2024-09-30

DATABASES 2

LAB 2 – ADVANCED QUERIES

Paulina Czarnota C21365726

PART 1: PARIS 2024 ATHLETICS RESULTS

INTRODUCTION

This report analyzes the dataset from the Paris 2024 Athletics Results, utilizing advanced SQL queries and window functions in PostgreSQL to derive insights from the competition results. The dataset contains information on various athletic events, including details about participants, results, and performance metrics.

PROBLEM DESCRIPTION

The file “Paris 2024 Result.csv” contains the results of all finals from the recent Olympic Games in Paris. The columns in the CSV include:

- **Date:** The date of the final in the format “2024-08-07”.
- **Event Name:** The name of the event (e.g., “High Jump”).
- **Gender:** The gender of the athlete or team (X denotes a mixed team competition).
- **Participant Name:** The name of the athlete.
- **Participant Type:** Indicates whether the entry is a “Person” or a “Team”.
- **Participant Country Code:** A unique code for the country (e.g., “IRL” for Ireland).
- **Participant Country:** The name of the country (e.g., “Ireland”).
- **Result:** The result achieved by the athlete, represented as a decimal number. For timed events, it is expressed in seconds; for distance events, it is expressed in meters.
- **Result Type:** Indicates whether the event is “Distance” or “Time”.

POSTGRESQL IMPLEMENTATION

Step 0: Clean Up Existing Tables and Views

```
DROP VIEW IF EXISTS medal_table CASCADE;

DROP VIEW IF EXISTS event_ranking CASCADE;

DROP TABLE IF EXISTS paris_2024_results CASCADE;
```

EXPLANATION

The initial commands drop any existing views or tables to prevent conflicts when creating new structures. Using `CASCADE` ensures that all dependencies are removed as well.

Step 1: Create the `paris_2024_results` Table

```
CREATE TABLE paris_2024_results (
    date DATE,
    event_name VARCHAR(255),
    gender CHAR(1),
    participant_name VARCHAR(255),
    participant_type VARCHAR(10),
    participant_country_code CHAR(3),
    participant_country VARCHAR(255),
    result DECIMAL,
    result_type VARCHAR(10)
);
```

EXPLANATION

This command creates the `paris_2024_results` table, defining each column with appropriate data types. The use of `VARCHAR` and `DECIMAL` ensures proper storage of text and numerical values. Notably, the `result` column is defined as `DECIMAL` for precision.

Step 2: Load Data into `paris_2024_results`

```
COPY paris_2024_results
FROM 'C:/Program Files/PostgreSQL/17/Paris_2024_Result.csv'
WITH (
    FORMAT csv,
    HEADER true,
    DELIMITER ',',
    QUOTE '"',
    ESCAPE '"',
    NULL ''
);
```

EXPLANATION

The `COPY` command imports data from the CSV file into the `paris_2024_results` table. The command specifies options to handle CSV formatting, including headers, delimiters, and quoting, ensuring accurate data loading.

Step 3a: Create a View `event_ranking` to Rank Participants

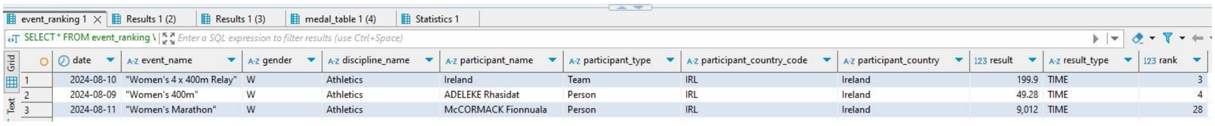
```
CREATE VIEW event_ranking AS
SELECT
  date,
  event_name,
  gender,
  participant_name,
  participant_type,
  participant_country_code,
  participant_country,
  result,
  result_type,
  RANK() OVER (
    PARTITION BY event_name, gender
    ORDER BY
      CASE
        WHEN result_type = 'TIME' THEN result
        WHEN result_type = 'DISTANCE' THEN result * -1
      END ASC
  ) AS rank
FROM
  paris_2024_results;
```

EXPLANATION

This command creates a view named `event_ranking` that ranks participants for each event based on their results. The `RANK()` function is used with `PARTITION BY` to separate rankings for different events and genders. The `ORDER BY` clause specifies the ranking order, differentiating between timed and distance events.

TABLE STRUCTURE

event_ranking:



```
SELECT * FROM event_ranking
```

	date	event_name	gender	discipline_name	participant_name	participant_type	participant_country_code	participant_country	result	result_type	rank
1	2024-08-10	"Women's 4 x 400m Relay"	W	Athletics	Ireland	Team	IRL	Ireland	199.9	TIME	3
2	2024-08-09	"Women's 400m"	W	Athletics	ADELEKE Rhasidat	Person	IRL	Ireland	49.28	TIME	4
3	2024-08-11	"Women's Marathon"	W	Athletics	McCORMACK Fionnuala	Person	IRL	Ireland	9,012	TIME	28

Step 3b: Identify the Best 5 Irish Athletes Based on Their Ranking

```
SELECT *

FROM event_ranking

WHERE participant_country_code = 'IRL'

ORDER BY rank

LIMIT 5;
```

EXPLANATION

This query retrieves information about the top five ranked athletes from Ireland (identified by the country code 'IRL') from the `event_ranking` view. The results are ordered by rank to display the best performers.

Step 3c: Check the Ranking of the Women's Marathon Winner in the Men's Marathon Event

```
WITH womens_marathon_winner AS (

   SELECT result

   FROM event_ranking

   WHERE event_name = 'Women''s Marathon' AND rank = 1

),

mens_marathon_ranks AS (

   SELECT result,

         RANK() OVER (ORDER BY result ASC) AS men_rank

   FROM event_ranking

   WHERE event_name = 'Men''s Marathon'

)

SELECT men_rank

FROM mens_marathon_ranks
```

```
WHERE result >= (SELECT result FROM womens_marathon_winner)
ORDER BY men_rank
LIMIT 1;
```

EXPLANATION

This query uses Common Table Expressions (CTEs) to first identify the winning result of the Women's Marathon. The second CTE ranks the results from the Men's Marathon. The final selection determines the rank position of the Women's Marathon winner if they had competed in the Men's event.

Step 3d: Generate the Medal Table View

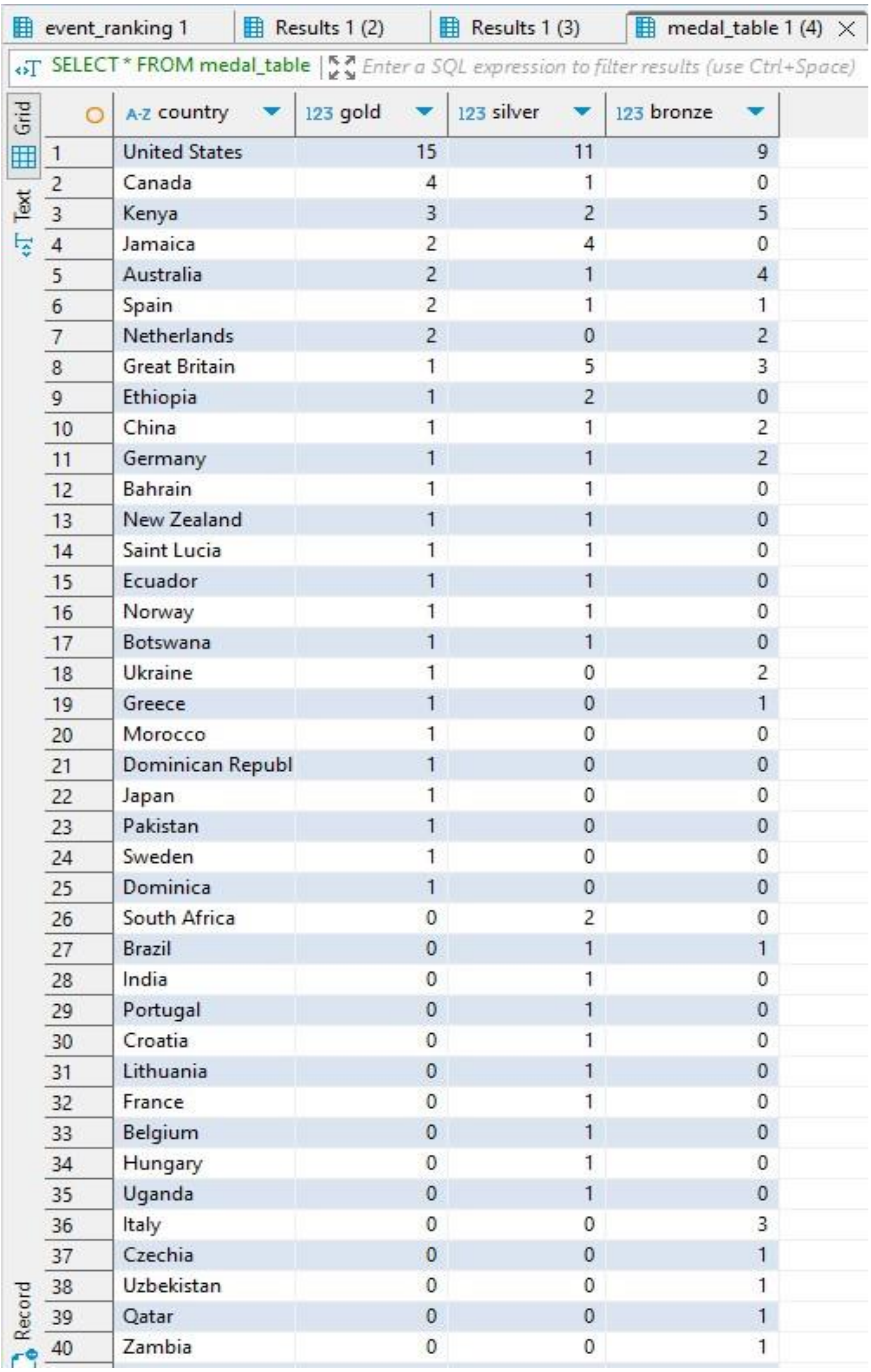
```
CREATE VIEW medal_table AS
SELECT
    participant_country AS Country,
    COUNT(CASE WHEN rank = 1 THEN 1 END) AS Gold,
    COUNT(CASE WHEN rank = 2 THEN 1 END) AS Silver,
    COUNT(CASE WHEN rank = 3 THEN 1 END) AS Bronze
FROM
    event_ranking
GROUP BY
    participant_country
ORDER BY
    Gold DESC, Silver DESC, Bronze DESC;
```

EXPLANATION

This command creates the `medal_table` view, summarizing the total counts of Gold, Silver, and Bronze medals for each country based on their rankings. The aggregation counts the medals by grouping results according to country, with the final output sorted by the number of Gold medals, then Silver and Bronze.

TABLE STRUCTURE

medal_table:



```
SELECT * FROM medal_table
```

	country	gold	silver	bronze
1	United States	15	11	9
2	Canada	4	1	0
3	Kenya	3	2	5
4	Jamaica	2	4	0
5	Australia	2	1	4
6	Spain	2	1	1
7	Netherlands	2	0	2
8	Great Britain	1	5	3
9	Ethiopia	1	2	0
10	China	1	1	2
11	Germany	1	1	2
12	Bahrain	1	1	0
13	New Zealand	1	1	0
14	Saint Lucia	1	1	0
15	Ecuador	1	1	0
16	Norway	1	1	0
17	Botswana	1	1	0
18	Ukraine	1	0	2
19	Greece	1	0	1
20	Morocco	1	0	0
21	Dominican Republ	1	0	0
22	Japan	1	0	0
23	Pakistan	1	0	0
24	Sweden	1	0	0
25	Dominica	1	0	0
26	South Africa	0	2	0
27	Brazil	0	1	1
28	India	0	1	0
29	Portugal	0	1	0
30	Croatia	0	1	0
31	Lithuania	0	1	0
32	France	0	1	0
33	Belgium	0	1	0
34	Hungary	0	1	0
35	Uganda	0	1	0
36	Italy	0	0	3
37	Czechia	0	0	1
38	Uzbekistan	0	0	1
39	Qatar	0	0	1
40	Zambia	0	0	1

Step 3e: Determine if EU Countries Won More Medals Than the USA

```
WITH eu_countries AS (
    SELECT unnest(ARRAY['IRL', 'FRA', 'GER', 'ITA', 'ESP', 'NED', 'SWE', 'POL', 'ROU',
'GRE']) AS country_code
),
eu_medals AS (
    SELECT
        COUNT(*) AS total_medals
    FROM
        event_ranking er
    JOIN
        eu_countries eu ON er.participant_country_code = eu.country_code
    WHERE
        rank <= 3
),
usa_medals AS (
    SELECT
        COUNT(*) AS total_medals
    FROM
        event_ranking
    WHERE
        participant_country_code = 'USA' AND rank <= 3
)
SELECT
    'EU' AS region, total_medals
FROM
    eu_medals
UNION ALL
SELECT
    'USA', total_medals
FROM
```

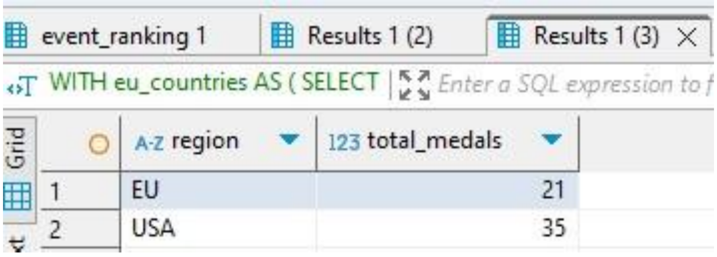
```
usa_medals;
```

EXPLANATION

This SQL code checks whether EU countries won more medals than the USA. It uses CTEs to count total medals for EU countries and the USA, comparing their rankings. The results are combined using a `UNION ALL` clause to show totals for both regions.

TABLE STRUCTURE

medal_table:



```
WITH eu_countries AS ( SELECT
```

	region	total_medals
1	EU	21
2	USA	35

Final Verification of the Medal Table Structure

```
SELECT * FROM medal_table;
```

EXPLANATION

This simple query retrieves all records from the `medal_table` view to confirm the accuracy of the medal counts and ensure data integrity.

PART 2: STOCK MARKET DATA ANALYSIS

INTRODUCTION

This report analyzes stock market data for six major companies in 2019, utilizing SQL queries and window functions in PostgreSQL to extract valuable insights. The dataset contains daily closing prices and trading volumes for the following stocks: Apple (AAPL), IBM, Cisco Systems (CSCO), Amazon (AMZN), Intel Corporation (INTC), and Google (GOOG). The analysis aims to load the data into a PostgreSQL table, compute moving averages, assess stock performance, and summarize daily and monthly trends.

PROBLEM DESCRIPTION

The file "stock_2019.csv" contains the prices of six stocks throughout the year 2019. The columns in the CSV include:

- **Date:** The date in the format "YYYY-MM-DD" (e.g., "2019-04-12").
- **Stock Symbol:** The unique ID of the stock (e.g., AAPL for Apple).
- **Closing Price:** The stock's daily closing price in dollars.
- **Volume:** The number of shares traded that day.

POSTGRESQL IMPLEMENTATION

Step 0: Clean Up Existing Tables and Views

```
DROP VIEW IF EXISTS moving_average CASCADE;

DROP TABLE IF EXISTS stocks CASCADE;
```

EXPLANATION

These commands remove any existing views or tables that may conflict with the new structures being created. The `CASCADE` option ensures that all dependencies related to these structures are also dropped.

Step 1: Create the `stocks` Table

```
CREATE TABLE IF NOT EXISTS stocks (

    date DATE,                    -- Date of stock price

    stock_symbol VARCHAR(10),      -- Unique ID for the stock (e.g., AAPL)

    closing_price DECIMAL(10, 4),  -- Closing price in dollars with 4 decimal precision
    volume BIGINT                   -- Volume of shares traded
);
```

EXPLANATION

This command creates the `stocks` table, defining each column with appropriate data types. The use of `DATE`, `VARCHAR`, and `DECIMAL` types ensures proper storage of date and numerical values. Notably, the `closing_price` column is defined as `DECIMAL` to accommodate precise financial data.

Step 2: Load Data into `stocks`

```
COPY stocks (date, stock_symbol, closing_price, volume)
FROM 'C:/Program Files/PostgreSQL/17/stock_2019.csv'  -- Update this path as necessary
WITH (FORMAT csv, HEADER true);
```

EXPLANATION

The `COPY` command imports data from the CSV file into the `stocks` table. The command specifies options to handle CSV formatting, including the presence of headers, which allows PostgreSQL to map the data correctly.

Step 3: Create a View for the 5-Day Moving Average

```
CREATE OR REPLACE VIEW moving_average AS
SELECT
    date,
    stock_symbol,
    closing_price,
    AVG(closing_price) OVER (PARTITION BY stock_symbol ORDER BY date ROWS
BETWEEN 4 PRECEDING AND CURRENT ROW) AS moving_avg
FROM stocks;
```

EXPLANATION

This command creates a view named `moving_average` that calculates the 5-day moving average of closing prices for each stock. The `AVG()` function is used with the `OVER` clause to partition the data by `stock_symbol`, enabling the calculation of moving averages separately for each stock.

Step 4: Calculate Minimum and Maximum Prices for Each Stock

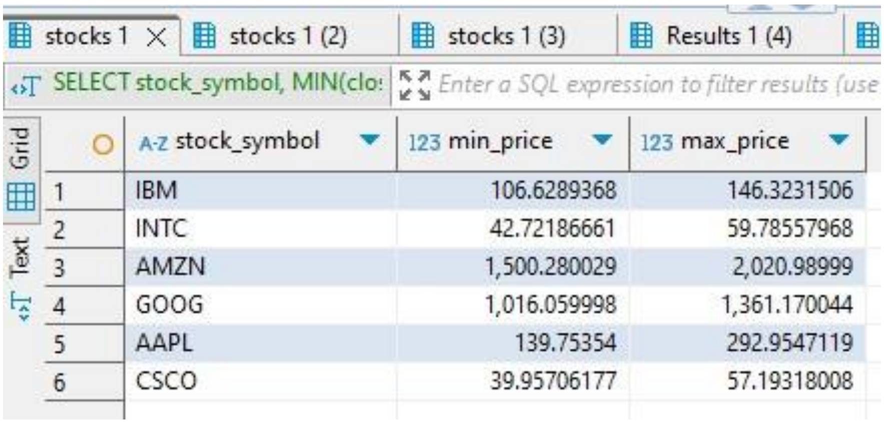
```
SELECT
    stock_symbol,
    MIN(closing_price) AS min_price,
    MAX(closing_price) AS max_price
FROM stocks
WHERE date BETWEEN '2019-01-01' AND '2019-12-31'
GROUP BY stock_symbol;
```

EXPLANATION

This query retrieves the minimum and maximum closing prices for each stock during the year 2019. The `MIN()` and `MAX()` functions calculate the lowest and highest prices, respectively, while the `GROUP BY` clause organizes the results by stock symbol.

TABLE STRUCTURE

stocks (min/max prices):



```
SELECT stock_symbol, MIN(clo:
```

	stock_symbol	min_price	max_price
1	IBM	106.6289368	146.3231506
2	INTC	42.72186661	59.78557968
3	AMZN	1,500.280029	2,020.98999
4	GOOG	1,016.059998	1,361.170044
5	AAPL	139.75354	292.9547119
6	CSCO	39.95706177	57.19318008

Step 5: Identify the Stock that Gained the Most in 2019

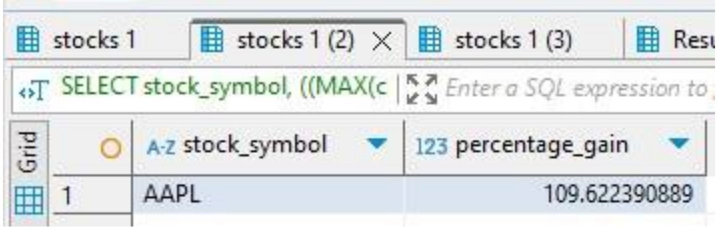
```
SELECT
    stock_symbol,
    ((MAX(closing_price) - MIN(closing_price)) / MIN(closing_price)) * 100 AS percentage_gain

FROM stocks

WHERE date BETWEEN '2019-01-01' AND '2019-12-31'

GROUP BY stock_symbol

ORDER BY percentage_gain DESC

LIMIT 1;
```

EXPLANATION

This query calculates the percentage gain for each stock and identifies the one with the highest gain. The percentage gain is computed by comparing the maximum and minimum closing prices, and the results are sorted to display the top performer.

TABLE STRUCTURE

stocks (percentage gain):



	stock_symbol	percentage_gain
1	AAPL	109.622390889

Step 6: Daily Gain/Loss for Each Stock

```
SELECT

    date,

    stock_symbol,

    CASE

        WHEN closing_price > LAG(closing_price) OVER (PARTITION BY stock_symbol ORDER BY date) THEN 1  -- Price went up

        WHEN closing_price < LAG(closing_price) OVER (PARTITION BY stock_symbol ORDER BY date) THEN 0  -- Price went down

        ELSE NULL  -- No change

    END AS gain

FROM stocks
```

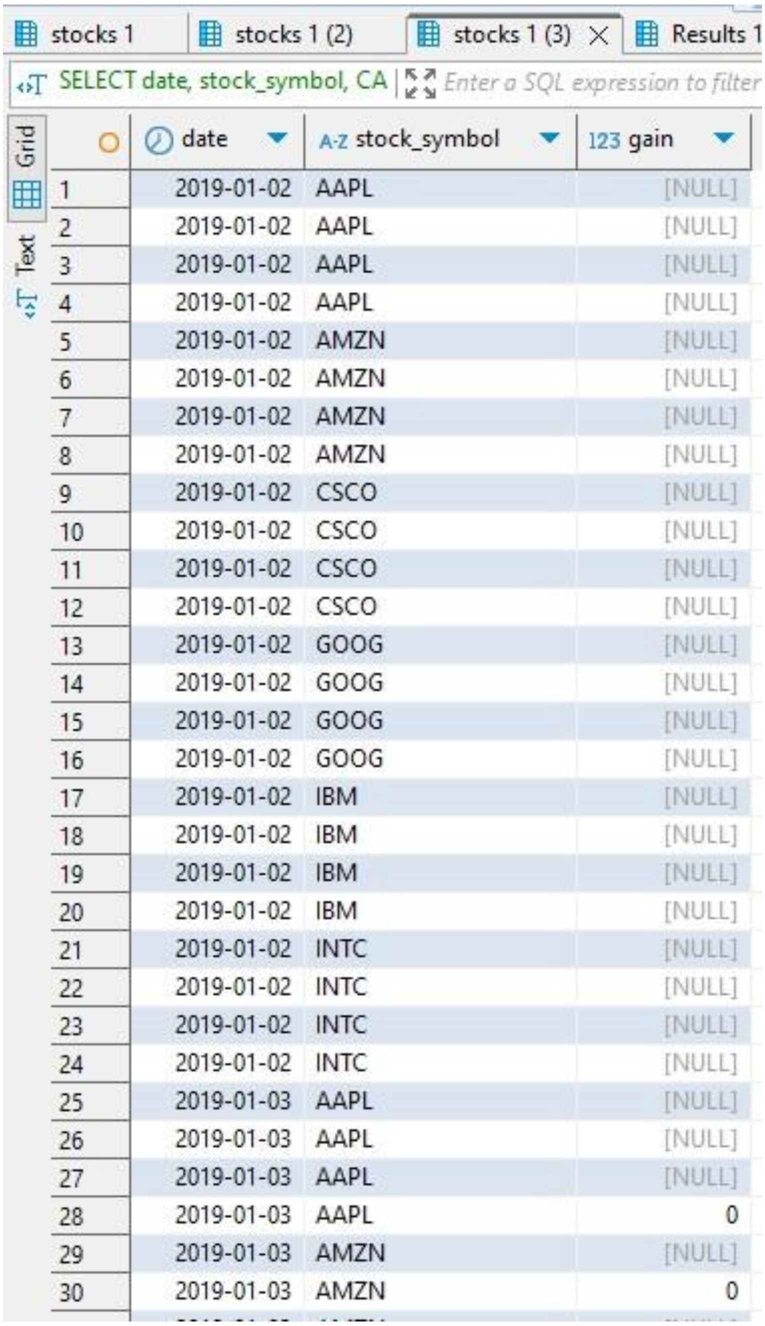
ORDER BY date, stock_symbol;

EXPLANATION

This query evaluates the daily performance of each stock by comparing the current day's closing price to the previous day's price. The `LAG()` function retrieves the previous day's closing price, and the `CASE` statement assigns a value of `1` for gains and `0` for losses.

TABLE STRUCTURE

stocks (daily gains):



SELECT date, stock_symbol, CA

	date	stock_symbol	gain
1	2019-01-02	AAPL	[NULL]
2	2019-01-02	AAPL	[NULL]
3	2019-01-02	AAPL	[NULL]
4	2019-01-02	AAPL	[NULL]
5	2019-01-02	AMZN	[NULL]
6	2019-01-02	AMZN	[NULL]
7	2019-01-02	AMZN	[NULL]
8	2019-01-02	AMZN	[NULL]
9	2019-01-02	CSCO	[NULL]
10	2019-01-02	CSCO	[NULL]
11	2019-01-02	CSCO	[NULL]
12	2019-01-02	CSCO	[NULL]
13	2019-01-02	GOOG	[NULL]
14	2019-01-02	GOOG	[NULL]
15	2019-01-02	GOOG	[NULL]
16	2019-01-02	GOOG	[NULL]
17	2019-01-02	IBM	[NULL]
18	2019-01-02	IBM	[NULL]
19	2019-01-02	IBM	[NULL]
20	2019-01-02	IBM	[NULL]
21	2019-01-02	INTC	[NULL]
22	2019-01-02	INTC	[NULL]
23	2019-01-02	INTC	[NULL]
24	2019-01-02	INTC	[NULL]
25	2019-01-03	AAPL	[NULL]
26	2019-01-03	AAPL	[NULL]
27	2019-01-03	AAPL	[NULL]
28	2019-01-03	AAPL	0
29	2019-01-03	AMZN	[NULL]
30	2019-01-03	AMZN	0

Step 7: Count of Days Apple Closed with Gain

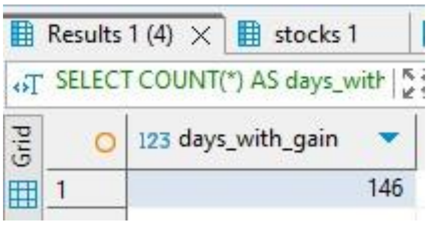
```
SELECT
    COUNT(*) AS days_with_gain
FROM (
    SELECT
        date,
        closing_price,
        CASE
            WHEN closing_price > LAG(closing_price) OVER (ORDER BY date) THEN 1
            ELSE 0  -- No gain
        END AS gain
    FROM stocks
    WHERE stock_symbol = 'AAPL'
) AS apple_gains
WHERE gain = 1;  -- Count only days with gain
```

EXPLANATION

This query counts the number of days that Apple’s closing price increased compared to the previous day. The inner query identifies days with gains for Apple, while the outer query aggregates the count of these gain days.

TABLE STRUCTURE

Results (days with gain):



```
SELECT COUNT(*) AS days_with
```

	days_with_gain
1	146

Step 8: Monthly Gain for Each Stock

```
WITH monthly_prices AS (
    SELECT
```

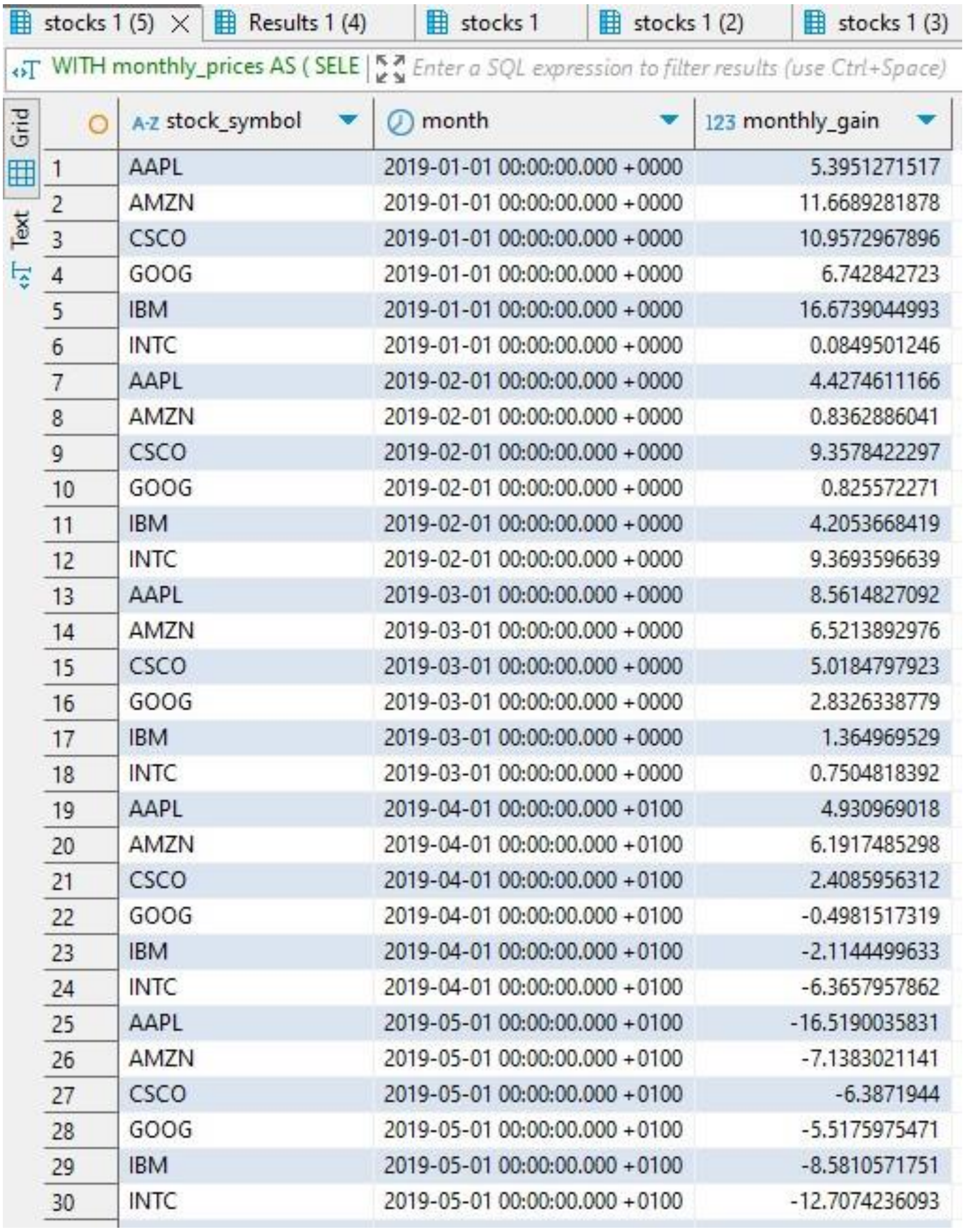
```
        stock_symbol,
        DATE_TRUNC('month', date) AS month,
        FIRST_VALUE(closing_price) OVER (PARTITION BY stock_symbol, DATE_TRUNC('month', date) ORDER BY date) AS first_price,
        LAST_VALUE(closing_price) OVER (PARTITION BY stock_symbol, DATE_TRUNC('month', date) ORDER BY date
            ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS last_price
    FROM stocks
    WHERE date BETWEEN '2019-01-01' AND '2019-12-31'
)
SELECT
    stock_symbol,
    month,
    ((last_price - first_price) / first_price) * 100 AS monthly_gain
FROM monthly_prices
GROUP BY stock_symbol, month, first_price, last_price
ORDER BY month, stock_symbol;
```

EXPLANATION

This query computes the monthly gain for each stock by comparing the closing prices at the beginning and end of each month. The `WITH` clause creates a Common Table Expression (CTE) to calculate the first and last prices for each month, allowing the final query to derive percentage gains for each stock.

TABLE STRUCTURE

stocks (monthly gain):


stocks 1 (5)
Results 1 (4)
stocks 1
stocks 1 (2)
stocks 1 (3)
WITH monthly_prices AS (SELE
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
Text

	stock_symbol	month	monthly_gain
1	AAPL	2019-01-01 00:00:00.000 +0000	5.3951271517
2	AMZN	2019-01-01 00:00:00.000 +0000	11.6689281878
3	CSCO	2019-01-01 00:00:00.000 +0000	10.9572967896
4	GOOG	2019-01-01 00:00:00.000 +0000	6.742842723
5	IBM	2019-01-01 00:00:00.000 +0000	16.6739044993
6	INTC	2019-01-01 00:00:00.000 +0000	0.0849501246
7	AAPL	2019-02-01 00:00:00.000 +0000	4.4274611166
8	AMZN	2019-02-01 00:00:00.000 +0000	0.8362886041
9	CSCO	2019-02-01 00:00:00.000 +0000	9.3578422297
10	GOOG	2019-02-01 00:00:00.000 +0000	0.825572271
11	IBM	2019-02-01 00:00:00.000 +0000	4.2053668419
12	INTC	2019-02-01 00:00:00.000 +0000	9.3693596639
13	AAPL	2019-03-01 00:00:00.000 +0000	8.5614827092
14	AMZN	2019-03-01 00:00:00.000 +0000	6.5213892976
15	CSCO	2019-03-01 00:00:00.000 +0000	5.0184797923
16	GOOG	2019-03-01 00:00:00.000 +0000	2.8326338779
17	IBM	2019-03-01 00:00:00.000 +0000	1.364969529
18	INTC	2019-03-01 00:00:00.000 +0000	0.7504818392
19	AAPL	2019-04-01 00:00:00.000 +0100	4.930969018
20	AMZN	2019-04-01 00:00:00.000 +0100	6.1917485298
21	CSCO	2019-04-01 00:00:00.000 +0100	2.4085956312
22	GOOG	2019-04-01 00:00:00.000 +0100	-0.4981517319
23	IBM	2019-04-01 00:00:00.000 +0100	-2.1144499633
24	INTC	2019-04-01 00:00:00.000 +0100	-6.3657957862
25	AAPL	2019-05-01 00:00:00.000 +0100	-16.5190035831
26	AMZN	2019-05-01 00:00:00.000 +0100	-7.1383021141
27	CSCO	2019-05-01 00:00:00.000 +0100	-6.3871944
28	GOOG	2019-05-01 00:00:00.000 +0100	-5.5175975471
29	IBM	2019-05-01 00:00:00.000 +0100	-8.5810571751
30	INTC	2019-05-01 00:00:00.000 +0100	-12.7074236093