Project – 3

Operation Analytics and Investigating Metric Spike

```
/* 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).*/
create database project3;
use project3;
CREATE TABLE job_data (
   job_id INT,
   actor_id INT,
   event VARCHAR(20),
   language VARCHAR(50),
   time_spent INT,
   org VARCHAR(100),
   ds VARCHAR(10)
);
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');
```

TASKS

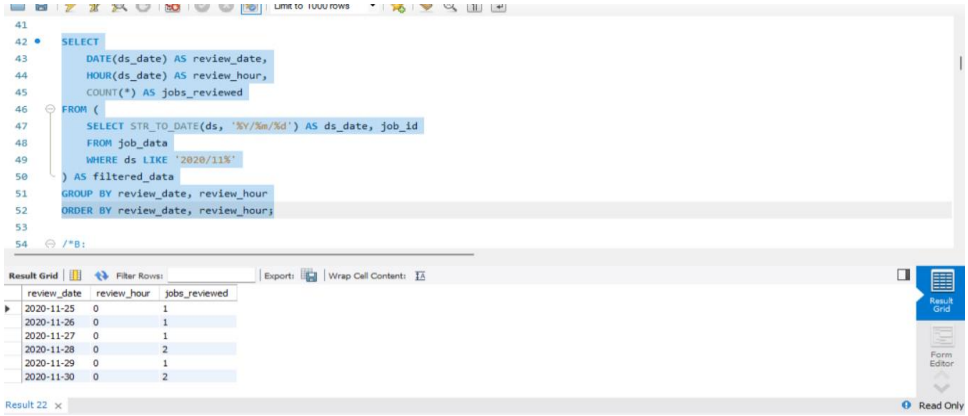
/*A:

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.*/

```
SELECT
   DATE(ds_date) AS review_date,
   HOUR(ds_date) AS review_hour,
   COUNT(*) AS jobs_reviewed
FROM (
   SELECT STR_TO_DATE(ds, '%Y/%m/%d') AS ds_date, job_id
   FROM job_data
   WHERE ds LIKE '2020/11%'
) AS filtered_data
GROUP BY review_date, review_hour
ORDER BY review_date, review_hour;
```



review_date	review_hour	jobs_reviewed
2020-11-25	0	1
2020-11-26	0	1
2020-11-27	0	1
2020-11-28	0	2
2020-11-29	0	1
2020-11-30	0	2

/*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.*/

```
SELECT
   language,
   COUNT(*) * 100.0 / (
      SELECT COUNT(*)
      FROM job_data
```

```
    WHERE ds >= '2020/11/01' AND ds <= '2020/11/30'
  ) AS language_share_percentage
FROM job_data
WHERE ds >= '2020/11/01' AND ds <= '2020/11/30'
GROUP BY language
ORDER BY language_share_percentage DESC;
```

Result Grid | Filter Rows: | Export

language	language_share_percentage
Persian	37.50000
English	12.50000
Arabic	12.50000
Hindi	12.50000
French	12.50000
Italian	12.50000

```
/* 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.*/
SELECT
  language,
  COUNT(*) * 100.0 / (
    SELECT COUNT(*)
    FROM job_data
    WHERE STR_TO_DATE(ds, '%Y/%m/%d') >= CURDATE() - INTERVAL 30 DAY
  ) AS language_share_percentage
FROM job_data
WHERE STR_TO_DATE(ds, '%Y/%m/%d') >= CURDATE() - INTERVAL 30 DAY
GROUP BY language
ORDER BY language_share_percentage DESC;
```

Output will be Null

Because there is no data present.

```
/* 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.*/
```

```
SELECT
    job_id,
    actor_id,
    event,
    language,
    time_spent,
    org,
    ds,
    COUNT(*) AS duplicate_count
FROM job_data
GROUP BY
    job_id,
    actor_id,
    event,
    language,
    time_spent,
    org,
    ds
HAVING COUNT(*) > 1;
```

OUTPUT: NO output no data present

/* 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.*/

```
# Table 1
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    created_at varchar(100),
    company_id INT,
    language VARCHAR(50),
    activated_at varchar(100),
```

```
    state VARCHAR(50)
);

show variables LIKE 'secure_file_priv';
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;

select * from users

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;

# Table 2
create table events(
user_id int,
event_type varchar(50),
event_name varchar(100),
location varchar(100) ,
device varchar(50) ,
user_type int,
occurred_at varchar(100)
);

load data infile "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/events.csv"
into table events
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 rows;
```

```
desc events;

select * from events

ALTER TABLE events ADD COLUMN temp_created_at datetime;
UPDATE events SET temp_created_at = STR_TO_DATE(ocurred_at_at, '%d-%m-%Y %H:%i');
alter table events drop column occured_at;
alter table events change column temp_created_at occured_at datetime;
```

Table 3

```
create table email_events (
user_id int,
action varchar(100),
user_type int,
occurred_at varchar(100)
);

load data infile "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/email_events.csv"
into table email_events
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 rows;

select * from email_events

ALTER TABLE email_events ADD COLUMN temp_created_at datetime;
UPDATE email_events SET temp_created_at = STR_TO_DATE(ocurred_at_at, '%d-%m-%Y %H:%i');
alter table email_events drop column occured_at;
alter table email_events change column temp_created_at occured_at datetime;
```

#Tasks:

/* 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.*/

```
select * from events

SELECT YEAR(occurred_at) AS year,

    WEEK(occurred_at) AS week_number,

    COUNT(DISTINCT user_id) AS active_users

FROM events

GROUP BY year, week_number

ORDER BY year, week_number;
```

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

year	week_number	active_users
2014	17	689
2014	18	1661
2014	19	1737
2014	20	1675
2014	21	166
2014	22	102
2014	23	65
2014	24	43
2014	25	4

/* 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.*/

```
select * from users

SELECT DATE_FORMAT(created_at, '%Y-%m') AS month,

    COUNT(DISTINCT user_id) AS new_users

FROM users

GROUP BY month

ORDER BY month;
```

Result Grid | Filter Rows: | Export: | Wrap

month	new_users
2013-01	160
2013-02	160
2013-03	150
2013-04	181
2013-05	214
2013-06	213
2013-07	284
2013-08	316
2013-09	330

/* 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.*/

```
select * from email_events

SELECT DATE_FORMAT(created_at, '%Y-%m-%d') AS cohort_week,

    YEAR(created_at) AS year,

    WEEK(created_at) AS week_number,

    COUNT(DISTINCT u.user_id) AS cohort_size,

    COUNT(DISTINCT CASE WHEN DATEDIFF(e.occurred_at, u.created_at) BETWEEN 0 AND 6 THEN e.user_id END) AS retained_users

FROM users u

LEFT JOIN events e ON u.user_id = e.user_id

GROUP BY cohort_week, year, week_number

ORDER BY cohort_week, year, week_number;
```

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

cohort_week	year	week_number	cohort_size	retained_users
2013-01-01	2013	0	7	0
2013-01-02	2013	0	7	0
2013-01-03	2013	0	6	0
2013-01-04	2013	0	1	0
2013-01-05	2013	0	2	0
2013-01-06	2013	1	3	0
2013-01-07	2013	1	4	0
2013-01-08	2013	1	2	0
2013-01-09	2013	1	6	0

Result 27

/* 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.*/

```
select * from events

SELECT

   YEAR(occurred_at) AS year,

   WEEK(occurred_at) AS week_number,

   device,

   COUNT(DISTINCT user_id) AS active_users

FROM events

GROUP BY year, week_number, device

ORDER BY year, week_number, device;
```

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

year	week_number	device	active_users
2014	17	acer aspire desktop	11
2014	17	acer aspire notebook	22
2014	17	amazon fire phone	12
2014	17	asus chromebook	14
2014	17	dell inspiron desktop	18
2014	17	dell inspiron notebook	23
2014	17	hp pavilion desktop	19
2014	17	htc one	6
2014	17	ipad air	10

Result 28

/* 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.*/

```
select * from email_events

SELECT
   YEAR(occurred_at) AS year,
   MONTH(occurred_at) AS month,
   COUNT(DISTINCT user_id) AS unique_users,
   COUNT(*) AS total_actions
FROM email_events
GROUP BY year, month
ORDER BY year, month;
```

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

year	month	unique_users	total_actions
2014	5	3270	3270
2014	6	878	878
2014	7	950	950
2014	8	1081	1081