

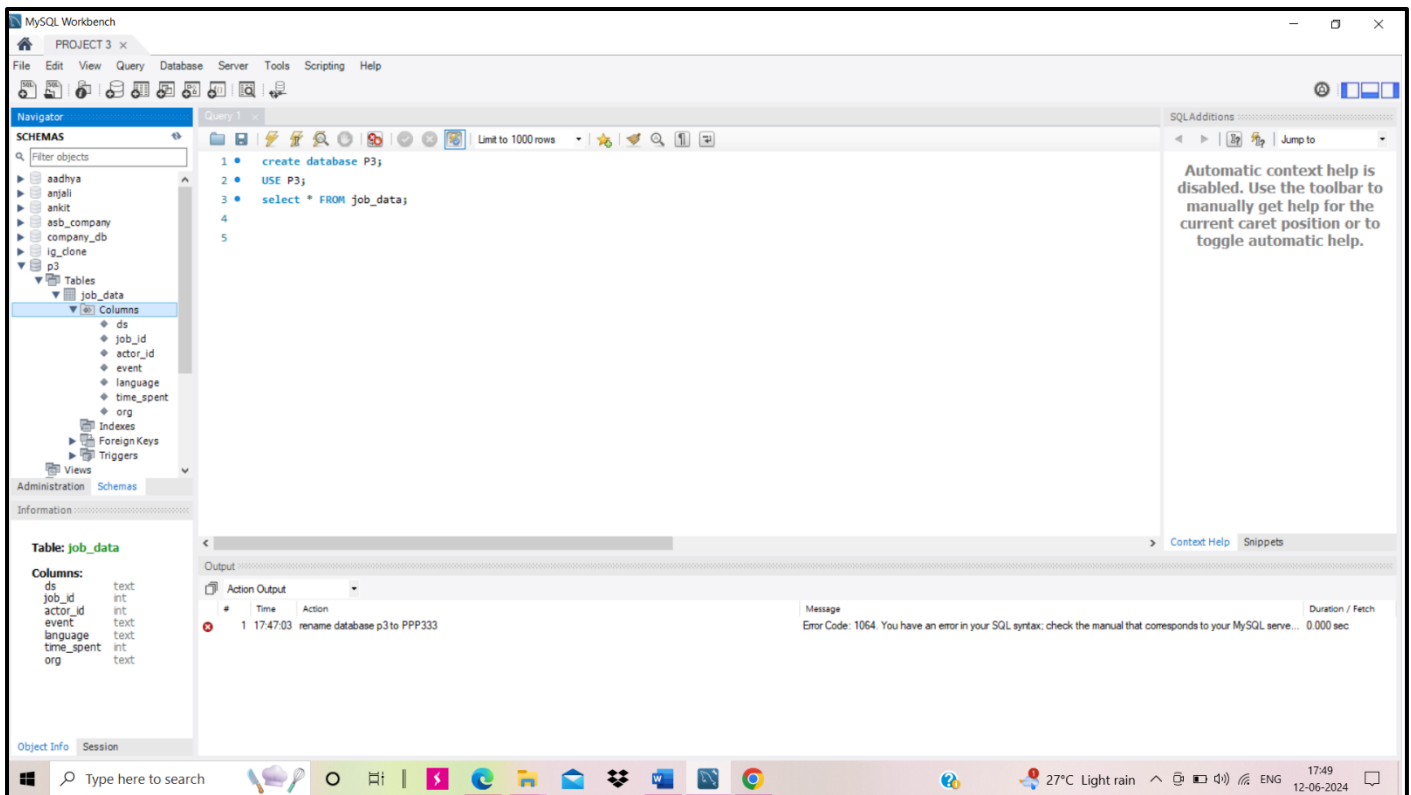
Operation Analytics and Investigating Metric Spike

- A. Project Description :-** . Analyzing a company's entire operational cycle is a critical step in the process of operational analytics. Finding areas for improvement within the organization is made easier by this analysis. In this project, with various datasets and tables I have to derive insights from this data to answer questions posed by different departments within the company by using advanced SQL skills to analyze the data and provide valuable insights that can help improve the company's operations and understand sudden changes in key metrics.
- B. Project approach :-** First the csv file data was imported in table of sql and then by using advance SQL skills the questions solutions were obtained.
- C. Tech-Stack Used :-** The tech stack included MySQL Workbench v8.0.30.0, a great tool for database queries because of its accuracy, speed, simplicity, and ease of use. It enables you to work with database objects, design, create, and browse your schemas; it also lets you design and execute SQL queries to manipulate with data that has been stored. Also Microsoft Excel was used to import/export data , Microsoft word was used to create presentation pdf document.

Project Insight

Case Study 1: Job Data Analysis

Before going for analysis all the raw data i.e CSV file is imported in the MY SQL in p3 database for futher analysis is inserted in MYSQL which can be seen in the following screenshots :-



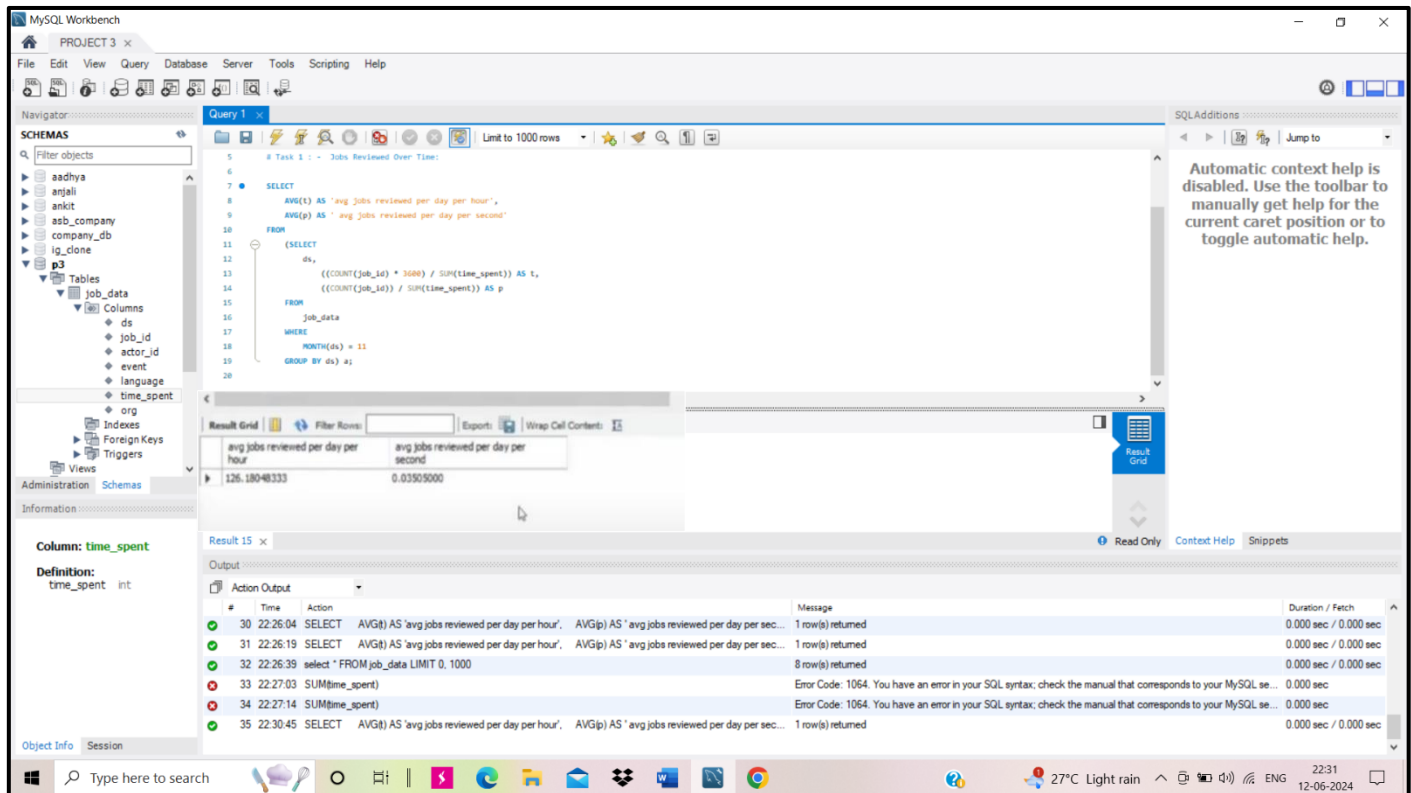
A) Jobs Reviewed Over Time:

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

Conclusion /result :- avg jobs reviewed per day per hour =126.18048333

avg jobs reviewed per day per second = 0.03505000

- screenshot of code and after running code its output



Code used :-

SELECT

AVG(t) AS 'avg jobs reviewed per day per hour',

AVG(p) AS 'avg jobs reviewed per day per second'

FROM

(SELECT

ds,

((COUNT(job_id) * 3600) / SUM(time_spent)) AS t,

((COUNT(job_id)) / SUM(time_spent)) AS p

FROM

job_data

WHERE

MONTH(ds) = 11

GROUP BY ds) a;

(B) Throughput Analysis

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

Conclusion /result :-

Dates	Daily Throughput
11/25/2020	0.02
11/26/2020	0.02
11/27/2020	0.01
11/28/2020	0.06
11/29/2020	0.05
11/30/2020	0.05

Weekly Throughput 0.03

- screenshot of code and after running code its output

The screenshot displays the MySQL Workbench interface. The 'Query' tab is active, showing two SQL queries. The first query calculates the 'weekly throughput' by rounding the count of events divided by the sum of time spent to 2 decimal places. The second query calculates the 'daily throughput' by rounding the same calculation to 2 decimal places and grouping by dates. The 'Result Grid' shows the output of the second query, displaying dates from 11/25/2020 to 11/30/2020 and their corresponding daily throughput values. The 'Output' tab at the bottom shows the execution log, indicating that the queries were executed successfully.

Code used :-

Select Round(count(event)/sum(time_spent), 2) as 'weekly throghput'
from job_data;

select ds as dates, round(count(event)/sum(time_spent), 2) as '
daily Throughput' from job_data

group by ds order by ds;

(C)Language Share Analysis:

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

• Conclusion /result :-	Languages	percentage	total
•	Arabic	12.50	8
	English	12.50	8
	French	12.50	8
	Hindi	12.50	8
	Italian	12.50	8
	Persian	37.50	8

➤ screenshot of code and after running code its output

The screenshot displays the MySQL Workbench interface. The 'Query Editor' window shows a SQL query for 'Language Share Analysis'. The query calculates the percentage share of each language in the last 30 days by grouping data by language and calculating the total count for each language, then dividing by the overall total count and multiplying by 100. The 'Results' window shows the output of the query, which is a table with three columns: 'Languages', 'percentage', and 'total'. The results show that Persian has the highest percentage share at 37.50%, followed by Arabic, English, French, Hindi, and Italian, all with a total count of 8.

```
#Task3 - Language Share Analysis:

Select language as Languages, round(100* count(*)/total, 2) as percentage, sub.total
from job_data
cross join (select count(*) as total from job_data) as sub
group by language, sub.total;
```

Languages	percentage	total
English	12.50	8
Arabic	12.50	8
Persian	37.50	8
Hindi	12.50	8
French	12.50	8
Italian	12.50	8

Code used :-

Select language as Languages, round(100* count(*)/total, 2) as percentage,
sub.total

from job_data

cross join (select count(*) as total from job_data) as sub

group by language, sub.total;

[4] Duplicate Rows Detection:

- **Objective: Identify duplicate rows in the data.**

- **Conclusion/Result :-**

Actor_id	duplicates
1003	2

- screenshot of code and after running code its output

The screenshot displays the MySQL Workbench interface. The 'Query Editor' window contains the following SQL code:

```
32
33 # Task 4 - Duplicate Rows Detection:
34
35 select actor_id, count(*) as Duplicates from job_data
36 group by actor_id having count(*)> 1;
37
38
```

The 'Result Grid' shows the output of the query:

actor_id	Duplicates
1003	2

The 'Output' window shows the execution details:

#	Time	Action	Message	Duration / Fetch
1	20:22:15	select actor_id, count(*) as Duplicates from job_data group by actor_id having count(*)> 1 LIMIT 0, 1000	1 row(s) returned	0.016 sec / 0.000 sec
2	20:22:30	select actor_id, count(*) as Duplicates from job_data group by actor_id having count(*)> 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

Code used :-

```
select actor_id, count(*) as Duplicates from job_data
group by actor_id having count(*)> 1;
```

Case Study 2: Investigating Metric Spike

There are three tables with which are doing this case study 2 as follows:-

- **users:** Contains one row per user, with descriptive information about that user's account.
- **events:** Contains one row per event, where an event is an action that a user has taken (e.g., login, messaging, search).
- **email_events:** Contains events specific to the sending of emails.

✚ Before going for analysis all the raw data i.e CSV file is imported in the MY SQL in “ case 2” database for futher which can be seen in the following screenshots :-

TABLE 1 – USERS

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' tree with 'case_2' selected. The central editor shows a SQL script for creating the 'users' table and loading data from a CSV file. The bottom pane shows the 'Result Grid' with 19 rows of user data.

SQL Script:

```
1 create database case_2;
2 use case_2;
3 # TABLE 1 - USERS
4 create table users (
5   user_id int,
6   company_id varchar(100),
7   language varchar(50),
8   activated_at varchar(100),
9   state varchar(50),
10  created_at datetime
11 );
12 select * from users;
13
14 load data infile "C:/ProgramData/MySQL/MySQL Server 8.0/uploads/users.csv"
15 into table users
16 fields terminated by ','
17 enclosed by '"'
18 lines terminated by '\n'
19 ignore 1 rows;
20
21 alter table users add column temp_created_at datetime;
22 update users set temp_created_at = str_to_date( created_at, '%Y-%m-%d %H:%M:%S' );
23 alter table users drop column created_at;
24 alter table users change column temp_created_at created_at datetime;
```

Table: users

user_id	company_id	language	activated_at	state	created_at
0	5737	english	01-01-2013 21:01	active	2013-01-01 20:59:00
3	2800	german	01-01-2013 18:42	active	2013-01-01 18:40:00
4	5110	indian	01-01-2013 14:39	active	2013-01-01 14:37:00
6	11699	english	01-01-2013 18:38	active	2013-01-01 18:37:00
7	4765	french	01-01-2013 16:20	active	2013-01-01 16:19:00
8	2698	french	01-01-2013 04:40	active	2013-01-01 04:38:00
11	3745	english	01-01-2013 08:09	active	2013-01-01 08:07:00
13	4025	english	02-01-2013 12:29	active	2013-01-02 12:27:00
15	4259	english	02-01-2013 15:41	active	2013-01-02 15:39:00
17	6076	german	01-01-2013 10:47	active	2013-01-01 10:46:00

TABLE 2 –Events

The screenshot shows the MySQL Workbench interface. The 'SCHEMAS' pane on the left lists various databases, with 'case_2' selected. The 'Query' editor in the center contains SQL code to create the 'events' table and load data from a CSV file. The 'Result Grid' at the bottom displays the data loaded into the table.

SQL Code:

```

-- Table 2 - EVENTS
-- create table events (
--   user_id INT,
--   occurred_at VARCHAR(100),
--   event_type VARCHAR(50),
--   event_name VARCHAR(100),
--   location VARCHAR(50),
--   device VARCHAR(50),
--   user_type INT
-- );
--
-- select * from events;
--
-- load data infile "C:/ProgramData/MySQL/MySQL Server 8.0/uploads/events.csv"
-- into table events
-- fields terminated by ','
-- enclosed by '"'
-- lines terminated by '\n'
-- ignore 1 rows;
--
-- alter table events add column temp_occurred_at datetime;
-- update events set temp_occurred_at = str_to_date(occurred_at, '%d-%m-%Y %H:%i');
-- alter table events drop column occurred_at;
-- alter table events change column temp_occurred_at occurred_at datetime;

```

Result Grid:

user_id	event_type	event_name	location	device	user_type	occurred_at
10522	engagement	login	Japan	dell inspiron notebook	3	2014-05-02 11:02:00
10522	engagement	home_page	Japan	dell inspiron notebook	3	2014-05-02 11:02:00
10522	engagement	like_message	Japan	dell inspiron notebook	3	2014-05-02 11:03:00
10522	engagement	view_inbox	Japan	dell inspiron notebook	3	2014-05-02 11:04:00
10522	engagement	search_run	Japan	dell inspiron notebook	3	2014-05-02 11:03:00
10522	engagement	search_run	Japan	dell inspiron notebook	3	2014-05-02 11:03:00
10612	engagement	login	Netherlands	iphone 5	1	2014-05-01 09:59:00
10612	engagement	like_message	Netherlands	iphone 5	1	2014-05-01 10:00:00
10612	engagement	send_message	Netherlands	iphone 5	1	2014-05-01 10:00:00
10617	engagement	home_page	Netherlands	iphone 5	1	2014-05-01 10:01:00

TABLE 3 –Email_events

The screenshot shows the MySQL Workbench interface. The 'SCHEMAS' pane on the left lists various databases, with 'case_2' selected. The 'Query' editor in the center contains SQL code to create the 'email_events' table and load data from a CSV file. The 'Result Grid' at the bottom displays the data loaded into the table.

SQL Code:

```

-- Table 3 - EMAIL_EVENTS
-- create table email_events(
--   user_id int,
--   occurred_at varchar(100),
--   action varchar(50),
--   user_type int
-- );
--
-- select * from email_events;
--
-- load data infile "C:/ProgramData/MySQL/MySQL Server 8.0/uploads/email_events.csv"
-- into table email_events
-- fields terminated by ','
-- enclosed by '"'
-- lines terminated by '\n'
-- ignore 1 rows;
--
-- alter table email_events add column temp_occurred_at datetime;
-- update email_events set temp_occurred_at = str_to_date(occurred_at, '%d-%m-%Y %H:%i');
-- alter table email_events drop column occurred_at;
-- alter table email_events change column temp_occurred_at occurred_at datetime;

```

Result Grid:

user_id	action	user_type	occurred_at
0	sent_weekly_digest	1	2014-05-06 09:30:00
0	sent_weekly_digest	1	2014-05-13 09:30:00
0	sent_weekly_digest	1	2014-05-20 09:30:00
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
0	sent_weekly_digest	1	2014-06-17 09:30:00
0	email_open	1	2014-06-17 09:30:00
0	sent_weekly_digest	1	2014-06-24 09:30:00
0	sent_weekly_digest	1	2014-07-01 09:30:00

Task 1 - Weekly User Engagement:

- **Objective:** Measure the activeness of users on a weekly basis.
- **Conclusion/Result :-**

<u>Week_number</u>	<u>Active_user</u>
17	663
18	1068
19	1113
20	1154
21	1121
22	1186
23	1232
24	1275
25	1264

<u>Week_number</u>	<u>Active_user</u>
26	1302
27	1372
28	1365
29	1376
30	1467
31	1299
32	1225
33	1225
34	1204
35	104

➤ Screenshot of code and after running code its output

The screenshot displays the MySQL Workbench interface. The 'Query Editor' window contains the following SQL code:

```
# Task 1 - Weekly User Engagement
select extract(week from occurred_at) as week_number,
count(distinct user_id) as active_user
from events
group by week_number
```

The 'Result Grid' window shows the output of the query, which is a table with two columns: 'week_number' and 'active_user'. The data is as follows:

week_number	active_user
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

The 'Output' window at the bottom shows the execution details: '151 20:40:14 select extract(week from occurred_at) as week_number, count(distinct user_id) as active_user from events group by week_number 19 row(s) returned'. The status bar at the bottom indicates the system time as 20:43 on 14-06-2024.

Code used:-

```
select extract(week from occurred_at) as
week_number,
count(distinct user_id) as active_user
from events
group by week_number
```


Task 2 - User Growth Analysis

- Objective: Analyze the growth of users over time for a product.
- Conclusion/Result :- 33th week of 2014 has greatest number of users i.e 261 , The lowest number of users are 35th week of 2014 i.e 18 seen from the output obtained.

➤ Screenshot of code and after running code its output

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

```
select year, week_num, num_users, sum(num_users)
over(order by year, week_num) as cum_users
from(
select extract(year from created_at) as year, extract(week from created_at) as week_num,
count(distinct user_id) as num_users
from users
where state = 'active'
group by year , week_num
order by year, week_num)sub
```

The Results grid shows the output of the query, displaying columns: year, week_num, num_users, and cum_users. The data shows a steady increase in users over time, with the highest cumulative count reaching 542 in week 14 of 2013.

year	week_num	num_users	cum_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

The bottom status bar indicates that 89 rows were returned in 0.016 seconds.

Code used:-

select year, week num, num users,sum(num users)

over(order by year,week num) as cum users

from(

select extract(year from created at) as year, extract(week from
created at) as week num,

count(distinct user id) as num users

from users

where state ='active'

group by year ,week num

order by year, week num)sub

Task 3- Weekly Retention Analysis

- **Objective: Analyze the retention of users on a weekly basis after signing up for a product**
- **Conclusion/Result:-**

Total engaged users	Total retained users
317	236

Screenshot of code and after running code its output

The screenshot displays the MySQL Workbench interface. The central pane shows a SQL query titled 'Task 3- Weekly Retention Analysis'. The query uses Common Table Expressions (CTEs) to calculate the number of engaged and retained users. The 'Result Grid' at the bottom shows the output of the query, which consists of two columns: 'total_engaged_users' with a value of 317, and 'retained_users' with a value of 236. The left sidebar shows the 'SCHEMAS' pane with a tree view of the database structure, including tables like 'email_events', 'events', and 'users'. The bottom status bar indicates the current session and the date and time.

```
102 with cte1 as (  
103   select distinct user_id,  
104   extract (week from occurred_at) as signup_week  
105   from events  
106   where event_type = 'signup_flow'  
107   and event_name = complete_signup and extract (week from occurred_at)),  
108 cte2 as(select distinct user_id,  
109   extrxt (week from occurred_at) as engagement_week  
110   from events  
111   where event_type = 'engagement')  
112 select count(user_id) total_engaged_users,  
113 sum(case when retension_week > 0 then 1 else end) as retained_users  
114 from (select a.user_id , a.signup_week,  
115   b.engagemnt_week,b.engagemnt_week-a.signup_week as retention_week  
116   from cte1 a  
117   left join cte2 b  
118   on a.user_id =b.user_id  
119   order by a.user_id)sub  
120
```

total_engaged_users	retained_users
317	236

Code used:-

with cte1 as (

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)),

cte2 as(select distinct user id,

extrxt (week from occurred at) as engagement week

from events

where event_type = 'engagement')

select count(user_id) total_engaged_users,

sum(case when retention_week > 0 then 1 else end) as retained_users

from (select a.user_id, a.signup_week,

b.engagement_week, b.engagement_week - a.signup_week as retention_week

from cte1 a

left join cte2 b

on a.user_id = b.user_id

order by a.user_id)sub

Task 4 -Weekly Engagement Per Device

- **Objective: Measure the activeness of users on a weekly basis per device**
- **Conclusion/Result:-**

weeknum	device	Usercnt
2014-18	Acer aspire desktop	10
2014-18	Acer aspire notebook	21
2014-18	Amazon fire phone	4
2014-18	Asus chromebook	23
2014-18	Dell inspiron desktop	21

- **Screenshot of code and after running code its output**

The screenshot displays the MySQL Workbench interface. The SQL editor contains the following code:

```
124
125
126 # task 4 - weekly user engagement
127 with cte as (select extract (year from occurred_at)||'-'||extract(week from
128 occurred_at) as weeknum device ,
129 count(distinct user_id) as usercnt
130 from event
131 where event_type ='engagement'
132 group by weeknum,device
133 order by weeknum)
134 select weeknum, device , usercnt
135 from cte
```

The Results tab shows the output of the query:

weeknum	device	usercnt
2014-18	acer aspire desktop	10
2014-18	acer aspire notebook	21
2014-18	amazon fire phone	4
2014-18	asus chromebook	23
2014-18	dell inspiron desktop	21

The bottom panel shows the Action Output and Messages. The Action Output indicates that 19 rows were returned. The Messages panel shows three error messages (Error Code: 1064) related to SQL syntax errors in the query.

Code used - with cte as (select extract (year from occurred_at)||'-'||extract(week from occurred_at) as weeknum device ,
count(distinct user_id) as usercnt
from event
where event_type ='engagement'
group by weeknum,device
order by weeknum)
select weeknum, device , usercnt

Task 5 - Email Engagement Analysis

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

- **Conclusion/Result:-**

Email open rate	Email click rate
31.1921	10.4745

Out of total sent mail only 35.73 % mail are opened and 15.74% mail are only clicked.

- Screenshot of code and after running code its output

The screenshot displays the MySQL Workbench interface. The 'Query' tab is active, showing a SQL query for email engagement analysis. The query calculates the email open rate and click rate. The 'Result Grid' shows the output with two columns: 'email_open_rate' and 'email_click_rate', with values 31.1921 and 10.4745 respectively. The 'Table: events' structure is visible on the left, and the 'Action Output' pane at the bottom shows the execution log.

```
SELECT  
100 sum(case when email_cat 'email_open' then 1 else 0 end)/  
sum(case when email_cat 'email sent' then 1 else 0 end) as email_open_rate, 100 sum(case when email_cat 'email clicked' then 1 else 0 end)/ sum(case when email_cat 'email sent' then 1 else 0  
end) as email_click_rate from  
(select*,CaseWhen action in ('sent_weekly_digest', 'sent_reengagement_email') then 'email_sent' when action in ('email_open')  
then 'email_open' when action in ('email_clickthrough') then 'email_clicked' end as email_cat  
from email_events) sub
```

email_open_rate	email_click_rate
31.1921	10.4745

Code used:-

select

100 sum(case when email_cat 'email_open' then 1 else 0 end)/

sum(case when email_cat 'email sent' then 1 else 0 end) as email_open_rate, 100 sum(case when email_cat 'email clicked' then 1 else 0 end)/ sum(case when email_cat 'email sent' then 1 else 0 end)

as email_click_rate from

(select*,CaseWhen action in ('sent_weekly_digest', 'sent_reengagement_email') then 'email_sent' when action in ('email_open')

then 'email_open' when action in ('email_clickthrough') then 'email_clicked' end as email_cat from email_events) sub