Wildcards in MySQL

Wildcards Characters

A wildcard character is used to substitute one or more charaters 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.

Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit

The wildcards can be also be used in combinations! Here are some examples showing different LIKE operators with "%" and "_" wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE "a%"	Finds any values that starts with "a"
WHERE CustomerName LIKE "%a"	Finds any values that end with "a"
WHERE CustomerName LIKE "%or%"	Finds any values that have "or" in any position
WHERE CustomerName LIKE "_r%"	Finds any values that have "r" in the second position
WHERE CustomerName LIKE "a_%_%"	Finds any values that starts with "a" and are at least 3 characters in length

LIKE Operator	Description
WHERE CustomerName LIKE "a%o"	Finds any values that starts with "a" and ends with "o"

Using the % Wildcard

The following SQL statement selects all customers with a City starting with "ber".

Example:

```
SELECT * FROM sales
WHERE City LIKE "ber%";
```

The following SQL statement selects all customers with a City containing the pattern "es":

Example:

```
SELECT * FROM sales
WHERE city LIKE "%es%";
```

Using the _ Wildcard

The following SQL statement selects all customers with a City starting with any character, followed by "ondon".

Example:

```
SELECT * FROM sales
WHERE city LIKE "_ondon";
```

The following SQL statement selects all customers with a city starting with "L", followed by any character, followed by "n", followed by any character, followed by "on":

Example:

```
SELECT * FROM sales
WHERE City LIKE "L_n_on";
```

CASE Statement

The CASE statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

CASE Syntax

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

CASE Examples

The following SQL goes through conditions and returns a value when the first condition is met:

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN "The quantity is greater than 30"
    WHEN Quantity = 30 THEN "The quantity is 30"
    ELSE "The quantity is under 30"
END AS QuantityText
FROM Sales;
```

The following SQL will order the customer by city. However, if City is NULL, then order by Country:

```
SELECT CustomerName, City, Country
FROM Sales
ORDER BY
(CASE
    WHEN City IS NULL THEN Country
    ELSE City
END;
```

Comments

Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.

Single Line Comments

Single line comments start with --.
Any text between -- and then end of the line will be ignored (will be not executed).
The following example uses a single-line comment as an explanation.
Example:

```
-- Select all:
SELECT * FROM Sales;
```

The following example uses a single-line comment to ignore the end of a line.
Example:

```
SELECT* FROM Sales -- WHERE city = "Berlin";
```

The following example uses a single-line comment to ignore a statement:

Example:

```
--SELECT * FROM Sales;
SELECT * FROM Products;
```

Multi-line Comments

Multi-line comments start with /* and end with */.

Any text between /* and */ will be ignored.

The following example uses a multi-line comment as an explanation:

Example:

```
/*Select all the columns
of all the records
in the customers table: */
SELECT * FROM Sales;
```

The following example uses a multi-line comment to ignore many statements:

Example:

```
/*SELECT * FROM Sales;
SELECT * FROM Products;
SELECT * FROM Orders;
SELECT * FROM Categories; */
SELECT * FROM Suppliers;
```

To ignore just a part of a statement, also use the /* */ comment.

The following example uses a comment to ignore part of a line:

Example:

```
SELECT CustomerName,/* City,*/ Country FROM Sales;
```

The following example uses a comment to ignore part of a statement.

Example:

```
SELECT * FROM Sales WHERE (CustomerName LIKE "L%"
OR CustomerName LIKE "R%" /*OR CustomerName LIKE "S%"
OR CustomerName LIKE "T%" */ OR CustomerName LIKE "W%")
AND Country = "USA"
ORDER BY CustomerName;
```

Aliases

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.

Alias Column Syntax

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

Alias Table Syntax

```
SELECT column_name (s)
FROM table_name AS alias_name;
```

Alias for Columns Examples

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

Example:

```
SELECT CustomerID AS ID, CustomerName AS Customer
FROM Sales;
```

The following SQL statement creates two aliases, one for the CustomerName column and one for the ContactName column. Note: Single or double quotation marks are required if the alias name contains spaces:

Example:

```
SELECT CustomerName AS Customer, ContactName AS "Contact person"
FROM sales;
```

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

Example:

```
SELECT CustomerName, CONCAT_WS(', ', Address, PostalCode,
City, Country) AS Address
FROM Sales;
```

Alias for Tables Example

The following SQL statement selects all the orders from the customer with CustomerID=4 (Around the Horn). We use the "sales" and "orders" tables, and give them the table aliases of "s" and "o" respectively (Here we use aliases to make the SQL shorter):

Example:

```
SELECT o.OrderID, o.OrderDate, s.CustomerName
FROM Sales AS s, Orders AS o
WHERE s.CustomerName = "Around the Horn" AND
s.CustomerID = o.CustomerID;
```

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

Example:

```
SELECT Orders.OrderID, Orders.OrderDate, Sales.CustomerName
FROM Sales, Orders
WHERE Sales.CustomerName = "Around the Horn" AND
Sales.CustomerID = Orders.CustomerID;
```