Sql  Query :-

```
create database job_data;
use job_data;
create table `Job data`(
   job_id bigint ,
   ds varchar(30),          actor_id int,
            event varchar (30),
            language varchar(50),
            time_spent varchar(30),
            org varchar(10))
;

INSERT INTO `Job data` (
  job_id, ds,actor_id, event, language, time_spent_hours, org
)
VALUES
  ('21', '30/11/2020', '1001', 'skip', 'English', '15', 'A'),
  ('22', '30/11/2020', '1006', 'transfer', 'Arabic', '25', 'B'),
  ('23', '29/11/2020', '1003', 'decision', 'Persian', '20', 'C'),
  ('23', '28/11/2020', '1005', 'transfer', 'Persian', '22', 'D'),
  ('25', '28/11/2020', '1002', 'decision', 'Hindi', '11', 'B'),
  ('11', '27/11/2020', '1007', 'decision', 'French', '104', 'D'),
  ('23', '26/11/2020', '1004', 'skip', 'Persian', '56', 'A'),
  ('20', '25/11/2020', '1003', 'transfer', 'Italian', '45', 'C');
```

## ==The above sql command will form table (Job data) in database job_data==========

## 1. Jobs Reviewed Over Time

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

## Sql Query:

```
SELECT
   ds,
   FLOOR (time_spent_hours) AS review_hour,
   COUNT(*) AS jobs_reviewed
FROM
   `Job data`
WHERE
   ds BETWEEN '2020/11/01' AND '2020/11/30'
GROUP BY
   ds,  review_hour
ORDER BY
   ds,  review_hour;
```

## Output:-

| Ds | 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 |

As time spent in reviewing is in seconds above answer gives time spent in hours as zero

2. **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.

## Sql Query:-

```
  WITH daily_throughput AS (
  SELECT
    ds,
    COUNT(*) AS total_events
  FROM `Job data`
  GROUP BY ds
),
rolling_average AS (
  SELECT
    dt.ds,
    AVG(dt.total_events * 1.0 / 86400) OVER (
      ORDER BY dt.ds
      ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
    ) AS rolling_avg_throughput
  FROM daily_throughput dt
)
SELECT
  ds,
  rolling_avg_throughput
FROM rolling_average
ORDER BY ds;
```

## Output:-

| ds | Rolling_avg_throughput |
|---|---|
| 2020/11/25 | 0.000010000 |
| 2020/11/26 | 0.000010000 |
| 2020/11/27 | 0.000010000 |
| 2020/11/28 | 0.000012500 |
| 2020/11/29 | 0.000012000 |
| 2020/11/30 | 0.000013333 |

Step 1: Calculate Daily Throughput

- In the daily_throughput Common Table Expression (CTE), the query groups data by the date (ds) and counts the total number of events (COUNT(*)) for each day.

Step 2: Calculate the Rolling Average

- The rolling_average CTE calculates the 7-day rolling average using the AVG() window function.
- The AVG() function includes the current day and the preceding 6 days (ROWS BETWEEN 6 PRECEDING AND CURRENT ROW).
- The daily throughput is converted to events per second by dividing the event count by 86,400 (number of seconds in a day).

Step 3: Final Output

- The final output includes the date (ds) and the 7-day rolling average throughput (rolling_avg_throughput).

## Daily Metric vs. 7-Day Rolling Average for Throughput

## Daily Metric:

- **Pros:**
    - **Provides granular, day-to-day insights into throughput.**

    - **Useful for identifying sudden spikes or drops in throughput on specific dates.**

- **Cons:**
    - **Highly volatile and may be affected by daily variations, holidays, or special events.**
    - **Not ideal for understanding broader trends or consistent performance.**

**7-Day Rolling Average:**

- **Pros:**
    - **Smoothens out short-term fluctuations, offering a more stable view of trends.**
    - **Helps in identifying long-term patterns and performance consistency.**
    - **Reduces the impact of outliers or one-off anomalies.**

- **Cons:**
    - **Less effective in identifying abrupt changes on a specific day.**
    - **Requires more data points and computational resources.**

---

**Preference:**

**I prefer the 7-day rolling average for throughput analysis as it offers a clearer picture of overall trends, reduces noise from daily variations, and is more reliable for making informed decisions about performance over time. While daily metrics are helpful for detailed monitoring, the rolling average is better for strategic analysis and planning.**

3. **Language Share Analysis:**
    a. Objective: Calculate the percentage share of each language in the last 30 days.
    b. Your Task: Write an SQL query to calculate the percentage share of each language over the last 30 days.

```
  WITH last_30_days AS (
   SELECT
      language,
      COUNT(*) AS language_count
   FROM `Job Data`
   WHERE ds BETWEEN STR_TO_DATE('2020-11-25', '%Y-%m-%d')
                          AND STR_TO_DATE('2020-11-30', '%Y-%m-%d')
   GROUP BY language
),
total_events AS (
   SELECT
      SUM(language_count) AS total_count
   FROM last_30_days
)
SELECT
   l30.language,
   l30.language_count,
   ROUND((l30.language_count * 1.0 / te.total_count) * 100, 2) AS percentage_share
FROM last_30_days l30
CROSS JOIN total_events te
ORDER BY percentage_share DESC;
```

**OUTPUT:-**

| Language | Language_count | Percentage_share |
|---|---|---|
| Persian | 3 | 37.50 |
| Italian | 1 | 12.50 |
| Hindi | 1 | 12.50 |
| French | 1 | 12.50 |
| English | 1 | 12.50 |
| Arabic | 1 | 12.50 |

4. **Duplicate Rows Detection:**
    a. Objective: Identify duplicate rows in the data.
    b. Your Task: Write an SQL query to display duplicate rows from the job_data table.

Sql Query;-

```
SELECT
   job_id,
   actor_id,
   event,
   language,
   time_spent_hours,
   org,
   ds,
   COUNT(*) AS duplicate_count
FROM `Job data`
GROUP BY
   job_id,
   actor_id,
```

```
    event,
    language,
    time_spent_hours,
    org,
    ds
HAVING COUNT(*) > 1;
```

**There are no duplicate values in table `Job_data`.**