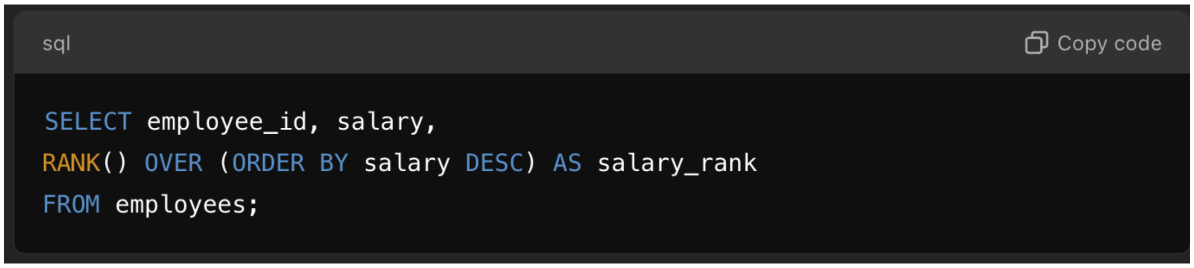
Mandatory SQL Topics for IT Professionals

1. Window functions allow you to perform calculations across a set of table rows that are related to the current row. They are useful for tasks like ranking, calculating running totals, and more, without collapsing rows into a single output.

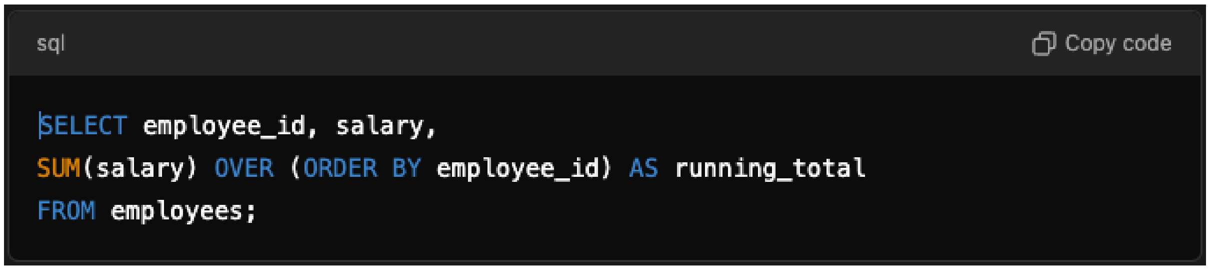
Examples:

Ranking Employees by Salary:



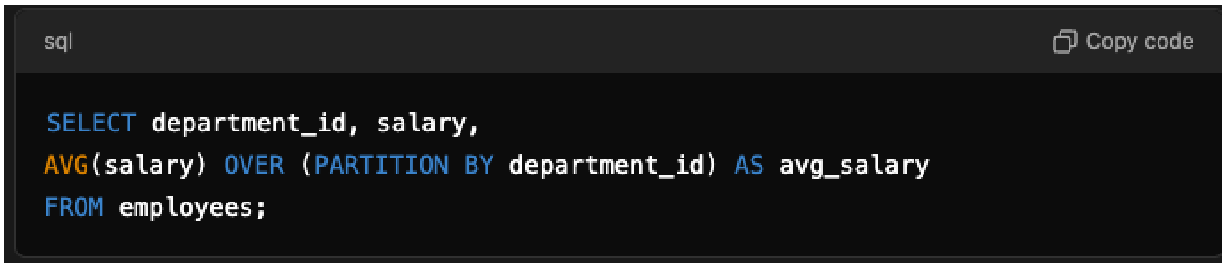
```sql
SELECT employee_id, salary,
RANK() OVER (ORDER BY salary DESC) AS salary_rank
FROM employees;
```

Calculating a Running Total:



```sql
SELECT employee_id, salary,
SUM(salary) OVER (ORDER BY employee_id) AS running_total
FROM employees;
```

Calculating the Average Salary per Department:



```sql
SELECT department_id, salary,
AVG(salary) OVER (PARTITION BY department_id) AS avg_salary
FROM employees;
```

Additional Information:
Common window functions include:

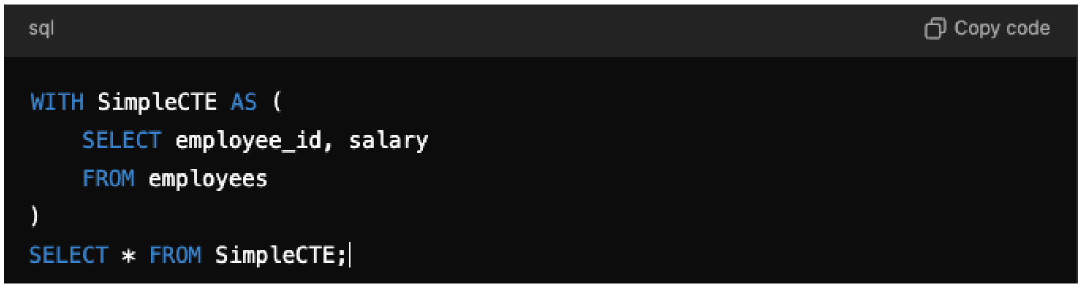
- RANK(): Assigns a rank to each row within a partition.
- ROW_NUMBER(): Assigns a unique number to each row.
- DENSE_RANK(): Similar to RANK() but without gaps in ranking.
- NTILE(n): Divides the result set into n buckets.
- SUM(), AVG(): Can be used as window functions to calculate totals over a set of rows.

2. Common Table Expressions (CTEs)

Description:
CTEs provide a way to define temporary result sets that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement. They help organize complex queries and improve readability.

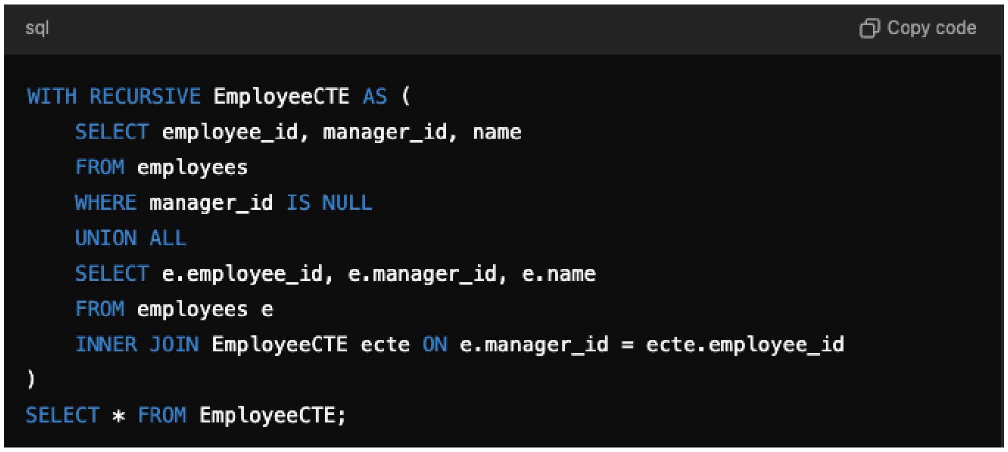
Types:

- Simple CTE:
  A simple CTE contains a single query that returns a result set.



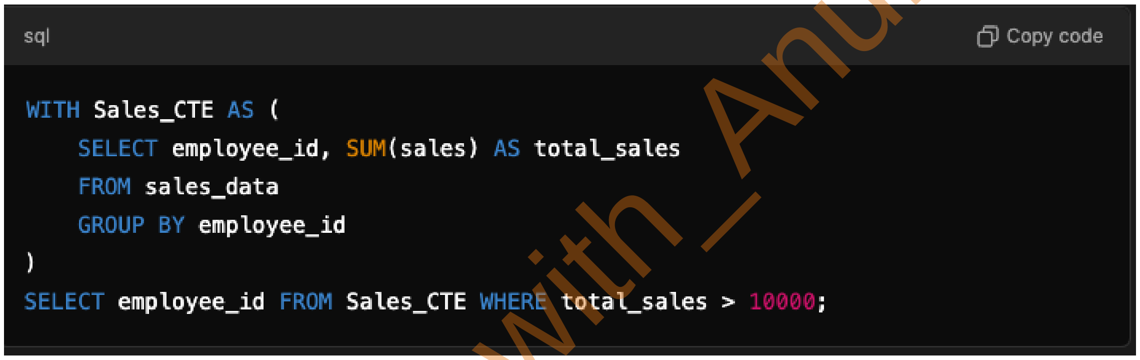
```sql
WITH SimpleCTE AS (
    SELECT employee_id, salary
    FROM employees
)
SELECT * FROM SimpleCTE;
```

Recursive CTE:
A recursive CTE references itself, allowing you to perform operations that require iteration, such as traversing hierarchical data.



```sql
WITH RECURSIVE EmployeeCTE AS (
    SELECT employee_id, manager_id, name
    FROM employees
    WHERE manager_id IS NULL
    UNION ALL
    SELECT e.employee_id, e.manager_id, e.name
    FROM employees e
    INNER JOIN EmployeeCTE ecte ON e.manager_id = ecte.employee_id
)
SELECT * FROM EmployeeCTE;
```

Example of Using CTE for Sales Data:



```sql
WITH Sales_CTE AS (
    SELECT employee_id, SUM(sales) AS total_sales
    FROM sales_data
    GROUP BY employee_id
)
SELECT employee_id FROM Sales_CTE WHERE total_sales > 10000;
```

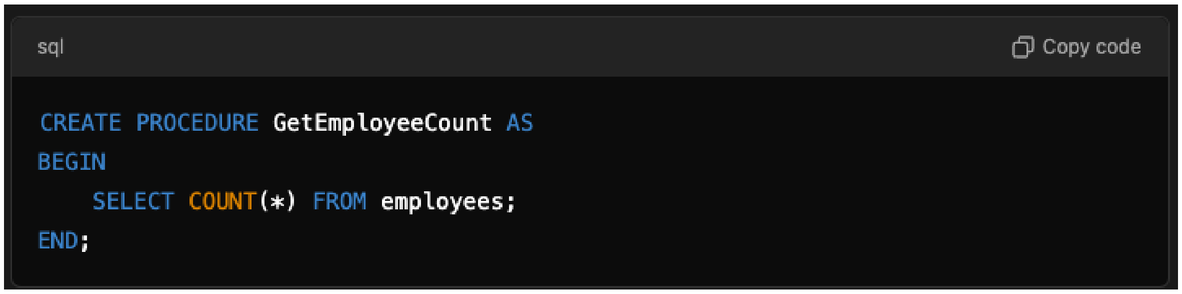
Additional Information:
Benefits of using CTEs:

- Simplifies complex joins and subqueries.
- Improves the readability of your SQL code.

3. Stored Procedures and Functions

Description:
Stored procedures are a collection of SQL statements that can be executed as a single command. Functions are similar but can return values and be used in SQL expressions. They help with code reusability and maintainability.

Example of Creating a Stored Procedure:



```sql
CREATE PROCEDURE GetEmployeeCount AS
BEGIN
    SELECT COUNT(*) FROM employees;
END;
```

Additional Information:
Stored procedures can:

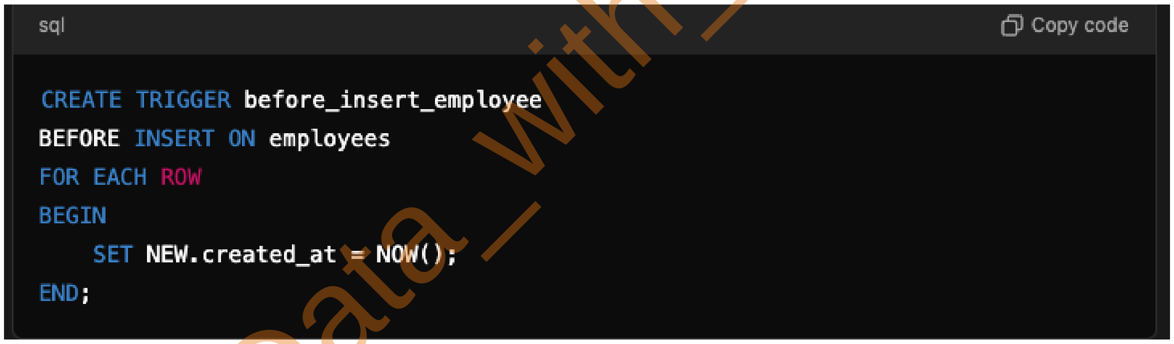
- Accept parameters to customize their behavior.
- Perform complex logic and operations, such as looping and conditional statements.

---

4. Triggers

Description:
Triggers are special types of stored procedures that automatically execute in response to certain events on a particular table or view. They can enforce business rules or maintain audit trails.

Example of a Trigger that Sets Created_at Timestamp:



```sql
CREATE TRIGGER before_insert_employee
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
    SET NEW.created_at = NOW();
END;
```

Additional Information:
Triggers can:

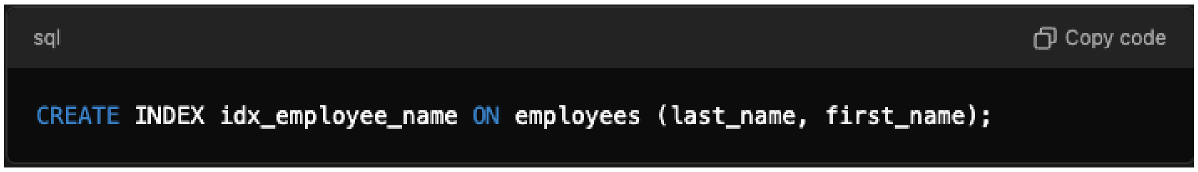
- Respond to INSERT, UPDATE, or DELETE operations.
- Be used for logging changes or validating data before it is committed.

---

5. Indexing Strategies

Description:
Indexes are used to speed up the retrieval of rows from a database table. They create a data structure that allows the database to find data quickly, improving query performance.

Example of Creating an Index:



```sql
CREATE INDEX idx_employee_name ON employees (last_name, first_name);
```

Additional Information:
Types of indexes include:

- **Unique Index: Ensures that all values in a column are different.**
- **Composite Index: An index on multiple columns to improve search efficiency.**

6. Data Modeling

Description:
Data modeling is the process of creating a visual representation of a data system, including the data entities, their relationships, and constraints. It is essential for efficient database design.

Example:

For effective data modeling, consider creating an ERD (Entity-Relationship Diagram) that outlines:

- Entities: Represent real-world objects (e.g., Employee, Department).
- Relationships: Define how entities interact (e.g., Employees belong to Departments).

Additional Information:
Benefits of good data modeling include:

- Improved data integrity and consistency.
- Easier maintenance and scalability of databases.

Window Functions Overview

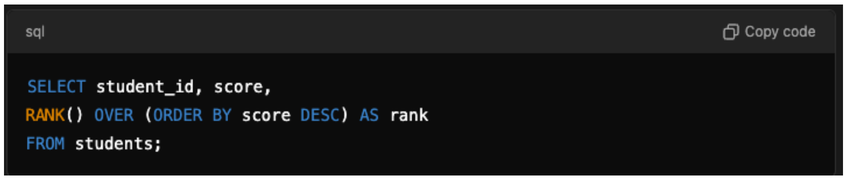
1. RANK()

Description:
Assigns a rank to each row within a partition of a result set. Rows with equal values receive the same rank, and the next rank is the number of distinct values in the partition plus one.

Use Case:
Used for ranking students by their scores in an exam.

Question:
What is the rank of each student based on their scores?

Query:



```sql
SELECT student_id, score,
RANK() OVER (ORDER BY score DESC) AS rank
FROM students;
```

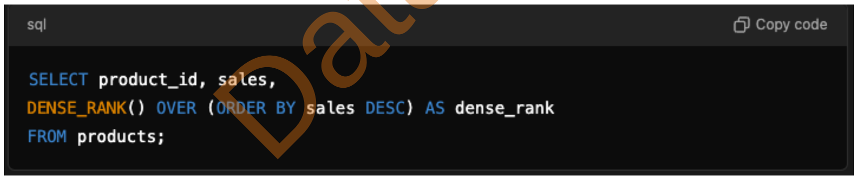
2. DENSE_RANK()

Description:
Similar to RANK(), but does not leave gaps in ranking when there are ties. All rows with the same values receive the same rank, and the next rank is the immediate next integer.

Use Case:
Used for ranking products by sales without gaps.

Question:
What is the dense rank of products based on sales?

Query:



sql Copy code

```sql
SELECT product_id, sales,
DENSE_RANK() OVER (ORDER BY sales DESC) AS dense_rank
FROM products;
```

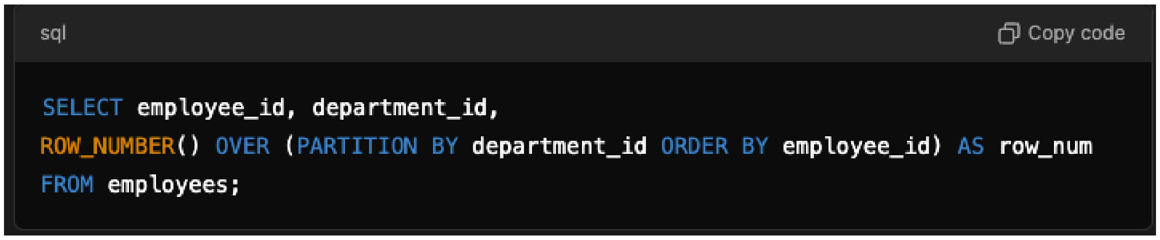
3. ROW_NUMBER()

Description:
Assigns a unique sequential integer to rows within a partition, starting at 1 for the first row.

Use Case:
Used to provide a unique identifier for each row in a result set.

Question:
What is the row number of each employee in the department?

Query:



sql Copy code

```sql
SELECT employee_id, department_id,
ROW_NUMBER() OVER (PARTITION BY department_id ORDER BY employee_id) AS row_num
FROM employees;
```

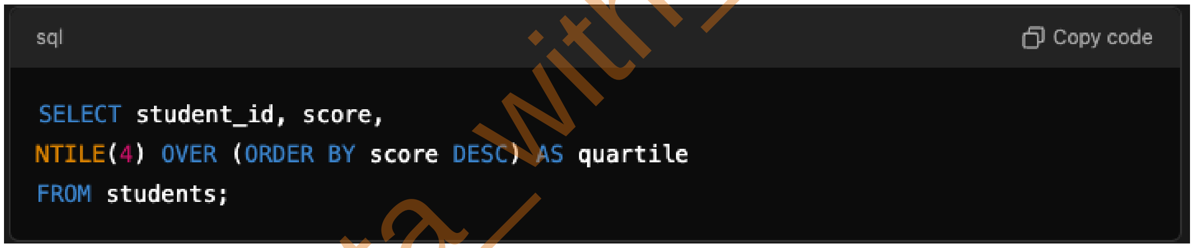
4. NTILE(n)

Description:
Divides the result set into 'n' number of buckets. Each bucket contains an approximately equal number of rows.

Use Case:
Used for dividing data into quartiles or deciles.

Question:
In which quartile does each student fall based on their score?

Query:



```sql
SELECT student_id, score,
NTILE(4) OVER (ORDER BY score DESC) AS quartile
FROM students;
```

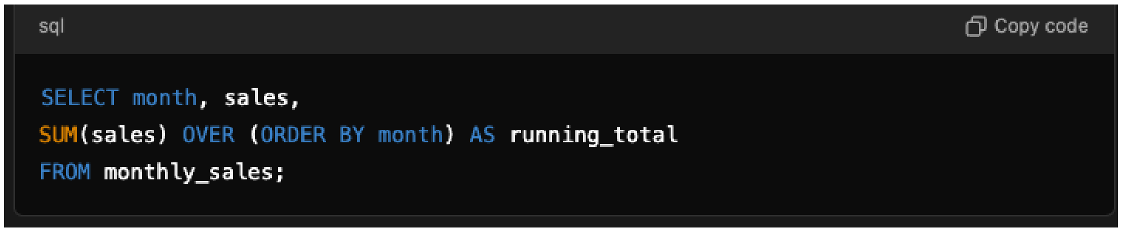
5. SUM()

Description:
Calculates the sum of a numeric column over a set of rows related to the current row.

Use Case:
Used to calculate running totals or subtotals.

Question:
What is the running total of sales for each month?

Query:



```sql
SELECT month, sales,
SUM(sales) OVER (ORDER BY month) AS running_total
FROM monthly_sales;
```

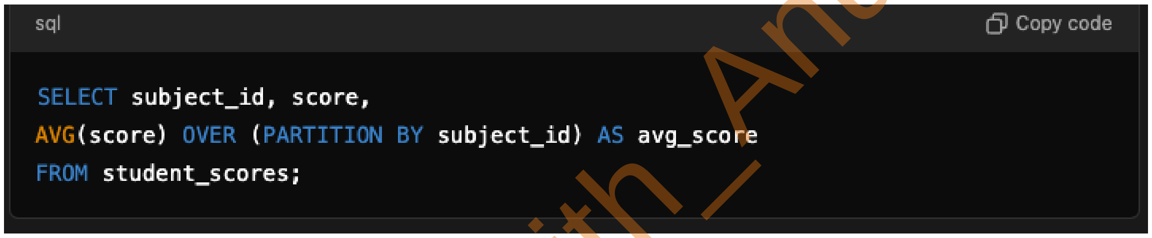
6. AVG()

Description:
Calculates the average value of a numeric column over a set of rows related to the current row.

Use Case:
Used to calculate the average score of students in each subject.

Question:
What is the average score of students in each subject?

Query:



```sql
SELECT subject_id, score,
AVG(score) OVER (PARTITION BY subject_id) AS avg_score
FROM student_scores;
```

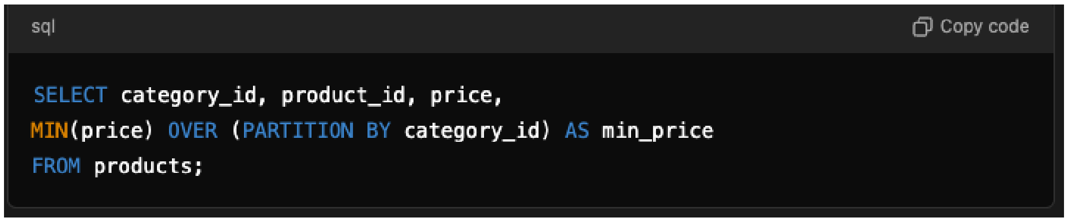
7. MIN()

Description:
Returns the minimum value of a column over a set of rows related to the current row.

Use Case:
Used to find the minimum price of products in each category.

Question:
What is the minimum price of products in each category?

Query:



```sql
SELECT category_id, product_id, price,
MIN(price) OVER (PARTITION BY category_id) AS min_price
FROM products;
```

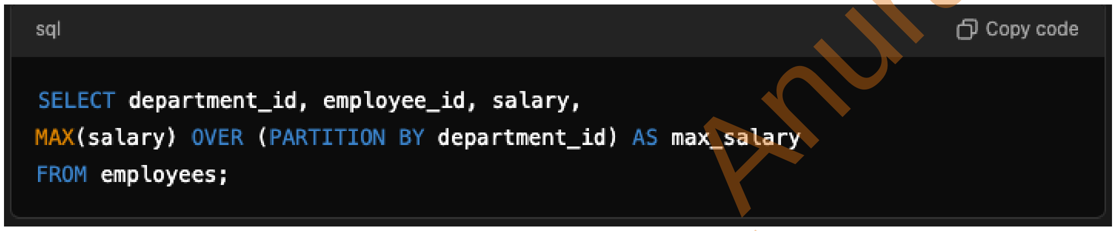
8. MAX()

Description:
Returns the maximum value of a column over a set of rows related to the current row.

Use Case:
Used to find the maximum salary in each department.

Question:
What is the maximum salary of employees in each department?

Query:



```sql
SELECT department_id, employee_id, salary,
MAX(salary) OVER (PARTITION BY department_id) AS max_salary
FROM employees;
```