**SQL Wildcards, IN, BETWEEN Operator, Aliases and Joins**

-----------------------------------------------------------------------------------------------------------------------------

-- SQL Wildcard Characters

-- A wildcard character is used to substitute one or more characters in a string.

-- Wildcard characters are used with the LIKE operator.

-- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

-- Return all customers that starts with the letter 'a':

SELECT \* FROM Customers

WHERE CustomerName REGEXP 'a%';

/\*

Wildcard Characters

Symbol Description

% Represents zero or more characters

\_ Represents a single character

[] Represents any single character within the brackets \*

^ Represents any character not in the brackets \*

- Represents any single character within the specified range \*

{} Represents any escaped character \*\*

\* Not supported in PostgreSQL and MySQL databases.

\*\* Supported only in Oracle databases.

\*/

-- Using the % Wildcard

-- The % wildcard represents any number of characters, even zero characters.

-- Return all customers that ends with the pattern 'es':

SELECT \* FROM Customers

WHERE CustomerName LIKE '%es';

-- Return all customers that contains the pattern 'mer':

SELECT \* FROM Customers

WHERE CustomerName LIKE '%mer%';

/\*Using the \_ Wildcard

The \_ wildcard represents a single character.

It can be any character or number, but each \_ represents one, and only one, character.

Return all customers with a City starting with any character, followed by "ondon":\*/

SELECT \* FROM Customers

WHERE City LIKE '\_ondon';

-- Return all customers with a City starting with "L", followed by any 3 characters, ending with "on":

SELECT \* FROM Customers

WHERE City LIKE 'L\_\_\_on';

-- Using the [] Wildcard

-- The [] wildcard returns a result if any of the characters inside gets a match.

-- Return all customers starting with either "b", "s", or "p":

SELECT \* FROM Customers

WHERE CustomerName LIKE 'bsp%';

-- Using the - Wildcard

-- The - wildcard allows you to specify a range of characters inside the [] wildcard.

-- Return all customers starting with "a", "b", "c", "d", "e" or "f":

SELECT \* FROM Customers

WHERE CustomerName LIKE '[A-F]%';

-- Combine Wildcards

-- Any wildcard, like % and \_ , can be used in combination with other wildcards.

-- Return all customers that starts with "a" and are at least 3 characters in length:

SELECT \* FROM Customers

WHERE CustomerName LIKE 'a\_\_%';

-- Return all customers that have "r" in the second position:

SELECT \* FROM Customers

WHERE CustomerName LIKE '\_r%';

-- Without Wildcard

-- If no wildcard is specified, the phrase has to have an exact match to return a result.

-- Return all customers from Spain:

SELECT \* FROM Customers

WHERE Country LIKE 'Spain';

-----------------------------------------------------------------------------------------------------------------------------

-- SQL IN Operator

-----------------------------------------------------------------------------------------------------------------------------

/\*

The SQL IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.\*/

-- Return all customers from 'Germany', 'France', or 'UK'

SELECT \* FROM Customers

WHERE Country IN ('Germany', 'France', 'UK');

-- Syntax

-- SELECT column\_name(s)

-- FROM table\_name

-- WHERE column\_name IN (value1, value2, ...);

-- NOT IN

-- By using the NOT keyword in front of the IN operator, you return all records that are NOT any of the values in the list.

-- Return all customers that are NOT from 'Germany', 'France', or 'UK':

SELECT \* FROM Customers

WHERE Country NOT IN ('Germany', 'France', 'UK');

-- IN (SELECT)

-- You can also use IN with a subquery in the WHERE clause.

-- With a subquery you can return all records from the main query that are present in the result of the subquery.

-- Return all customers that have an order in the Orders table:

SELECT \* FROM Customers

WHERE CustomerID IN (SELECT CustomerID FROM Orders);

-- NOT IN (SELECT)

-- The result in the example above returned 74 records, that means that there are 17 customers that haven't placed any orders.

-- Let us check if that is correct, by using the NOT IN operator.

-- Return all customers that have NOT placed any orders in the Orders table:

SELECT \* FROM Customers

WHERE CustomerID NOT IN (SELECT CustomerID FROM Orders);

-----------------------------------------------------------------------------------------------------------------------------

-- SQL BETWEEN Operator

-----------------------------------------------------------------------------------------------------------------------------

/\*

The SQL BETWEEN Operator

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. \*/

-- Selects all products with a price between 10 and 20:

SELECT \* FROM Products

WHERE Price BETWEEN 10 AND 20;

-- Syntax

-- SELECT column\_name(s)

-- FROM table\_name

-- WHERE column\_name BETWEEN value1 AND value2;

-- NOT BETWEEN

-- To display the products outside the range of the previous example, use NOT BETWEEN:

SELECT \* FROM Products

WHERE Price NOT BETWEEN 10 AND 20;

-- BETWEEN with IN

-- The following SQL statement selects all products with a price between 10 and 20. In addition, the CategoryID must be either 1,2, or 3:

SELECT \* FROM Products

WHERE Price BETWEEN 10 AND 20

AND CategoryID IN (1,2,3);

SELECT \* FROM Products

WHERE Price BETWEEN 10 AND 20

OR CategoryID IN (1,2,3);

-- BETWEEN Text Values

-- The following SQL statement selects all products with a ProductName alphabetically between Carnarvon Tigers and Mozzarella di Giovanni:

-- SELECT \* FROM Products

-- WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'

-- ORDER BY ProductName;

-- The following SQL statement selects all products with a ProductName between Carnarvon Tigers and Chef Anton's Cajun Seasoning:

SELECT \* FROM Products

WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Gula Malacca'

ORDER BY PRODUCTNAME;

-- NOT BETWEEN Text Values

-- The following SQL statement selects all products with a ProductName not between Carnarvon Tigers and Mozzarella di Giovanni:

SELECT \* FROM Products

WHERE ProductName NOT BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'

ORDER BY ProductName;

-- BETWEEN Dates

-- The following SQL statement selects all orders with an OrderDate between '01-July-1996' and '31-July-1996':

SELECT \* FROM Orders

WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;

-- OR:

SELECT \* FROM Orders

WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';

-----------------------------------------------------------------------------------------------------------------------------

-- SQL Aliases

-----------------------------------------------------------------------------------------------------------------------------

/\*

SQL Aliases

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.\*/

SELECT CustomerID AS ID

FROM Customers;

-- AS is Optional

-- Actually, in most database languages, you can skip the AS keyword and get the same result:

SELECT CustomerID ID

FROM Customers;

-- Syntax

-- When alias is used on column:

SELECT column\_name AS alias\_name

FROM table\_name;

-- When alias is used on table:

-- SELECT column\_name(s)

-- FROM table\_name AS alias\_name;

-- Alias for Columns

-- The following SQL statement creates two aliases, one for the CustomerID column and one for the CustomerName column:

SELECT CustomerID AS ID, CustomerName AS Customer

FROM Customers;

-- Using Aliases With a Space Character

-- If you want your alias to contain one or more spaces, like "My Great Products", surround your alias with square brackets or double quotes.

-- Using [square brackets] for aliases with space characters:

SELECT ProductName AS [My Great Products]

FROM Products;

-- Using "double quotes" for aliases with space characters:

SELECT ProductName AS "My Great Products"

FROM Products;

-- Note: Some database systems allows both [] and "", and some only allows one of them.

-- Concatenate Columns

-- The following SQL statement creates an alias named "Address" that combine four columns (Address, PostalCode, City and Country):

SELECT CustomerName, Address || ', ' || PostalCode || ' ' || City || ', ' || Country AS Address

FROM Customers;

select 'hello' ||' '||'world';

select 1+2;

-- Note: To get the SQL statement above to work in MySQL use the following:

SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,', ',Country) AS Address

FROM Customers;

-- Note: To get the SQL statement above to work in Oracle use the following:

SELECT CustomerName, (Address || ', ' || PostalCode || ' ' || City || ', ' || Country) AS Address

FROM Customers;

-- Alias for Tables

-- The same rules applies when you want to use an alias for a table.

-- Refer to the Customers table as Persons instead:

SELECT \* FROM Customers AS Persons;

-- It might seem useless to use aliases on tables, but when you are using more than one table in your queries,

-- it can make the SQL statements shorter.

-- The following SQL statement selects all the orders from the customer with CustomerID=4 (Around the Horn).

-- We use the "Customers" and "Orders" tables, and give them the table aliases of "c" and "o" respectively (Here we use aliases to make the SQL shorter):

SELECT o.OrderID, o.OrderDate, c.CustomerName,c.customerid as cust\_from\_c,o.CustomerID as cust\_id\_ord

FROM Customers AS cust, Orders AS ord

WHERE c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;

-- The following SQL statement is the same as above, but without aliases:

SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName||customerid

FROM Customers, Orders

WHERE Customers.CustomerName='Around the Horn' AND Customers.CustomerID=Orders.CustomerID;

-- Aliases can be useful when:

-- There are more than one table involved in a query

-- Functions are used in the query

-- Column names are big or not very readable

-- Two or more columns are combined together

-----------------------------------------------------------------------------------------------------------------------------