

# Trainity Assignment-3

Muthiah Sivavelan  
Ph:8525021258

## Project Description:

Operational Analytics is a crucial process that involves analyzing a company's end-to-end operations. This analysis helps identify areas for improvement within the company. As a Data Analyst, you'll work closely with various teams, such as operations, support, and marketing, helping them derive valuable insights from the data they collect.

One of the key aspects of Operational Analytics is investigating metric spikes. This involves understanding and explaining sudden changes in key metrics, such as a dip in daily user engagement or a drop in sales. As a Data Analyst, you'll need to answer these questions daily, making it crucial to understand how to investigate these metric spikes.

In this project I used 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. In this project, I used SQL and MySQL Workbench as my tool to analyze the given data and answer questions from different scenarios posed by the operations team.

The approach I took for providing the insights regarding various queries by marketing team and development team are as follows:

- I created a database and created various tables in it.
- I filled every table with their given data
- I carefully analyzed the attributes in each table and their relationships with other table (which helped in joining the tables).
- I meticulously drafted sql queries based on the questions given. I created views in some queries to make the queries easy to read and simple.

## Case Study 1: Job Data Analysis

### Tasks:

## A) Jobs Reviewed Over Time:

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

### Approach:

In the SQL query I select the day, total number of distinct job\_id from the job\_data table(named as c1) and I filtered the resultant rows with dates only on November 2020 and time\_spent on the job less than or equal to 3600 (1 hour).

### Query:

```
/* Write an SQL query to calculate the number of jobs
reviewed per hour for each day in November 2020.
Note: the time_spent is given in seconds therefore
in the following query time_spent<=3600
would check if the time_spent is less than an hour */

select
    ds AS day,
    count(distinct job_id) as total_jobs_handled
from c1
where
    ds >= '2020-11-01' and ds < '2020-12-01'
    and time_spent <= 3600
group by ds
order by ds;
```

### Output:

| # | day        | total_jobs_handle |
|---|------------|-------------------|
| 1 | 2020-11-25 | 1                 |
| 2 | 2020-11-26 | 1                 |
| 3 | 2020-11-27 | 1                 |
| 4 | 2020-11-28 | 2                 |
| 5 | 2020-11-29 | 1                 |
| 6 | 2020-11-30 | 2                 |

## B) Throughput Analysis:

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

### Approach:

Here, I assumed that throughput is the number of events per second. Therefore to calculate the 7-day rolling average of throughput, I selected the total number of events divided by sum(time\_spent)[named as events\_per\_sec] for each date as the subquery and then from the subquery I selected the date, average of events\_per\_sec and used the window function to specify the number of rows as preceding 6 rows along with the current row. If I need to check only long-term trends then I would prefer 7-day rolling average throughput while if I need real-time monitoring such as day to day changes then I would choose daily metric. Personally, I think 7 day rolling average throughput would be better in this scenario where we can find the long term trends.

### Query:

```
/* 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.*/

select ds,
avg(events_per_sec) over(order by ds rows between 6 preceding and current row) as rolling_avg_throughput
from
(select ds, count(*)/sum(time_spent) as events_per_sec from c1 group by ds) subquery;
```

### Output:

| # | ds         | rolling_avg_throughput |
|---|------------|------------------------|
| 1 | 2020-11-25 | 0.02220000             |
| 2 | 2020-11-26 | 0.02005000             |
| 3 | 2020-11-27 | 0.01656667             |
| 4 | 2020-11-28 | 0.02757500             |
| 5 | 2020-11-29 | 0.03206000             |
| 6 | 2020-11-30 | 0.03505000             |

### C) Language Share Analysis:

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

### Approach:

In this SQL query, I selected the language and the total number of rows containing the language as a subquery and then I selected the total number of rows as another subquery and then I selected the language and the percentage of each language( $\text{count} \times 100 / \text{sum}$ ) from the subqueries.

### Query:

```
/* Write an SQL query to calculate the percentage share of each language over the last 30 days. */  
  
with  
  t2 as (select language, count(*) as count from c1 group by language),  
  t1 as (select sum(count) as total from t2)  
select language, (count*100)/total as pct from t1, t2;
```

### Output:

| # | language | pct     |
|---|----------|---------|
| 1 | English  | 12.5000 |
| 2 | Arabic   | 12.5000 |
| 3 | Persian  | 37.5000 |
| 4 | Hindi    | 12.5000 |
| 5 | French   | 12.5000 |
| 6 | Italian  | 12.5000 |

### D) Duplicate Rows Detection:

**Objective:** Identify duplicate rows in the data. Write an SQL query to display duplicate rows from the job\_data table.

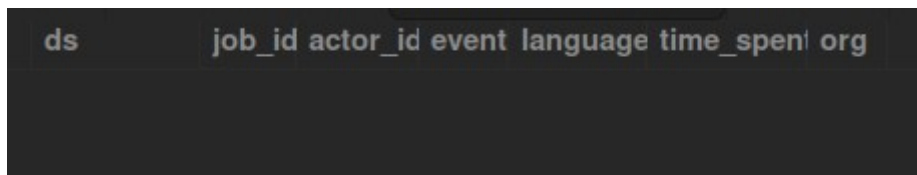
### Approach:

I selected all the rows and grouped by all the columns that have count greater than one, this essentially checks for the duplicate row.

### Query:

```
/* Write an SQL query to display duplicate rows from the job_data table. */  
  
select ds, job_id, actor_id, event, language, time_spent, org  
from c1  
group by ds, job_id, actor_id, event, language, time_spent, org  
having count(*) > 1;
```

### Output:



| ds | job_id | actor_id | event | language | time_spent | org |
|----|--------|----------|-------|----------|------------|-----|
|----|--------|----------|-------|----------|------------|-----|

There is no duplicate rows in the table.

## Case Study 2: Investigating Metric Spike

### Tasks:

#### A) Weekly User Engagement:

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

### Approach:

I selected the user\_id, week(occurred\_at), total number of rows in each group as total\_engagements where I first filtered the rows according to the “engagement” event\_type and then grouped the resultant according to the user\_id and week. This would give me the user engagement for each user in each week.

### Query:

```

/* Write a SQL query to calculate the weekly user engagement. */

select
    user_id,
    concat(year(occurred_at), '/', week(occurred_at)) as `Week`,
    count(*) as total_engagements
from events
where event_type='engagement'
group by user_id, `Week`;

```

## Output:

| #  | user_id | Week    | total_engagement |
|----|---------|---------|------------------|
| 1  | 4       | 2014/19 | 4                |
| 2  | 4       | 2014/20 | 8                |
| 3  | 4       | 2014/21 | 29               |
| 4  | 4       | 2014/22 | 4                |
| 5  | 4       | 2014/23 | 15               |
| 6  | 4       | 2014/24 | 8                |
| 7  | 4       | 2014/25 | 7                |
| 8  | 4       | 2014/26 | 10               |
| 9  | 4       | 2014/27 | 8                |
| 10 | 8       | 2014/17 | 2                |
| 11 | 8       | 2014/18 | 15               |
| 12 | 8       | 2014/19 | 3                |

Other rows where not shown. Total 1000 rows where returned.

## B) User Growth Analysis:

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

## Approach:

In this SQL query, I selected the month, number of users in that month, difference of users in consecutive months (used window function

to achieve this) as growth which I grouped by month and ordered by month.

### Query:

```
/* Write an SQL query to calculate the user growth for the product. */
```

```
select
    date_format(created_at, "%b") as month,
    count(distinct user_id) as users,
    coalesce(count(distinct user_id) - lag(count(distinct user_id),1)
             over(order by date_format(created_at, "%b")),
             count(distinct user_id)) as growth
from users
group by month
order by month;
```

### Output:

| #  | month | users | growth |
|----|-------|-------|--------|
| 1  | Apr   | 907   | 907    |
| 2  | Aug   | 1347  | 440    |
| 3  | Dec   | 486   | -861   |
| 4  | Feb   | 685   | 199    |
| 5  | Jan   | 712   | 27     |
| 6  | Jul   | 1281  | 569    |
| 7  | Jun   | 1086  | -195   |
| 8  | Mar   | 765   | -321   |
| 9  | May   | 993   | 228    |
| 10 | Nov   | 399   | -594   |
| 11 | Oct   | 390   | -9     |
| 12 | Sep   | 330   | -60    |

### C) Weekly Retention Analysis:

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

### Approach:

I considered the activated\_at date to be the cohort start date. To calculate the retention of users on a weekly basis, I first selected the cohort\_start\_date, week of occurred\_at, total number of users, retained\_users (which is total number of users in current row – previous

row, i.e., the last week), retention rate (which is  $\text{retained\_users} * 100 / \text{total number of users in current week}$ ) from users left join events table (because left join will give all the rows for the left table, i.e., for all the users) based on the user\_id and I filtered the rows where the engagement\_type = engagement and that the occurred\_at date is after the created\_at date. Finally I grouped the result based on cohort\_start\_date and week of occurred\_at and then ordered by the same.

## Query:

```
/* Write an SQL query to calculate the weekly retention of users based on their sign-up cohort. */
```

```
select
  date(activated_at) as cohort_start_date,
  week(occurred_at, 1) as `week`,
  count(distinct users.user_id) AS total_users,
  count(distinct users.user_id) - lag(count(distinct users.user_id), 1)
    over(partition by date(activated_at) order by week(occurred_at, 1)) as retained_users,
  coalesce((count(distinct users.user_id) - lag(count(distinct users.user_id), 1)
    over(partition by date(activated_at) order by week(occurred_at, 1))) /
    count(distinct users.user_id) * 100, 0) as retention_rate
from users
left join events on users.user_id = events.user_id
where event_type = 'engagement'
  and occurred_at >= created_at
group by cohort_start_date, week(occurred_at, 1)
order by cohort_start_date, week;
```

## Output:



| #  | cohort_start_date | week | total_users | retained_users | retention_rate |
|----|-------------------|------|-------------|----------------|----------------|
| 1  | 2013-01-01        | 18   | 1           | NULL           | 0.0000         |
| 2  | 2013-01-01        | 19   | 1           | 0              | 0.0000         |
| 3  | 2013-01-01        | 20   | 2           | 1              | 50.0000        |
| 4  | 2013-01-01        | 21   | 2           | 0              | 0.0000         |
| 5  | 2013-01-01        | 22   | 1           | -1             | -100.0000      |
| 6  | 2013-01-01        | 23   | 1           | 0              | 0.0000         |
| 7  | 2013-01-01        | 24   | 1           | 0              | 0.0000         |
| 8  | 2013-01-01        | 25   | 2           | 1              | 50.0000        |
| 9  | 2013-01-01        | 26   | 2           | 0              | 0.0000         |
| 10 | 2013-01-01        | 27   | 1           | -1             | -100.0000      |
| 11 | 2013-01-01        | 28   | 1           | 0              | 0.0000         |
| 12 | 2013-01-01        | 31   | 2           | 1              | 50.0000        |
| 13 | 2013-01-01        | 32   | 1           | -1             | -100.0000      |
| 14 | 2013-01-02        | 18   | 1           | NULL           | 0.0000         |
| 15 | 2013-01-02        | 19   | 2           | 1              | 50.0000        |
| 16 | 2013-01-02        | 20   | 1           | -1             | -100.0000      |
| 17 | 2013-01-02        | 21   | 1           | 0              | 0.0000         |
| 18 | 2013-01-02        | 22   | 1           | 0              | 0.0000         |
| 19 | 2013-01-02        | 23   | 2           | 1              | 50.0000        |

6494 rows were returned in total. Other rows are not shown above.

#### D) Weekly Engagement Per Device:

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

#### Approach:

In this SQL query, I select the device, week of occurred\_at and the count of rows as total\_engagements where I grouped by the device and week. Before grouping, I filtered the rows that contain event\_type = "engagement".

#### Query:

```

/* Write an SQL query to calculate the weekly engagement per device. */

select device, count(*) as total_engagements,
       concat(year(occurred_at), '/', week(occurred_at)) as `Week`
from events
where event_type='engagement'
group by device, `Week`;

```

## Output:

| #  | device              | total_engagement | Week    |
|----|---------------------|------------------|---------|
| 1  | acer aspire desktop | 67               | 2014/17 |
| 2  | acer aspire desktop | 295              | 2014/18 |
| 3  | acer aspire desktop | 242              | 2014/19 |
| 4  | acer aspire desktop | 226              | 2014/20 |
| 5  | acer aspire desktop | 328              | 2014/21 |
| 6  | acer aspire desktop | 255              | 2014/22 |
| 7  | acer aspire desktop | 240              | 2014/23 |
| 8  | acer aspire desktop | 289              | 2014/24 |
| 9  | acer aspire desktop | 263              | 2014/25 |
| 10 | acer aspire desktop | 313              | 2014/26 |
| 11 | acer aspire desktop | 296              | 2014/27 |
| 12 | acer aspire desktop | 303              | 2014/28 |
| 13 | acer aspire desktop | 212              | 2014/29 |
| 14 | acer aspire desktop | 403              | 2014/30 |

A total of 494 rows were being returned.

## E) Email Engagement Analysis:

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

## Approach:

I have written two different queries for this problem statement. The first query explains the distribution of each action for each date while the second query explains the action, number of people who chose that action and how many times the action has occurred (considering same user can choose the same action again) per week.

In the first query, I selected the date, number of rows for each action, open rate as  $100 \times \text{number of rows where the action is email\_open} / \text{number}$

of rows where the action is sent\_weekly\_digest. Similarly I have also selected clickthrough\_rate as the percentage of action=email\_clickthrough w.r.t the email\_open case.

In the second query, I selected the action, count of unique users, count(\*) as total\_actions, week of occurred\_at grouped by action and week.

## Query:

i)

```
/* Your Task: Write an SQL query to calculate the email engagement metrics. */
```

```
select
  date(occurred_at) as date,
  count(distinct case when action = 'sent_weekly_digest' then user_id end) as weekly_digests_sent,
  count(distinct case when action = 'email_open' then user_id end) as emails_opened,
  count(distinct case when action = 'email_clickthrough' then user_id end) as emails_clicked,
  count(distinct case when action = 'sent_reengagement_email' then user_id end) as reengagement_emails_sent,
  100.0 * count(distinct case when action = 'email_open' then user_id end)
    / count(distinct case when action = 'sent_weekly_digest' then user_id end) as open_rate,
  100.0 * count(distinct case when action = 'email_clickthrough' then user_id end)
    / count(distinct case when action = 'email_open' then user_id end) as clickthrough_rate
from email_events
group by DATE(email_events.occurred_at)
order by date;
```

ii)

```
select action, count(distinct user_id) as unique_users,
count(*) as total_actions, concat(year(occurred_at),'/',week(occurred_at)) as `Week`
from email_events
group by action, `Week`;
```

---

## Output:

i)

| #  | date       | weekly_digests_sent | emails_opened | emails_clicked | reengagement_emails_sent | open_rate | clickthrough_rate |
|----|------------|---------------------|---------------|----------------|--------------------------|-----------|-------------------|
| 1  | 2014-05-01 | 467                 | 145           | 61             | 7                        | 31.04925  | 42.06897          |
| 2  | 2014-05-02 | 441                 | 142           | 82             | 39                       | 32.19955  | 57.74648          |
| 3  | 2014-05-03 | 0                   | 23            | 23             | 27                       | NULL      | 100.00000         |
| 4  | 2014-05-04 | 0                   | 22            | 21             | 25                       | NULL      | 95.45455          |
| 5  | 2014-05-05 | 760                 | 255           | 115            | 34                       | 33.55263  | 45.09804          |
| 6  | 2014-05-06 | 477                 | 168           | 82             | 30                       | 35.22013  | 48.80952          |
| 7  | 2014-05-07 | 435                 | 141           | 63             | 8                        | 32.41379  | 44.68085          |
| 8  | 2014-05-08 | 479                 | 156           | 65             | 9                        | 32.56785  | 41.66667          |
| 9  | 2014-05-09 | 451                 | 148           | 64             | 24                       | 32.81596  | 43.24324          |
| 10 | 2014-05-10 | 0                   | 22            | 20             | 27                       | NULL      | 90.90909          |
| 11 | 2014-05-11 | 0                   | 29            | 25             | 32                       | NULL      | 86.20690          |
| 12 | 2014-05-12 | 766                 | 258           | 117            | 33                       | 33.68146  | 45.34884          |
| 13 | 2014-05-13 | 491                 | 193           | 92             | 31                       | 39.30754  | 47.66839          |
| 14 | 2014-05-14 | 447                 | 151           | 62             | 7                        | 33.78076  | 41.05960          |
| 15 | 2014-05-15 | 494                 | 146           | 57             | 6                        | 29.55466  | 39.04110          |
| 16 | 2014-05-16 | 467                 | 162           | 92             | 29                       | 34.68951  | 56.79012          |

123 rows were being returned.

ii)

| #  | action             | unique_users | total_actions | Week    |
|----|--------------------|--------------|---------------|---------|
| 1  | email_clickthrough | 166          | 166           | 2014/17 |
| 2  | email_clickthrough | 425          | 430           | 2014/18 |
| 3  | email_clickthrough | 476          | 477           | 2014/19 |
| 4  | email_clickthrough | 501          | 507           | 2014/20 |
| 5  | email_clickthrough | 436          | 443           | 2014/21 |
| 6  | email_clickthrough | 478          | 488           | 2014/22 |
| 7  | email_clickthrough | 529          | 538           | 2014/23 |
| 8  | email_clickthrough | 549          | 554           | 2014/24 |
| 9  | email_clickthrough | 524          | 530           | 2014/25 |
| 10 | email_clickthrough | 550          | 556           | 2014/26 |
| 11 | email_clickthrough | 613          | 621           | 2014/27 |
| 12 | email_clickthrough | 594          | 599           | 2014/28 |
| 13 | email_clickthrough | 583          | 590           | 2014/29 |
| 14 | email_clickthrough | 625          | 630           | 2014/30 |
| 15 | email_clickthrough | 444          | 445           | 2014/31 |
| 16 | email_clickthrough | 416          | 418           | 2014/32 |

75 rows were being returned.

### Tech Stack Used:

I have used MySQL Workbench for my project. MySQL Workbench is cross platform and it offers a modern and intuitive graphical user interface (GUI) that combines all the functionalities in a single window. It provides visual representation of databases, schema design tools, SQL editor, and query execution capabilities.

## **Insights:**

### **Case study 1:**

- The number of jobs reviewed per hour for each day in November 2020 is less than or equal to 2.
- Temporary dip(0.022-0.016) followed by recovery(0.016-0.035) of throughput.
- Over the last 30 days, Persian has the most percentage among other languages. All other languages are in equal proportion.
- There are no duplicate rows in the job\_data table.

### **Case study 2:**

- The total engagement of each user in each week has been found.
- The maximum growth of users for the product was found in July month [569 increased in one week].
- The weekly engagement of each device has been found which can give meaningful information.
- Email open rate is around 32 percent and the email clickthrough rate is around 45 percent [more than half of the opened emails go unclicked].

### **Result:**

This project imitates the real world scenario and has given me an opportunity to test my knowledge on SQL. I have learnt how to use critical thinking and the knowledge of SQL to solve various problems. I have found various insights and these insights should actually help the operations team a lot if it is a real world case.