The SQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

* The AND operator displays a record if all the conditions separated by AND are TRUE.
* The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

### **AND Syntax**

SELECT column1, column2, ...  
FROM table\_name  
WHERE condition1 AND condition2 AND condition3 ...;

### **OR Syntax**

SELECT column1, column2, ...  
FROM table\_name  
WHERE condition1 OR condition2 OR condition3 ...;

### **NOT Syntax**

SELECT column1, column2, ...  
FROM table\_name  
WHERE NOT condition;

### **Example**

SELECT \* FROM Customers  
WHERE Country='Germany' AND (City='Berlin' OR City='München');

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

SELECT \* FROM Customers  
WHERE NOT Country='Germany' AND NOT Country='USA';

## The SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

### **ORDER BY Syntax**

SELECT column1, column2, ...  
FROM table\_name  
ORDER BY column1, column2, ... ASC|DESC;

SELECT \* FROM Customers  
ORDER BY Country DESC;

SELECT \* FROM Customers  
ORDER BY Country ASC, CustomerName DESC;

## What is a NULL Value?

A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

## How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

### **IS NULL Syntax**

SELECT column\_namesFROM table\_name  
WHERE column\_name IS NULL;

### **IS NOT NULL Syntax**

SELECT column\_namesFROM table\_name  
WHERE column\_name IS NOT NULL;

### **Example**

SELECT CustomerName, ContactName, Address  
FROM Customers  
WHERE Address IS NOT NULL;

## The SQL SELECT TOP Clause

The SELECT TOP clause is used to specify the number of records to return.

The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

**MySQL Syntax:**

SELECT *column\_name(s)*  
FROM *table\_name*WHERE *condition*  
LIMIT *number*;

### **Example**

SELECT \* FROM Customers  
WHERE Country='Germany'  
LIMIT 3;

## The SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

### **MIN() Syntax**

SELECT MIN(column\_name)  
FROM table\_name  
WHERE condition;

### **MAX() Syntax**

SELECT MAX(column\_name)  
FROM table\_name  
WHERE condition;

## MAX() Example

The following SQL statement finds the price of the most expensive product:

### **Example**

SELECT MAX(Price) AS LargestPrice  
FROM Products;

## The SQL COUNT(), AVG() and SUM() Functions

The COUNT() function returns the number of rows that matches a specified criteria.

The AVG() function returns the average value of a numeric column.

The SUM() function returns the total sum of a numeric column.

### **COUNT() Syntax**

SELECT COUNT(column\_name)  
FROM table\_name  
WHERE condition;

### **AVG() Syntax**

SELECT AVG(column\_name)  
FROM table\_name  
WHERE condition;

### **SUM() Syntax**

SELECT SUM(column\_name)  
FROM table\_name  
WHERE condition;

### **Example**

SELECT COUNT(ProductID)  
FROM Products;

SELECT AVG(Price)  
FROM Products;

SELECT SUM(Quantity)  
FROM OrderDetails;

## The SQL LIKE Operator

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

There are two wildcards often used in conjunction with the LIKE operator:

* % - The percent sign represents zero, one, or multiple characters
* \_ - The underscore represents a single character

**Note:** MS Access uses an asterisk (\*) instead of the percent sign (%), and a question mark (?) instead of the underscore (\_).

The percent sign and the underscore can also be used in combinations!

### **LIKE Syntax**

SELECT column1, column2, ...  
FROM table\_name  
WHERE columnN LIKE pattern;

|  |  |
| --- | --- |
| **LIKE Operator** | **Description** |
| WHERE CustomerName LIKE 'a%' | Finds any values that start 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 start with "a" and are at least 3 characters in length |
| WHERE ContactName LIKE 'a%o' | Finds any values that start with "a" and ends with "o" |

### **Example**

SELECT \* FROM Customers  
WHERE CustomerName LIKE 'a%';

SELECT \* FROM Customers  
WHERE CustomerName LIKE '%a';

SELECT \* FROM Customers  
WHERE CustomerName LIKE '%or%';

SELECT \* FROM Customers  
WHERE CustomerName LIKE '\_r%';

SELECT \* FROM Customers  
WHERE CustomerName LIKE 'a\_\_%';

SELECT \* FROM Customers  
WHERE ContactName LIKE 'a%o';

SELECT \* FROM Customers  
WHERE CustomerName NOT LIKE 'a%';

## 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.

### **BETWEEN Syntax**

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name BETWEEN value1 AND value2;

SELECT \* FROM Products  
WHERE Price BETWEEN 10 AND 20;

SELECT \* FROM Products  
WHERE Price BETWEEN 10 AND 20  
AND NOT CategoryID IN (1,2,3);

SELECT \* FROM Products  
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'  
ORDER BY ProductName;

SELECT \* FROM Products  
WHERE ProductName BETWEEN "Carnarvon Tigers" AND "Chef Anton's Cajun Seasoning"  
ORDER BY ProductName;

## 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 the query.

### **Alias Column Syntax**

SELECT column\_name AS alias\_name  
FROM table\_name;

SELECT column\_name(s)  
FROM table\_name AS alias\_name;

### **Example**

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

SELECT CustomerID AS ID, CustomerName AS Customer  
FROM Customers;

SELECT CustomerName AS Customer, ContactName AS [Contact Person]  
FROM Customers;

### **Example**

SELECT o.OrderID, o.OrderDate, c.CustomerName  
FROM Customers AS c, Orders AS o  
WHERE c.CustomerName="Around the Horn" AND c.CustomerID=o.CustomerID;

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

## SQL JOIN

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

Let's look at a selection from the "Orders" table

SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate  
FROM Orders  
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;

## The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

### **GROUP BY Syntax**

SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)ORDER BY column\_name(s);

### **Example**

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country;

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
ORDER BY COUNT(CustomerID) DESC;

SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders  
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID  
GROUP BY ShipperName;

## The SQL ANY and ALL Operators

The ANY and ALL operators are used with a WHERE or HAVING clause.

The ANY operator returns true if any of the subquery values meet the condition.

The ALL operator returns true if all of the subquery values meet the condition.

### **ANY Syntax**

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name operator ANY  
(SELECT column\_name FROM table\_name WHERE condition);

### **ALL Syntax**

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name operator ALL  
(SELECT column\_name FROM table\_name WHERE condition);

### **Example**

SELECT ProductName  
FROM Products  
WHERE ProductID = ANY (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);

SELECT ProductName  
FROM Products  
WHERE ProductID = ANY (SELECT ProductID FROM OrderDetails WHERE Quantity > 99);

## The SQL SELECT INTO Statement

The SELECT INTO statement copies data from one table into a new table.

### **SELECT INTO Syntax**

Copy all columns into a new table:

SELECT \*  
INTO newtable [IN externaldb]  
FROM oldtableWHERE condition;

Copy only some columns into a new table:

SELECT column1, column2, column3, ...  
INTO newtable [IN externaldb]  
FROM oldtableWHERE condition;

## Examples

SELECT CustomerName, ContactName INTO CustomersBackup2017  
FROM Customers;

SELECT \* INTO CustomersGermany  
FROM Customers  
WHERE Country = 'Germany';