

Project Report: Job Data Analysis

1. Project Description

The goal of this project is to analyze job review data, focusing on metrics such as jobs reviewed over time, throughput, language share, and detection of duplicate records. This analysis aims to uncover patterns in job processing, improve organizational efficiency, and provide data-driven insights for decision-making. To handle the analysis, I structured and imported the dataset into a MySQL database, enabling effective querying and data manipulation for each analytical task.

2. Approach

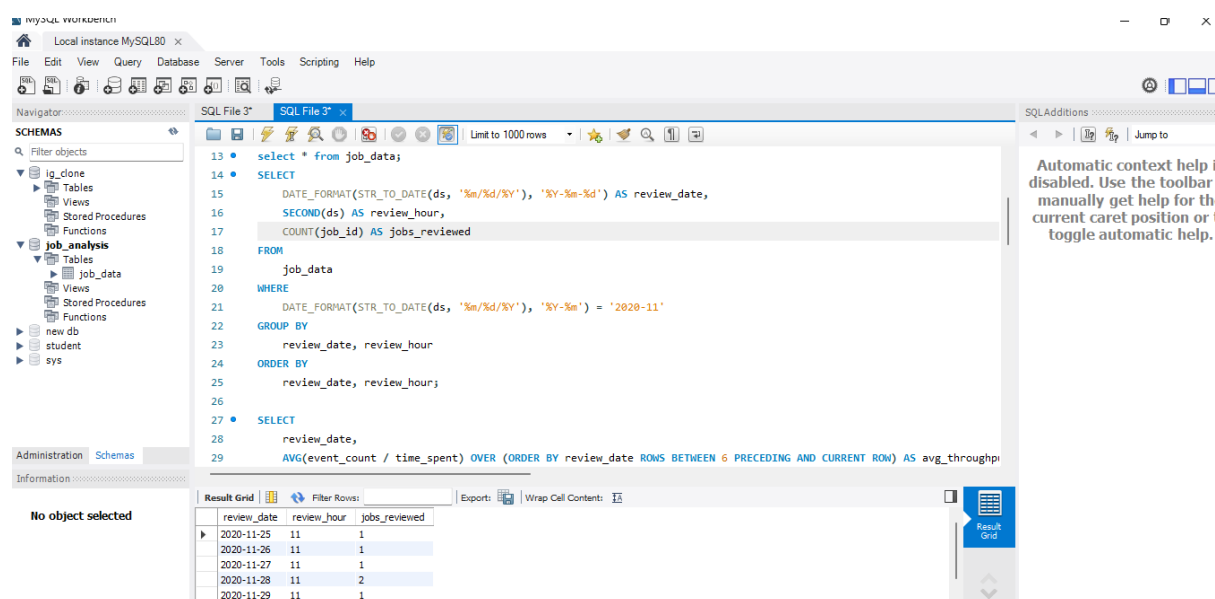
The approach involved several key steps:

- **Data Preparation:** The CSV file was imported into MySQL Workbench, creating a structured `job_data` table to facilitate SQL queries.
- **Query Execution:** For each case study question, I wrote SQL queries to retrieve the required metrics. Each query focused on extracting and summarizing data from the `job_data` table to meet specific objectives.
- **Analysis:** I calculated various metrics, including jobs reviewed per hour, a 7-day rolling average for throughput, language share percentages, and identified duplicate records. This approach enabled thorough examination and provided a clear view of the dataset's behavior.

3. Tech-Stack Used

- **MySQL Workbench:** Used as the primary tool for database management, data import, and SQL query execution.

Task 1: Jobs Reviewed Over Time



The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' panel with a tree view containing 'job_analysis' and 'job_data'. The main editor window shows a SQL query file with the following code:

```
13 select * from job_data;
14 SELECT
15     DATE_FORMAT(STR_TO_DATE(ds, '%m/%d/%Y'), '%Y-%m-%d') AS review_date,
16     SECOND(ds) AS review_hour,
17     COUNT(job_id) AS jobs_reviewed
18 FROM
19     job_data
20 WHERE
21     DATE_FORMAT(STR_TO_DATE(ds, '%m/%d/%Y'), '%Y-%m') = '2020-11'
22 GROUP BY
23     review_date, review_hour
24 ORDER BY
25     review_date, review_hour;
26
27 SELECT
28     review_date,
29     AVG(event_count / time_spent) OVER (ORDER BY review_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS avg_throughput;
```

The 'Result Grid' at the bottom shows the output of the first query, displaying columns 'review_date', 'review_hour', and 'jobs_reviewed'.

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

```

13 • select * from job_data;
14 • SELECT
15     DATE_FORMAT(STR_TO_DATE(ds, '%m/%d/%Y'), '%Y-%m-%d') AS review_date,
16     SECOND(ds) AS review_hour,
17     COUNT(job_id) AS jobs_reviewed
18 FROM
19     job_data
20 WHERE
21     DATE_FORMAT(STR_TO_DATE(ds, '%m/%d/%Y'), '%Y-%m-%d') = '2020-11'
22 GROUP BY
23     review_date, review_hour
24 ORDER BY
25     review_date, review_hour;

```

Insights

- Jobs Reviewed Over Time:

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

Task 2: Throughput Analysis

MySQL Workbench interface showing a query for throughput analysis. The query calculates the average throughput (event_count / time_spent) over a 6-row window, grouped by review_date and review_hour.

Query:

```

27 • SELECT
28     review_date,
29     AVG(event_count / time_spent) OVER (ORDER BY review_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS avg_throughput
30 FROM (
31     SELECT
32         DATE_FORMAT(STR_TO_DATE(ds, '%m/%d/%Y'), '%Y-%m-%d') AS review_date,
33         COUNT(event) AS event_count,
34         SUM(time_spent) AS time_spent
35     FROM
36         job_data
37     GROUP BY
38         review_date
39 ) AS daily_stats
40 ORDER BY
41     review_date;

```

Result Grid:

review_date	avg_throughput
2020-11-25	0.02220000
2020-11-26	0.02005000
2020-11-27	0.01656667
2020-11-28	0.02757500
2020-11-29	0.03206000

```

27 • SELECT
28     review_date,
29     AVG(event_count / time_spent) OVER (ORDER BY review_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS avg_throughput
30 FROM (
31     SELECT
32         DATE_FORMAT(STR_TO_DATE(ds, '%m/%d/%Y'), '%Y-%m-%d') AS review_date,
33         COUNT(event) AS event_count,
34         SUM(time_spent) AS time_spent
35     FROM
36         job_data
37     GROUP BY
38         review_date
39 ) AS daily_stats
40 ORDER BY
41     review_date;

```

SELECT

review_date,

AVG(event_count / time_spent) OVER (ORDER BY review_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS avg_throughput

FROM (

SELECT

DATE_FORMAT(STR_TO_DATE(ds, '%m/%d/%Y'), '%Y-%m-%d') AS review_date,

COUNT(event) AS event_count,

SUM(time_spent) AS time_spent

FROM

job_data

GROUP BY

review_date

) AS daily_stats

ORDER BY

review_date;

Insights

- **Throughput Analysis:**

	review_date	avg_throughput
	2020-11-25	0.02220000
	2020-11-26	0.02005000
	2020-11-27	0.01656667
	2020-11-28	0.02757500
	2020-11-29	0.03206000
	2020-11-30	0.03505000

Task 3: Language Share Analysis

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows a tree view with 'job_analysis' selected. The main editor displays a SQL query (lines 57-69) that calculates the average throughput for each language over the last 1600 days. The query uses a subquery to calculate the average throughput and then joins it with the 'job_data' table to get the language share. The results grid at the bottom shows the output of the query, with columns 'language' and 'language_share'.

language	language_share
Persian	37.50
English	12.50
Arabic	12.50
Hindi	12.50
French	12.50

```

59 • SELECT
60     language,
61     ROUND((COUNT(*) / (SELECT
62         COUNT(*)
63         FROM
64             job_data
65         WHERE
66             STR_TO_DATE(ds, '%m/%d/%Y') >= DATE_SUB(CURDATE(), INTERVAL 1600 DAY))) * 100,
67     2) AS language_share
68 FROM
69     job_data
70 WHERE
71     STR_TO_DATE(ds, '%m/%d/%Y') >= DATE_SUB(CURDATE(), INTERVAL 1600 DAY)
72 GROUP BY language
73 ORDER BY language_share DESC;
74

```

Insights

Language Share:

	language	language_share
▶	Persian	37.50
	English	12.50
	Arabic	12.50
	Hindi	12.50
	French	12.50
	Italian	12.50

Task 4: Duplicate Rows Detection

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows the 'job_analysis' database selected. The central SQL editor contains the following query:

```

75
76 • SELECT
77     ds,
78     job_id,
79     actor_id,
80     event,
81     language,
82     time_spent,
83     org,
84     COUNT(*) AS duplicate_count
85 FROM
86     job_data
87 GROUP BY ds , job_id , actor_id , event , language , time_spent , org
88 HAVING duplicate_count > 1;
89
90
91

```

Below the query editor, the 'Result Grid' tab is active, showing the following columns: ds, job_id, actor_id, event, language, time_spent, org, duplicate_count. The status bar indicates 'No object selected'.

```

--
76 • SELECT
77     ds,
78     job_id,
79     actor_id,
80     event,
81     language,
82     time_spent,
83     org,
84     COUNT(*) AS duplicate_count
85 FROM
86     job_data
87 GROUP BY ds , job_id , actor_id , event , language , time_spent , org
88 HAVING duplicate_count > 1;
89

```

Insights:

Duplicate Detection

	ds	job_id	actor_id	event	language	time_spent	org	duplicate_count
--	----	--------	----------	-------	----------	------------	-----	-----------------

Insights

The analysis produced valuable insights:

- **Jobs Reviewed Over Time:** The hourly distribution of reviewed jobs provided visibility into peak and low activity periods.
- **Throughput Analysis:** The 7-day rolling average highlighted trends in job processing rates, allowing for the identification of consistent workflow patterns.
- **Language Share:** The percentage distribution of languages used in job reviews helped determine language popularity and could inform resource allocation for multilingual support.
- **Duplicate Detection:** No duplicates identified.