OPERATION ANALYTICS

BY NAZIREEN SANIA

PROJECT 3

CASE STUDY :

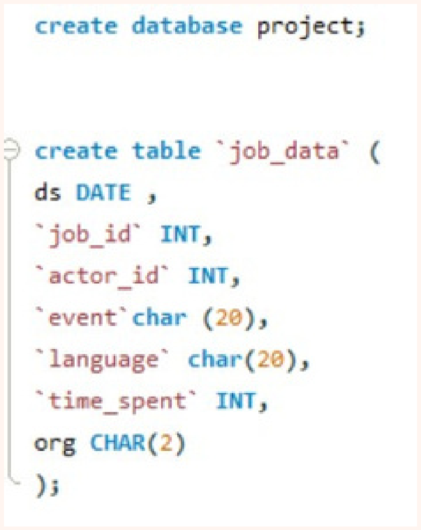
JOB DATA ANALYSIS

For the first section of the project, we work on 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

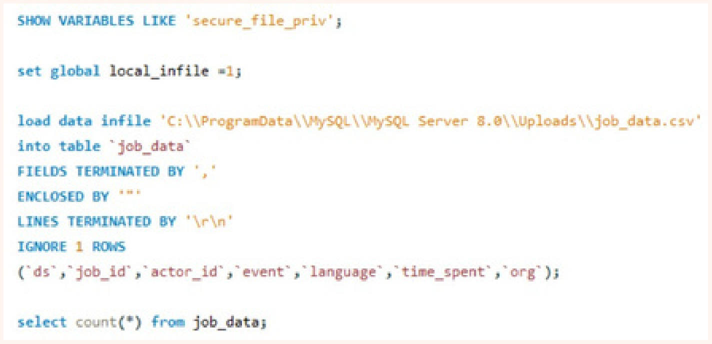
METHODOLOGY

We first load the table into MySQL. First we create a database named ‘project’ and create the table ‘job_data’ under it with the following columns.



```sql
create database project;


create table `job_data` (
ds DATE ,
`job_id` INT,
`actor_id` INT,
`event`char (20),
`language` char(20),
`time_spent` INT,
org CHAR(2)
);
```

We then determine the file path to store our excel file using the ‘secure_file_priv’ function. Before we load the file, we check for any missing values using the ‘Select & Find’ option. We do this instead of Table Data Import Wizard option because the values were not getting loaded completely on using that option.After determining the file path, we load the data using the ‘load data infile’ command.

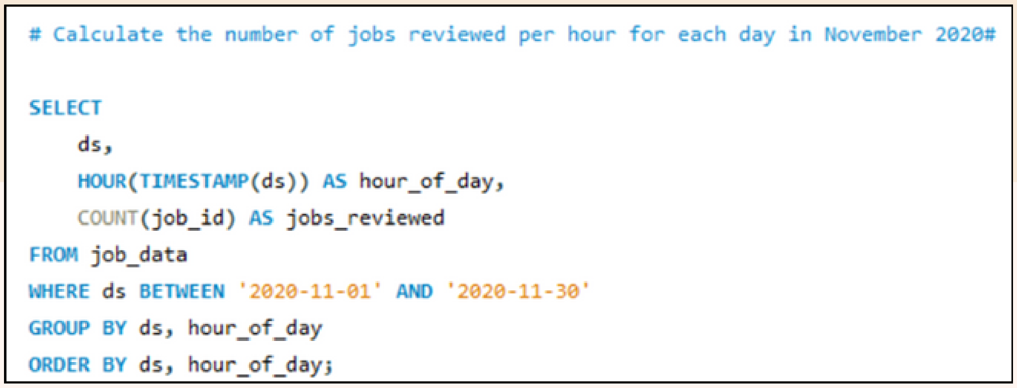


```sql
SHOW VARIABLES LIKE 'secure_file_priv';

set global local_infile =1;

load data infile 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\job_data.csv'
into table `job_data`
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS
(`ds`,`job_id`,`actor_id`,`event`,`language`,`time_spent`,`org`);

select count(*) from job_data;
```

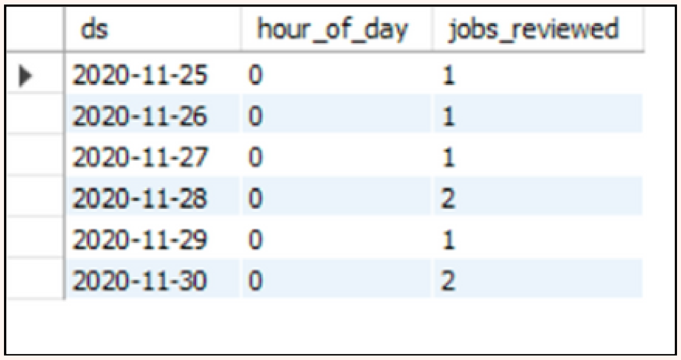
Once we have ensured that all entries are loaded, we begin with the tasks.

(A) Jobs Reviewed Over Time:

Objective: Calculate the number of jobs reviewed per hour for each day in November 2020.



```
# Calculate the number of jobs reviewed per hour for each day in November 2020#

SELECT
    ds,
    HOUR(TIMESTAMP(ds)) AS hour_of_day,
    COUNT(job_id) AS jobs_reviewed
FROM job_data
WHERE ds BETWEEN '2020-11-01' AND '2020-11-30'
GROUP BY ds, hour_of_day
ORDER BY ds, hour_of_day;
```

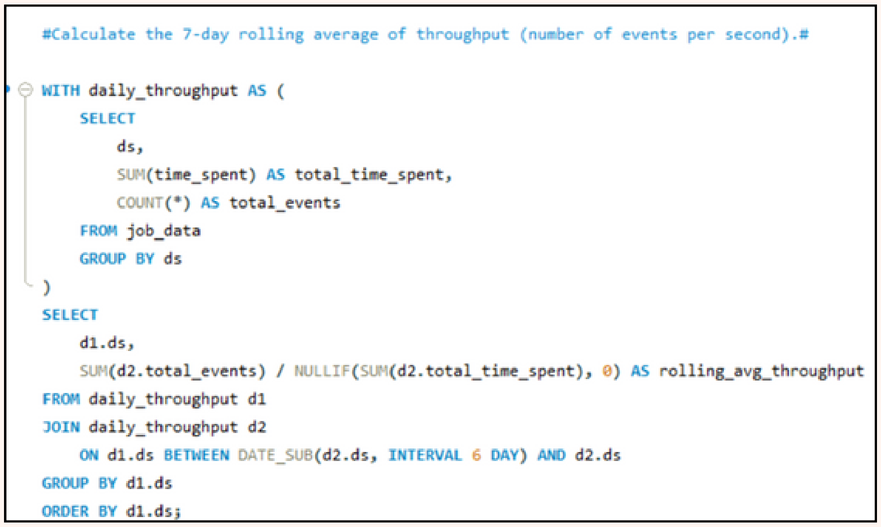


ds	hour_of_day	jobs_reviewed
2020-11-25	0	1
2020-11-26	0	1
2020-11-27	0	1
2020-11-28	0	2
2020-11-29	0	1
2020-11-30	0	2

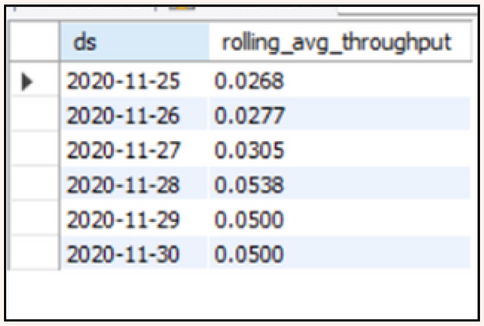
- The WHERE ds BETWEEN '2020-11-01' AND '2020-11-30' clause ensures that we only consider jobs reviewed in November 2020.
- HOUR(TIMESTAMP(ds)) extracts the hour from the ds column (assuming ds contains timestamps; otherwise, we'd need a separate time column).
- COUNT(job_id) counts the number of job reviews per hour.
- GROUP BY ds, hour_of_day groups the records by date and hour.
- ORDER BY ds, hour_of_day sorts the output to maintain chronological order.

(B) Throughput Analysis

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



```sql
#Calculate the 7-day rolling average of throughput (number of events per second).#

WITH daily_throughput AS (
    SELECT
        ds,
        SUM(time_spent) AS total_time_spent,
        COUNT(*) AS total_events
    FROM job_data
    GROUP BY ds
)
SELECT
    d1.ds,
    SUM(d2.total_events) / NULLIF(SUM(d2.total_time_spent), 0) AS rolling_avg_throughput
FROM daily_throughput d1
JOIN daily_throughput d2
    ON d1.ds BETWEEN DATE_SUB(d2.ds, INTERVAL 6 DAY) AND d2.ds
GROUP BY d1.ds
ORDER BY d1.ds;
```



ds	rolling_avg_throughput
2020-11-25	0.0268
2020-11-26	0.0277
2020-11-27	0.0305
2020-11-28	0.0538
2020-11-29	0.0500
2020-11-30	0.0500

- The daily_throughput Common Table Expression (CTE) calculates:

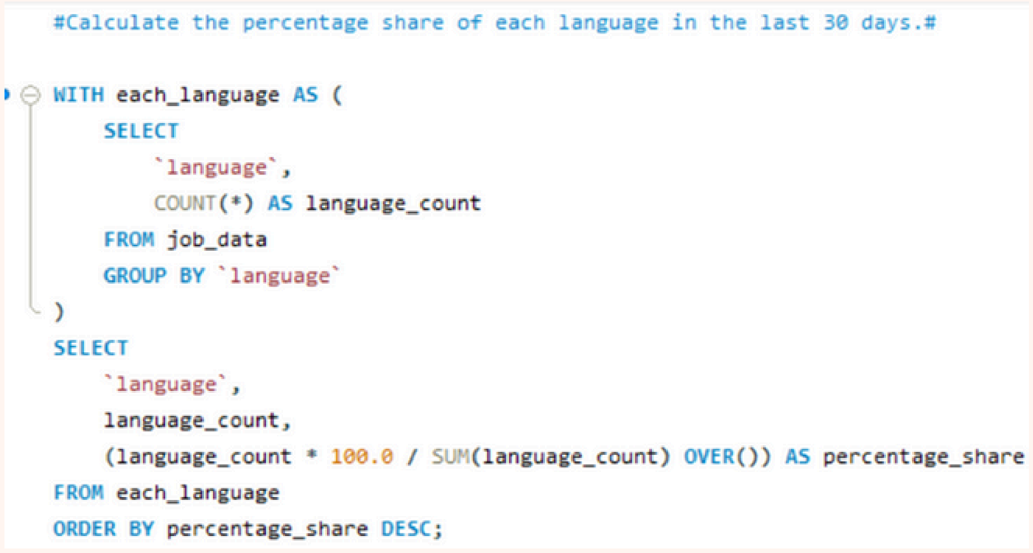
 - SUM(time_spent) AS total_time_spent → Total time spent by users on job reviews for each day.

 - COUNT(*) AS total_events → Total number of job events (decision, skip, transfer) for each day.

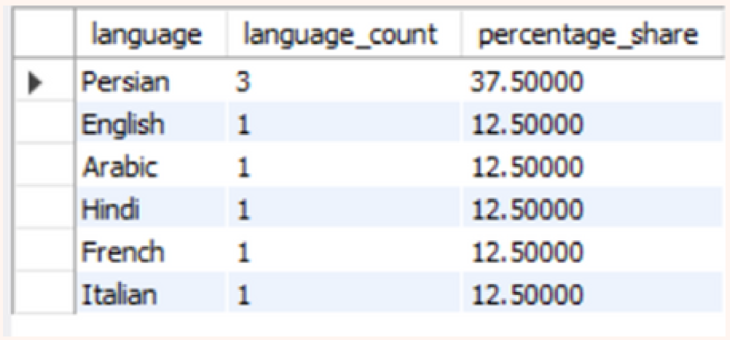
- The main query joins daily_throughput with itself to get a 7-day window for each date (ds).
- The ON d1.ds BETWEEN DATE_SUB(d2.ds, INTERVAL 6 DAY) AND d2.ds condition ensures that for each date d1.ds, we take data from the past 7 days.
-
- SUM(d2.total_events) / NULLIF(SUM(d2.total_time_spent), 0) computes throughput as events per second while avoiding division by zero.
- ORDER BY d1.ds ensures chronological order.

© Language Share Analysis:

Objective: Calculate the percentage share of each language in the last 30 days.



```
#Calculate the percentage share of each language in the last 30 days.#

WITH each_language AS (
    SELECT
        `language`,
        COUNT(*) AS language_count
    FROM job_data
    GROUP BY `language`
)
SELECT
    `language`,
    language_count,
    (language_count * 100.0 / SUM(language_count) OVER()) AS percentage_share
FROM each_language
ORDER BY percentage_share DESC;
```



language	language_count	percentage_share
Persian	3	37.50000
English	1	12.50000
Arabic	1	12.50000
Hindi	1	12.50000
French	1	12.50000
Italian	1	12.50000

- each_language CTE groups jobs by language and counts them.
- Calculating Percentage Share

- language & language_count Selection : This simply selects language and language_count from the each_language CTE.

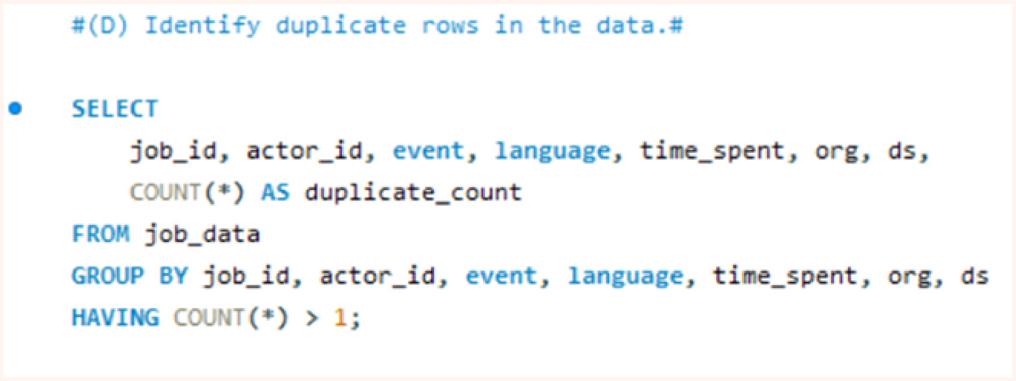
- SUM(language_count) OVER():

- This computes the total number of job records across all languages.
- The window function OVER() ensures that this sum is available in every row.

- (language_count * 100.0 / SUM(language_count) OVER()) AS percentage_share - This calculates how much each language contributes as a percentage of all job records.

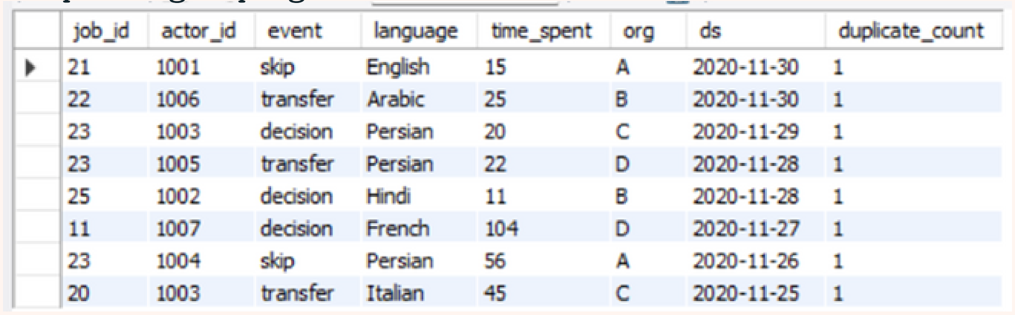
(D) Duplicate Rows Detection:

Objective: Identify duplicate rows in the data.



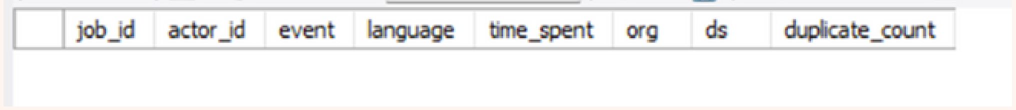
```
#(D) Identify duplicate rows in the data.#

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 for grouping of rows



job_id	actor_id	event	language	time_spent	org	ds	duplicate_count
21	1001	skip	English	15	A	2020-11-30	1
22	1006	transfer	Arabic	25	B	2020-11-30	1
23	1003	decision	Persian	20	C	2020-11-29	1
23	1005	transfer	Persian	22	D	2020-11-28	1
25	1002	decision	Hindi	11	B	2020-11-28	1
11	1007	decision	French	104	D	2020-11-27	1
23	1004	skip	Persian	56	A	2020-11-26	1
20	1003	transfer	Italian	45	C	2020-11-25	1

Output for checking if there are any duplicate rows



job_id	actor_id	event	language	time_spent	org	ds	duplicate_count

- Grouping by All Columns:

The GROUP BY job_id, actor_id, event, language, time_spent, org, ds groups records that have identical values across all columns.

- Counting Duplicates:

COUNT(*) AS duplicate_count counts occurrences of each unique row.

- Filtering Duplicates:

HAVING COUNT(*) > 1 ensures only duplicate rows (i.e., those appearing more than once) are displayed.

- Since no rows are displayed in the output for HAVING COUNT(*) > 1, we can conclude that there are no duplicate rows.