Project Summary: Operational Analytics

Objective: Operational Analytics plays a vital role in analysing a company's operations from start to finish. This process helps pinpoint areas for improvement within the organisation. As a Data Analyst, I work closely with various teams, such as operations, support, and marketing, to extract valuable insights from the data they collect. One of my key responsibilities is investigating metric spikes—understanding and explaining sudden changes in key metrics like daily user engagement or sales dips.

In this project, I take on the role of a Lead Data Analyst at a company like Microsoft. I am provided with various datasets and tables, and my task is to derive insights from this data to answer questions posed by different departments within the company. Using advanced SQL skills, I analyse the data and provide valuable insights to help improve the company's operations and understand sudden changes in key metrics.

Case Study 1: Job Data Analysis

Table: `job_data`

Columns:

- `job_id`: Unique identifier of jobs
- `actor_id`: Unique identifier of actor
- `event`: Type of event (decision/skip/transfer)
- `language`: Language of the content
- `time_spent`: Time spent reviewing the job (in seconds)
- `org`: Organization of the actor
- `ds`: Date (yyyy/mm/dd format, stored as text)

Tasks:

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.

```sql
    SELECT
    DATE(STR_TO_DATE(ds, '%m/%d/%Y')) AS review_day,
    COUNT(*) / 24 AS jobs_per_hour
FROM
    job_data
WHERE
    STR_TO_DATE(ds, '%m/%d/%Y') >= '2020-11-01' AND STR_TO_DATE(ds, '%m/%d/%Y') < '2020-12-01'
GROUP BY
    review_day;
```

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

review_day	jobs_per_hour
2020-11-30	0.0833
2020-11-29	0.0417
2020-11-28	0.0833
2020-11-27	0.0417
2020-11-26	0.0417
2020-11-25	0.0417

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 I prefer using the daily metric or the 7-day rolling average for throughput, and why.

Daily throughput :

```sql
select event_date, events_count * 1.0 / 86400  as throughput_per_second from (
 SELECT
        STR_TO_DATE(ds, '%m/%d/%Y') AS event_date,
        COUNT(*) AS events_count

    FROM job_data
    GROUP BY STR_TO_DATE(ds, '%m/%d/%Y'))as sub
    GROUP BY event_date order by event_date;
```

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

event_date	throughput_per_second
2020-11-25	0.00001
2020-11-26	0.00001
2020-11-27	0.00001
2020-11-28	0.00002
2020-11-29	0.00001
2020-11-30	0.00002

Interpretation

1. Daily Metric Analysis:

- **Consistent Throughput**: For most days, the throughput remains consistent at `0.00001` events per second.
- **Spike on 2020-11-28 and 2020-11-30**: There are noticeable spikes on 2020-11-28 and 2020-11-30, where the throughput doubles to `0.00002`.

```sql
WITH daily_events AS (
    SELECT
        STR_TO_DATE(ds, '%m/%d/%Y') AS event_date,  COUNT(*) AS events_count
    FROM job_data
    GROUP BY STR_TO_DATE(ds, '%m/%d/%Y')
),
daily_throughput AS (
    SELECT event_date, events_count,
    events_count * 1.0 / 86400 AS throughput_per_second -- 86400 seconds in a day
    FROM daily_events)
SELECT
    dt1.event_date,
    AVG(dt2.throughput_per_second) AS avg_throughput_per_second
FROM
    daily_throughput dt1
    JOIN daily_throughput dt2 ON dt1.event_date BETWEEN DATE_SUB(dt2.event_date, INTERVAL 6 DAY) AND dt2.event_date
GROUP BY dt1.event_date ORDER BY dt1.event_date;
```

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

event_date	avg_throughput_per_second
2020-11-25	0.000013333
2020-11-26	0.000014000
2020-11-27	0.000015000
2020-11-28	0.000016666
2020-11-29	0.000015000
2020-11-30	0.000020000

Interpretation

7-Day Rolling Average Analysis:

- **Smoothed Trends**: The rolling average shows a smoother trend without abrupt changes. It gradually increases from `0.000013333` on 2020-11-25 to `0.000020000` on 2020-11-30.
- **Less Impact of Spikes**: The rolling average mitigates the impact of the daily spikes observed on 2020-11-28 and 2020-11-30, providing a more gradual increase in throughput.

Given these observations, I prefer using the 7-day rolling average for throughput because:

- **Smoother Trends**: It provides a clearer view of overall trends and performance.
- **Noise Reduction**: It minimises the impact of daily noise and anomalies, making it easier to detect genuine trends.

- **Better Decision Making**: Long-term trends are more reliable for making strategic decisions, as they are less likely to be influenced by short-term irregularities.

However, the daily metric is still valuable for identifying and addressing specific days with significant changes, so both metrics have their uses depending on the context and needs of the analysis.

Language Share Analysis

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

```sql
with recent_data as (
select language ,
count(*) as lang_count
from job_data where str_to_date(ds,'%m/%d/%Y') between date_sub('2020-12-05', INTERVAL 30 DAY) AND CURDATE()
group by language),
total_language_count as(
select sum(lang_count) as total_count from recent_data)

select t.language,t.lang_count, (t.lang_count *100 / m.total_count ) as language_percent
from recent_data t , total_language_count m
order by language_percent desc;
```

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

language	lang_count	language_percent
Persian	3	37.5000
English	1	12.5000
Arabic	1	12.5000
Hindi	1	12.5000
French	1	12.5000

Let's assume the current date is ‘2020-12-05’ as the data available in this table is of 2020 and to calculate the percentage share of each language in the last 30 days won't make any sense if take the current date which is year 2024 month 7 july

This query filters the data for last 30 days

Count the event per language

Calculate the total event within the 30 days

Calculate the percentage share of each language

Duplicate Rows Detection

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

```sql
select * from job_data;

select count(*),str_to_date(ds,'%m/%d/%Y') as job_date,job_id,actor_id,event , language,
time_spent , org from job_data group by job_date,job_id,actor_id,event , language,
time_spent , org having count(*)>1;
```

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

count(*)	job_date	job_id	actor_id	event	language	time_spent	org

There is no duplicate data , this query group by each column value and count the occurrence of any set of rows similar occurring more than one that is duplicate which will be displayed as of now there is no duplicate in the given data .

Case Study 2: Investigating Metric Spike

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.
    - **Task:** Write an SQL query to calculate weekly user engagement.

```sql
select * from users;
select * from events;

select distinct event_type from events;
```

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

event_type
engagement
signup_flow

I looked for different event type in the events table

```sql
SELECT
    occurred_at,
    YEAR(occurred_at) AS year,
    MONTH(occurred_at) AS month,
    WEEK(occurred_at, 1) AS year_week,  -- Week number with Monday as the first day of the we
    DAY(occurred_at) AS day,
    DAYOFWEEK(occurred_at) AS day_of_week,  -- 1 = Sunday, 2 = Monday, ..., 7 = Saturday
    HOUR(occurred_at) AS hour,
    MINUTE(occurred_at) AS minute,
    SECOND(occurred_at) AS second,
    DATE_FORMAT(occurred_at, '%Y-%m-%d %H:%i:%s') AS formatted_date
FROM
    events;
```

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

occurred_at	year	month	year_week	day	day_of_week	hour	minute	second	formatted_date
2014-05-02 11:02:00	2014	5	18	2	6	11	2	0	2014-05-02 11:02:00
2014-05-02 11:02:00	2014	5	18	2	6	11	2	0	2014-05-02 11:02:00
2014-05-02 11:03:00	2014	5	18	2	6	11	3	0	2014-05-02 11:03:00
2014-05-02 11:04:00	2014	5	18	2	6	11	4	0	2014-05-02 11:04:00
2014-05-02 11:03:00	2014	5	18	2	6	11	3	0	2014-05-02 11:03:00
2014-05-02 11:03:00	2014	5	18	2	6	11	3	0	2014-05-02 11:03:00
2014-05-01 09:59:00	2014	5	18	1	5	9	59	0	2014-05-01 09:59:00
2014-05-01 10:00:00	2014	5	18	1	5	10	0	0	2014-05-01 10:00:00
2014-05-01 10:00:00	2014	5	18	1	5	10	0	0	2014-05-01 10:00:00

Result 23

To find weekly user engagement i need find out the weekyear and this query helped me to understand the date format very well

```
select f.user_id , yearweek(occurred_at,1) as year_week,
count(e.event_name) as event_count from users f , events e
where f.user_id = e.user_id group by yearweek(e.occurred_at,1),f.user_id
order by year_week  ;


select * from events where user_id = 8
```

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

user_id	year_week	event_count
8	201418	2
22	201418	8
66	201418	14
163	201418	22
172	201418	15
209	201418	9
232	201418	47
239	201418	9
253	201418	24

Result 29

This gives the weekly engagement of user here 201418 (2014 is year and 18 is the 18th week of this year)

User Growth Analysis

- **Objective:** Analyze user growth over time for a product.
- **Task:** Write an SQL query to calculate user growth.

```sql
SELECT
    DATE_FORMAT(created_at, '%Y-%m') AS month,
    COUNT(user_id) AS new_users,
    IFNULL(LAG(COUNT(user_id)) OVER (ORDER BY DATE_FORMAT(created_at, '%Y-%m')), 0) AS previous_month_users,
    COUNT(user_id) - IFNULL(LAG(COUNT(user_id)) OVER (ORDER BY DATE_FORMAT(created_at, '%Y-%m')), 0) AS user_growth
FROM
    users
GROUP BY
    DATE_FORMAT(created_at, '%Y-%m')
ORDER BY
    DATE_FORMAT(created_at, '%Y-%m');
```

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

month	new_users	previous_month_users	user_growth
2013-01	160	0	160
2013-02	160	160	0
2013-03	150	160	-10
2013-04	181	150	31
2013-05	214	181	33

Using `LAG(COUNT(user_id)) OVER (ORDER BY DATE_FORMAT(created_at, '%Y-%m'))`, we get the count of users from the previous month. `IFNULL` is used to handle cases where there is no previous month

User growth is calculated previous month user - the current month user

This gives the clear details about user growth of every month

Weekly Retention Analysis

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

```sql
with sign_up as (
SELECT
    u.user_id,
    u.created_at,
    e.occurred_at,
    YEARWEEK(u.created_at, 3) AS signup_week,
    YEARWEEK(e.occurred_at, 3) AS event_week,
    TIMESTAMPDIFF(WEEK, u.created_at, e.occurred_at) AS week_diff
FROM
    users u
JOIN
    events e
ON
    u.user_id = e.user_id)

 select signup_week, week_diff, count(distinct user_id ) as retained_users
 from sign_up group by signup_week, week_diff order by signup_week, week_diff
```

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

signup_week	week_diff	retained_users
201301	69	3
201301	70	4
201301	71	3
201301	72	3
201301	73	4
201301	74	3
201301	75	3

Interpretation

- `signup_week`: The week number when the users signed up.
- `week_diff`: The difference in weeks between the sign-up date and the event date.
- `retained_users`: The count of distinct users who performed an event after a certain number of weeks since signing up.

Weekly Engagement Per Device

- **Objective:** Measure user activeness on a weekly basis per device.
- **Task:** Write an SQL query to calculate weekly engagement per device.

```sql
select * from events;

SELECT count(distinct user_id) as active_users,
    YEARWEEK(occurred_at, 1) AS event_year_week,
    device
FROM
    events
GROUP BY
    YEARWEEK(occurred_at, 1), device
ORDER BY
    event_year_week;
```

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

active_users	event_year_week	device
10	201418	acer aspire desktop
21	201418	acer aspire notebook
4	201418	amazon fire phone
23	201418	asus chromebook
21	201418	dell inspiron desktop
49	201418	dell inspiron notebook

Result 14

- `user_count`: The number of distinct users who were active in each week.
- `event_date`: The year and week number during which the users were active.
- `device`: The type of device used by the users.

Email Engagement Analysis

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

```sql
SELECT
    COUNT(*) AS total_actions,
    COUNT(DISTINCT user_id) AS unique_users,
    user_type,
    action,
    COUNT(*) AS action_count,
    DATE(occurred_at) AS date
FROM
    email_events
GROUP BY
    user_type,
    action,
    DATE(occurred_at)
ORDER BY
    DATE(occurred_at), user_type, action;
```

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

total_actions	unique_users	user_type	action	action_count	date
18	18	1	email_clickthrough	18	2014-05-01
50	50	1	email_open	50	2014-05-01
158	158	1	sent_weekly_digest	158	2014-05-01
14	14	2	email_clickthrough	14	2014-05-01
33	33	2	email_open	33	2014-05-01
2	2	2	sent_reengagement_email	2	2014-05-01
119	119	2	sent_weekly_digest	119	2014-05-01
29	29	3	email_clickthrough	29	2014-05-01
62	62	3	email_open	62	2014-05-01
5	5	3	sent_reengagement_email	5	2014-05-01

By using these clauses together, the query provides a comprehensive set of email engagement metrics, including:

- The total number of actions.
- The number of unique users interacting with the service.
- The breakdown of actions by user type and action type.
- The number of actions on a daily basis.