



OPERATION ANALYTICS AND INVESTIGATING METRIC SPIKES

Project description:

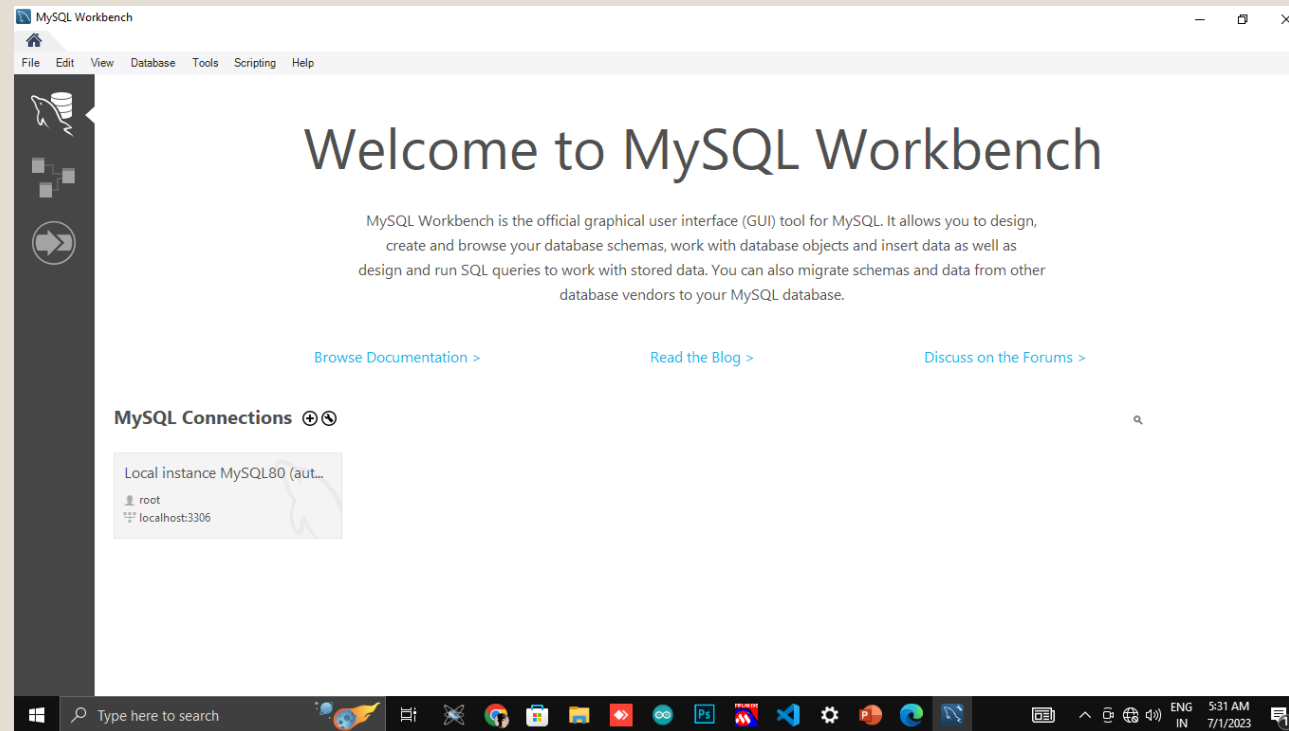
- Operation Analytics is the analysis done for the complete end to end operations of a company.
- With provided data sets, tables from the which required insights and answers for the questions has been provided.
- This project helps in investigating metric spike.
- With the help of this project, the company finds the areas on which it must improve upon.

Approach:

- In this project , I have handled case study-1 (job data) and case study-2 (investigating metric spike) separately.
- In this project, I have used case statements and week() functions to extract information on weekly basis.

Tech-Stack used:

- For this project I have used “MySQL workbench” to perform the queries.



Insights:

- This project is very difficult to accomplish, and it is very useful for those who are looking for advanced knowledge of SQL.
- Through this project I gained better understanding of the given dataset .

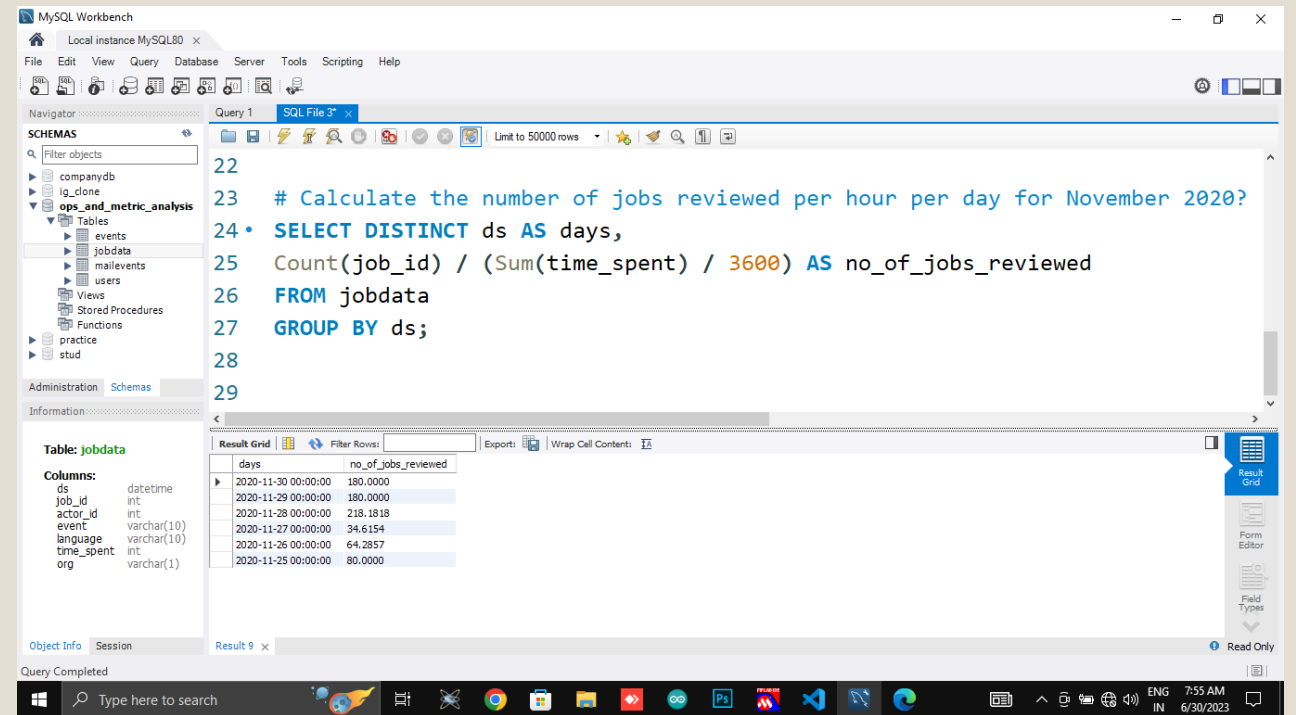
Results:

- Through this project I have achieved the knowledge of writing some complex queries.

Case Study 1 (Job Data)

Number of jobs reviewed:

The number of jobs has been reviewed per hour per day for November 2020.



The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' tree with 'companydb' expanded, revealing tables like 'events', 'jobdata', 'mailevents', and 'users'. The main query editor shows a SQL query to calculate the number of jobs reviewed per hour per day for November 2020. The 'Result Grid' at the bottom shows the output of the query.

```
22
23 # Calculate the number of jobs reviewed per hour per day for November 2020?
24 • SELECT DISTINCT ds AS days,
25     Count(job_id) / (Sum(time_spent) / 3600) AS no_of_jobs_reviewed
26 FROM jobdata
27 GROUP BY ds;
28
29
```

days	no_of_jobs_reviewed
2020-11-30 00:00:00	180.0000
2020-11-29 00:00:00	180.0000
2020-11-28 00:00:00	218.1818
2020-11-27 00:00:00	34.6154
2020-11-26 00:00:00	64.2857
2020-11-25 00:00:00	80.0000

Table: jobdata

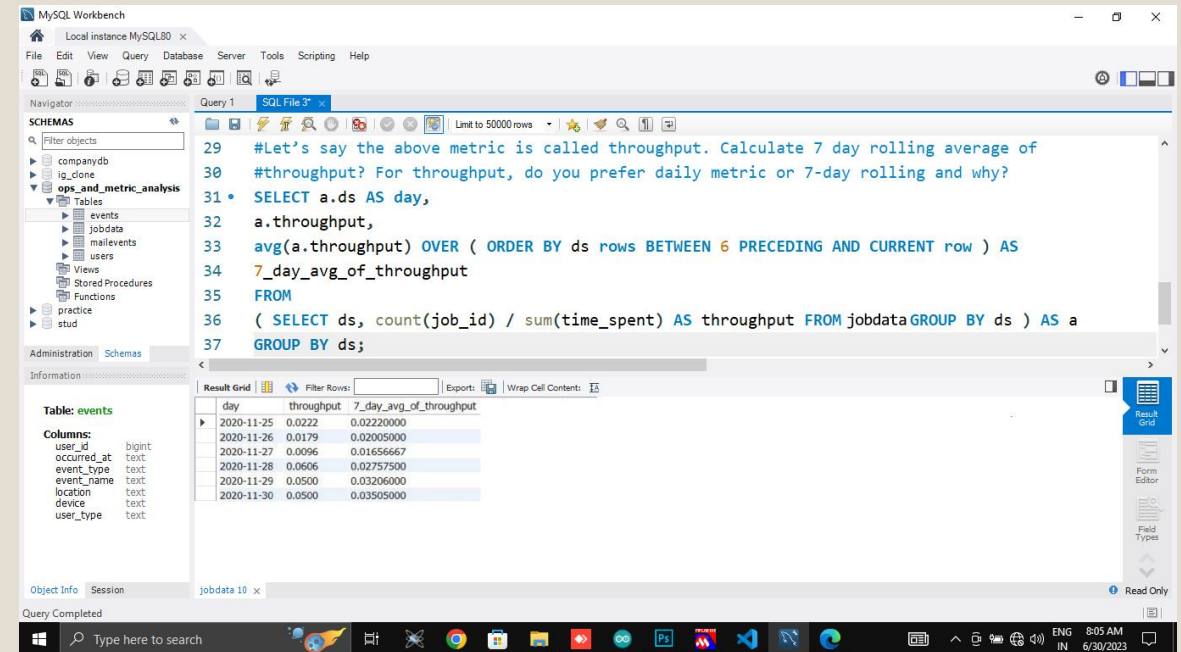
Columns:

- ds: datetime
- job_id: int
- actor_id: int
- event: varchar(10)
- language: varchar(10)
- time_spent: int
- org: varchar(1)

Query Completed

Throughput:

7 day rolling average of throughput has been calculated.



The screenshot shows the MySQL Workbench interface. The SQL Editor contains a query to calculate a 7-day rolling average of throughput. The query is as follows:

```
29 #Let's say the above metric is called throughput. Calculate 7 day rolling average of
30 #throughput? For throughput, do you prefer daily metric or 7-day rolling and why?
31 • SELECT a.ds AS day,
32       a.throughput,
33       avg(a.throughput) OVER ( ORDER BY ds rows BETWEEN 6 PRECEDING AND CURRENT row ) AS
34       7_day_avg_of_throughput
35 FROM
36 ( SELECT ds, count(job_id) / sum(time_spent) AS throughput FROM jobdata GROUP BY ds ) AS a
37 GROUP BY ds;
```

The Result Grid shows the following data:

day	throughput	7_day_avg_of_throughput
2020-11-25	0.0222	0.02220000
2020-11-26	0.0179	0.02005000
2020-11-27	0.0096	0.01656667
2020-11-28	0.0606	0.02757500
2020-11-29	0.0500	0.03206000
2020-11-30	0.0500	0.03505000

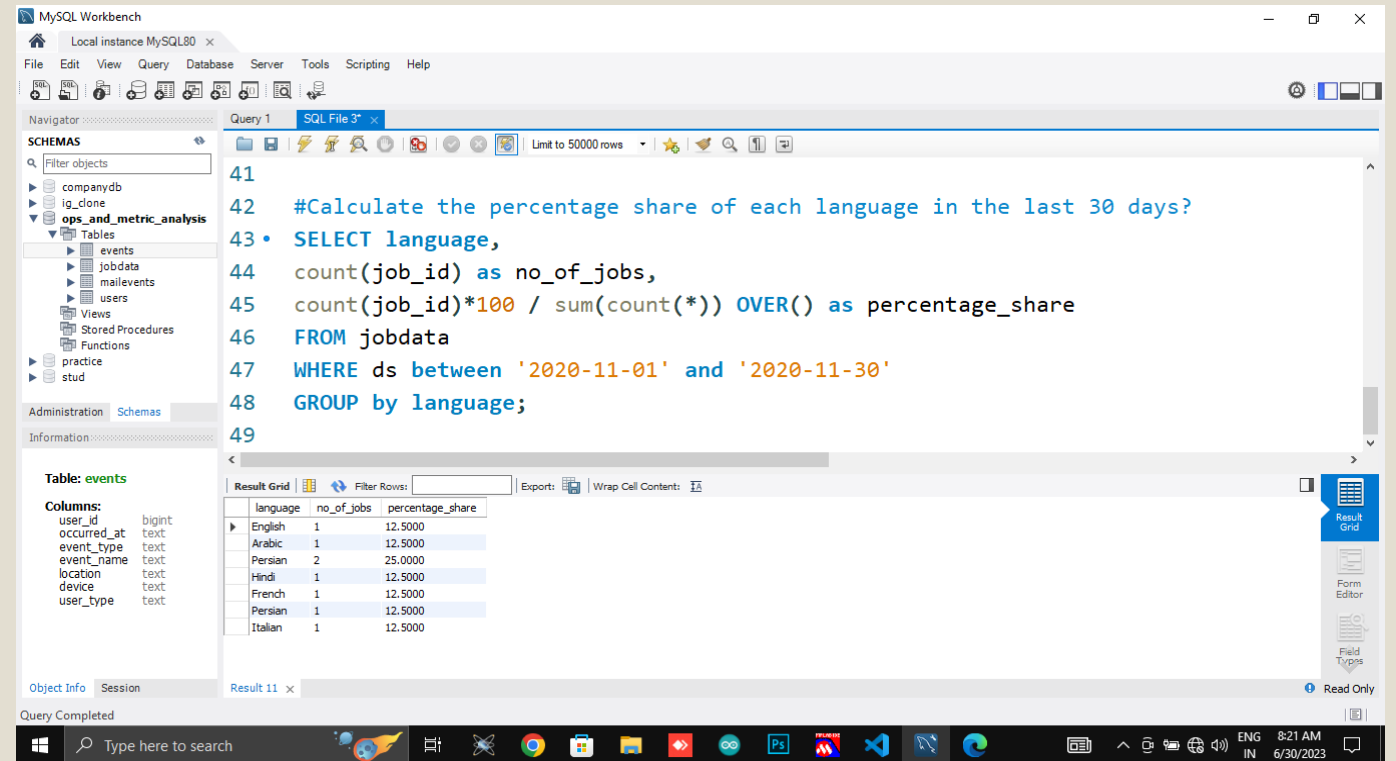
The left sidebar shows the Schemas pane with the following structure:

- companydb
 - ig_done
 - ops_and_metric_analysis
 - events
 - jobdata
 - mail_events
 - users
 - Views
 - Stored Procedures
 - Functions
 - pradice
 - stud

The bottom status bar indicates "Query Completed".

Percentage share of each language:

The percentage share of each language in the last 30 days has been calculated.



The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' panel with a tree view of databases and tables. The 'ops_and_metric_analysis' database is selected, showing tables like 'events', 'jobdata', 'mail_events', 'users', 'Views', 'Stored Procedures', 'Functions', 'practice', and 'stud'. The 'Table: events' is expanded, showing columns: 'user_id' (bigint), 'occurred_at' (text), 'event_type' (text), 'event_name' (text), 'location' (text), 'device' (text), and 'user_type' (text).

The main query editor shows the following SQL query:

```
41
42 #Calculate the percentage share of each language in the last 30 days?
43 • SELECT language,
44 count(job_id) as no_of_jobs,
45 count(job_id)*100 / sum(count(*)) OVER() as percentage_share
46 FROM jobdata
47 WHERE ds between '2020-11-01' and '2020-11-30'
48 GROUP by language;
49
```

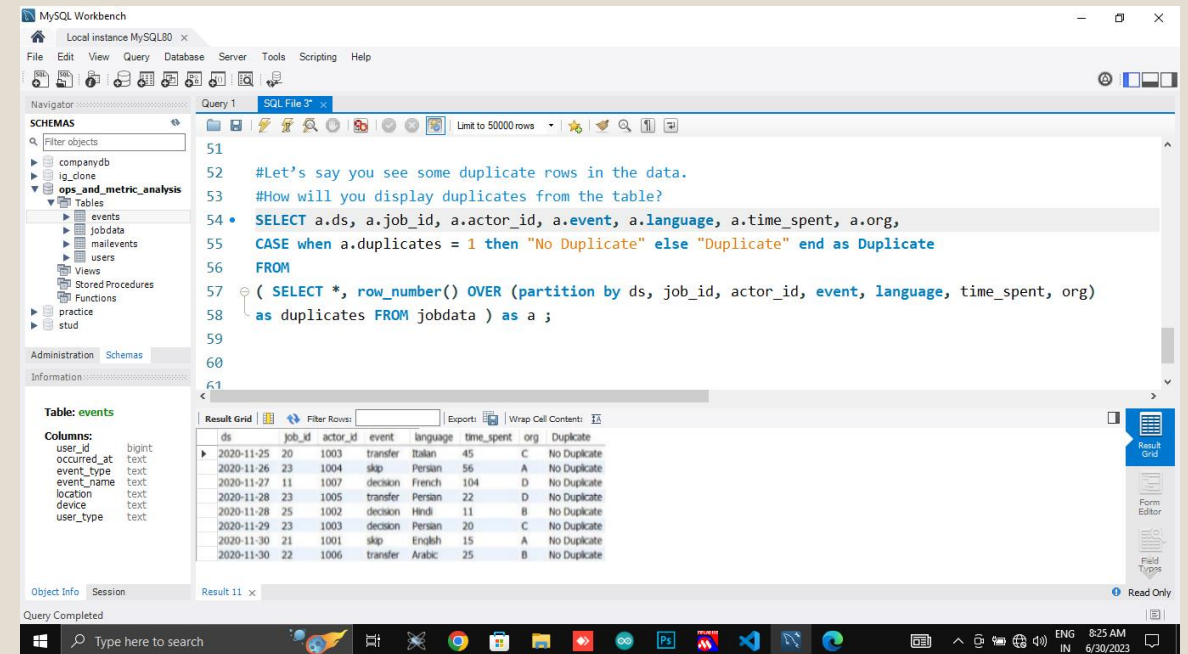
The 'Result Grid' shows the following data:

language	no_of_jobs	percentage_share
English	1	12.5000
Arabic	1	12.5000
Persian	2	25.0000
Hindi	1	12.5000
French	1	12.5000
Persian	1	12.5000
Italian	1	12.5000

The bottom status bar indicates 'Query Completed' and 'Result 11 x'. The Windows taskbar at the bottom shows the date and time as 8:21 AM on 6/30/2023.

Duplicate rows:

Rows without duplicates
has been displayed from
the table.



The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'companydb' selected, and 'events' table highlighted under 'Tables'. The main editor shows a SQL query to identify duplicate rows in the 'events' table. The query uses a subquery with 'row_number()' over a partition of columns to flag duplicates. The 'Result Grid' at the bottom shows the output of the query, with columns for 'ds', 'job_id', 'actor_id', 'event', 'language', 'time_spent', 'org', and 'Duplicate'. The 'Duplicate' column contains values 'No Duplicate' or 'Duplicate'.

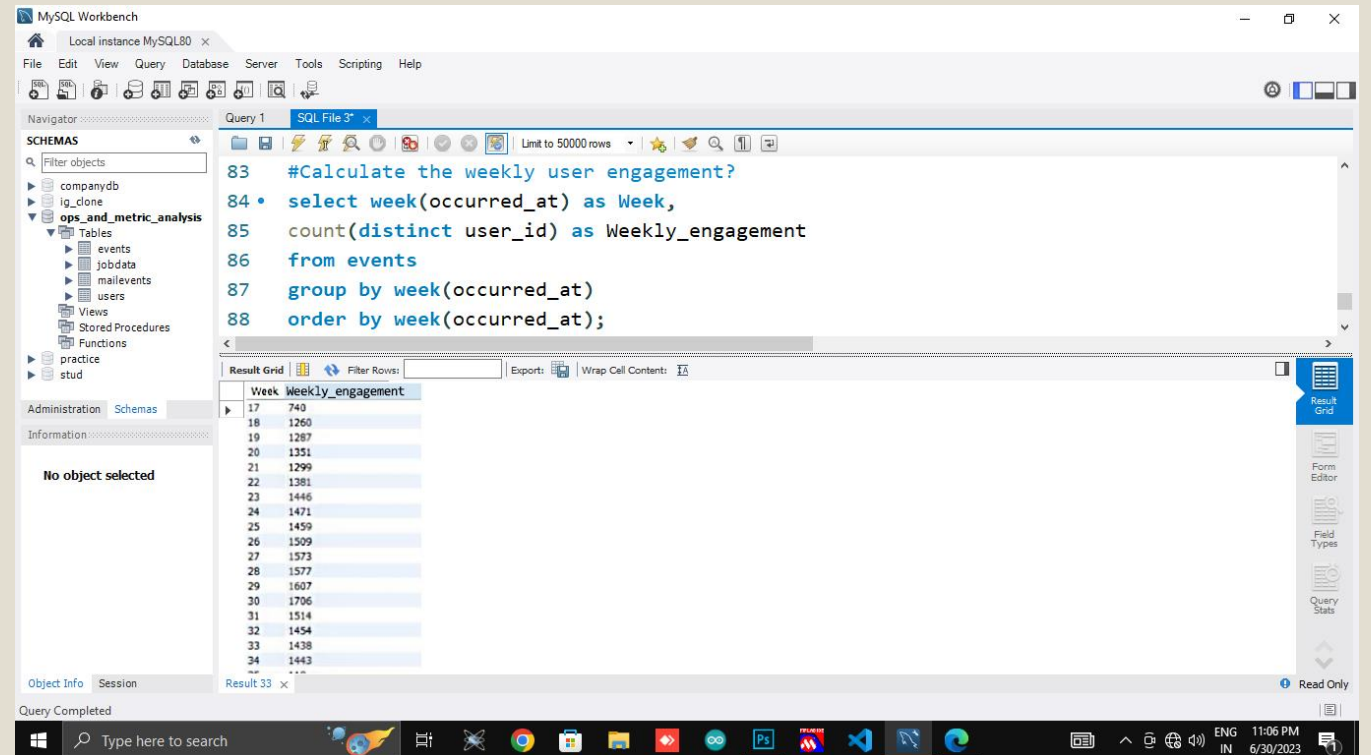
```
51
52 #Let's say you see some duplicate rows in the data.
53 #How will you display duplicates from the table?
54 • SELECT a.ds, a.job_id, a.actor_id, a.event, a.language, a.time_spent, a.org,
55 CASE when a.duplicates = 1 then "No Duplicate" else "Duplicate" end as Duplicate
56 FROM
57 ( SELECT *, row_number() OVER (partition by ds, job_id, actor_id, event, language, time_spent, org)
58 as duplicates FROM jobdata ) as a ;
59
60
61
```

ds	job_id	actor_id	event	language	time_spent	org	Duplicate
2020-11-25	20	1003	transfer	Italian	45	C	No Duplicate
2020-11-26	23	1004	skip	Persian	56	A	No Duplicate
2020-11-27	11	1007	decision	French	104	D	No Duplicate
2020-11-28	23	1005	transfer	Persian	22	D	No Duplicate
2020-11-28	25	1002	decision	Hindi	11	B	No Duplicate
2020-11-29	23	1003	decision	Persian	20	C	No Duplicate
2020-11-30	21	1001	skip	English	15	A	No Duplicate
2020-11-30	22	1006	transfer	Arabic	25	B	No Duplicate

Case Study 2 (Investigating metric spike)

User Engagement:

The weekly user engagement has been selected with week.



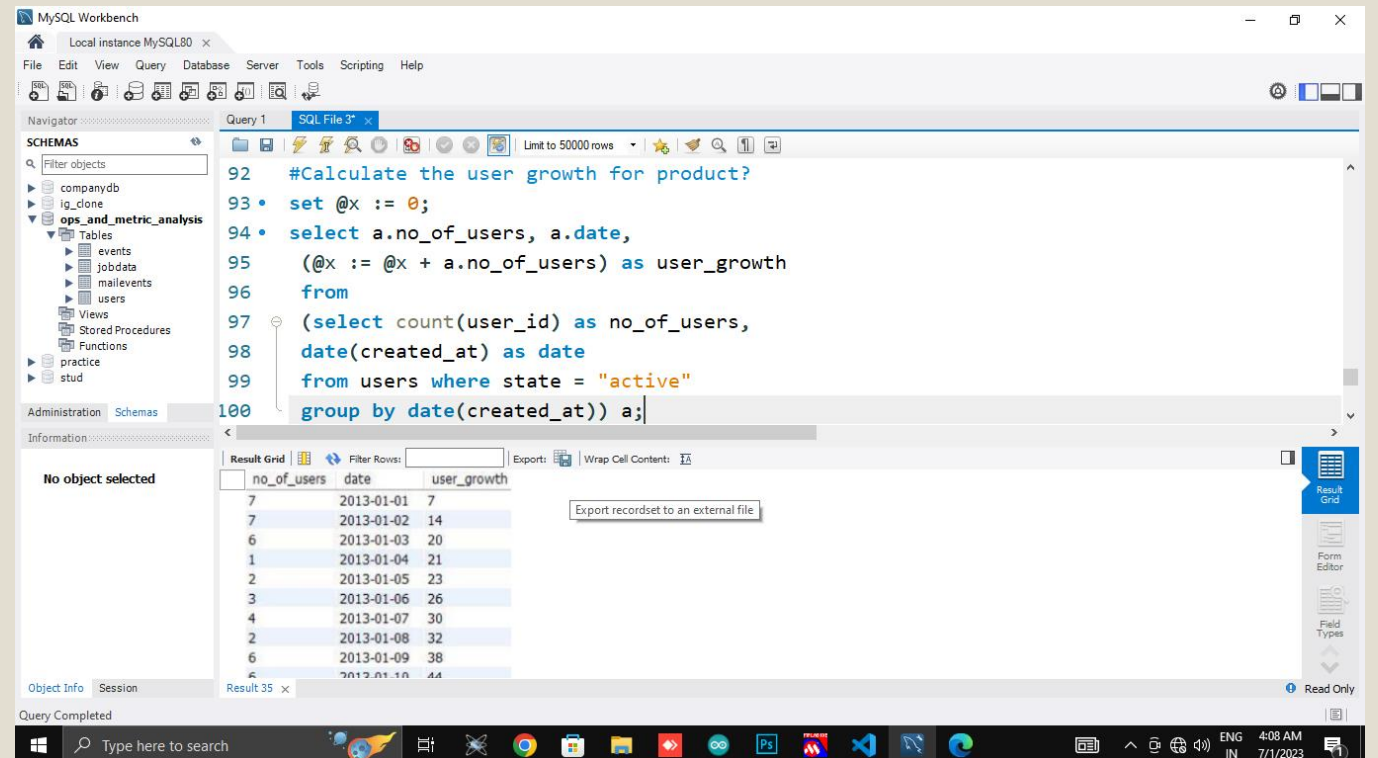
The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' tree with 'companydb' expanded, and 'ops_and_metric_analysis' selected. The main editor shows a SQL query: `#Calculate the weekly user engagement?`
`select week(occurred_at) as Week,`
`count(distinct user_id) as Weekly_engagement`
`from events`
`group by week(occurred_at)`
`order by week(occurred_at);`
The 'Result Grid' at the bottom shows the query results in a table with two columns: 'Week' and 'Weekly_engagement'. The data is as follows:

Week	Weekly_engagement
17	740
18	1260
19	1287
20	1351
21	1299
22	1381
23	1446
24	1471
25	1459
26	1509
27	1573
28	1577
29	1607
30	1706
31	1514
32	1454
33	1438
34	1443

The status bar at the bottom indicates 'Query Completed'.

User Growth:

Amount of users growing over time for a product has been selected



The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' panel with a tree view containing 'companydb', 'ig_clone', and 'ops_and_metric_analysis'. The 'ops_and_metric_analysis' schema is expanded, showing tables like 'events', 'jobdata', 'mailevents', and 'users'. The main editor window shows a SQL query titled 'Query 1' with the following code:

```
92 #Calculate the user growth for product?
93 • set @x := 0;
94 • select a.no_of_users, a.date,
95       (@x := @x + a.no_of_users) as user_growth
96 from
97 (select count(user_id) as no_of_users,
98  date(created_at) as date
99  from users where state = "active"
100  group by date(created_at)) a;
```

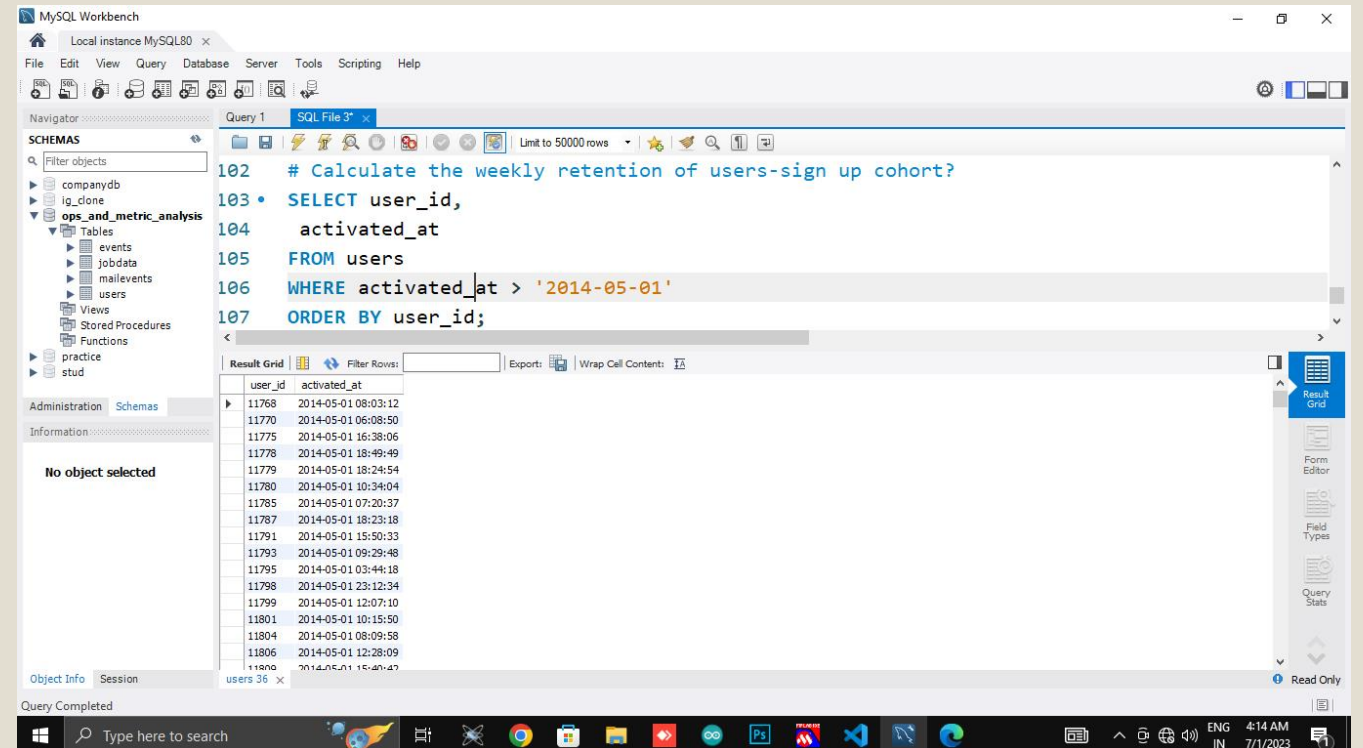
Below the query editor, the 'Result Grid' is visible, showing the output of the query. The table has three columns: 'no_of_users', 'date', and 'user_growth'. The data is as follows:

no_of_users	date	user_growth
7	2013-01-01	7
7	2013-01-02	14
6	2013-01-03	20
1	2013-01-04	21
2	2013-01-05	23
3	2013-01-06	26
4	2013-01-07	30
2	2013-01-08	32
6	2013-01-09	38
6	2013-01-10	44

The bottom status bar indicates 'Query Completed'.

Weekly Retention:

The weekly retention of users-sign up cohort has been calculated.



The screenshot displays the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows a tree view with databases like 'companydb', 'ig_clone', and 'ops_and_metric_analysis'. The 'ops_and_metric_analysis' database is selected, showing tables such as 'events', 'jobdata', 'mailevents', and 'users'. The main query editor on the right contains the following SQL code:

```
102 # Calculate the weekly retention of users-sign up cohort?
103 • SELECT user_id,
104       activated_at
105 FROM users
106 WHERE activated_at > '2014-05-01'
107 ORDER BY user_id;
```

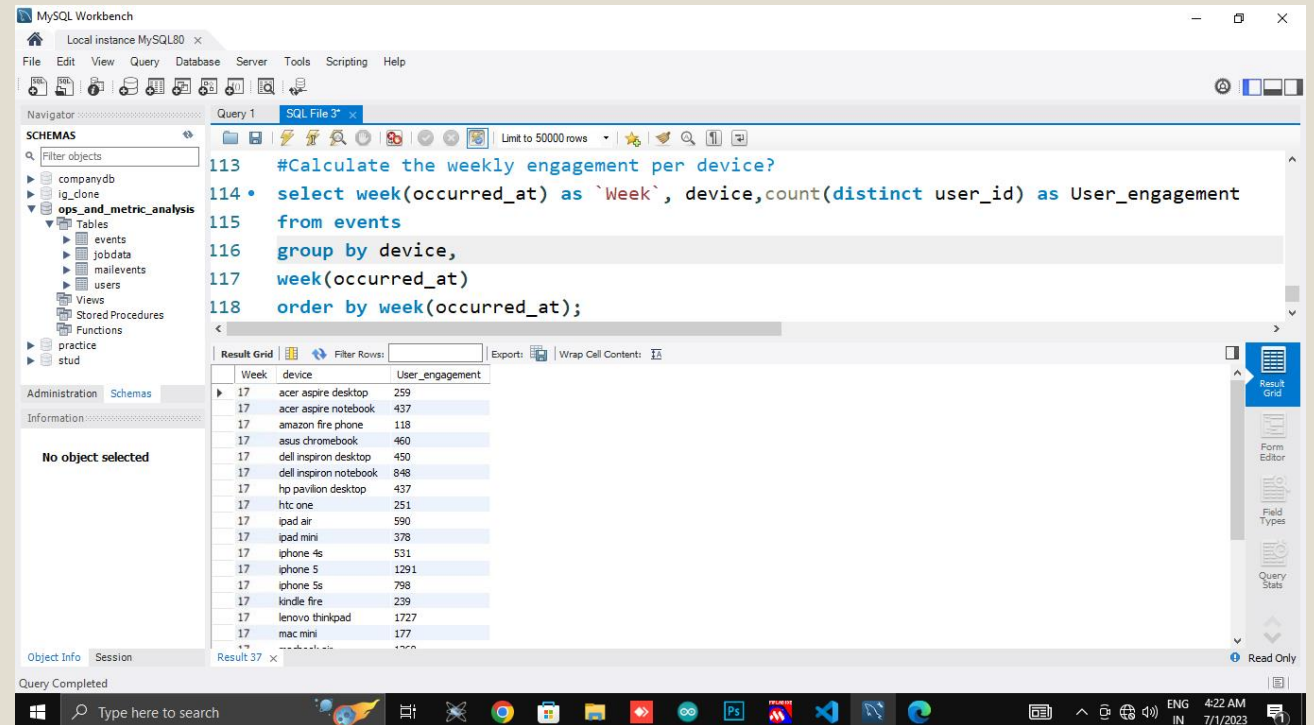
Below the query editor, the 'Result Grid' shows the output of the query. It contains two columns: 'user_id' and 'activated_at'. The results are as follows:

user_id	activated_at
11768	2014-05-01 08:03:12
11770	2014-05-01 06:08:50
11775	2014-05-01 16:38:06
11778	2014-05-01 18:49:49
11779	2014-05-01 18:24:54
11780	2014-05-01 10:34:04
11785	2014-05-01 07:20:37
11787	2014-05-01 18:23:18
11791	2014-05-01 15:50:33
11793	2014-05-01 09:29:48
11795	2014-05-01 03:44:18
11798	2014-05-01 23:12:34
11799	2014-05-01 12:07:10
11801	2014-05-01 10:15:50
11804	2014-05-01 08:09:58
11806	2014-05-01 12:28:09
11809	2014-05-01 15:40:47

The bottom status bar indicates 'Query Completed' and 'users 36 x'. The Windows taskbar at the bottom shows the time as 4:14 AM on 7/1/2023.

Weekly Engagement:

The weekly user engagement per device has been calculated.



The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' tree with 'companydb' expanded, containing 'events', 'jobdata', 'mailevents', and 'users'. The main editor window shows a SQL query in 'Query 1' (SQL File 3*):

```
113 #Calculate the weekly engagement per device?
114 • select week(occurred_at) as `Week`, device, count(distinct user_id) as User_engagement
115 from events
116 group by device,
117 week(occurred_at)
118 order by week(occurred_at);
```

The 'Result Grid' at the bottom shows the query results for 37 rows. The columns are 'Week', 'device', and 'User_engagement'. The data is sorted by 'Week'.

Week	device	User_engagement
17	acer aspire desktop	259
17	acer aspire notebook	437
17	amazon fire phone	118
17	asus chromebook	460
17	dell inspiron desktop	450
17	dell inspiron notebook	848
17	hp pavilion desktop	437
17	htc one	251
17	ipad air	590
17	ipad mini	378
17	iphone 4s	531
17	iphone 5	1291
17	iphone 5s	798
17	kindle fire	239
17	lenovo thinkpad	1727
17	mac mini	177
17	macbook air	1220

The bottom status bar indicates 'Query Completed' and the system clock shows 4:22 AM on 7/1/2023.

Email Engagement:

The user email engagement metrics has been calculated.

The screenshot displays the MySQL Workbench interface with a SQL query executed in the Query Editor. The query calculates email engagement metrics by grouping data by week. The results are shown in a table with columns: Week, weekly_digest, reengagement_mail, opened_email, and email_clickthrough.

```
121 #Calculate the email engagement metrics?
122 • select week(occurred_at) as Weeks,
123 count( distinct ( case when action = "sent_weekly_digest"
124 then user_id end )) as weekly_digest,
125 count( distinct ( case when action = "sent_reengagement_email"
126 then user_id end )) as reengagement_mail,
127 count( distinct ( case when action = "email_open"
128 then user_id end )) as opened_email,
129 count( distinct ( case when action = "email_clickthrough"
130 then user_id end )) as email_clickthrough
131 from mailevents
132 group by week(occurred_at)
133 order by week(occurred_at);
```

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

The interface also shows the Navigator pane on the left with a tree view of databases and tables, and the Information pane at the bottom.

Conclusion:

- Through this project, I gained advanced knowledge of SQL queries.

- Thank you.