```
/*
    ======================
    == Window Functions ==
    ======================
    - Perform calculations across rows WITHOUT grouping them into one row
    - They keep all rows and just add a new calculated column

    ----------------------------
    PARTS OF A WINDOW FUNCTION
    ----------------------------
    1. FUNCTION
       - The calculation you want to perform
       - Examples include SUM, AVG, COUNT, ROW_NUMBER, RANK, DENSE_RANK, LAG, LEAD
       - Determines what type of analysis you are performing

    2. OVER()
       - Defines the "window" of rows the function should consider
       - Every window function must have OVER()
       - Without OVER(), it is not a window function

    3. PARTITION BY (optional)
       - Divides the rows into logical groups (like GROUP BY)
       - Each group has its own window for calculations
       - Unlike GROUP BY, rows within each partition are NOT collapsed
       - Useful for per-category calculations

    4. ORDER BY (optional)
       - Defines the order of rows within the partition
       - Essential for functions like ROW_NUMBER, LAG, LEAD, and running totals
       - Determines how calculations progress row by row

    ----------------------------
    TYPES OF WINDOW FUNCTIONS
    ----------------------------
    - Aggregate Window Functions
      - SUM, AVG, COUNT, MIN, MAX
      - Calculate aggregated values over the window without removing rows

    - Ranking Functions
      - ROW_NUMBER: unique sequential number for each row
      - RANK: rank rows, duplicates share the same rank, gaps exist
      - DENSE_RANK: rank rows, duplicates share the same rank, no gaps

    - Analytic / Access Functions
      - LAG: retrieves a value from a previous row
      - LEAD: retrieves a value from the next row
      - Useful for comparisons, trends, or differences between rows

    ----------------------------
    KEY POINTS TO REMEMBER
    ----------------------------
    - Window functions always keep the original rows
    - Calculations are relative to the "window" defined by OVER()
    - Partitioning allows group-wise calculations without collapsing rows
    - Ordering controls how row-specific functions (like LAG/LEAD) behave
    - Common real-life uses: rankings, running totals, cumulative averages, trend
analysis
*/
```

```sql
USE MyDatabase;


/*
    ROW_NUMBER()

    Task:
    - Sort customers by score from highest to lowest
    - Assign a unique number to each row based on that order
    - Even if two customers have the same score, each gets a different number
*/
SELECT
    id,
    first_name,
    score,
    ROW_NUMBER() OVER(ORDER BY score DESC) AS RankNumber
FROM Customers;



/*
    RANK()

    Task:
    - Rank customers based on their score
    - Customers with the same score get the same rank
    - When a tie happens, the next rank number is skipped
*/
SELECT
    first_name,
    score,
    RANK() OVER(ORDER BY score DESC) AS RankPosition
FROM Customers;



/*
    DENSE_RANK()

    Task:
    - Similar to RANK()
    - Customers with equal scores share the same rank
    - No rank numbers are skipped after a tie
*/
SELECT
    first_name,
    score,
    DENSE_RANK() OVER(ORDER BY score DESC) AS DenseRank
FROM Customers;
```

```sql
/*
    SUM() OVER()

    Task:
    - Calculate the total score of all customers in the table
    - Show that total value on every row
    - No rows are grouped or removed
*/
SELECT
    first_name,
    score,
    SUM(score) OVER() AS TotalScore
FROM Customers;



/*
    SUM() with PARTITION BY

    Task:
    - Divide customers into groups based on country
    - Calculate total score inside each country
    - Each customer sees the total score of their own country
*/
SELECT
    first_name,
    country,
    score,
    SUM(score) OVER(PARTITION BY country) AS CountryTotal
FROM Customers;



/*
    AVG() OVER with PARTITION BY

    Task:
    - Group customers logically by country
    - Calculate the average score within each country
    - Keep all individual rows visible
*/
SELECT
    first_name,
    country,
    score,
    AVG(score) OVER(PARTITION BY country) AS AvgCountryScore
FROM Customers;



/*
    LAG()

    Task:
    - Sort customers by ID
    - Show the score from the previous row
    - The first row has no previous value, so it returns NULL
*/
```

```sql
SELECT
    first_name,
    score,
    LAG(score) OVER(ORDER BY id) AS PreviousScore
FROM Customers;



/*
    LEAD()

    Task:
    - Sort customers by ID
    - Show the score from the next row
    - The last row has no next value, so it returns NULL
*/
SELECT
    first_name,
    score,
    LEAD(score) OVER(ORDER BY id) AS NextScore
FROM Customers;



/*
    Running Total

    Task:
    - Sort customers by ID
    - Add scores progressively from the first row to the current row
    - Each row shows a cumulative total up to that point
*/
SELECT
    first_name,
    score,
    SUM(score) OVER(ORDER BY id) AS RunningTotal
FROM Customers;
```
