***Filtering, Functions, Subquries***

***The WHERE Statement***

**The WHERE Statement**

The **WHERE** clause is used to extract only those records that fulfill a specified criterion.

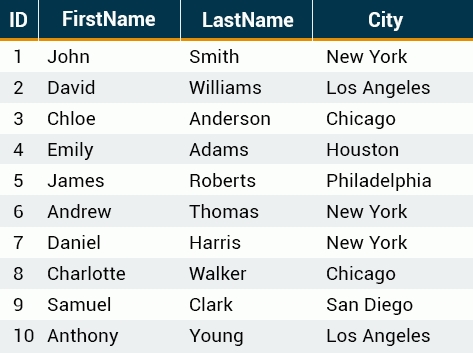
The syntax for the WHERE clause:

SELECT column\_list

FROM table\_name

**WHERE** condition;

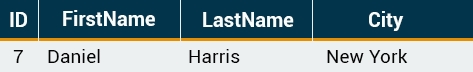
Consider the following table:



In the above table, to SELECT a specific record:

SELECT \* FROM customers

**WHERE ID = 7**;

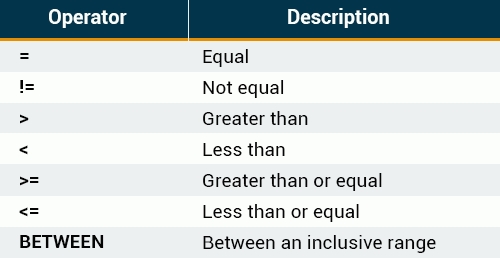


*The WHERE clause is used to extract only those records that fulfill a specified criterion.*

**SQL Operators**

**Comparison Operators** and **Logical Operators** are used in the WHERE clause to filter the data to be selected.

The following comparison operators can be used in the WHERE clause:

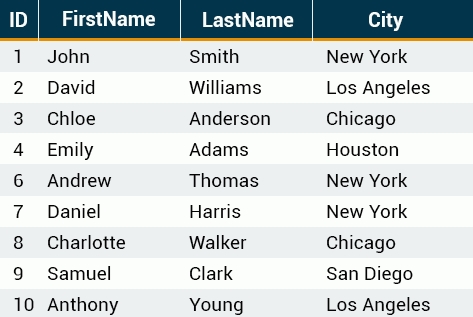


For example, we can display all customers names listed in our table, with the exception of the one with ID 5.

SELECT \* FROM customers

**WHERE ID != 5;**

Result:



*As you can see, the record with ID=5 is excluded from the list.*

**The BETWEEN Operator**

The **BETWEEN** operator selects values within a range. The first value must be lower bound and the second value, the upper bound.

The syntax for the BETWEEN clause is as follows:

SELECT column\_name(s)

FROM table\_name

WHERE column\_name **BETWEEN** value1 **AND** value2;

The following SQL statement selects all records with IDs that fall between 3 and 7:

SELECT \* FROM customers

**WHERE** ID **BETWEEN** 3 **AND** 7;

Result:



*As you can see, the lower bound and upper bound are both included in the range.*

**Text Values**

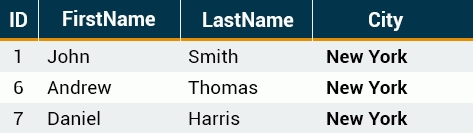
When working with text columns, surround any text that appears in the statement with **single quotation marks (')**.

The following SQL statement selects all records in which the *City* is equal to 'New York'.

SELECT ID, FirstName, LastName, City

FROM customers

WHERE City = **'New York'**;



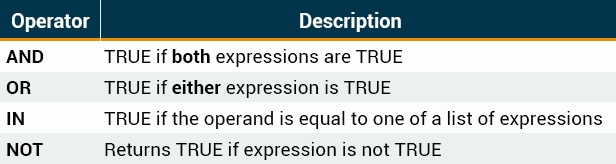
*If your text contains an apostrophe (single quote), you should use two single quote characters to escape the apostrophe. For example: 'Can''t'.*

***Filtering with AND, OR***

**Logical Operators**

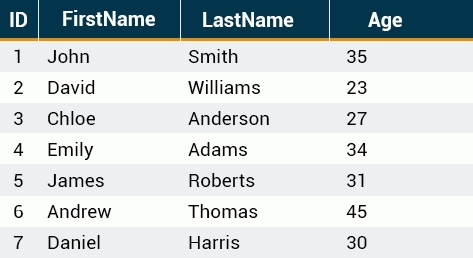
Logical operators can be used to combine two Boolean values and return a result of **true**, **false**, or **null**.

The following operators can be used:



When retrieving data using a SELECT statement, use logical operators in the WHERE clause to combine multiple conditions.

If you want to select rows that satisfy all of the given conditions, use the logical operator, **AND**.



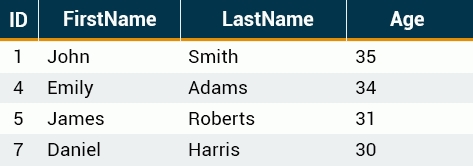
To find the names of the customers between 30 to 40 years of age, set up the query as seen here:

SELECT ID, FirstName, LastName, Age

FROM customers

WHERE Age >= 30 **AND** Age <= 40;

This results in the following output:

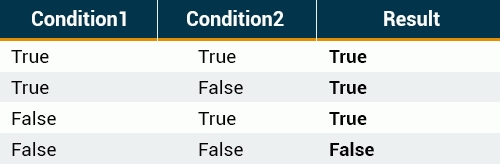


*You can combine as many conditions as needed to return the desired results.*

**OR**

If you want to select rows that satisfy at least one of the given conditions, you can use the logical OR operator.

The following table describes how the logical OR operator functions:

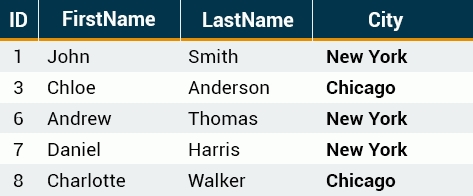


For example, if you want to find the customers who live either in New York or Chicago, the query would look like this:

SELECT \* FROM customers

WHERE City = 'New York' **OR** City = 'Chicago';

Result:



*You can OR two or more conditions.*

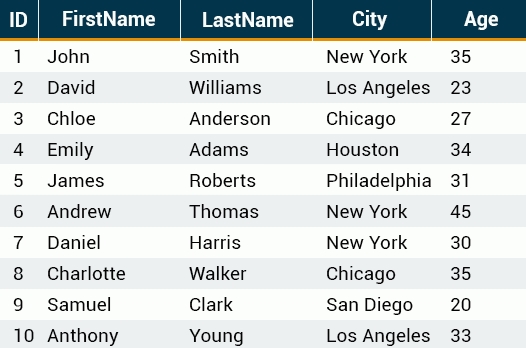
**Combining AND & OR**

The SQL AND and OR conditions may be combined to test multiple conditions in a query.

These two operators are called **conjunctive operators**.

When combining these conditions, it is important to use **parentheses**, so that the order to evaluate each condition is known.

Consider the following table:



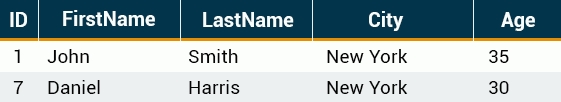
The statement below selects all customers from the city "New York" AND with the age equal to "30" OR “35":

SELECT \* FROM customers

WHERE City = 'New York'

**AND** (Age=30 **OR** Age=35);

Result:



*You can nest as many conditions as you need.*

***IN, NOT IN Statements***

**The IN Operator**

The IN operator is used when you want to compare a column with more than one value.

For example, you might need to select all customers from New York, Los Angeles, and Chicago.

With the **OR** condition, your SQL would look like this:

SELECT \* FROM customers

WHERE City = 'New York'

**OR** City = 'Los Angeles'

**OR** City = 'Chicago';

Result:



*The IN operator is used when you want to compare a column with more than one value.*

**The IN Operator**

You can achieve the same result with a single IN condition, instead of the multiple OR conditions:

SELECT \* FROM customers

WHERE City **IN** (**'New York'**, **'Los Angeles'**, **'Chicago'**);

This would also produce the same result:



*Note the use of parentheses in the syntax.*

**The NOT IN Operator**

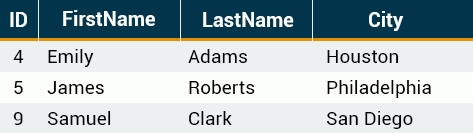
The **NOT** IN operator allows you to exclude a list of specific values from the result set.

If we add the **NOT** keyword before **IN** in our previous query, customers living in those cities will be excluded:

SELECT \* FROM customers

WHERE City **NOT** **IN** ('New York', 'Los Angeles', 'Chicago');

Result:



*The NOT IN operator allows you to exclude a list of specific values from the result set.*

***Custom Columns***

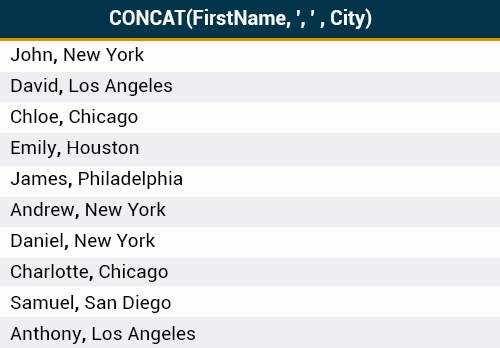
**The CONCAT Function**

The **CONCAT** function is used to concatenate two or more text values and returns the concatenating string.

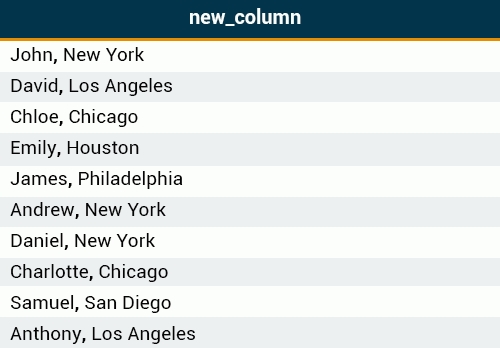
Let's concatenate the *FirstName* with the *City*, separating them with a *comma*:

SELECT **CONCAT**(FirstName, ', ' , City) FROM customers;

The output result is:



*The CONCAT() function takes two or more parameters.*



*A concatenation results in a new column.*

**The AS Keyword**

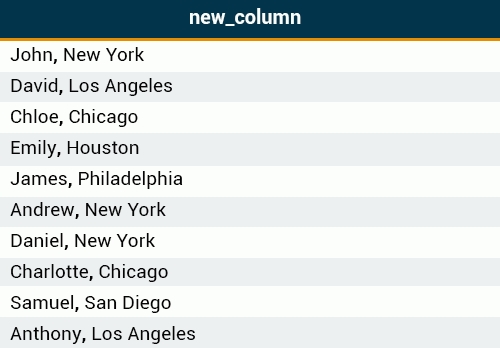
A concatenation results in a new column. The default column name will be the CONCAT function.

You can assign a custom name to the resulting column using the AS keyword:

SELECT CONCAT(FirstName,', ', City) **AS** new\_column

FROM customers;

And when you run the query, the column name appears to be changed.

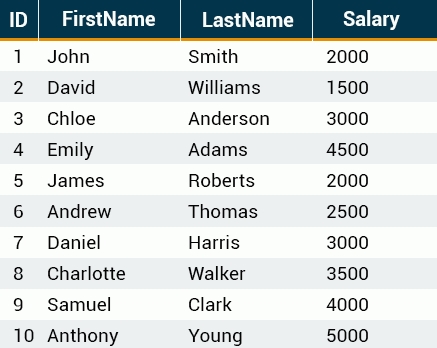
****

*A concatenation results in a new column.*

**Arithmetic Operators**

Arithmetic operators perform arithmetical operations on numeric operands. The Arithmetic operators include addition (+), subtraction (-), multiplication (\*) and division (/).

The following **employees** table shows employee names and salaries:

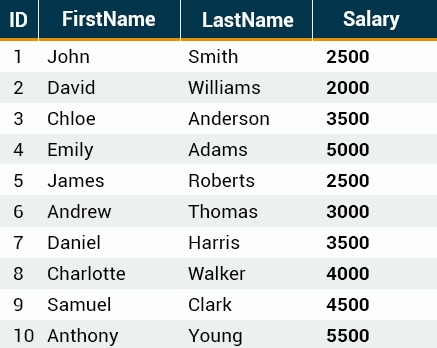


The example below adds 500 to each employee's salary and selects the result:

SELECT ID, FirstName, LastName, **Salary+500 AS Salary**

FROM employees;

Result:



*Parentheses can be used to force an operation to take priority over any other operators. They are also used to improve code readability.*

***Functions***

**The UPPER Function**

The **UPPER** function converts all letters in the specified string to uppercase.

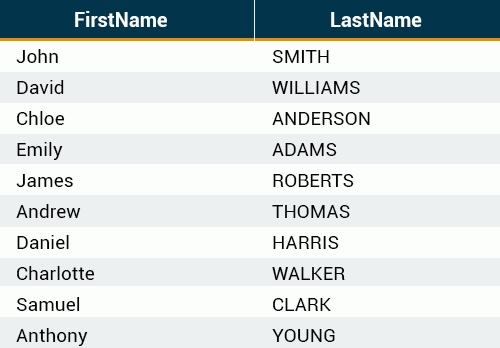
The **LOWER** function converts the string to lowercase.

The following SQL query selects all *LastNames* as uppercase:

SELECT FirstName, **UPPER(LastName)** AS LastName

FROM employees;

Result:



*If there are characters in the string that are not letters, this function will have no effect on them.*

**SQRT and AVG**

The **SQRT** function returns the square root of given value in the argument.

Let's calculate the square root of each Salary:

SELECT Salary, **SQRT(Salary)**

FROM employees;

Result:



Similarly, the **AVG** function returns the average value of a numeric column:

SELECT **AVG(Salary)** FROM employees;

Result:



*Another way to do the SQRT is to use POWER with the 1/2 exponent. However, SQRT seems to work faster than POWER in this case.*

**The SUM function**

The **SUM** function is used to calculate the sum for a column's values.

For example, to get the sum of all of the salaries in the employees table, our SQL query would look like this:

SELECT **SUM(Salary)** FROM employees;

Result:



*The sum of all of the employees' salaries is 31000.*

***Subqueries***

**Subqueries**

A **subquery** is a query within another query.

Let's consider an example. We might need the list of all employees whose salaries are greater than the average.

First, calculate the average:

SELECT AVG(Salary) FROM employees;

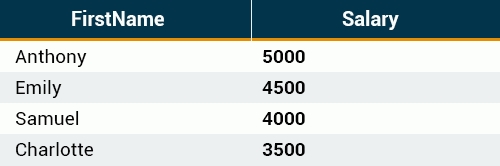
As we already know the average, we can use a simple WHERE to list the salaries that are greater than that number.

SELECT FirstName, Salary FROM employees

**WHERE Salary > 3100**

ORDER BY Salary DESC;

Result:



The **DESC** keyword sorts results in **descending** order.

Similarly, **ASC** sorts the results in **ascending** order.

**Subqueries**

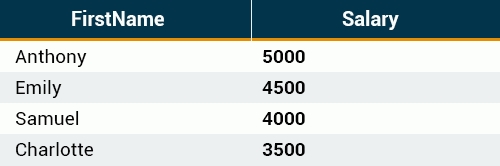
A single subquery will return the same result more easily.

SELECT FirstName, Salary FROM employees

WHERE Salary > **(SELECT AVG(Salary) FROM employees)**

ORDER BY Salary DESC;

The same result will be produced.

****

*Enclose the subquery in* ***parentheses****.*

*Also, note that there is no semicolon at the end of the subquery, as it is part of our single query.*

***LIKE and MIN***

**The Like Operator**

The **LIKE** keyword is useful when specifying a **search condition** within your WHERE clause.

SELECT column\_name(s)

FROM table\_name

WHERE column\_name **LIKE pattern**;

SQL **pattern** matching enables you to use "\_" to match any single character and "%" to match an arbitrary number of characters (including zero characters).

For example, to select employees whose *FirstNames* begin with the letter **A**, you would use the following query:

SELECT \* FROM employees

WHERE FirstName **LIKE 'A%'**;

Result:

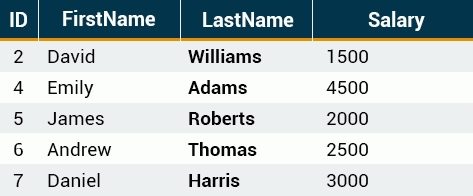


As another example, the following SQL query selects all employees with a *LastName* ending with the letter "s":

SELECT \* FROM employees

WHERE LastName **LIKE** **'%s'**;

Result:



*The % wildcard can be used* ***multiple*** *times within the same pattern.*

**The MIN Function**

The **MIN** function is used to return the minimum value of an expression in a SELECT statement.

For example, you might wish to know the minimum salary among the employees.

SELECT **MIN**(Salary) AS Salary FROM employees;



*All of the SQL functions can be combined together to create a single expression.*