the SQL script was saved to 'C:\Users\abhi\OneDrive\Documents\MYSQL ASSIGNMENT\ASSIGNMENT 3, USER_ANALYTICS\CASE 2\STR_TO_DATE FOR CASE 2(users).sql'.

Events table format Change:- We use the STR_TO_DATE function for this. Converting the String format to DateTime format for column 'occurred_at'.

The screenshot shows the MySQL Workbench interface with the 'events' table being created and modified. The SQL editor contains the following queries:

```

1 use assignment3;
2
3 CREATE TABLE events(
4   user_id int not null,
5   occurred_at varchar(50) not null,
6   event_type varchar(50) not null,
7   event_name varchar(50) not null,
8   location varchar(50) not null,
9   device varchar(50) not null,
10  user_type int not null);
11
12 /* Lacks of data importing technique using "load data into" function*/
13
14 show variables like 'secure_file_priv';
15
16 LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/events.csv"
17 INTO TABLE events
18 FIELDS TERMINATED BY ','
19 ENCLOSED BY '"'
20 LINES TERMINATED BY '\n'
21 IGNORE 1 ROWS;
22
23 select * from events;
24
25 ALTER TABLE events ADD COLUMN temp2_occurred_at DATETIME;
26 UPDATE events SET temp2_occurred_at=STR_TO_DATE(occurred_at, '%d-%m-%Y %H:%i');
27 ALTER TABLE events DROP COLUMN occurred_at;
28 ALTER TABLE events CHANGE COLUMN temp2_occurred_at occurred_at DATETIME;
  
```

The 'Result Grid' shows the following data:

user_id	event_type	event_name	location	device	user_type	occurred_at
10522	engagement	login	Japan	del inspiron notebook	3	2014-05-02 11:02:00
10522	engagement	home_page	Japan	del inspiron notebook	3	2014-05-02 11:02:00
10522	engagement	like_message	Japan	del inspiron notebook	3	2014-05-02 11:03:00
10522	engagement	view_post	Japan	del inspiron notebook	3	2014-05-02 11:04:00
10522	engagement	search_run	Japan	del inspiron notebook	3	2014-05-02 11:03:00
10522	engagement	search_run	Japan	del inspiron notebook	3	2014-05-02 11:03:00
10612	engagement	login	Netherlands	iphone 5	1	2014-05-01 09:39:00
10612	engagement	like_message	Netherlands	iphone 5	1	2014-05-01 10:00:00

Email Events table format Change:- We use the STR_TO_DATE function for this. Converting the String format to DateTime format for column 'occurred_at'.

The screenshot shows the MySQL Workbench interface with the 'email_events' table being created and modified. The SQL editor contains the following queries:

```

2
3 create table email_events(
4   user_id int not null,
5   occurred_at varchar(50) not null,
6   action varchar(50) not null,
7   user_type int not null);
8
9 show variables like 'secure_file_priv';
10
11 load data infile "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/email_events.csv"
12 INTO TABLE email_events
13 FIELDS TERMINATED BY ','
14 ENCLOSED BY '"'
15 LINES TERMINATED BY '\n'
16 IGNORE 1 ROWS;
17
18 select * from email_events;
19
20 alter table email_events add column temp3_occurred_at datetime;
21 update email_events set temp3_occurred_at=STR_TO_DATE(occurred_at, '%d-%m-%Y %H:%i');
22 alter table email_events drop column occurred_at;
23 alter table email_events change column temp3_occurred_at occurred_at datetime;
  
```

The 'Result Grid' shows the following data:

user_id	action	user_type	occurred_at
0	sent_weekly_digest	1	2014-05-27 09:30:00
0	sent_weekly_digest	1	2014-06-03 09:30:00
0	email_open	1	2014-06-03 09:30:00
0	sent_weekly_digest	1	2014-06-10 09:30:00
0	email_open	1	2014-06-10 09:30:00

The 'Output' tab shows the execution results of the queries:

#	Time	Action	Message	Duration / Fetch
229	21:42:20	alter table email_events add column temp3_occurred_at datetime	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.047 sec
230	21:56:50	update email_events set temp3_occurred_at=STR_TO_DATE(occurred_at, '%d-%m-%Y %H:%i')	90389 row(s) affected Rows matched: 90389 Changed: 90389 Warnings: 0	5.375 sec
231	21:57:06	select * from email_events LIMIT 0, 1000	1000 row(s) returned	0.000 sec / 0.016 sec
232	21:59:29	alter table email_events drop column occurred_at	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.031 sec
233	22:00:32	alter table email_events change column temp3_occurred_at occurred_at datetime	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.016 sec
234	22:00:34	select * from email_events LIMIT 0, 1000	1000 row(s) returned	0.000 sec / 0.000 sec

Task 1: Weekly User Engagement

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

My Task: Write an SQL query to calculate the weekly user engagement

Code Written:

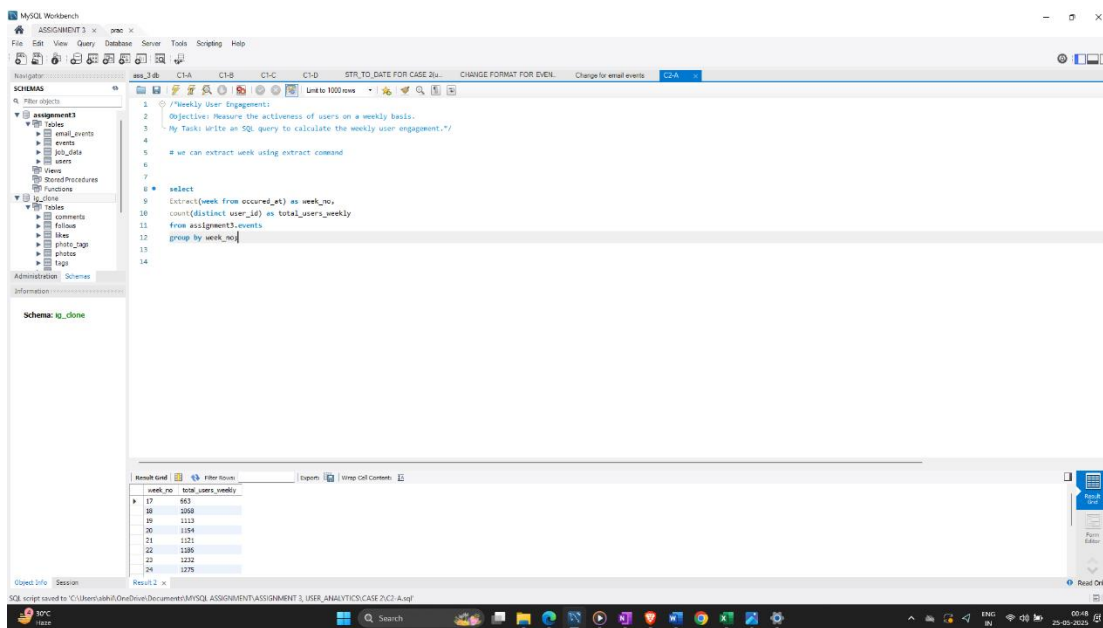
select

Extract(week from occurred_at) as week_no,

count(distinct user_id) as total_users_weekly

from assignment3.events

group by week_no;



Output:

Result Grid		Filter Rows:
week_no	total_users_weekly	
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	

Insights Derived:

- The total users weekly was highest in week 30 whereas lowest in week 35
- The highest total users weekly was recorded as 1467, whereas the lowest recorded was 105.

Task 2: User Growth Analysis:

- Objective: Analyse the growth of users over time for a product.
- My Task: Write an SQL query to calculate the user growth for the product.
- Code Written:

```
SELECT year, week_num, num_of_users, sum(num_of_users) over(order by year, week_num) as cumulative_users
```

```
from (
```

```
select extract(year from created_at) as year, extract(week from created_at) as week_num,
```

```
count(distinct user_id) as num_of_users
```

```
from users where state='active'
```

```
group by year, week_num
```

```
order by year, week_num)derived;
```

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' panel with a tree view of the 'assignment3' database, including tables like 'email_events', 'events', 'job_data', and 'users'. The main editor window contains a SQL query titled '/* User Growth Analysis: Objective: Analyze the growth of users over time for a product. My Task: Write an SQL query to calculate the user growth for the product */'. The query is as follows:

```
1  /* User Growth Analysis:
2  Objective: Analyze the growth of users over time for a product.
3  My Task: Write an SQL query to calculate the user growth for the product */
4
5  SELECT year, week_num, num_of_users, sum(num_of_users) over(order by year, week_num) as cumulative_users
6  from (
7  select extract(year from created_at) as year, extract(week from created_at) as week_num,
8  count(distinct user_id) as num_of_users
9  from users where state='active'
10 group by year, week_num
11 order by year, week_num)derived;
```

Below the query editor, the 'Result Grid' shows the output of the query. It is a table with four columns: 'year', 'week_num', 'num_of_users', and 'cumulative_users'. The data spans from year 2013, week 0 to week 30. The 'num_of_users' column shows a general upward trend, starting at 23 in week 0 and reaching 67 in week 30. The 'cumulative_users' column shows the total number of users up to that week, starting at 23 and reaching 1389 in week 30.

year	week_num	num_of_users	cumulative_users
2013	0	23	23
2013	1	30	53
2013	2	48	101
2013	3	36	137
2013	4	30	167
2013	5	48	215
2013	6	38	253
2013	7	42	295
2013	8	34	329
2013	9	43	372
2013	10	32	404
2013	11	31	435
2013	12	33	468
2013	13	39	507
2013	14	35	542
2013	15	43	585
2013	16	46	631
2013	17	49	680
2013	18	44	724
2013	19	57	781
2013	20	39	820
2013	21	49	869
2013	22	54	923
2013	23	50	973
2013	24	45	1018
2013	25	57	1075
2013	26	56	1131
2013	27	52	1183
2013	28	72	1255
2013	29	67	1322
2013	30	67	1389

The bottom status bar indicates the SQL script was saved to 'C:\Users\abhi\OneDrive\Documents\MYSQL ASSIGNMENT\ASSIGNMENT 3, USER_ANALYTICS\CASE 2\C2-B.sql'.

○ Output:

year	week_num	num_of_users	cumulative_users
2013	0	23	23
2013	1	30	53
2013	2	48	101
2013	3	36	137
2013	4	30	167
2013	5	48	215
2013	6	38	253
2013	7	42	295
2013	8	34	329
2013	9	43	372
2013	10	32	404
2013	11	31	435
2013	12	33	468
2013	13	39	507
2013	14	35	542
2013	15	43	585
2013	16	46	631
2013	17	49	680
2013	18	44	724
2013	19	57	781
2013	20	39	820
2013	21	49	869
2013	22	54	923
2013	23	50	973
2013	24	45	1018
2013	25	57	1075
2013	26	56	1131
2013	27	52	1183
2013	28	72	1255
2013	29	67	1322
2013	30	67	1389
2013	31	67	1456
2013	32	71	1527
2013	33	73	1600
2013	34	78	1678
2013	35	63	1741
2013	36	72	1813
2013	37	85	1898

2013	38	90	1988
2013	39	84	2072
2013	40	87	2159
2013	41	73	2232
2013	42	99	2331
2013	43	89	2420
2013	44	96	2516
2013	45	91	2607
2013	46	88	2695
2013	47	102	2797
2013	48	97	2894
2013	49	116	3010
2013	50	124	3134
2013	51	102	3236
2013	52	47	3283
2014	0	83	3366
2014	1	126	3492
2014	2	109	3601
2014	3	113	3714
2014	4	130	3844
2014	5	133	3977
2014	6	135	4112
2014	7	125	4237
2014	8	129	4366
2014	9	133	4499
2014	10	154	4653
2014	11	130	4783
2014	12	148	4931
2014	13	167	5098
2014	14	162	5260
2014	15	164	5424
2014	16	179	5603
2014	17	170	5773
2014	18	163	5936
2014	19	185	6121
2014	20	176	6297
2014	21	183	6480

2014	22	196	6676
2014	23	196	6872
2014	24	229	7101
2014	25	207	7308
2014	26	201	7509
2014	27	222	7731
2014	28	215	7946
2014	29	221	8167
2014	30	238	8405
2014	31	193	8598
2014	32	245	8843
2014	33	261	9104
2014	34	259	9363
2014	35	18	9381

Derived Insights:-

- Week no 33 had the highest number of users i.e. 261 number of users
- Whereas Week 25 had the lowest number of users i.e. 18 number of users

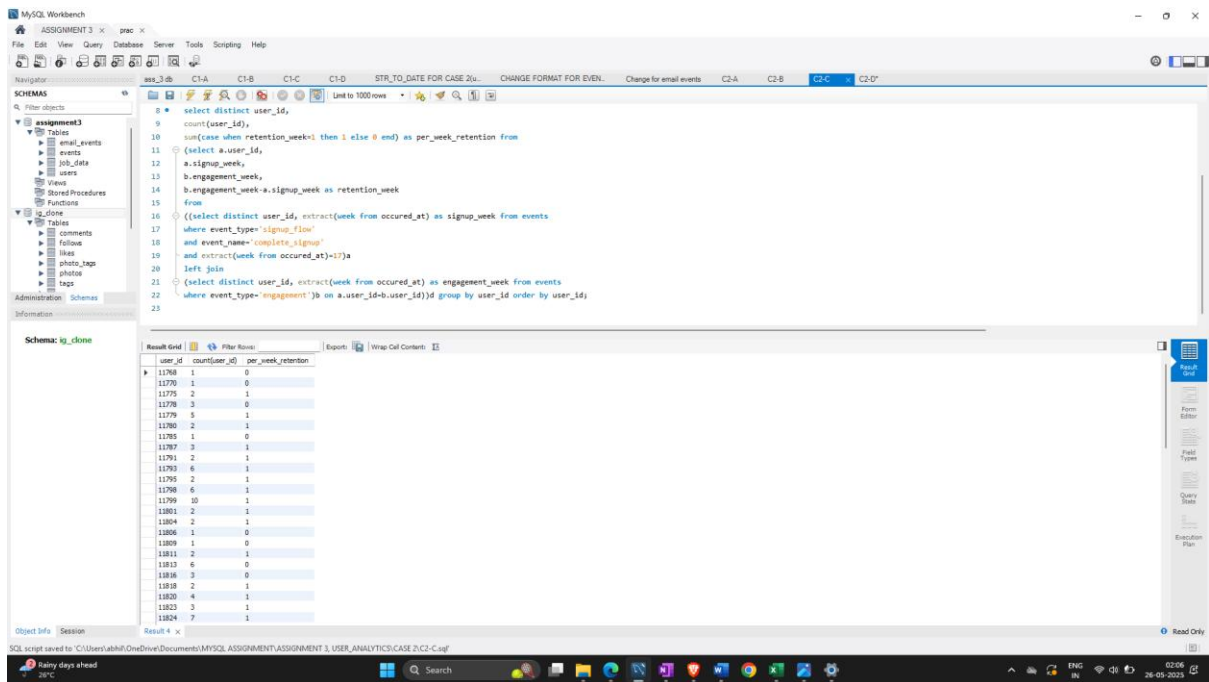
Task 3: Weekly Retention Analysis:

- Objective: Analyze the retention of users on a weekly basis after signing up for a product.
- My Task: Write an SQL query to calculate the weekly retention of users based on their sign-up cohort.
- Code Written:

```

SELECT DISTINCT
    user_id,
    COUNT(user_id),
    SUM(CASE
        WHEN retention_week = 1 THEN 1
        ELSE 0
    END) AS per_week_retention
FROM
    (SELECT
        a.user_id,
        a.signup_week,
        b.engagement_week,
        b.engagement_week - a.signup_week AS retention_week
    FROM
        ((SELECT DISTINCT
            user_id, EXTRACT(WEEK FROM occurred_at) AS signup_week
        FROM
            events
        WHERE
            event_type = 'signup_flow'
            AND event_name = 'complete_signup'
            AND EXTRACT(WEEK FROM occurred_at) = 17) 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)) d
GROUP BY user_id
ORDER BY user_id;

```



Output:

user_id	count(user_id)	per_week_retention
11768	1	0
11770	1	0
11775	2	1
11778	3	0
11779	5	1
11780	2	1
11785	1	0
11787	3	1
11791	2	1
11793	6	1
11795	2	1
11798	6	1
11799	10	1
11801	2	1
11804	2	1
11806	1	0
11809	1	0
11811	2	1
11813	6	0
11816	3	0
11818	2	1
11820	4	1
11823	3	1
11824	7	1
11825	3	1
11826	2	1
11828	3	1
11829	3	1
11832	4	1
11833	14	1
11834	2	1
11838	2	1
11839	1	0
11840	2	1
11841	6	1
11842	6	1
11843	3	1
11844	6	1
11849	3	1
11850	3	0
11852	5	1
11854	3	1
11858	6	1
11859	4	1
11863	6	1
11864	2	1
11865	3	1
11868	9	1
11872	2	1
11874	2	1
11875	2	1
11876	2	1
11877	8	1
11879	2	1
11881	6	1
11882	2	1
11884	1	0
11887	2	1
11888	5	0
11889	2	1
11893	14	1
11894	2	1
11895	3	1
11896	2	1
11898	1	0
11899	4	1
11900	6	1
11901	4	1
11906	12	1
11908	3	1
11909	7	1
11914	6	1

Insights Derived:

- User id 11893 & 11833 had the highest count i.e. 14

Task 4: Weekly Engagement Per Device:

- Objective: Measure the activeness of users on a weekly basis per device.
- My Task: Write an SQL query to calculate the weekly engagement per device.
- Code Written:

```
select
extract(year from occurred_at) as year_num,
extract(week from occurred_at) as week_num,
device,
count(distinct user_id) as no_of_users
from events
where event_type='engagement'
group by device, week_num, year_num
order by week_num;
```

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' panel with 'assignment3' selected, containing tables like 'email_events', 'events', 'job_data', and 'users'. The main editor window shows the SQL query for 'Weekly Engagement Per Device'. The 'Result Grid' at the bottom shows the output of the query, listing weekly engagement data for various devices.

year_num	week_num	device	no_of_users
2014	17	acer aspire desktop	9
2014	17	acer aspire notebook	20
2014	17	amazon fire phone	4
2014	17	asus chromebook	21
2014	17	dell inspiron desktop	18
2014	17	dell inspiron notebook	46
2014	17	hp pavilion desktop	14
2014	17	htc one	16
2014	17	ipad air	27
2014	17	ipad mini	19
2014	17	iphone 4s	21
2014	17	iphone 5	65
2014	17	iphone 5s	42
2014	17	kindle fire	6
2014	17	lenovo thinkpad	86
2014	17	mac mini	6
2014	17	macbook air	54
2014	17	macbook pro	143
2014	17	nexus 10	16
2014	17	nexus 5	40
2014	17	nexus 7	18
2014	17	nokia lumia 635	17
2014	17	samsung galaxy tablet	8
2014	17	samsung galaxy note	7
2014	17	samsung galaxy s4	52
2014	17	windows surface	10
2014	18	acer aspire desktop	26
2014	18	acer aspire notebook	33

Output:-

2014	30	lenovo thinkpad	206
2014	30	mac mini	23
2014	30	macbook air	159
2014	30	macbook pro	322
2014	30	nexus 10	36
2014	30	nexus 5	84
2014	30	nexus 7	62
2014	30	nokia lumia 635	34
2014	30	samsung galaxy tablet	9
2014	30	samsung galaxy note	15
2014	30	samsung galaxy s4	103
2014	30	windows surface	19
2014	31	acer aspire desktop	31
2014	31	acer aspire notebook	55
2014	31	amazon fire phone	14
2014	31	asus chromebook	56
2014	31	dell inspiron desktop	44

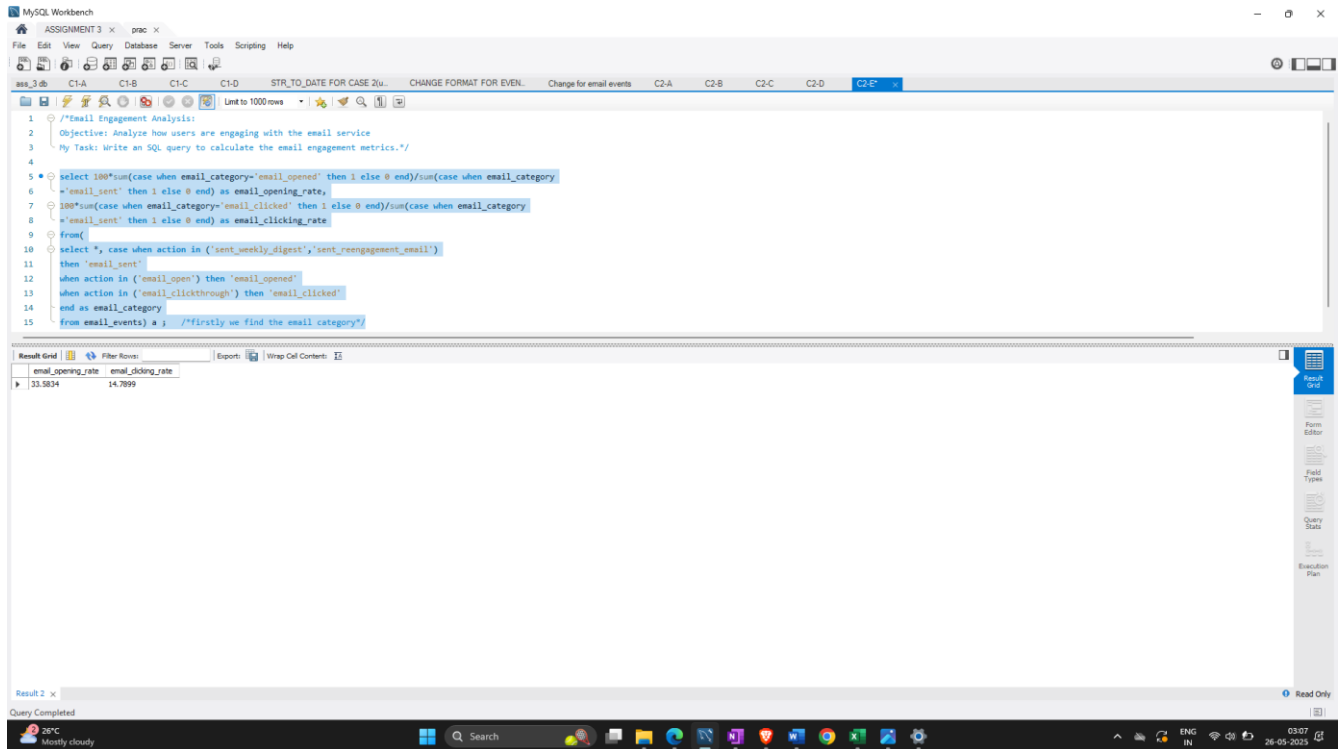
Insights Derived:-

- Week 30 has the highest number of users i.e. 322
- The most number of users are for MacBook Pro

Task 5:- Email Engagement Analysis:

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

```
select 100*sum(case when email_category='email_opened' then 1 else 0 end)/sum(case when email_category='email_sent' then 1 else 0 end) as email_opening_rate,  
100*sum(case when email_category='email_clicked' then 1 else 0 end)/sum(case when email_category='email_sent' then 1 else 0 end) as email_clicking_rate  
from(  
select *, case when action in ('sent_weekly_digest','sent_reengagement_email')  
then 'email_sent'  
when action in ('email_open') then 'email_opened'  
when action in ('email_clickthrough') then 'email_clicked'  
end as email_category  
from email_events) a ; /*firstly we find the email category*/
```

Output:-

Result Grid			Filter Rows:	Export:	Wrap Cell Content:
	email_opening_rate	email_clicking_rate			
▶	33.5834	14.7899			

Insights Derived:-

- It was observed that the email opening rate was 33.5834
- The email clicking rate was 14.7899

Overall Insights, Results and Observations:

Insights:-

- Practical knowledge gained on implementing SQL to real world scenarios
- Gained in-depth knowledge on joins
- Helped to gain insights into how MySQL can be used to solve real world problems coming up in the corporate world

Result:-

- a) Gained confidence on MySQL and how to use it in different scenarios and what different methods/codes can be used for a particular problem
- b) Helped to gain a grip on the different syntaxes to be used, what errors need to be avoided while coding in MySQL, I learnt how some common syntax errors can be avoided and what to look for.
- c) Thorough understanding of how to use different Window Functions to solve real world business problems.