Introduction to Databases (Postgres)

Zeyad Ashraf

LIKE

- Syntax :

```
SELECT * FROM table_name
WHERE column_name LIKE 'pattern';
```

- Example:

```
SELECT * FROM customers
WHERE customer_name LIKE 'A%';
```

The MAX() function returns the highest value in the selected column.

Operators in the WHERE clause

Symbol	Description	Example
%	Wildcard for 0 or more characters	'A%' → starts with A
_	Wildcard for a single character	'A_' → A + one character
[abc]	Matches any one character inside brackets	'J[ao]hn' → John or Jahn
[abc^]	Matches any character not in brackets	'J[^ae]hn' → not Jehn or Juhn
[a-z]	Matches any letter in the range	'S[a-z]m' → Sam, Sim, Som
\	Escape character (for literal %, _)	'20\%' → matches "20%"

EXAMPLES

```
-- Starts with A
SELECT * FROM customers WHERE name LIKE 'A%';

-- Ends with "n"
SELECT * FROM customers WHERE name LIKE '%n';

-- Contains "an"
SELECT * FROM customers WHERE name LIKE '%an%';

-- Starts with "J", any one character, then "n"
SELECT * FROM customers WHERE name LIKE 'J_n';

-- Name with exactly 4 characters
SELECT * FROM customers WHERE name LIKE '____';

-- Escape a percent symbol
SELECT * FROM products WHERE name LIKE '20\%%' ESCAPE '\';
```

EXAMPLES

```
SELECT * FROM customers
WHERE name LIKE 'A%' AND city NOT LIKE '%on';
```

IN

- Syntax :

```
SELECT * FROM table_name
WHERE column_name IN (value1, value2, ...);
```

- Example:

```
SELECT * FROM customers
WHERE country IN ('Germany', 'France', 'UK');
```

The IN operator allows you to specify a list of possible values in the WHERE clause.

NOT IN

- Syntax :

```
SELECT * FROM table_name
WHERE column_name NOT IN (value1, value2, ...);
```

- Example:

```
SELECT * FROM customers
WHERE country NOT IN ('Germany', 'France', 'UK');
```

The NOT IN operator returns rows that do not match any value in the list.

IN (SELECT)

- Syntax :

```
SELECT * FROM table_name
WHERE column_name IN (SELECT column_name FROM
another_table);
```

- Example:

```
SELECT * FROM customers
WHERE customer_id IN (SELECT customer_id FROM
orders);
```

You can also use a SELECT statement inside the parenthesis to return all records that are in the result of the SELECT statement.

NOT IN (SELECT)

- Syntax :

```
SELECT * FROM table_name
WHERE column_name NOT IN (SELECT column_name
FROM another_table);
```

- Example:

```
SELECT * FROM customers
WHERE customer_id NOT IN (SELECT customer_id FROM
orders);
```

BETWEEN

- Syntax :

```
SELECT * FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

- Example:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 15;
```

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.

Aliases

- Syntax :

```
SELECT column_name AS alias_name
FROM table_name;
```

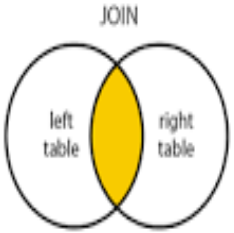
- Example:

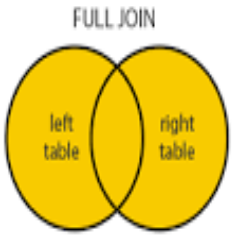
```
SELECT customer_id AS id
FROM customers;
```

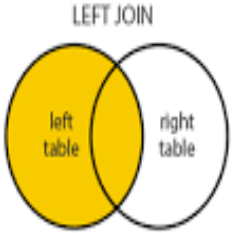
```
SELECT product_name || unit AS
product
FROM products;
```

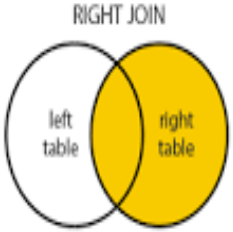
- SQL aliases are used to give a table, or a column in a table, a temporary name.
- Aliases are often used to make column names more readable.
- An alias only exists for the duration of that query.
- An alias is created with the AS keyword.

Join


JOIN
left table
right table


FULL JOIN
left table
right table


LEFT JOIN
left table
right table


RIGHT JOIN
left table
right table

JOIN

- JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Types of join

INNER JOIN: Returns records that have matching values in both tables

LEFT JOIN: Returns all records from the left table, and the matched records from the right table

RIGHT JOIN: Returns all records from the right table, and the matched records from the left table

FULL JOIN: Returns all records when there is a match in either left or right table

INNER JOIN

Note: `JOIN` and `INNER JOIN` will give the same result.

- Syntax :

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

- Example:

```
SELECT testproduct_id, product_name, category_name
FROM testproducts
INNER JOIN categories
ON testproducts.category_id = categories.category_id;
```

INNER JOIN returns only the rows where there is a match in both tables (i.e., matching category_id in both testproducts and categories).

LEFT JOIN

- Syntax :

```
SELECT columns
FROM table1
LEFT JOIN table2
ON table1.column = table2.column;
```

- Example:

```
SELECT testproduct_id, product_name, category_name
FROM testproducts
LEFT JOIN categories
ON testproducts.category_id = categories.category_id;
```

LEFT JOIN returns all rows from the left table (testproducts),
and the matched rows from the right table (categories).
If there is no match, NULLs are returned for the right table's columns.

RIGHT JOIN

- Syntax :

```
SELECT columns
FROM table1
RIGHT JOIN table2
ON table1.column = table2.column;
```

- Example:

```
SELECT testproduct_id, product_name, category_name
FROM testproducts
RIGHT JOIN categories
ON testproducts.category_id = categories.category_id;
```

The RIGHT JOIN keyword selects ALL records from the "right" table, and the matching records from the "left" table. The result is 0 records from the left side if there is no match.

Note: `RIGHT JOIN` and `RIGHT OUTER JOIN` will give the same result.

FULL JOIN

The `FULL JOIN` keyword selects ALL records from both tables, even if there is not a match. For rows with a match the values from both tables are available, if there is not a match the empty fields will get the value `NULL`.

- Syntax :

```
SELECT columns
FROM table1
FULL JOIN table2
ON table1.column = table2.column;
```

- Example:

```
SELECT testproduct_id, product_name, category_name
FROM testproducts
FULL JOIN categories
ON testproducts.category_id = categories.category_id;
```

CROSS JOIN

- The CROSS JOIN keyword matches ALL records from the "left" table with EACH record from the "right" table.
- Use with caution. The number of returned rows can grow very large if both tables contain many records.

- Syntax :

```
SELECT columns
FROM table1
CROSS JOIN table2;
```

- Example:

```
SELECT testproduct_id, product_name, category_name
FROM testproducts
CROSS JOIN categories;
```

Advanced Data Grouping and Merging Techniques in SQL

UNION & UNION ALL

The UNION operator is used to combine the result-set of two or more queries.

RULES

- They must have the same number of columns
- The columns must have the same data types
- The columns must be in the same order

- Syntax :

```
SELECT column1, column2
FROM table1
UNION
SELECT column1, column2
FROM table2
ORDER BY column;
```

- Example:

```
SELECT product_id, product_name
FROM products
UNION
SELECT testproduct_id, product_name
FROM testproducts
ORDER BY product_id;
```

GROUP BY

- The GROUP BY clause groups rows that have the same values into summary rows
- often used with aggregate functions like
- Syntax :

```
SELECT aggregate_function(column), group_column
FROM table
GROUP BY group_column;
```

- Example:

```
SELECT COUNT(customer_id), country
FROM customers
GROUP BY country;
```

```
SELECT customers.customer_name,
COUNT(orders.order_id)
FROM orders
LEFT JOIN customers ON
orders.customer_id =
customers.customer_id
GROUP BY customer_name;
```

HAVING

The HAVING clause was added to SQL because the WHERE clause cannot be used with aggregate functions.

- Syntax :

```
SELECT aggregate_function(column), group_column
FROM table
GROUP BY group_column
HAVING condition;
```

- Example:

```
SELECT COUNT(customer_id), country
FROM customers
GROUP BY country
HAVING COUNT(customer_id) > 5;
```

EXISTS

- The EXISTS operator is used to test for the existence of any record in a sub query.
- The EXISTS operator returns TRUE if the sub query returns one or more records.
- Syntax :

```
SELECT column_name
FROM table1
WHERE EXISTS (
  SELECT 1
  FROM table2
  WHERE table2.column = table1.column);
```

- Example:

```
SELECT customers.customer_name
FROM customers
WHERE EXISTS (
  SELECT order_id
  FROM orders
  WHERE customer_id = customers.customer_id);
```

```
SELECT customers.customer_name
FROM customers
WHERE NOT EXISTS (
  SELECT order_id
  FROM orders
  WHERE customer_id = customers.customer_id);
```

ANY

- Allows comparison between a single value and a set of values.
- Commonly used with operators like: =, !=, <, >, <=, >=.
- Returns a Boolean result (TRUE or FALSE).
- Returns TRUE if any value in the subquery meets the condition.
- In short: the condition is true if it matches at least one value from the set.
- Example:

```
SELECT product_name
FROM products
WHERE product_id = ANY (
  SELECT product_id
  FROM order_details
  WHERE quantity > 120);
```

ALL

- returns a Boolean value as a result
- returns TRUE if ALL of the sub query values meet the condition
- is used with SELECT, WHERE and HAVING statements
- Example:

```
SELECT product_name
FROM products
WHERE product_id = ALL (
  SELECT product_id
  FROM order_details
  WHERE quantity > 10
);
```

CASE

- CASE works like an if-then-else statement.
- It checks conditions in order, and returns the result of the first true condition.
- Once a condition is met, it stops evaluating the rest.
- If no condition is true, it returns the value from the ELSE clause.
- If there's no ELSE clause and no condition is met, it returns NULL.

- Syntax :

```
SELECT column1,
  CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ...
    ELSE default_result
  END
FROM table_name;
```

- Example:

```
SELECT product_name,
  CASE
    WHEN price < 10 THEN 'Low price product'
    WHEN price > 50 THEN 'High price product'
    ELSE 'Normal product'
  END
FROM products;
```

Subqueries

- A query **inside another query**.
- Used in WHERE, FROM, or SELECT.

- Example :

```
SELECT name
FROM customers
WHERE id IN (
  SELECT customer_id
  FROM orders
  WHERE total > 1000
);
```

Thanks

zeyadashraf015@gmail.com

01097143595

Zeyad Elmalky