Operation Analytics and Investigating Metric Spike

Submitted by:
Nadendla Dharani

Project Description

The project focuses on Operational Analytics, aiming to analyze end-to-end operations within the company to identify areas for improvement. As a Lead Data Analyst, the goal is to investigate and explain sudden changes in key metrics, addressing questions from various departments. By using advanced SQL skills, the project aims to derive valuable insights that can enhance the company's operations and shed light on metric spikes.

Approach

In the context of Operational Analytics, the initial step involves a comprehensive data exploration phase. This includes a thorough examination of the structure of provided datasets to identify relevant columns, relationships, and potential key metrics.

Following data exploration, the focus shifts to the crafting of SQL queries tailored to extract, transform, and analyze data to address specific inquiries from different departments. MySQL Workbench is utilized as the primary tool for database design, querying, and visualization, providing an integrated environment for efficient analysis. Statistical analysis tools are incorporated where necessary to delve deeper into insights. The insights generated contribute to data-driven decision-making processes across departments, fostering collaboration and promoting a holistic understanding of company operations.

The overall approach combines meticulous data exploration, strategic SQL query design, and collaborative efforts to derive actionable insights.

Tech-Stack Used

The technology stack employed for this project includes MySQL Workbench and SQL queries. MySQL Workbench serves as the primary tool for visualizations and provides a robust platform for interpreting the analyzed data. SQL queries, on the other hand, are instrumental for extracting, manipulating, and analyzing the user data from the given databases efficiently. This combination was chosen for its effectiveness in handling the specific requirements of the project, allowing for seamless data exploration and extraction of meaningful insights from the given data.

Insights

- Peaks in job reviews during specific hours can be identified, aiding in scheduling resource allocation and workload management.
- Daily throughput and the 7-day rolling average provide different perspectives. Daily metrics offer a granular view, while the rolling average smoothens short-term fluctuations, making it preferable for trend analysis.
- This query reveals the percentage share of each language over the last 30 days, assisting in language-related decision-making and content prioritization.
- Identified duplicate rows based on multiple columns, providing a basis for data cleansing and ensuring data integrity.
- By analyzing weekly user engagement, trends in user activity can be identified, potentially aligning with marketing campaigns or product releases.
- Understanding user growth over time is crucial for assessing the product's performance and potential market saturation.
- Weekly retention analysis sheds light on user stickiness, highlighting areas for improvement in onboarding and user engagement strategies.
- Understanding how user engagement varies across devices can inform device-specific optimizations and feature development.
- Analyzing email engagement metrics provides insights into the effectiveness of email campaigns, helping refine communication strategies.

These insights collectively empower data-driven decision-making across departments, enabling proactive responses to operational challenges and driving improvements in overall business performance.

Results

Case Study 1: Job Data Analysis

You will be working with a table named job_data with the following columns:

- **job_id:** Unique identifier of jobs
- **actor_id:** Unique identifier of actor
- **event:** The type of event (decision/skip/transfer).
- **language:** The Language of the content
- **time_spent:** Time spent to review the job in seconds.
- **org:** The Organization of the actor
- **ds:** The date in the format yyyy/mm/dd (stored as text).

Creating the job_data table:

```sql
create database jobs;
create table job_data (
    ds date,
    job_id int not null,
    actor_id int not null,
    event varchar(50) not null,
    language varchar(50) not null,
    time_spent int not null,
    org char(5)
);
insert into job_data (ds, job_id, actor_id, event, language, time_spent, org)
values ('2020-11-30', 21, 1001, 'skip', 'English', 15, 'A'),
    ('2020-11-30', 22, 1006, 'transfer', 'Arabic', 25, 'B'),
    ('2020-11-29', 23, 1003, 'decision', 'Persian', 20, 'C'),
    ('2020-11-28', 23, 1005,'transfer', 'Persian', 22, 'D'),
    ('2020-11-28', 25, 1002, 'decision', 'Hindi', 11, 'B'),
    ('2020-11-27', 11, 1007, 'decision', 'French', 104, 'D'),
    ('2020-11-26', 23, 1004, 'skip', 'Persian', 56, 'A'),
    ('2020-11-25', 20, 1003, 'transfer', 'Italian', 45, 'C');
select * from job_data;
```

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

ds	job_id	actor_id	event	language	time_spent	org
2020-11-30	21	1001	skip	English	15	A
2020-11-30	22	1006	transfer	Arabic	25	B
2020-11-29	23	1003	decision	Persian	20	C
2020-11-28	23	1005	transfer	Persian	22	D
2020-11-28	25	1002	decision	Hindi	11	B
2020-11-27	11	1007	decision	French	104	D
2020-11-26	23	1004	skip	Persian	56	A
2020-11-25	20	1003	transfer	Italian	45	C

Tasks:

1. **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.

```
22 • select ds, round((count(job_id)/sum(time_spent))*3600) as num_of_Jobs_Reviewed_per_hour
23   from job_data
24   group by ds
25   order by ds;
26
```

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

ds	num_of_Jobs_Reviewed_per_hour
2020-11-25	80
2020-11-26	64
2020-11-27	35
2020-11-28	218
2020-11-29	180
2020-11-30	180

Here I have calculated the number of jobs reviewed per hour by dividing the number of times a user reviewed by total time spent, later this result is multiplied by 3600 because time spent is given in seconds and we need the result in hours. Also round() function is used which returns the only integer value excluding decimal values.

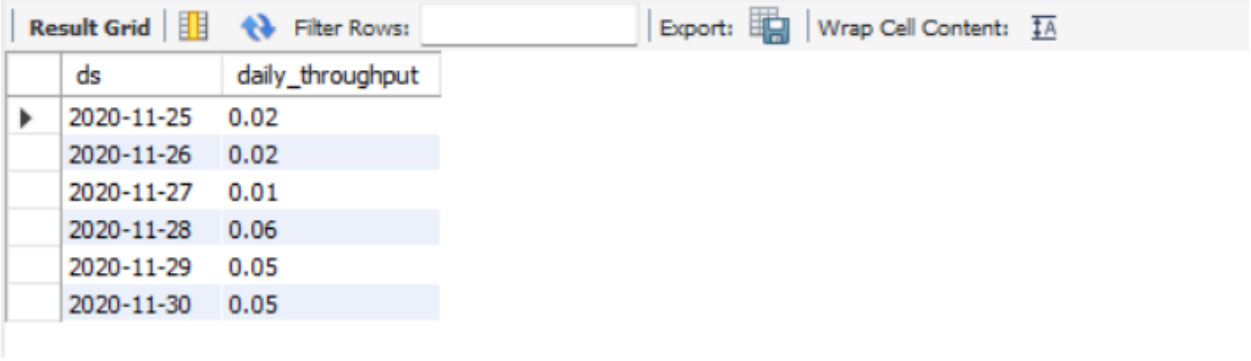
2. **Throughput Analysis:**
    - Objective: Calculate the 7-day rolling average of throughput (number of events per second).
    - Your Task: Write an SQL query to calculate the 7-day rolling average of throughput. Additionally, explain whether you prefer using the daily metric or the 7-day rolling average for throughput, and why.

```
27 • select round(count(event)/sum(time_spent),2) as 7day_throughput
28   from job_data;
29
```

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

7day_throughput
0.03

```sql
select ds, round(count(event)/sum(time_spent),2) as daily_throughput
from job_data
group by ds
order by ds;
```



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

ds	daily_throughput
2020-11-25	0.02
2020-11-26	0.02
2020-11-27	0.01
2020-11-28	0.06
2020-11-29	0.05
2020-11-30	0.05

Here after finding 7-day rolling throughput and daily throughput, I would say I prefer both, because both offer different perspectives for the trend analysis. So both are crucial for a proper data-driven decision.

3. **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.

```sql
select language, count(language) as language_count,
(count(language) / (select count(*) from job_data)) * 100 as percentage
from job_data
group by language
order by language DESC;
```

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

language	language_count	percentage
Persian	3	37.5000
Italian	1	12.5000
Hindi	1	12.5000
French	1	12.5000
English	1	12.5000
Arabic	1	12.5000

Here, there are languages that are repeated multiple times so I displayed the count of each language for better understanding and also found percentage using the basic formula, (number/total)*100.

4. **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.

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

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

actor_id	duplicates
1003	2

actor_id is a unique attribute, so I counted the number of times each ID repeated and displayed the ID which has counted more than 1.

Case Study 2: Investigating Metric Spike

You will be working with three tables:

- **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.

Tasks:

1. **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.

```
61 •  select week(occurred_at) as Week, count(distinct user_id) as Weekly_Active_Users
62    from events
63    where event_type = "engagement"
64    group by week(occurred_at);
```

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

Week	Weekly_Active_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

Here I have extracted week from the occured_at column, and in the event_type column I have selected rows having “engagement” and printed the output resulting number of active users in each particular week.

2. **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.

```
66 •  select months, total_users, round(((total_users/Lag(total_users, 1) over (order by months) - 1)*100),2) as Growth_inc
67    from(
68    select extract(month from created_at) as months, count(distinct user_id) as total_users
69    from users
70    group by months
71    order by months) sub;
```

months	total_users	Growth_inc
1	712	NULL
2	685	-3.79
3	765	11.68
4	907	18.56
5	993	9.48
6	1086	9.37
7	1281	17.96
8	1347	5.15
9	330	-75.50
10	390	18.18
11	399	2.31
12	486	21.80

Here I have used the ‘Lag’ function, which is used to access previous rows. It is a useful function in comparing the current row value from the previous row value.

3. **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.

```sql
with users_start as (
select user_id, extract(week from created_at) as start_week
from users
),
user_activity as (
select user_id, extract(week from occurred_at) as activity_week
from events
)
select users_start.start_week as cohort_week,
    user_activity.activity_week as retention_week,
    count(distinct user_activity.user_id) as total_users
from users_start
left join user_activity
on users_start.user_id = user_activity.user_id and user_activity.activity_week >= users_start.start_week
group by users_start.start_week, user_activity.activity_week
order by users_start.start_week, user_activity.activity_week;
```

Result Grid | Filter Rows: | Export:

cohort_week	retention_week	total_users
0	NULL	0
0	17	5
0	18	11
0	19	15
0	20	12
0	21	12
0	22	14
0	23	12
0	24	19
0	25	12
0	26	13
0	27	11
0	28	8
0	29	6

I solved the question using ‘WITH’ clause subquery and created two subqueries, users_start and user_activity. Also in the main subquery I performed a LEFT JOIN operation on users_start and user_activity.

Also, the output for this question is very large. So I have exported the output to an excel sheet and I am attaching the drive link to the sheet here. For referral, I have provided a small screenshot of the total output.

https://drive.google.com/file/d/1jLvo0geYZno3iAkMc9CcXb5pWVATKYWm/view?usp=sharing

4. **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.

```
141 •  select extract(week from occurred_at) as week, device, count(distinct user_id) as total_active_users,
142    count(*) as total_activities
143    from events
144    group by week, device
145    order by week, device;
146
```

Result Grid | Filter Rows: | Export:

week	device	total_active_users	total_activities
17	acer aspire desktop	9	113
17	acer aspire notebook	20	354
17	amazon fire phone	4	152
17	asus chromebook	21	413
17	dell inspiron desktop	18	315
17	dell inspiron notebook	46	771
17	hp pavilion desktop	14	220
17	htc one	16	241
17	ipad air	27	567
17	ipad mini	19	337
17	iphone 4s	21	382
17	iphone 5	65	1190
17	iphone 5s	42	757
17	kindle fire	6	99

Result 45

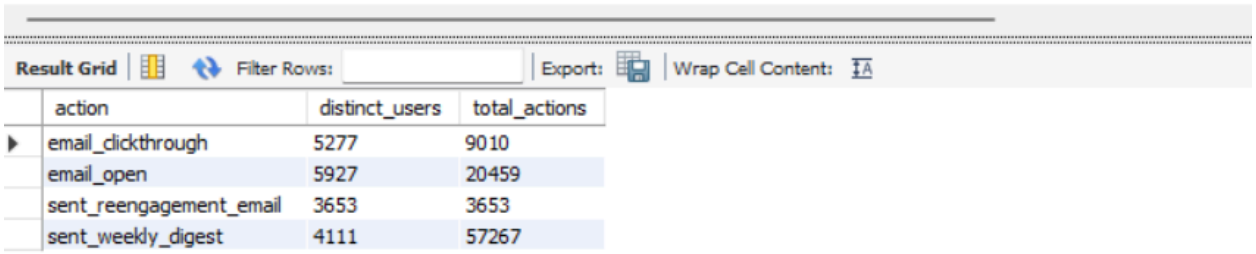
For this question the output is very large, so I have exported the output to an excel sheet and I am attaching the drive link to the sheet here. For referral, I have provided a small screenshot of the total output

https://drive.google.com/file/d/1Lubn0VJBwAsUjwyw1Vv4KJ67uj-mOJna/view?usp=sharing

5. **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.

```sql
select action, count(distinct user_id) as distinct_users, count(*) as total_actions
from email_events
group by action
order by action;

```



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

action	distinct_users	total_actions
email_clickthrough	5277	9010
email_open	5927	20459
sent_reengagement_email	3653	3653
sent_weekly_digest	4111	57267

Through the GROUP BY clause, I have provided the output of a unique number of users and total actions taken place in every action respectively.