

# Operation Analytics and Investigating Metric Spike

Advanced SQL

## Project Description:

Operational Analytics is a process involving analysis of a company's end-to-end operations. This analysis helps identify areas for improvement within the company. One of the key aspects of Operational Analytics is investigating metric spikes. It involves understanding and explaining sudden changes in key metrics, like sudden changes in daily user engagement or product sales. We have derived insights from the provided data and answered the questions that have been presented. We make use of advanced SQL skills to analyze the given data and provide insights to help improve the company's operations and understand sudden spikes and lows in key metrics. This will help provide valuable insights to developers and investors that can help the business grow.

## Approach:

We start by creating our database in MySQL.  
Next we run a command to use the said database that has been created.  
The data is stored in a tabular format.  
The problem statement is divided as Case Study 1 and Case Study 2.  
Case Study 1 includes a dataset called job\_data.  
We create and use a database named job\_data.  
Then we make use of Table Data Import Wizard to import the csv file on our MySQL Workbench.  
Case Study 2 has three datasets as users, events and email\_events.  
For Case Study 2, datasets are loaded in the form of tables in a database called metric\_square using load data infile query.  
Various Operational Analysis queries and Metric Spike queries have been used to retrieve information from the database according to our requirement.  
We use various DDL and DML queries like SELECT, FROM, GROUP BY, ORDER BY, etc to retrieve and show information from the database.

## Tech-Stack Used:

Intel Core i5  
Windows 10  
MySQL 8.0 CE Workbench  
Local host MySQ80

## Insights:

### For Case Study 1:

#### Creating a database in MySQL:

**CREATE DATABASE job\_data;**

Command has been used to create a database named job\_data.

**USE job\_data;**

**USE database** command is run to select the database that we work on.

#### Loading Dataset in table job\_data:

In the **schemas** column, under job\_data database, right click on **Table Data Import Wizard**. Next, choose the required file, choose required data types for each column and import.

## Result:

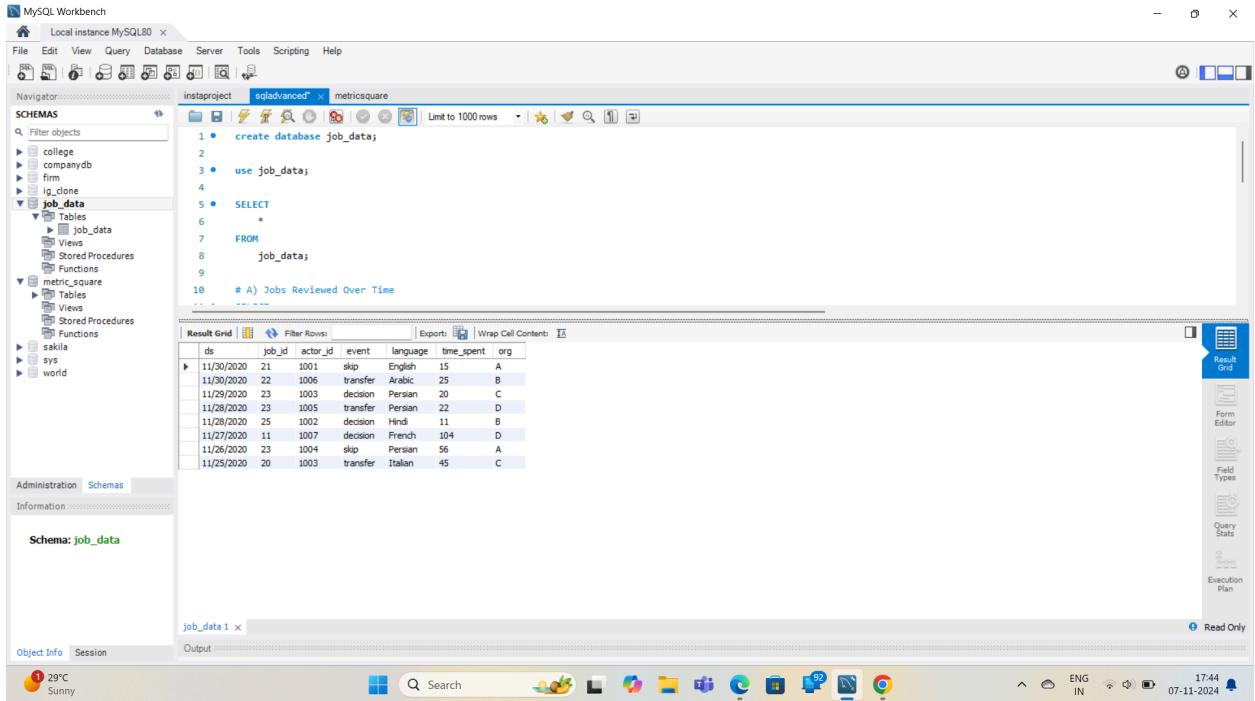
In order to visualize the data in tables we use query

**SELECT \* FROM tablename;**

To visualize data from users table we use command

**select \* from job\_data;**

we get output as



## 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.

### Query:

```

SELECT
    ds,
    COUNT(job_id) AS jobs_per_day,
    SUM(time_spent) AS hours_spent
FROM
    job_data
GROUP BY ds
ORDER BY hours_spent DESC;

```

### Explanation:

We use **SELECT** to show column **ds** as it is, **COUNT(job\_id)** gives the total number of entries in the **job\_id** column and **AS** is used to show this in a column **jobs\_per\_day**.

**SUM(time\_spent)** gives the total number of entries in the **time\_spent** column and **AS** is used to show this in a column **hours\_spent** **FROM** **job\_data** table.

**GROUP BY ds** is used to group the values by **ds** column.

**DESC** is used to give descending order of values in **hours\_spent** column using **ORDER BY**.

## Result:

The screenshot shows the MySQL Workbench interface. In the top navigation bar, the database 'instaproject' is selected. The main pane displays an SQL query and its results. The query is:

```
9
10  # A) Jobs Reviewed Over Time
11 •  SELECT
12    ds,
13    COUNT(job_id) AS jobs_per_day,
14    SUM(time_spent) AS hours_spent
15  FROM
16    job_data
17  GROUP BY ds
18  ORDER BY hours_spent DESC;
```

The results grid shows the following data:

ds	jobs_per_day	hours_spent
11/27/2020	1	104
11/26/2020	1	56
11/25/2020	1	45
11/30/2020	2	40
11/28/2020	2	33
11/29/2020	1	20

## Conclusion:

On 11/27/2020, 1 job took 104 hours. On 11/30/2020, 2 jobs took 40 hours and similar.

## 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.

### Query:

```
WITH CTE AS ( SELECT ds, COUNT(job_id) AS jobs, SUM(time_spent) AS times
  FROM job_data
 GROUP BY ds)
SELECT ds, SUM(jobs) OVER (ORDER BY ds ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) /
  SUM(times) OVER (ORDER BY ds ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)
 AS rolling_avg FROM CTE;
```

### Explanation:

**CTE** is used as Common Table Expression.

The **OVER** clause constructs a window that includes all the records returned by the query.

**ROWS BETWEEN** is used to calculate the running measure / aggregation in Analytical Functions.

**CURRENT ROW:** Range starts or ends at CURRENT ROW.

### Result:

The screenshot shows the MySQL Workbench interface. In the top-left, the Navigator pane displays the database schema with the 'job\_data' schema selected. The central area contains a query editor with the following SQL code:

```
16 job_data
17 GROUP BY ds
18 ORDER BY hours_spent DESC;
19
20 # B) Throughput Analysis
21 WITH CTE AS ( SELECT ds, COUNT(job_id) AS jobs, SUM(time_spent) AS times
22   FROM job_data
23  GROUP BY ds )
24   SELECT ds, SUM(jobs) OVER (ORDER BY ds ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) /
25     SUM(times) OVER (ORDER BY ds ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)
26   AS rolling_avg FROM CTE;
27
28 # C) Language Share Analysis
29 select language, round(((count(language)/s)*100),2) as lang_percent
```

The bottom section shows the 'Result Grid' containing the following data:

ds	rolling_avg
11/25/2020	0.0222
11/26/2020	0.0198
11/27/2020	0.0146
11/28/2020	0.0210
11/29/2020	0.0233
11/30/2020	0.0268

### Conclusion:

Preference of daily metric or 7 day rolling average depends on what is required.

**Daily metric** is preferred on small and stable data, where insights from a particular day are required to make real time decisions.

**7 day rolling average** is more useful for large and noisy datasets, where we require a smooth outcome to highlight longer and broader trends.

## 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.

### Query:

```

SELECT
    language,
    ROUND(((COUNT(language) / 8) * 100), 2) AS lang_percent
FROM
    job_data
GROUP BY language
ORDER BY lang_percent DESC;

```

### Explanation:

The **ROUND** function rounds a number to a specified number of decimal places, which we have defined as **2**.

The expression **COUNT(language) / 8 \* 100** applies the formula of percentage on the **language** column.

### Result:

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Toolbar:** Standard MySQL icons for connection, schema, table, view, stored procedure, function, and system objects.
- Navigator:** Shows the database structure under 'instaproject' schema, including 'job\_data' table.
- Query Editor:** Contains the SQL code for the query, starting with CTE definitions and ending with the final SELECT statement.
- Result Grid:** Displays the query results in a tabular format.

language	lang_percent
Persian	37.50
English	12.50
Arabic	12.50
Hindi	12.50
French	12.50
Italian	12.50

- Action Output:** Shows the execution log with one entry: "5 15:57:57 SELECT \* FROM job\_data LIMIT 0,1000".
- System Status:** Shows system information like CPU temperature (30°C), battery status (Haze), and system date/time (08-11-2024, 15:28).

### Conclusion:

**Persian** language has **37.50%** weightage, while the rest of the languages **Arabic, Hindi, French, Italian** each have a similar weightage of **12.50%**.

## 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.

### Query:

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

### Explanation:

**Having** clause is used to filter rows that satisfy specific conditions along with aggregate functions.

### Query:

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Navigator:** Schemas (college, companydb, firm, lgjone, job\_data, metric\_square, sakila, sys, world).
- Query Editor:** A tab titled "sqladvanced" is active, containing the SQL query provided in the text block above.
- Result Grid:** Shows the output of the query:

actor_id	total_actors
1003	2
- Output Tab:** Shows the execution log:

#	Time	Action	Message	Duration / Fetch
1	6 15:28:33	SELECT language, ROUND((COUNT(language) / 8) * 100), 2) AS lang_percent FROM job_data GROUP BY language ORDER BY lang_percent DESC;	6 row(s) returned	0.000 sec / 0.000 sec
2	6 15:28:33	SELECT actor_id, COUNT(actor_id) AS total_actors FROM job_data GROUP BY actor_id HAVING total_actors > 1;	1 row(s) returned	0.000 sec / 0.000 sec
- System Bar:** Shows system icons and status information (Finance headline, US stocks climb..., 15:40, 08-11-2024).

### Conclusion:

Actor ID 1003 has two rows.

## For Case Study 2:

### Creating a database in MySQL:

```
CREATE DATABASE metric_square;
```

Command has been used to create a database named metric\_square.

```
USE metric_square;
```

**USE database** command is run to select the database that we work on.

### Creating a table in MySQL:

Creating table USERS in our database.

#### **Query:**

```
CREATE TABLE users (
    user_id INT,
    created_at VARCHAR(50),
    company_id INT,
    language VARCHAR(50),
    activated_at VARCHAR(100),
    state VARCHAR(50)
);
```

We use the CREATE command to create tables.

We define names and datatypes of the columns in the table.

### Loading Data In The Table:

```
show variables like 'secure_file_priv';
```

Secure\_file\_priv gives the location where dataset.csv needs to be saved.

```
load data infile "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users.csv"
into table users
fields terminated by ','
enclosed by ""
lines terminated by '\n'
```

```
ignore 1 rows;
```

Load data infile gives location of the file that needs to be loaded on MySQL Workbench.  
Ignore 1 rows to ignore column headings.

## Changing datatype in column created\_at from string to datetime

```
alter table users add column temp_created_at datetime;  
UPDATE users  
SET  
    temp_created_at = STR_TO_DATE(created_at, '%d-%m-%Y %H:%i');  
alter table users drop column created_at;  
alter table users change column temp_created_at created_at datetime;
```

ALTER and DROP commands are used.

We have changed the data type from the created\_at column from STR\_TO\_DATE.

Similarly, we'll change, datatype of activated\_at table.

## Visualizing The Table Data:

In order to visualize the data in tables we use query

**SELECT \* FROM tablename;**

To visualize data from users table we use command

**select \* from users;**

We get output as

MySQL Workbench

Local instance MySQL80

Schemas: instaproject, sqldvanced, metricsquare\*

```

29      *
30  FROM
31  users;
32
33  # Changing datatype in column created_at from string to datetime
34 • alter table users add column temp_created_at datetime;
35 • UPDATE users
36 SET
37     temp_created_at = STR_TO_DATE(created_at, '%d-%m-%Y %H:%i');
38 • alter table users drop column created_at;
39 • alter table users change column temp_created_at created_at datetime;
40
41  # Changing datatype in column activated_at from string to datetime
42 • alter table users add column temp_activated_at datetime;

```

Result Grid:

user_id	company_id	language	state	created_at	activated_at	temp_created_at
0	5737	english	active	2013-01-01 20:59:00	2013-01-01 21:01:00	2013-01-01 20:59:00
3	2800	german	active	2013-01-01 18:40:00	2013-01-01 18:42:00	2013-01-01 18:40:00
4	5110	indian	active	2013-01-01 14:37:00	2013-01-01 14:39:00	2013-01-01 14:37:00
6	11699	english	active	2013-01-01 18:37:00	2013-01-01 18:38:00	2013-01-01 18:37:00
7	4765	french	active	2013-01-01 16:19:00	2013-01-01 16:20:00	2013-01-01 16:19:00
8	2698	french	active	2013-01-01 04:38:00	2013-01-01 04:40:00	2013-01-01 04:38:00
11	3745	english	active	2013-01-01 08:07:00	2013-01-01 08:09:00	2013-01-01 08:07:00
13	4025	english	active	2013-01-02 12:27:00	2013-01-02 12:29:00	2013-01-02 12:27:00
15	4259	english	active	2013-01-02 15:39:00	2013-01-02 15:41:00	2013-01-02 15:39:00
17	5025	japanese	active	2013-01-02 10:56:00	2013-01-02 10:57:00	2013-01-02 10:56:00
19	274	english	active	2013-01-03 00:00:00	2013-01-03 00:00:00	2013-01-03 00:00:00

Output:

#	Time	Action	Message	Duration / Fetch
2	18:01:19	use metric_square	0 rows affected	0.000 sec
3	18:01:20	SELECT * FROM `users` LIMIT 0, 1000	1000 rows returned	0.000 sec / 0.000 sec

Object Info Session

28°C Sunny

ENG IN 18:12 08-11-2024

Similarly, we create tables EVENTS and EMAIL\_EVENTS and change data types from string to datetime.

## Events Table

MySQL Workbench

Local instance MySQL80

Schemas: instaproject, sqldvanced, metricsquare\*

```

62  into table events
63  fields terminated by ','
64  enclosed by "'"
65  lines terminated by '\n'
66  ignore 1 rows;
67
68  # Visualizing data in EVENTS table
69 • SELECT
70  *
71 FROM
72  events;
73
74  # Changing datatype in column occurred_at from string to datetime
75 • alter table events add column temp_occurred_at datetime;

```

Result Grid:

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

Output:

#	Time	Action	Message	Duration / Fetch
4	18:20:33	SELECT * FROM `events` LIMIT 0, 1000	1000 row(s) returned	0.000 sec / 0.000 sec

Object Info Session

Finance headline US stocks climb...

ENG IN 18:23 08-11-2024

## Email\_Events Table

The screenshot shows the MySQL Workbench interface with a query editor and results grid. The query editor contains SQL code for creating an events table and selecting data from it. The results grid displays 10 rows of user engagement data. The operating system taskbar at the bottom shows various application icons.

```
62 into table events
63 fields terminated by ','
64 enclosed by """
65 lines terminated by '\n'
66 ignore 1 rows;
67
68 # Visualizing data in EVENTS table
69 • SELECT
70 *
71 FROM
72 events;
73
74 # Changing datatype in column occurred_at from string to datetime
75 • alter table events add column temp_occurred_at datetime;
```

user_id	event_type	event_name	location	device	user_type	temp_created_at	occurred_at
10522	engagement	login	Japan	del insprn notebook	3	2014-05-02 11:02:00	2014-05-02 11:02:00
10522	engagement	home_page	Japan	del insprn notebook	3	2014-05-02 11:02:00	2014-05-02 11:02:00
10522	engagement	like_message	Japan	del insprn notebook	3	2014-05-02 11:02:00	2014-05-02 11:02:00
10522	engagement	view_inbox	Japan	del insprn notebook	3	2014-05-02 11:02:00	2014-05-02 11:02:00
10522	engagement	search_run	Japan	del insprn notebook	3	2014-05-02 11:02:00	2014-05-02 11:02:00
10522	engagement	search_run	Japan	del insprn notebook	3	2014-05-02 11:02:00	2014-05-02 11:02:00
10612	engagement	login	Netherlands	iphone 5	1	2014-05-01 10:59:00	2014-05-01 10:59:00
10612	engagement	like_message	Netherlands	iphone 5	1	2014-05-01 10:00:00	2014-05-01 10:00:00
10612	engagement	send_message	Netherlands	iphone 5	1	2014-05-01 10:00:00	2014-05-01 10:00:00
10612	engagement	home_page	Netherlands	iphone 5	1	2014-05-01 10:01:00	2014-05-01 10:01:00

### 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.

**Query:**

```
SELECT
```

```
EXTRACT(WEEK FROM occurred_at) AS no_of_weeks,
```

```
COUNT(DISTINCT user_id) AS user_engagement
```

```
FROM
```

```
events
```

```
WHERE
```

```
event_type = 'Engagement'
```

```
GROUP BY no_of_weeks
```

```
ORDER BY no_of_weeks;
```

## **Explanation:**

**EXTRACT** function extracts a part from a given date.

The **WEEK** function returns the week number for a given date

**WHERE** clause is used to filter records. It is used to extract only those records that fulfill a specified condition.

**DISTINCT** statement is used to return only **distinct** (different) values.

## Result:

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Schemas:** Navigator pane shows the current schema is "instaproject". Other schemas listed include college, companydb, firm, igclone, job\_data, metricsquare, sakila, sys, and world.
- Query Editor:** The query being run is:

```
109 • alter table email_events drop column occurred_at;
110 • alter table email_events change column temp_occurred_at occurred_at datetime;
111
112 # A) Weekly User Engagement
113 • SELECT
114     EXTRACT(WEEK FROM occurred_at) AS no_of_weeks,
115     COUNT(DISTINCT user_id) AS user_engagement
116
117     FROM
118         events
119     WHERE
120         event_type = 'Engagement'
121     GROUP BY no_of_weeks
122     ORDER BY no_of_weeks;
```
- Result Grid:** The results of the query are displayed in a grid format:

no_of_weeks	user_engagement
17	663
18	1068
19	1113
20	1154
21	1121
22	1186
23	1232
24	1275
25	1264
26	1302
27	1999
- Object Info:** Shows the last selected object was a SELECT statement from the email\_events table.
- Session:** Shows the session status as active.
- System:** Shows the system status including temperature (27°C), battery level (Mostly clear), and system icons.
- Bottom Bar:** Includes the Start button, Search bar, Task View, and various application icons.

## Conclusion:

Number of users that are engaged for how many weeks is mentioned in the result grid.

#### 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.

## Query:

SELECT year, month, active\_users;

```
    sum(active_users) OVER(ROWS BETWEEN UNBOUNDED PRECEDING AND  
CURRENT ROW) AS user_growth,
```

```

CASE
    WHEN LAG(active_users) OVER (ORDER BY year, month) IS NULL THEN 0
    ELSE ROUND(((active_users - LAG(active_users) OVER (ORDER BY year, month)) /
LAG(active_users) OVER (ORDER BY year, month)) * 100, 2)
    END AS growth_percentage
FROM (
    SELECT YEAR(activated_at) AS year,
           EXTRACT(MONTH FROM activated_at) AS month,
           COUNT(DISTINCT user_id) AS active_users
      FROM users
     GROUP BY year,month
) as Active_Users_per_Month
ORDER BY year;

```

### Explanation:

**CASE** statement goes through conditions and returns a value when the first condition is met (like an IF-THEN-ELSE statement).

**LAG** function is used to get value from the row that precedes the current row.

### Result:

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The schema tree on the left shows several databases and tables, including `college`, `companydb`, `firm`, `ig\_done`, `job\_data`, and `metric\_square`.
- Query Editor:** The main area contains the SQL query provided above. The code uses a CASE statement to handle the first row where the LAG value is null by returning 0. It then calculates the growth percentage by dividing the difference between the current row's active\_users and the previous row's active\_users by the previous row's active\_users, multiplied by 100 and rounded to two decimal places.
- Result Grid:** Below the query editor is a result grid showing the output for each month from January to October 2013. The columns are `year`, `month`, `active\_users`, `user\_growth`, and `growth\_percentage`. The data is as follows:

year	month	active_users	user_growth	growth_percentage
2013	1	160	160	0.00
2013	2	160	320	0.00
2013	3	150	470	-6.25
2013	4	181	651	20.67
2013	5	214	865	18.23
2013	6	213	1078	-0.47
2013	7	284	1362	33.33
2013	8	316	1678	11.27
2013	9	330	2008	4.43
2013	10	390	2398	18.18
2013	11	390	???	???

- Output:** The bottom pane shows the execution log with one successful query (SELECT) and one warning message about unhandled warnings.

### Conclusion:

The result grid shows columns for year, month, active\_users, user\_growth and growth\_percentage.

## 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.

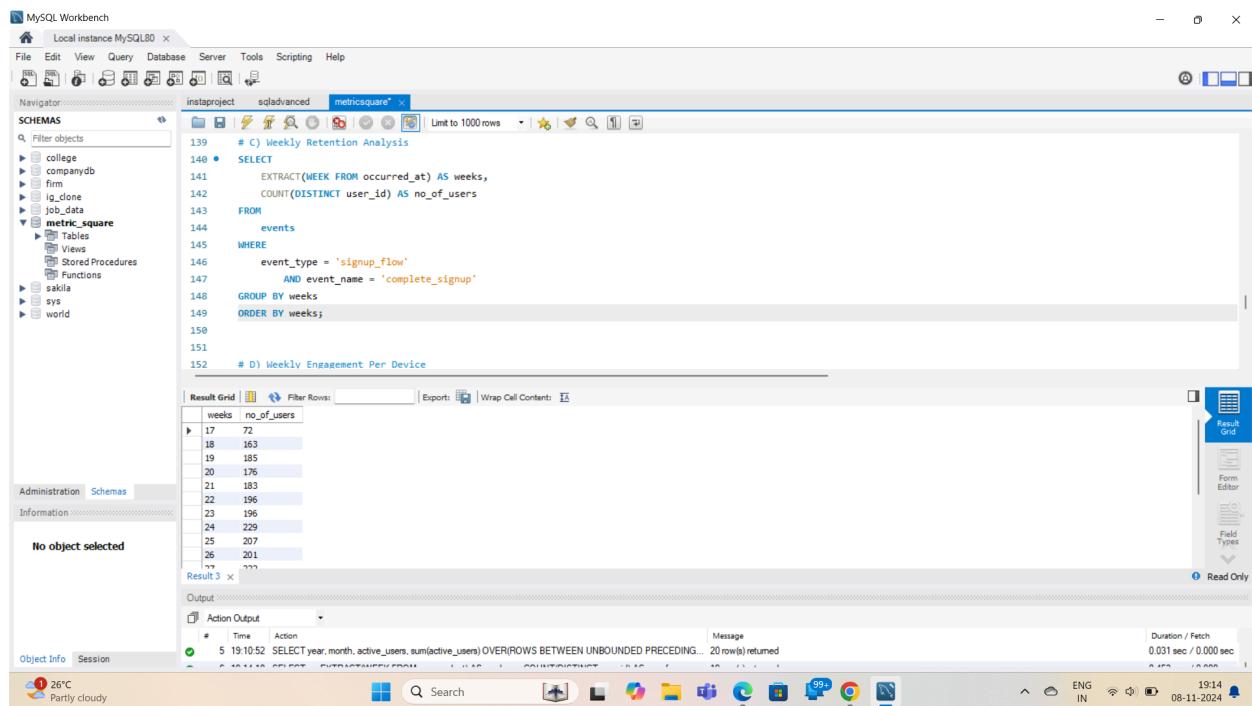
### Query:

```
SELECT
    EXTRACT(WEEK FROM occurred_at) AS weeks,
    COUNT(DISTINCT user_id) AS no_of_users
FROM
    events
WHERE
    event_type = 'signup_flow'
        AND event_name = 'complete_signup'
GROUP BY weeks
ORDER BY weeks;
```

### Explanation:

event\_type = 'signup\_flow' AND event\_name = 'complete\_signup' are used as conditions in the query.

### Result:



The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help
- Schemas:** instaproject, sqldvanced, metric\_square\*
- Query Editor:** Contains the SQL query for C) Weekly Retention Analysis.
- Result Grid:** Displays the results of the query, showing weeks and no\_of\_users.

weeks	no_of_users
17	72
18	163
19	185
20	176
21	183
22	196
23	196
24	229
25	207
26	201
27	222

- Output:** Shows the action output of the query execution.
- System Bar:** Includes icons for weather (26°C Partly cloudy), search, file explorer, taskbar, and system status (ENG IN, 19:14, 08-11-2024).

**Conclusion:**

Number of users for weeks is given in the result grid.

## 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.

**Query:**

```
SELECT  
    EXTRACT(YEAR FROM occurred_at) AS year,  
    EXTRACT(WEEK FROM occurred_at) AS week,  
    device,  
    COUNT(DISTINCT user_id) AS num_users  
FROM  
    events  
GROUP BY  
    year, week, device  
ORDER BY  
    year, week, device;
```

**Result:**

The screenshot shows the MySQL Workbench interface. The SQL editor tab is active, displaying a query titled '# D) Weekly Engagement Per Device'. The query uses the `EXTRACT` function to group data by year and week, then counts distinct user IDs for each device. The results are displayed in a grid:

year	week	device	num_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	lenovo ideapad	21

The result grid has 8 rows. The status bar at the bottom shows the duration of 0.469 sec / 0.000 sec.

## Conclusion:

Number of users engaging for number of weeks per year is given along with the names of devices that were used.

## 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.

### Query:

```

SELECT
    user_id,
    Emails_sent,
    Emails_opened,
    Emails_clicked,
    ROUND(SUM(Emails_opened) / SUM(Emails_sent), 2) * 100 AS Opening_rate,
    ROUND(SUM(Emails_clicked) / SUM(Emails_opened),
        2) * 100 AS Engagement_rate
FROM
    (SELECT
        user_id,
        SUM(CASE

```

```

WHEN `action` = 'sent_weekly_digest' THEN 1
ELSE 0
END) AS Emails_sent,
SUM(CASE
WHEN `action` = 'email_open' THEN 1
ELSE 0
END) AS Emails_opened,
SUM(CASE
WHEN `action` = 'email_clickthrough' THEN 1
ELSE 0
END) AS Emails_clicked
FROM
email_events
GROUP BY user_id) AS user_email_engagement
GROUP BY user_id;

```

**Explanation:**

**USER ENGAGEMENT**

Clickthrough rate, or CTR, is the number of clicks per email delivered. If your emails include links this metric can help you gauge how effective that is.

Formula to calculate rate

(email clicks / emails delivered) x 100

WHEN -- ELSE is similar to the if else statement in coding languages.

**Result:**

MySQL Workbench

Local instance MySQL80 X

File Edit View Query Database Server Tools Scripting Help

Navigator: instaproject sqldvanced metricsquare\*

SCHEMAS

- college
- companydb
- firm
- ig\_clone
- job\_data
- metricsquare\*
  - Tables
  - Views
  - Stored Procedures
  - Functions
- sakila
- sys
- world

181      END) AS Emails\_sent,

182      SUM(CASE

183        WHEN `action` = 'email\_open' THEN 1

184        ELSE 0

185      END) AS Emails\_opened,

186      SUM(CASE

187        WHEN `action` = 'email\_clickthrough' THEN 1

188        ELSE 0

189      END) AS Emails\_clicked

190     FROM

191        email\_events

192     GROUP BY user\_id) AS user\_email\_engagement

193     GROUP BY user\_id|

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch Rows: |

user_id	Emails_sent	Emails_opened	Emails_clicked	Opening_rate	Engagement_rate
0	17	5	0	29.00	0.00
4	17	5	4	29.00	80.00
8	17	3	1	18.00	33.00
11	17	5	2	29.00	40.00
17	17	4	1	24.00	25.00
19	17	5	1	29.00	20.00
20	17	8	3	47.00	38.00
22	17	7	3	41.00	43.00
30	18	6	1	33.00	17.00
49	17	5	1	29.00	20.00
50	17	6	?	20.00	60.00

Administration Schemas Information No object selected Result 5 x

Output Action Output # Time Action Message Duration / Fetch

Object Info Session 11 19:21:18 SELECT EXTRACT(YEAR FROM occurred\_at) AS year, EXTRACT(WEEK FROM occurred\_at) AS wee... Error Code: 1054. Unknown column 'occurred\_at' in field list 0.000 sec

25C Partly cloudy Search ENG IN 19:27 08-11-2024

## Conclusion:

Number of emails that have been sent, opened, clicked, along with their opening rate and engagement rate are given.