

Operation Analytics and Investigating Metric Spike

Case Study 1: Job Data Analysis

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

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
1 use assignments_2 ;
2 # 7 DAY ROLLING THROUGHPUT ;
3 SELECT
4     ds,
5     COUNT(job_id) / (SUM(time_spent) / 3600) AS daily_throughput,
6     avg(count(job_id) / (SUM(time_spent) / 3600)) over (order by ds rows between 6 preceding and current row) AS 7day_throughput
7 from
8     job_data
9 GROUP BY ds
10 order by ds
```

The result grid displays the following data:

ds	daily_throughput	7day_throughput
11/25/2020	80.0000	80.00000000
11/26/2020	64.2857	72.14285829
11/27/2020	34.6154	59.63370075
11/28/2020	218.1818	99.27073408
11/29/2020	180.0000	115.41659086
11/30/2020	180.0000	126.18049269

2. Write an SQL query to calculate the 7-day rolling average of throughput

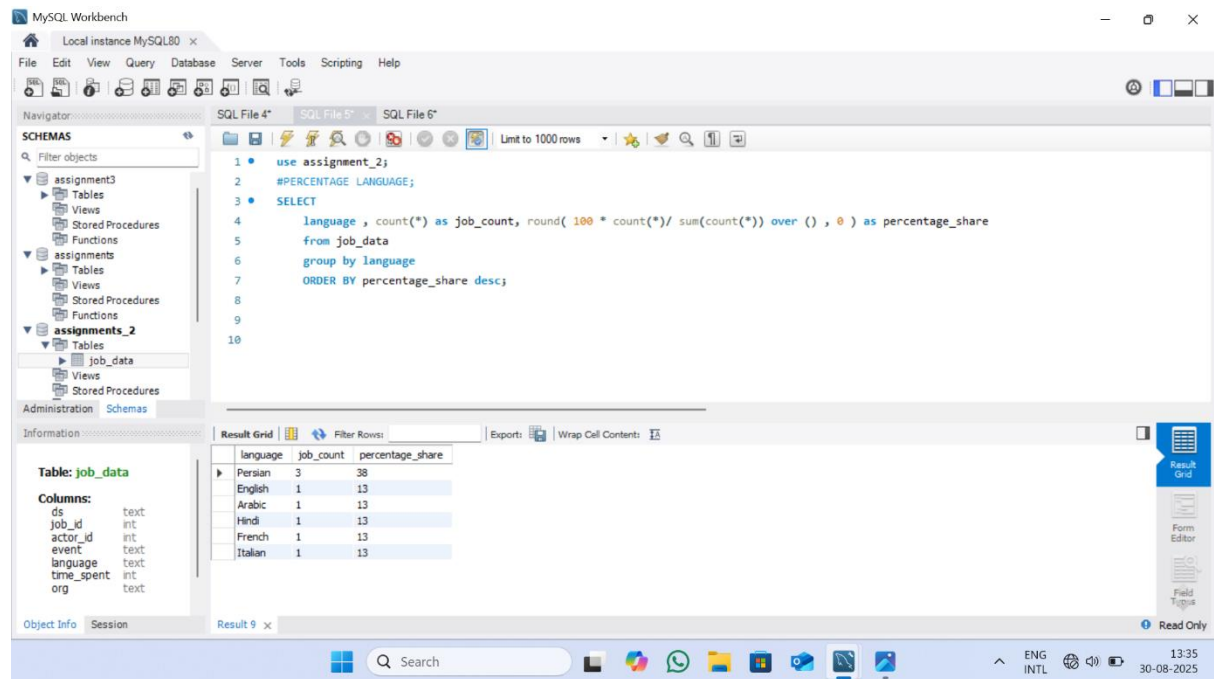
The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
1 use assignments ;
2 # jobs per hour ;
3 SELECT
4     ds,
5     COUNT(job_id) / (SUM(time_spent) / 3600) AS time_spent_hours
6 FROM
7     job_data
8 GROUP BY ds
9 ORDER BY ds
```

The result grid displays the following data:

ds	time_spent_hours
11/25/2020	80.0000
11/26/2020	64.2857
11/27/2020	34.6154
11/28/2020	218.1818
11/29/2020	180.0000
11/30/2020	180.0000

3. Write an SQL query to calculate the percentage share of each language over the last 30 days.



MySQL Workbench interface showing the SQL query and its results.

SQL Query:

```
1 use assignment_2;
2 #PERCENTAGE LANGUAGE;
3 SELECT
4     language , count(*) as job_count, round( 100 * count(*)/ sum(count(*) over () , 0 ) as percentage_share
5     from job_data
6     group by language
7     ORDER BY percentage_share desc;
8
9
10
```

Result Grid:

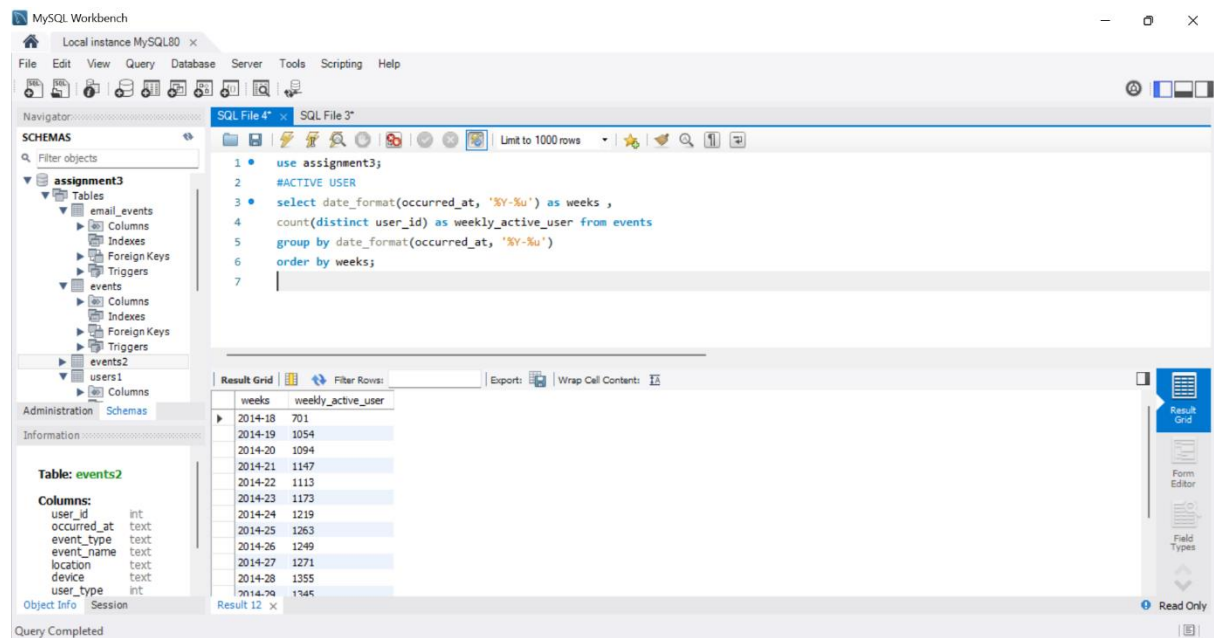
language	job_count	percentage_share
Persian	3	38
English	1	13
Arabic	1	13
Hindi	1	13
French	1	13
Italian	1	13

Table: job_data

Columns:

- ds: text
- job_id: int
- actor_id: int
- event: text
- language: text
- time_spent: int
- org: text

4. Write an SQL query to display duplicate rows fr om the job_data table .



MySQL Workbench interface showing the SQL query and its results.

SQL Query:

```
1 use assignment3;
2 #ACTIVE USER
3 select date_format(occurred_at, '%Y-%u') as weeks ,
4 count(distinct user_id) as weekly_active_user from events
5 group by date_format(occurred_at, '%Y-%u')
6 order by weeks;
7
```

Result Grid:

weeks	weekly_active_user
2014-18	701
2014-19	1054
2014-20	1094
2014-21	1147
2014-22	1113
2014-23	1173
2014-24	1219
2014-25	1263
2014-26	1249
2014-27	1271
2014-28	1355
2014-29	1345

Table: events2

Columns:

- user_id: int
- occurred_at: text
- event_type: text
- event_name: text
- location: text
- device: text
- user_type: int

Case Study 2: Investigating Metric Spike

1. Write an SQL query to calculate the weekly user engagement.

The screenshot shows MySQL Workbench with a SQL query in the editor and its results in the Result Grid.

SQL Query:

```
1 use assignment3;
2 #ACTIVE USER
3 select date_format(occurred_at, '%Y-%u') as weeks ,
4 count(distinct user_id) as weekly_active_user from events
5 group by date_format(occurred_at, '%Y-%u')
6 order by weeks;
```

Result Grid:

weeks	weekly_active_user
2014-18	701
2014-19	1054
2014-20	1094
2014-21	1147
2014-22	1113
2014-23	1173
2014-24	1219
2014-25	1263
2014-26	1249
2014-27	1271
2014-28	1355
2014-29	1345

2. Write an SQL query to calculate the user growth for the product.

The screenshot shows MySQL Workbench with a SQL query in the editor and its results in the Result Grid.

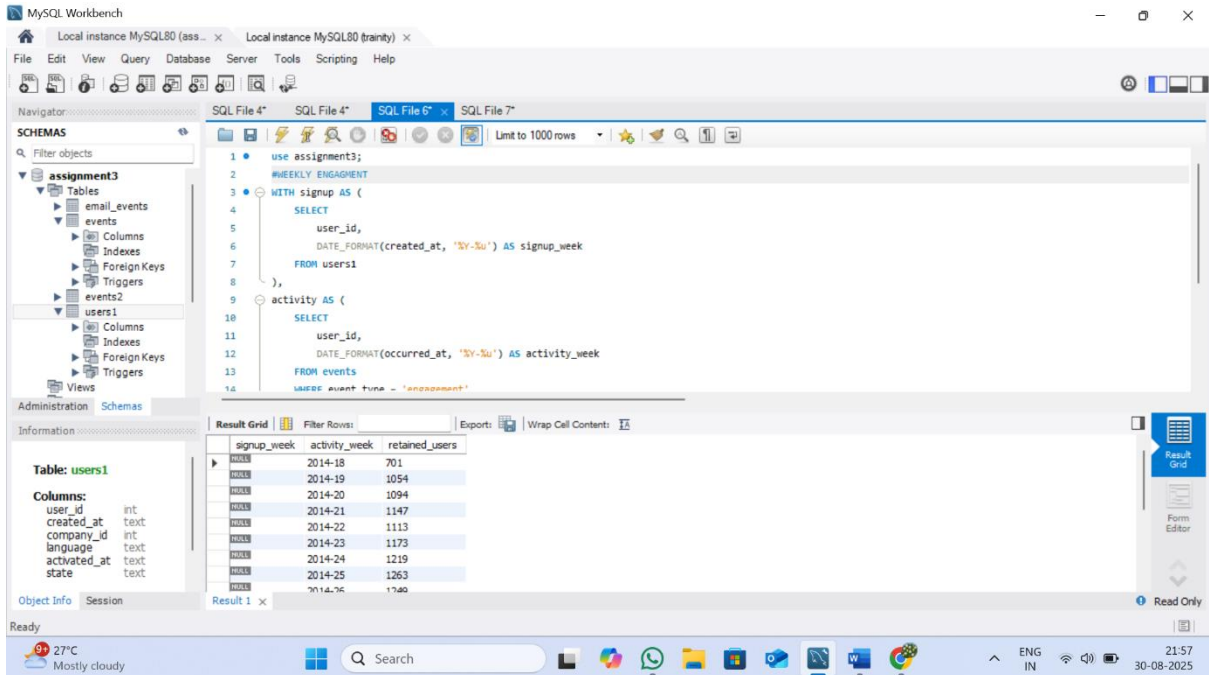
SQL Query:

```
1 use assignment3;
2 # USERS AFTER SIGNING UP FOR PRODUCT
3 SELECT
4 date(activated_at) as dates,
5 e.device,
6 COUNT(DISTINCT u.user_id) AS new_users,
7 SUM(COUNT(DISTINCT u.user_id)) OVER (PARTITION BY e.device ORDER BY date(activated_at) ) AS cumulative_users
8 FROM users1 u
9 JOIN events e ON u.user_id = e.user_id
10 GROUP BY date(activated_at) , e.device
11 ORDER BY date(activated_at) , e.device;
```

Result Grid:

dates	device	new_users	cumulative_users
2014-18	acer aspire desktop	198	198
2014-18	acer aspire notebook	338	338
2014-18	amazon fire phone	89	89
2014-18	asus chromebook	355	355
2014-18	dell inspiron desktop	360	360
2014-18	dell inspiron notebook	677	677
2014-18	hp pavilion desktop	339	339
2014-18	htc one	196	196
2014-18	inspired	478	478

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



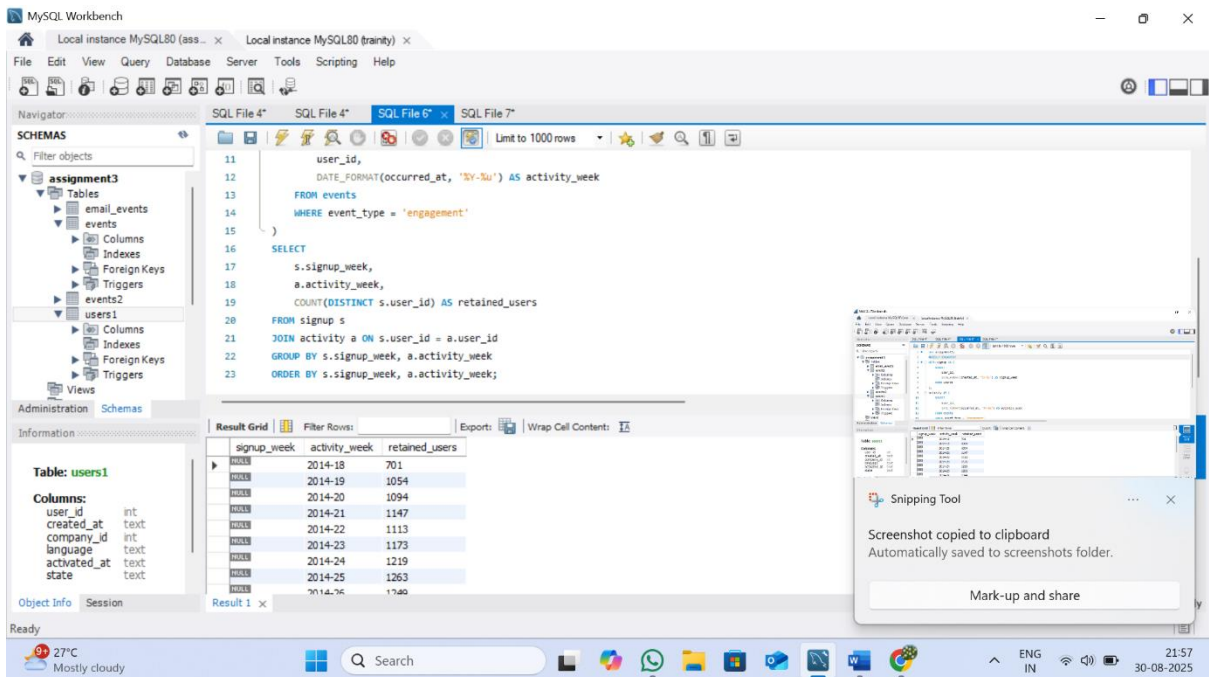
The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
1 use assignment3;
2 WEEKLY ENGAGEMENT
3 WITH signup AS (
4     SELECT
5         user_id,
6         DATE_FORMAT(created_at, '%Y-%w') AS signup_week
7     FROM users1
8 ),
9 activity AS (
10    SELECT
11        user_id,
12        DATE_FORMAT(occurred_at, '%Y-%w') AS activity_week
13    FROM events
14    WHERE event_type = 'engagement'
```

The result grid displays the following data:

signup_week	activity_week	retained_users
2014-18	2014-18	701
2014-19	2014-19	1054
2014-20	2014-20	1094
2014-21	2014-21	1147
2014-22	2014-22	1113
2014-23	2014-23	1173
2014-24	2014-24	1219
2014-25	2014-25	1263
2014-26	2014-26	1280

The left sidebar shows the database schema for 'assignment3', including tables like 'email_events', 'events', 'events2', and 'users1'. The 'users1' table structure is detailed in the 'Information' tab.



The screenshot shows the MySQL Workbench interface with the completed query and result grid. The SQL editor contains the following query:

```
11 user_id,
12 DATE_FORMAT(occurred_at, '%Y-%w') AS activity_week
13 FROM events
14 WHERE event_type = 'engagement'
15 )
16 SELECT
17     s.signup_week,
18     a.activity_week,
19     COUNT(DISTINCT s.user_id) AS retained_users
20 FROM signup s
21 JOIN activity a ON s.user_id = a.user_id
22 GROUP BY s.signup_week, a.activity_week
23 ORDER BY s.signup_week, a.activity_week;
```

The result grid displays the same data as the first screenshot:

signup_week	activity_week	retained_users
2014-18	2014-18	701
2014-19	2014-19	1054
2014-20	2014-20	1094
2014-21	2014-21	1147
2014-22	2014-22	1113
2014-23	2014-23	1173
2014-24	2014-24	1219
2014-25	2014-25	1263
2014-26	2014-26	1280

A 'Snipping Tool' window is overlaid on the right side of the screen, displaying the message: 'Screenshot copied to clipboard. Automatically saved to screenshots folder. Mark-up and share'.

4. Write an SQL query to calculate the email engagement metrics.

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```

1 use assignment3;
2 #EMAIL ENGAGEMENT
3 SELECT
4     DATE_FORMAT(occurred_at, '%Y-%u') AS weeks,
5     action,
6     COUNT(DISTINCT user_id) AS engaged_users
7 FROM email_events
8 GROUP BY DATE_FORMAT(occurred_at, '%Y-%u'), action
9 ORDER BY weeks, action;

```

The result grid shows the following data:

weeks	action	engaged_users
2015-01	email_clickthrough	1829
2015-01	email_open	2133
2015-01	sent_weekly_digest	2140

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

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```

1 use assignment3;
2 # USERS AFTER SIGNING UP FOR PRODUCT
3 SELECT
4     date(activated_at) as dates,
5     e.device,
6     COUNT(DISTINCT u.user_id) AS new_users,
7     SUM(COUNT(DISTINCT u.user_id)) OVER (PARTITION BY e.device ORDER BY date(activated_at)) AS cumulative_users
8 FROM users1 u
9 JOIN events e ON u.user_id = e.user_id
10 GROUP BY date(activated_at), e.device
11 ORDER BY date(activated_at), e.device;

```

The result grid shows the following data:

dates	device	new_users	cumulative_users
2015-01-01	acer aspire desktop	198	198
2015-01-01	acer aspire notebook	338	338
2015-01-01	amazon fire phone	89	89
2015-01-01	asus chromebook	355	355
2015-01-01	dell inspiron desktop	360	360
2015-01-01	dell inspiron notebook	677	677
2015-01-01	hp pavilion desktop	339	339
2015-01-01	htc one	196	196
2015-01-01	motorola	479	479

