SQL Queries: Job Data Analysis

Jobs Reviewed Over Time

Query 1: Calculate the number of jobs reviewed per hour for each day in November 2020.

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

Solution: For getting the correct output, we need to **update table job_data column ds.**

```
-- update ds column from date to datetime
ALTER TABLE job_data MODIFY COLUMN ds DATETIME;

-- Update ds with Random Time Values
SET SQL_SAFE_UPDATES = 0;

UPDATE job_data
SET ds = ds
        + INTERVAL FLOOR(RAND() * 24) HOUR
        + INTERVAL FLOOR(RAND() * 60) MINUTE
        + INTERVAL FLOOR(RAND() * 60) SECOND;

SET SQL_SAFE_UPDATES = 1;  -- Turn safe mode back on
```

Query:

```sql
SELECT
    DATE(ds) AS review_date,       -- Extract only the DATE part from ds
    HOUR(ds) AS review_hour,       -- Extract the HOUR part from ds
    COUNT(job_id) AS jobs_reviewed -- Count the number of jobs reviewed
FROM job_data
WHERE ds BETWEEN '2020-11-01' AND '2020-11-30' -- Filter only November 2020
GROUP BY review_date, review_hour              -- Group by Date and Hour
ORDER BY review_date, review_hour;             -- Sort results by Date and Hour
```

Output:

review_date	review_hour	jobs_reviewed
2020-11-07	2	1
2020-11-07	8	1
2020-11-12	2	1
2020-11-17	0	1
2020-11-17	11	1
2020-11-19	19	1
2020-11-19	20	1
2020-11-19	21	1
2020-11-20	19	1
2020-11-22	15	1
2020-11-25	4	1
2020-11-25	5	1
2020-11-26	5	1
2020-11-27	19	1
2020-11-28	2	1
2020-11-28	21	1

Throughput Analysis

Query 2: Calculate the 7-day rolling average of throughput (number of events per second).

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.

Query:

```sql
WITH event_counts AS (
    SELECT
        ds,
        COUNT(*) AS total_events,                -- Count of events per day
        SUM(time_spent) AS total_time_spent      -- Sum of time spent on jobs per day
    FROM job_data
    GROUP BY ds
),
throughput_per_day AS (
    SELECT
        ds,
        total_events / NULLIF(total_time_spent, 0) AS throughput_per_second  -- Calculate throughput
    FROM event_counts
),
rolling_avg AS (
    SELECT
        ds,
        throughput_per_second,
        AVG(throughput_per_second) OVER (
            ORDER BY ds ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
        ) AS rolling_7_day_avg
    FROM throughput_per_day
)
SELECT * FROM rolling_avg;
```

Daily Metric vs. 7-Day Rolling Average:

Metric	Pros	Cons
Daily Throughput	Captures daily trends accurately. Shows immediate changes.	Very volatile (fluctuates daily).
7-Day Rolling Avg	Smoothens fluctuations. Helps identify long-term trends.	Delays reaction to sudden changes.

Preferred Approach:

A 7-day rolling average is better for trend analysis because it removes daily noise and helps see patterns more clearly. If immediate performance tracking is required, the daily metric can be used alongside it.

Output:

ds	throughput_per_second	rolling_7_day_avg
2019-01-01 23:24:06	0.0476	0.04760000
2019-01-02 16:56:36	0.0400	0.04380000
2019-01-08 19:54:09	0.0909	0.05950000
2019-01-08 23:42:32	0.0417	0.05505000
2019-01-09 01:43:25	0.0370	0.05144000
2019-01-13 20:02:42	0.0435	0.05011667
2019-01-14 02:26:52	0.0345	0.04788571
2019-01-14 21:46:09	0.0400	0.04680000
2019-01-17 04:52:46	0.0400	0.04680000
2019-01-17 05:21:07	0.0833	0.04571429
2019-01-18 05:20:02	0.0455	0.04625714
2019-01-19 07:48:07	0.0417	0.04692857
2019-01-22 08:04:16	0.0909	0.05370000
2019-01-22 23:40:22	0.0909	0.06175714
2019-01-24 04:57:15	0.0714	0.06624286
2019-01-24 08:53:22	0.0500	0.06767143
2019-01-26 10:22:34	0.0833	0.06767143
2019-01-28 08:36:58	0.0588	0.06957143
2019-01-28 11:12:33	0.0588	0.07201429
2019-01-29 17:20:32	0.0769	0.07001429
2019-01-30 05:35:17	0.0909	0.07001429
2019-02-01 14:02:26	0.0333	0.06457143
2019-02-01 17:07:25	0.0435	0.06364286
2019-02-03 09:34:38	0.0769	0.06272857
2019-02-04 21:55:52	0.0455	0.06082857
2019-02-06 02:13:49	0.0357	0.05752857
2019-02-08 22:31:46	0.0417	0.05250000
2019-02-09 17:11:44	0.0333	0.04427143
2019-02-13 14:37:21	0.0667	0.04904286

Language Share Analysis

Query 3: Calculate the percentage share of each language in the last 30 days.

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

Query:

```sql
WITH language_counts AS (
    SELECT
        language,
        COUNT(*) AS language_count  -- Count jobs per language
    FROM job_data
    WHERE ds >= DATE_SUB(CURDATE(), INTERVAL 30 DAY)  -- Last 30 days
    GROUP BY language
),
total_jobs AS (
    SELECT SUM(language_count) AS total_jobs  -- Total jobs in last 30 days
    FROM language_counts
)
SELECT
    lc.language,
    lc.language_count,
    (lc.language_count / tj.total_jobs) * 100 AS percentage_share  -- Percentage calculation
FROM language_counts lc
JOIN total_jobs tj;
```

Output:

language	language_count	percentage_share
Arabic	5	21.7391
Persian	4	17.3913
Hindi	10	43.4783
English	4	17.3913

Duplicate Rows Detection

Query 4: Identify duplicate rows in the data.

Task: Write an SQL query to display duplicate rows from the job_data table.

Query:

```sql
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  -- Only show duplicates
ORDER BY duplicate_count DESC;
```

Output:

	job_id	actor_id	event	language	time_spent	org	ds	duplicate_count

There's nothing shown in the output because there's no duplicates in the dataset, as **we have taken job_id as the primary key**.

```sql
-- Step 1: Create the database (if not exists)
CREATE DATABASE IF NOT EXISTS job_DA;
USE job_DA;

-- Step 2: Create the table
CREATE TABLE IF NOT EXISTS job_data (
    job_id INT PRIMARY KEY,
    actor_id INT,
    event ENUM('decision', 'skip', 'transfer'),
    language VARCHAR(50),
    time_spent INT,
    org VARCHAR(50),
    ds DATE
);
```