Shwetank Singh
GritSetGrow - GSGLearn.com

DATA AND AI

WINDOW FUNCTIONS WITH REAL-TIME EXAMPLES & EXPLANATIONS

www.gsglearn.com

Introduction to Window Functions

Window functions perform calculations across a set of rows related to the current row, while retaining the original granularity of the dataset. They are essential for:

- Time-series analysis (e.g., stock price trends).
- Rankings (e.g., top-performing stocks).
- Cumulative metrics (e.g., running totals of trades).
- Comparative analysis (e.g., daily price changes).

Unlike aggregate functions (e.g., `SUM`, `AVG`), window functions do not collapse rows into summaries. Instead, they compute values for each row based on a defined **window** of rows.

---

2. Finance Dataset Design

We create a detailed dataset to simulate real-world financial analysis. The table includes **stock prices**, **dividends**, and **trading volumes** for three companies (AAPL, GOOGL, MSFT) over 10 trading days.

Table Structure: `stock_data`

Column	Description	Data Type
stock_symbol	Company ticker (e.g., AAPL)	VARCHAR(10)
date	Trading date	DATE
open_price	Opening stock price	DECIMAL(10,2)
high_price	Highest price during the day	DECIMAL(10,2)
low_price	Lowest price during the day	DECIMAL(10,2)
closing_price	Closing stock price	DECIMAL(10,2)
volume	Number of shares traded	INT
dividend	Dividend per share (if applicable)	DECIMAL(10,2)

Sample Data

```
-- Create table
CREATE TABLE stock_data (
    stock_symbol VARCHAR(10),
    date DATE,
    open_price DECIMAL(10,2),
    high_price DECIMAL(10,2),
    low_price DECIMAL(10,2),
    closing_price DECIMAL(10,2),
    volume INT,
    dividend DECIMAL(10,2)
);
```

```
-- Insert data for AAPL, GOOGL, MSFT (10 days)
INSERT INTO stock_data VALUES
```

```
('AAPL', '2023-10-01', 148.00, 152.00, 147.50, 150.00, 100000, 0.00),
('AAPL', '2023-10-02', 150.50, 153.00, 149.80, 152.00, 120000, 0.00),
...
('GOOGL', '2023-10-01', 2795.00, 2810.00, 2785.00, 2800.00, 50000, 0.00),
...
('MSFT', '2023-10-05', 308.00, 312.00, 307.50, 310.00, 95000, 2.50);
```

---

3. Types of Window Functions with Detailed Use Cases

3.1. Ranking Functions

a. `ROW_NUMBER()`

Assigns a unique sequential integer to rows within a partition.
Use Case: Identify the sequence of days when a stock's closing price increased consecutively.

```
SELECT
    stock_symbol,
    date,
    closing_price,
    ROW_NUMBER() OVER (
        PARTITION BY stock_symbol
        ORDER BY date
    ) AS trading_day_sequence
FROM stock_data;
```

Output Explanation:

- AAPL on 2023-10-01 gets `1`, 2023-10-02 gets `2`, etc.
- Helps track trading patterns over time (e.g., streaks of gains/losses).

b. `RANK()` vs. `DENSE_RANK()`

- **`RANK()`**: Skips ranks after ties (e.g., 1, 2, 2, 4).
- **`DENSE_RANK()`**: No rank gaps (e.g., 1, 2, 2, 3).
  Use Case: Rank stocks by daily trading volume.

```
SELECT
    date,
    stock_symbol,
    volume,
    RANK() OVER (PARTITION BY date ORDER BY volume DESC) AS volume_rank,
    DENSE_RANK() OVER (PARTITION BY date ORDER BY volume DESC) AS dense_volume_rank
FROM stock_data;
```

Output Example (2023-10-01):

date	stock_symbol	volume	volume_rank	dense_volume_rank
2023-10-01	AAPL	100000	1	1
2023-10-01	MSFT	80000	2	2

2023-10-01	GOOGL	50000	3	3

c. `NTILE(n)`

Divides rows into `n` buckets (e.g., quartiles, deciles).
Use Case: Segment stocks into quartiles based on closing price volatility.

```
SELECT
    stock_symbol,
    date,
    closing_price,
    NTILE(4) OVER (
        PARTITION BY stock_symbol
        ORDER BY (high_price - low_price) DESC
    ) AS volatility_quartile
FROM stock_data;
```

Insight:

- Stocks in quartile 1 have the highest daily price swings (high volatility).

3.2. Analytic Functions

a. `LAG()` and `LEAD()`

- **`LAG(column, offset)`**: Accesses data from a previous row.
- **`LEAD(column, offset)`**: Accesses data from a subsequent row.
  Use Case: Calculate day-over-day (DoD) price changes and 5-day momentum.

  ```
  SELECT
  stock_symbol,
  date,
  closing_price,
  LAG(closing_price, 1) OVER (PARTITION BY stock_symbol ORDER BY date) AS prev_close,
  closing_price - LAG(closing_price, 1) OVER (PARTITION BY stock_symbol ORDER BY date) AS dod_change,
  LEAD(closing_price, 5) OVER (PARTITION BY stock_symbol ORDER BY date) AS next_5day_close
  FROM stock_data;
  ```

- *Output Analysis**:
- `dod_change` shows daily price fluctuations.
- `next_5day_close` helps anticipate future prices for momentum strategies.

b. `FIRST_VALUE()` and `LAST_VALUE()`

- **`FIRST_VALUE()`**: Returns the first value in the window.
- **`LAST_VALUE()`**: Returns the last value in the window.
  Use Case: Compare the first and last closing prices of a stock in a 7-day window.

```
SELECT
stock_symbol,
date,
closing_price,
FIRST_VALUE(closing_price) OVER (
    PARTITION BY stock_symbol
    ORDER BY date
    ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
) AS week_start_price,
LAST_VALUE(closing_price) OVER (
    PARTITION BY stock_symbol
    ORDER BY date
    ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
) AS week_end_price
FROM stock_data;
```

- *Insight**:
- Track weekly performance trends (e.g., +5% increase from Monday to Friday).

3.3. Aggregate Functions with Window Frames

a. Moving Averages

Use Case: 7-day moving average to smooth price volatility.

```
SELECT
    stock_symbol,
    date,
    closing_price,
    AVG(closing_price) OVER (
        PARTITION BY stock_symbol
        ORDER BY date
        ROWS BETWEEN 6 PRECEDING AND CURRENT ROW
    ) AS moving_avg_7day
FROM stock_data;
```

Output:

stock_symbol	date	closing_price	moving_avg_7day
AAPL	2023-10-07	158.00	152.86

b. Cumulative Sum

Use Case: Track cumulative dividends paid over time.

```
SELECT
    stock_symbol,
    date,
    dividend,
    SUM(dividend) OVER (
        PARTITION BY stock_symbol
        ORDER BY date
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS cumulative_dividend
FROM stock_data;
```

Insight:

- MSFT's cumulative dividend reaches $10.00 by Q4 2023.

c. Rolling Total Volume

```
SELECT
    stock_symbol,
    date,
    volume,
    SUM(volume) OVER (
        PARTITION BY stock_symbol
        ORDER BY date
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    ) AS rolling_3day_volume
FROM stock_data;
```

Business Use:

- Identify spikes in trading activity (e.g., rolling volume > 300,000 shares).

3.4. Statistical Functions

a. `PERCENT_RANK()`

Computes the percentile rank of a row within a partition.
Use Case: Identify days where AAPL's closing price was in the top 10% historically.

```
SELECT
    stock_symbol,
    date,
    closing_price,
    PERCENT_RANK() OVER (
        PARTITION BY stock_symbol
        ORDER BY closing_price
    ) AS price_percentile
FROM stock_data
WHERE stock_symbol = 'AAPL';
```

Interpretation:

- A percentile of 0.9 means the price is higher than 90% of historical values.

b. `CUME_DIST()`

Calculates the cumulative distribution of a value (proportion of rows ≤ current value).
Use Case: Analyze the distribution of GOOGL's daily trading volumes.

```
SELECT
    stock_symbol,
    date,
    volume,
    CUME_DIST() OVER (
        PARTITION BY stock_symbol
        ORDER BY volume
    ) AS volume_cume_dist
FROM stock_data
WHERE stock_symbol = 'GOOGL';
```

Output:

- A CUME_DIST of 0.7 means 70% of days had volumes ≤ current day's volume.

---

4. Advanced Window Function Techniques

4.1. Custom Window Frames

Define the range of rows for calculations using `ROWS`, `RANGE`, or `GROUPS`.
Use Case: Compare a stock's closing price to the average of the prior 3 days (excluding current row).

```
SELECT
    stock_symbol,
    date,
    closing_price,
    AVG(closing_price) OVER (
        PARTITION BY stock_symbol
        ORDER BY date
        ROWS BETWEEN 3 PRECEDING AND 1 PRECEDING
    ) AS avg_prior_3days
FROM stock_data;
```

4.2. Multiple Window Partitions

Combine partitions for granular analysis.
Use Case: Rank stocks by volume within each sector (assuming a `sector` column exists).

```
SELECT
    stock_symbol,
    sector,
    date,
    volume,
    RANK() OVER (
        PARTITION BY sector, date
        ORDER BY volume DESC
    ) AS sector_volume_rank
FROM stock_data;
```

---

5. Real-World Finance Scenarios

Scenario 1: Identifying Support/Resistance Levels

Goal: Find price levels where a stock frequently reverses direction.
Approach: Use `LAG` and `LEAD` to detect local minima/maxima.

```
WITH price_changes AS (
    SELECT
        stock_symbol,
        date,
        closing_price,
        LAG(closing_price) OVER (PARTITION BY stock_symbol ORDER BY date) AS
prev_price,
        LEAD(closing_price) OVER (PARTITION BY stock_symbol ORDER BY date) AS
next_price
    FROM stock_data
)
SELECT
    stock_symbol,
    date,
    closing_price AS potential_support_resistance
FROM price_changes
WHERE (closing_price > prev_price AND closing_price > next_price) -- Resistance
    OR (closing_price < prev_price AND closing_price < next_price); -- Support
```

Scenario 2: Dividend Yield Analysis

Goal: Compare dividend yields across sectors.
Approach: Use `AVG` with partitions.

```
SELECT
    stock_symbol,
    date,
    dividend,
    closing_price,
    (dividend / closing_price) * 100 AS dividend_yield,
    AVG(dividend / closing_price) OVER (PARTITION BY sector) * 100 AS sector_avg_yield
FROM stock_data;
```

---

6. Performance Optimization

- **Indexing**: Add indexes on `stock_symbol` and `date` for faster partitioning.
- **Frame Size**: Use narrow frames (e.g., `ROWS 10 PRECEDING`) for large datasets.
- **Avoid `RANGE`**: Prefer `ROWS` over `RANGE` for precise control and faster execution.

---

7. Common Pitfalls

1. **Omitting `ORDER BY`**: Causes incorrect cumulative sums or rankings.
2. **Over-Partitioning**: Excessive partitions slow down queries.
3. **Ambiguous Window Frames**: Always define `ROWS`/`RANGE` explicitly.

---

8. Conclusion

Window functions empower financial analysts to:

- Track trends with moving averages.
- Rank assets by performance.
- Compute running totals (e.g., cumulative returns).

- Analyze distributions (e.g., volatility quartiles).

By combining partitioning, ordering, and custom frames, they unlock granular insights without aggregating data. For instance, a hedge fund might use `RANK()` to identify outperforming stocks or `LAG()` to model mean-reversion strategies. Mastery of window functions is critical for modern financial analytics.

Thanks, By Shwetank Singh - GSGLearn.com

Thank you

Shwetank Singh

GritSetGrow - GSGLearn.com