

Mayank Kumar Gandharv

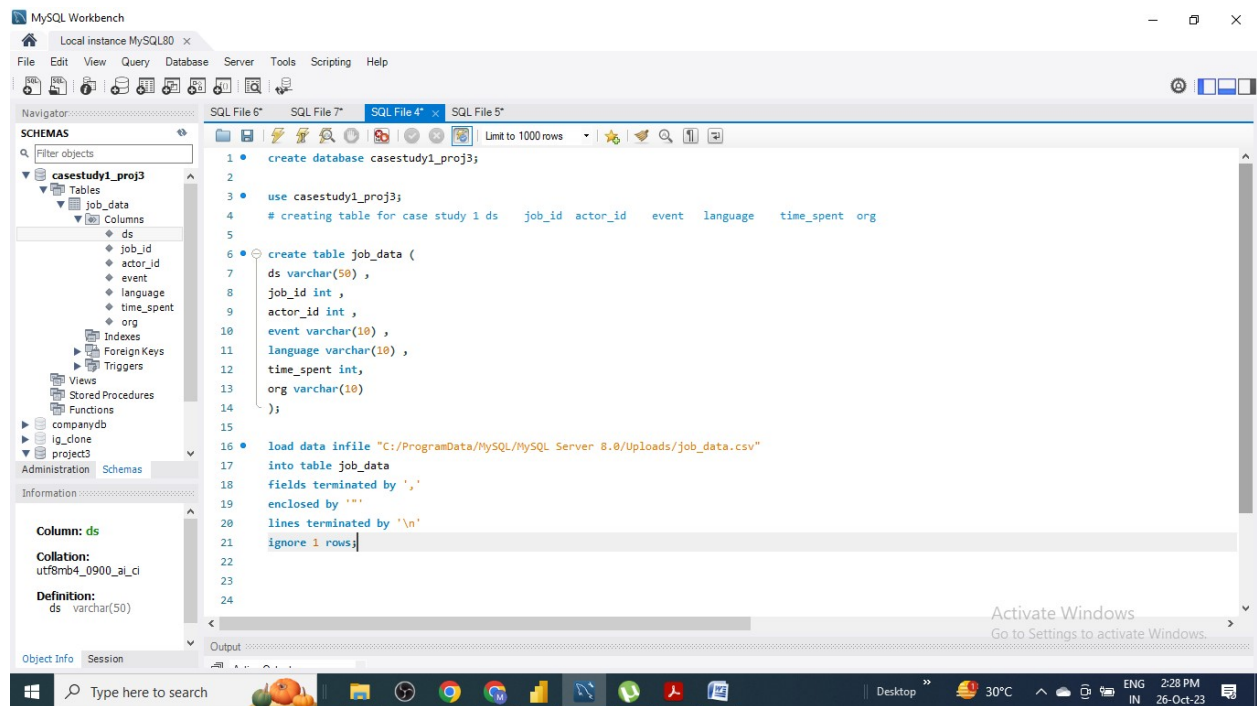
## Trinity Project 3

### Operation Analytics and Investigating Metric Spike

#### CASE STUDY 1: JOB DATA ANALYSIS

##### Creating Database and Tables:

First I imported the provided database of case study 1 in MySQL.



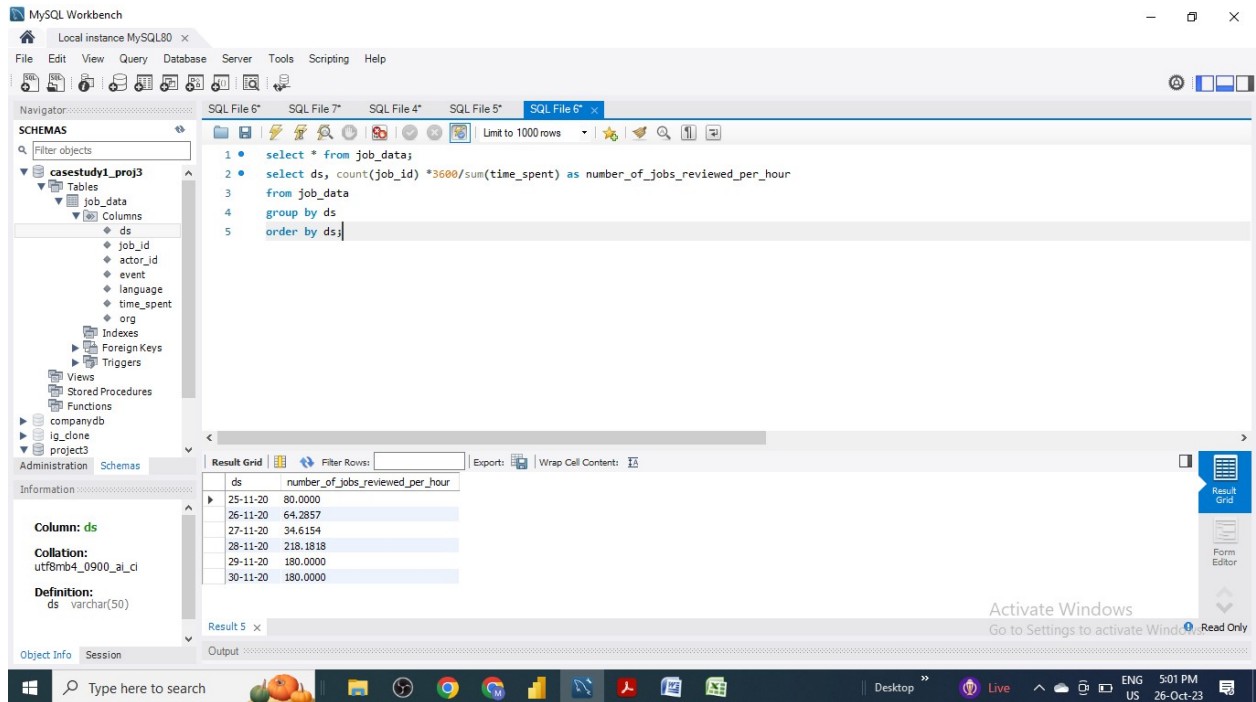
After importing the dataset by creating the table job\_data, I further did my analysis using MySQL Workbench.

##### 1. Jobs Reviewed Over Time:

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

For this query I followed following steps:

- First I selected ds(date format in text) ,count of jobs reviewed per day, sum of time spent for that particular day.
- I divided the count of job reviewed per day by sum of time spent (seconds) for that day . Like that I got the total number of jobs reviewed for each day per second.
- Now to calculate this rate (number of jobs per second) in hours I multiplied the whole equation by 3600(60min\*60sec).
- Lastly I got the desired output. The syntax is given in the screenshot below.



## 2. Throughput Analysis:

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

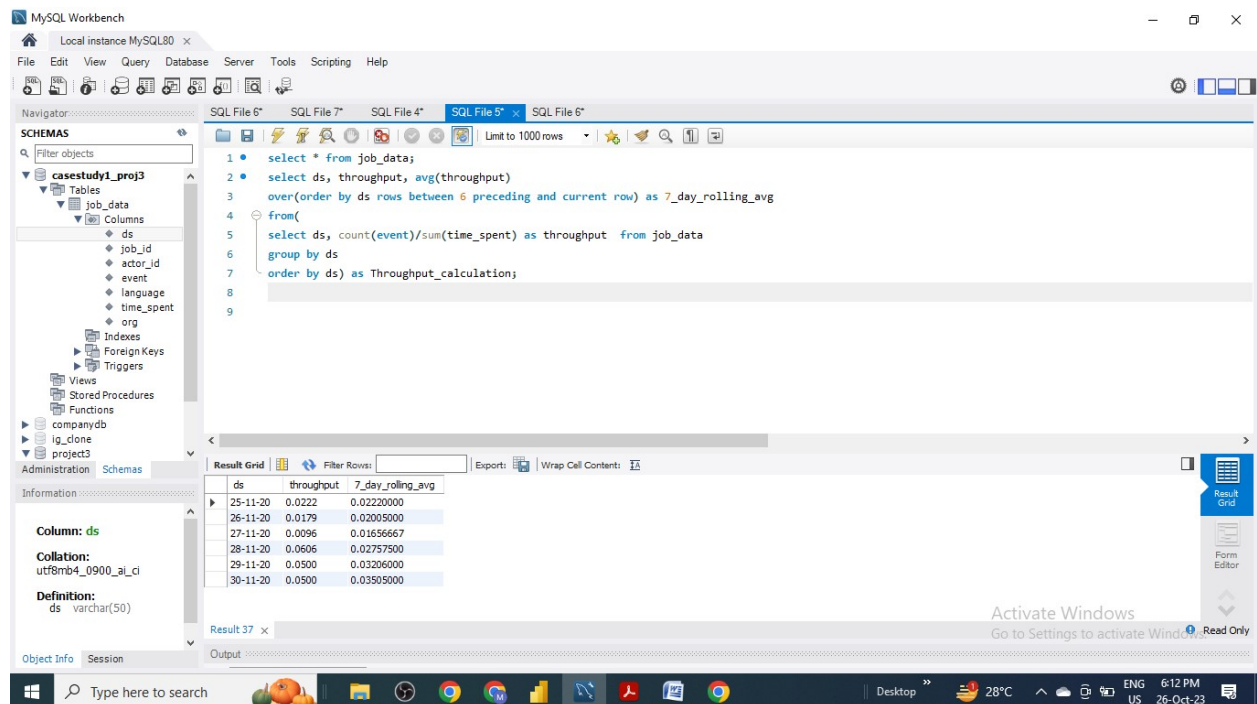
For this query I followed following steps:

- I selected ds (date), count(events), sum(time\_spent)
- Then to calculate throughput i.e number of events per second (count(events)/sum(time\_spent)), grouped it according to ds and ordered in ascending order.

ds	throughput
25-11-20	0.0222
26-11-20	0.0179
27-11-20	0.0096
28-11-20	0.0606
29-11-20	0.0500
30-11-20	0.0500

- c. Now, to calculate the 7 day rolling average or moving average, I used window function .
- d. For 7 day rolling average first I selected ds(date) and average of throughput on basis of ds such that **average should be inclined from the current row to 6 preceding rows.**

The syntax for this query is given in the screenshot below:



According to me calculating 7 day rolling average for this case study is not at all required because the dataset given is very less.

But, for big data moving average or rolling average is the best option in order to make a whole sense out of the data.

So, rolling average is better than daily metric but only for big data analysis because rolling average helps to derive the bigger picture in a small and quantifiable manner.

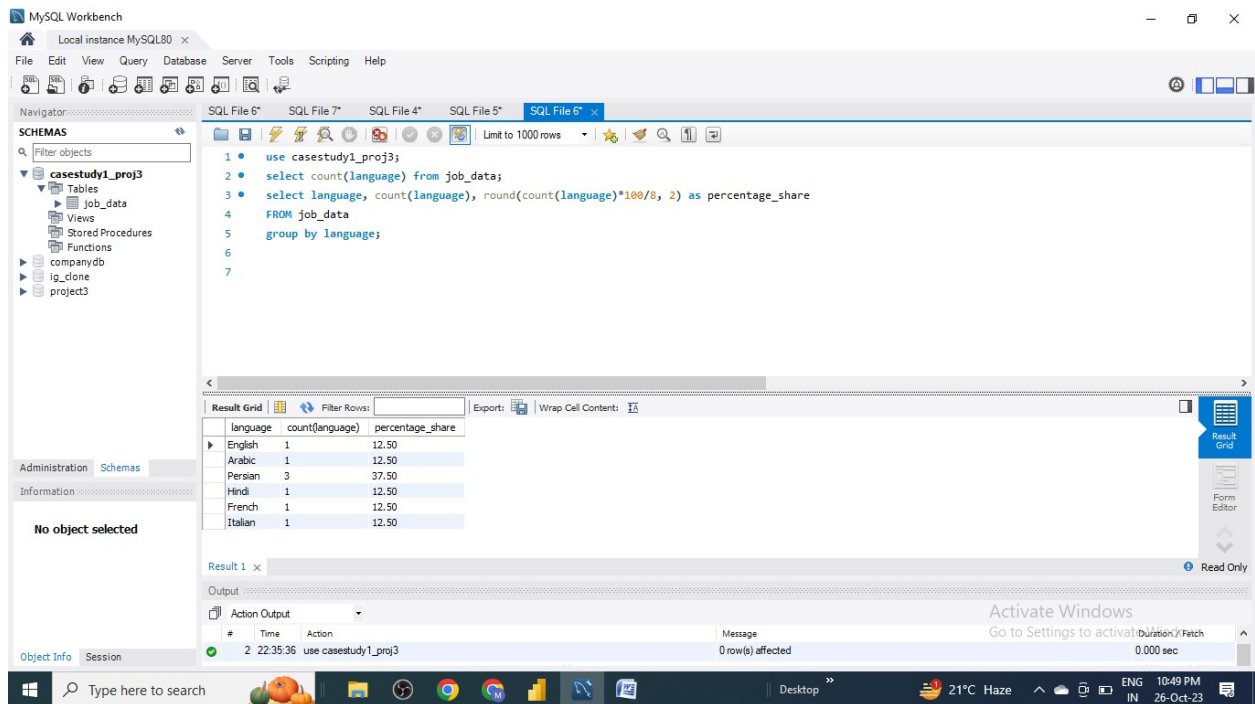
### 3. Language Share Analysis:

- a. Objective: Calculate the percentage share of each language in the last 30 days.
- b. Task: Write an SQL query to calculate the percentage share of each language over the last 30 days.

For this query I followed following step:

- a. First I calculated total count of the languages available i.e 8
- b. Then, I selected count of language grouped by language and used those grouped count and divided by 8 (total language occurrence) and finally multiplied the equation by 100. Also, rounded off the value to 2 decimal point.

Following is the syntax of the same:



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

```
1 use casestudy1_proj3;
2 select count(language) from job_data;
3 select language, count(language), round(count(language)*100/8, 2) as percentage_share
4 FROM job_data
5 group by language;
```

The Results window displays the output of the query:

language	count(language)	percentage_share
English	1	12.50
Arabic	1	12.50
Persian	3	37.50
Hindi	1	12.50
French	1	12.50
Italian	1	12.50

The status bar at the bottom indicates that 0 row(s) were affected and the execution took 0.000 seconds.

### OUTPUT:

language	count(language)	percentage_share
English	1	12.5
Arabic	1	12.5
Persian	3	37.5
Hindi	1	12.5
French	1	12.5
Italian	1	12.5

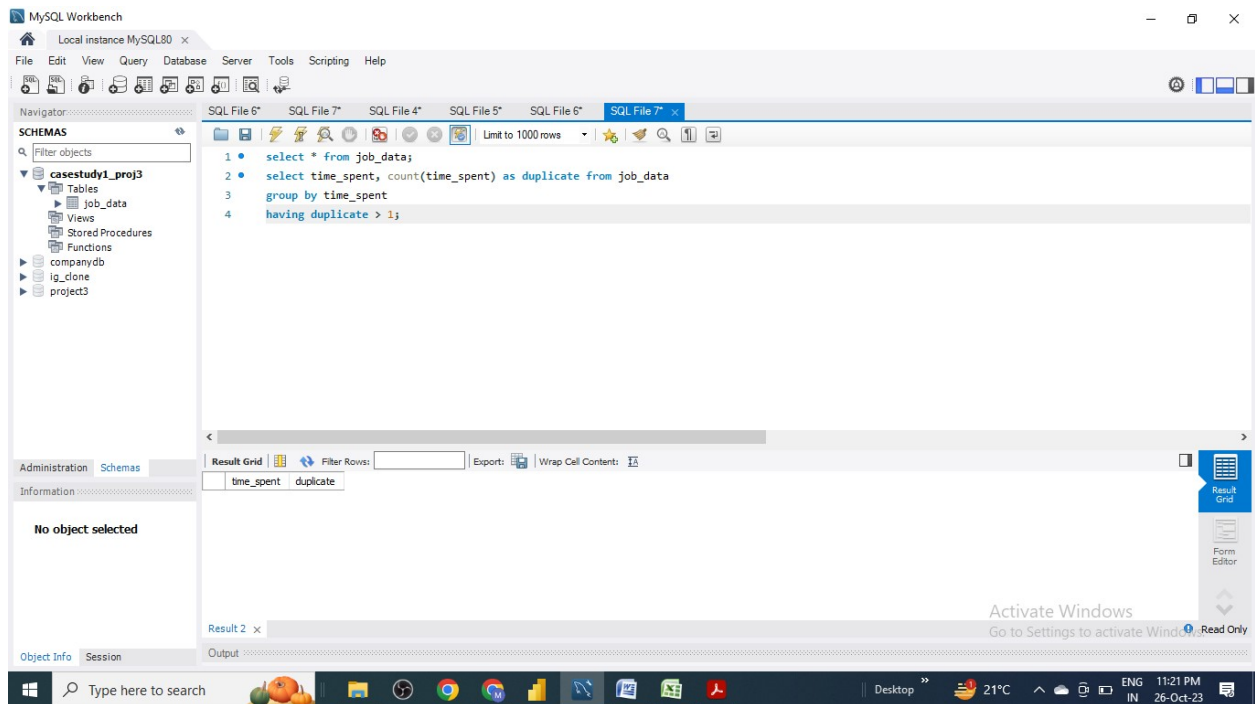
#### 4. Duplicate Rows Detection:

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

To check duplicate rows we need a variable that should occur in two or more rows.

Here every column is unique and can occur more than once except `time_spent` because it is nearly impossible to review a job at the exact same time with accuracy that too in seconds. So, in this dataset to check whether there is duplicate data we can simply have a count of time spent and group it with itself.

Here the outcome is null because there is no duplicate rows whatsoever.



## CASE STUDY 2

After importing data from given csv file into MySQL by following steps from the reference video. I was able to do the following tasks on MySQL

### A. Weekly User Engagement:

- Objective: Measure the activeness of users on a weekly basis.
- Your Task: Write an SQL query to calculate the weekly user engagement.

In order to do this sql query I used following steps:

1. First I **extracted week** (number\_of\_week) from events table, extracting from occurred\_at.
2. After extracting week I gave a **distinct count** of user\_id so that we could get filtered data on every week and no same user\_id can occur twice in the same week. Also, using **where clause** for “engagement” I filtered data on the basis of engagement type.
3. Lastly I grouped everything on the basis of extracted week (number\_of\_week).

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

```
1 use project3;
2 select * from events;
3 select extract(week from occurred_at) as number_of_week,
4 count(distinct(user_id)) as number_of_users
5 from events
6 where event_type = 'engagement'
7 group by number_of_week
8 order by number_of_week;
```

The results are displayed in a table with two columns: number\_of\_week and number\_of\_users. The data is as follows:

number_of_week	number_of_users
35	104
17	663
18	1068
19	1113
21	1121
20	1154
22	1186
34	1204
32	1225
33	1225
23	1232
25	1264
24	1275

The interface also shows a sidebar with a schema tree on the left and a 'Result Grid' panel on the right. The status bar at the bottom indicates 'Result 54' and 'Output'.

## OUTPUT:

number_of_week	number_of_users
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 :

The maximum user engagement is shown on week number 30 and the minimum is shown on week number 35.

### B. User Growth Analysis:

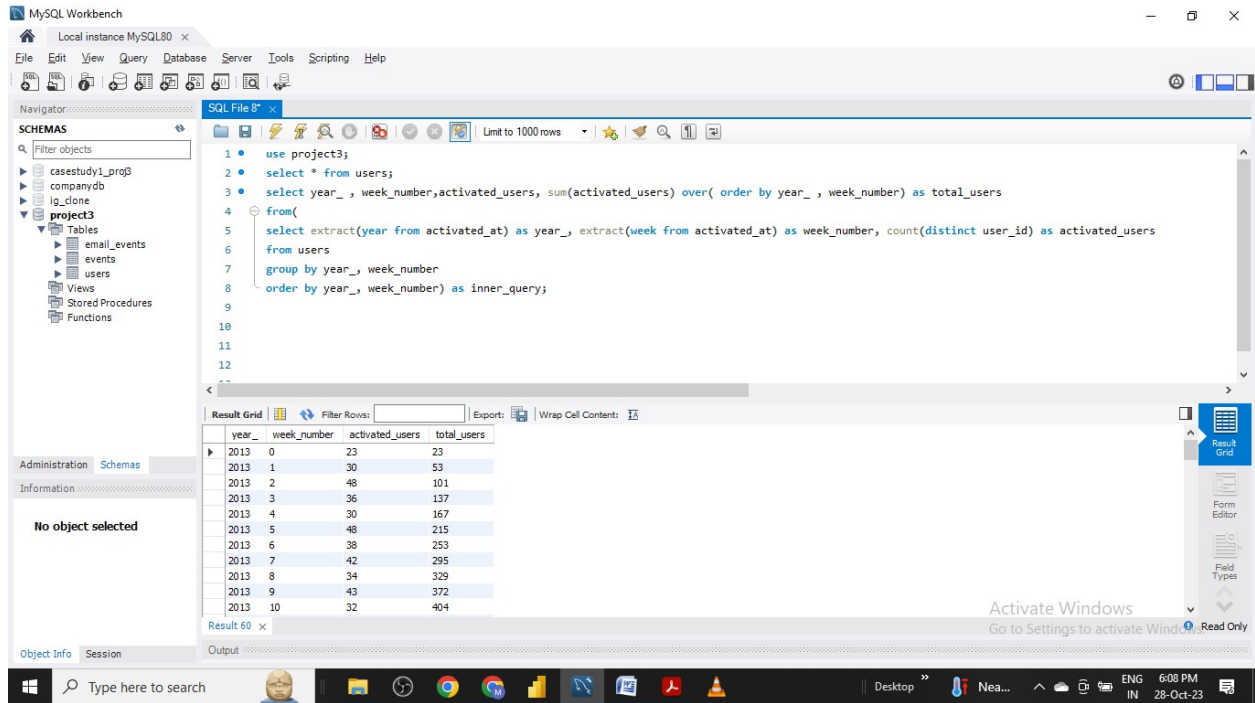
- Objective: Analyze the growth of users over time for a product.
- Your Task: Write an SQL query to calculate the user growth for the product.

In order to do this Sql query I followed the following steps:

1. Since all the users are active and distinct already, we don't need to sort data in terms of active users.
2. In order to do this query I **first extracted the active users on yearly and weekly bases** using their user\_id (which is distinct).
3. Then, I **grouped the dataset by giving a count distinct function** to the user\_id as per the extracted year and week.

4. This whole process helped to find the user growth of every week for each year (2013, 2014).
5. This query was named as **inner query** because I also calculated the total users since the starting i.e 0th week till date using sum.

The syntax is given in the screenshot below:



OUTPUT:

year_	week_number	activated_users	total_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

#### INSIGHTS:

The **maximum** number of users growth was seen in **33<sup>rd</sup> week of 2014** and **minimum** users growth in **35<sup>th</sup> week of 2014**.

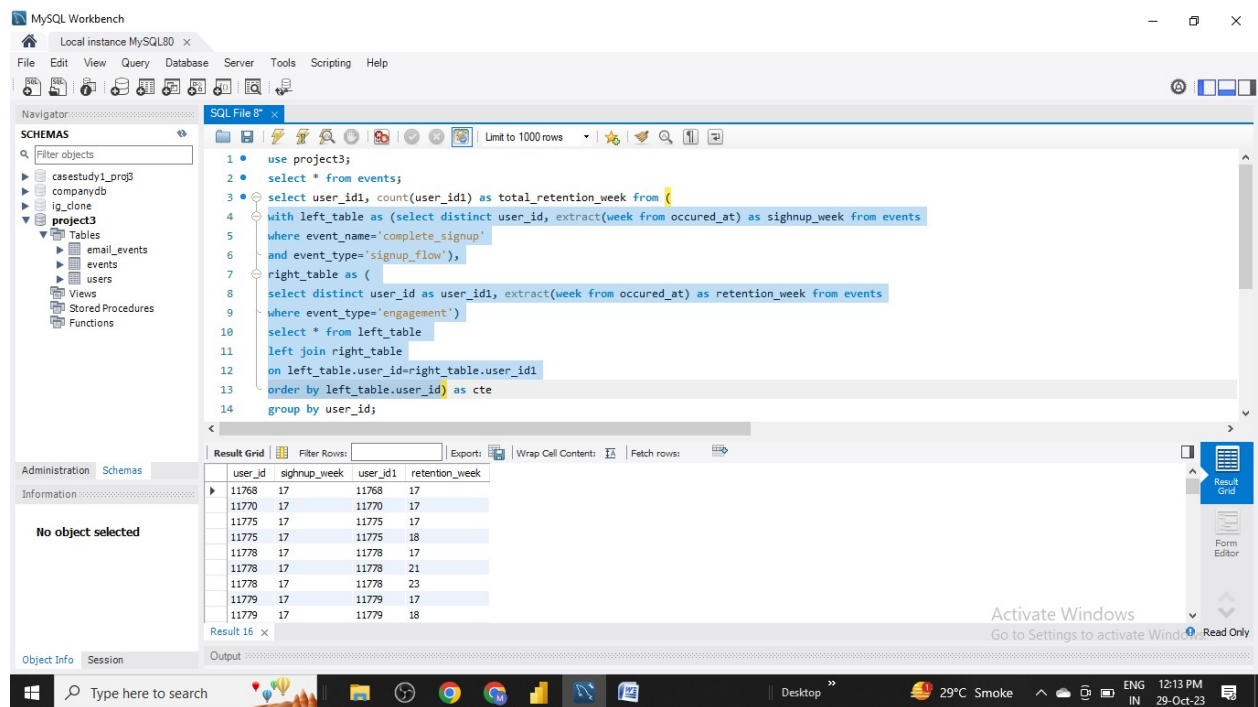
Total number of users since starting till end is **9381**.

### C. Weekly Retention Analysis:

- 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

In order to do this query I followed the following steps:

1. Since, I need to know how many users retained after signing up, I will make 2 separate temporary tables using common table expressions. 1<sup>st</sup> to know the users signed up on weekly basis and 2<sup>nd</sup> to know the users after signing up retained and performed engaging tasks.
2. After making these CTE tables I left joined tables according to user\_id's of both the tables.
3. Lastly to calculate the most retained user, I grouped all the retained user\_id's with itself so that I could get a count of it and ordered in descending order.



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

```
1 use project3;
2 select * from events;
3 select user_id, count(user_id) as total_retention_week from
4 with left_table as (select distinct user_id, extract(week from occurred_at) as signup_week from events
5 where event_name='complete_signup'
6 and event_type='signup_flow'),
7 right_table as (
8 select distinct user_id as user_id, extract(week from occurred_at) as retention_week from events
9 where event_type='engagement')
10 select * from left_table
11 left join right_table
12 on left_table.user_id=right_table.user_id
13 order by left_table.user_id as cte
14 group by user_id;
```

The Results Grid shows the following data:

user_id	signup_week	user_id	retention_week
11768	17	11768	17
11770	17	11770	17
11775	17	11775	17
11775	17	11775	18
11778	17	11778	17
11778	17	11778	21
11778	17	11778	23
11779	17	11779	17
11779	17	11779	18

## INSIGHTS:

After calculating the most retained users on weekly basis, user number **12976** is the most retained user of all after signing up i.e 15 weeks. Also, the total retained users are **3680** from **6142** users. The syntax is shown in the screenshot below:

The screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query is as follows:

```
2 select * from events;
3 select user_id1, count(user_id1) as total_retention_week from (
4 with left_table as (select distinct user_id, extract(week from occurred_at) as signup_week from events
5 where event_name='complete_signup'
6 and event_type='signup_flow'),
7 right_table as (
8 select distinct user_id as user_id1, extract(week from occurred_at) as retention_week from events
9 where event_type='engagement')
10 select * from left_table
11 left join right_table
12 on left_table.user_id=right_table.user_id1
13 order by left_table.user_id) as cte
14 group by user_id
15 order by total_retention_week desc;
```

The result grid shows the following data:

user_id1	total_retention_week
12976	15
11833	14
11893	14
12034	14
12389	14
12617	14
12995	14
13247	14
12164	13

The screenshot shows the MySQL Workbench interface with a query editor and a result grid. The query is as follows:

```
2 select count(distinct user_id) from events;
3 select count(distinct user_id1) as total_retained_users from (
4 with left_table as (select distinct user_id, extract(week from occurred_at) as signup_week from events
5 where event_name='complete_signup'
6 and event_type='signup_flow'),
7 right_table as (
8 select distinct user_id as user_id1, extract(week from occurred_at) as retention_week from events
9 where event_type='engagement')
10 select * from left_table
11 left join right_table
12 on left_table.user_id=right_table.user_id1
13 order by left_table.user_id) as cte
14
```

The result grid shows the following data:

total_retained_users
3680

#### 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.

In order to solve this query I used following steps:

1. First I selected **device, total distinct users using that device, extracted week and year for those users.**
2. In order to get the engagement information I selected users **whose event\_type is 'engagement' using where clause.**
3. Lastly I **grouped** the query with respect to extracted week, extracted year and device used. Also, ordered everything **in descending order w.r.t users.**

The query used is shown in screenshot below:

The screenshot displays the MySQL Workbench interface. The SQL Editor window contains the following query:

```
1 use project3;
2 select * from events;
3 select device, extract(year from occurred_at) as year_no, extract(week from occurred_at) as week_no, count(distinct user_id) as users from events
4 where event_type='engagement';
5 group by week_no, year_no, device
6 order by users desc ;
```

The Results window shows the output of the query in a table format:

device	year_no	week_no	users
macbook pro	2014	30	322
macbook pro	2014	31	321
macbook pro	2014	33	312
macbook pro	2014	32	307
macbook pro	2014	27	302
macbook pro	2014	28	295
macbook pro	2014	29	295
macbook pro	2014	34	292
macbook pro	2014	25	275

#### INSIGHTS:

Week **30 and 31 of year 2014** had most users (332 and 321 resp.) and the device used was **macbook pro**.

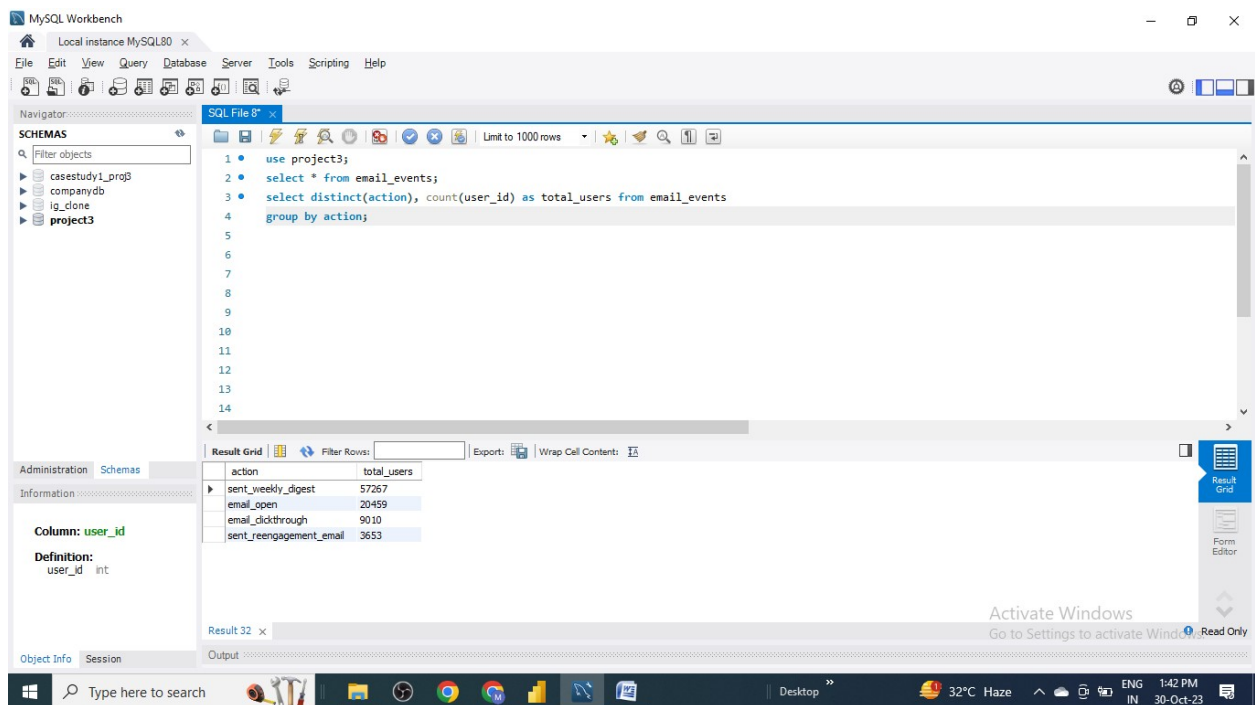
### E. Email Engagement Analysis:

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

To perform this task I followed following steps:

1. I selected the email type and grouped it according to the count of users.
2. After grouping each email type is grouped according to the users activity.

The syntax is showed in following screenshot:



### INSIGHTS:

The total email sent to users (sent\_weekly\_digest + sent\_reengagement\_email) are 60,920 out of which only 20,459 are opened and 9010 were clicked through the opened mail.

### RESULT:

Both the case studies were very insightful and after doing these case studies I am feeling more confident while performing tasks on Sql.

