

Top 30 most important SQL Queries for beginners:-

1. Retrieving Data From All Columns

This is a very basic query to display all data from a table. Notice that this query only has one character after `SELECT`: `"*"` (this denotes all columns). Therefore, you don't need to list the names of the columns. Of course, remember to write `FROM` and the name of the table from which you want to retrieve data. In this example, we are retrieving data from the table `animal`.

```
SELECT *  
FROM animal;
```

2. Retrieving Data From Certain Columns

The query above displays all of the data from the table `animal`. If you would like to only retrieve data from certain columns, list them after `SELECT`. In this example, we are retrieving data from the `id` and `name` columns.

```
SELECT id, name  
FROM animal;
```

3. Filtering Data Using WHERE Clause

In addition to retrieving data from certain columns, you can also filter data by listing conditions after `WHERE`. In this example, there is one condition: `age>=2`. We are looking for records with a value of 2 or more in the column `age`.

```
SELECT id, name,  
age  
FROM animal  
WHERE age>=2;
```

4. Filtering Data Using Conditions Joined by AND Operator

If you want to filter data using more than one condition, you can use `AND`. In this example, we are looking for records with a value of 2 or more in the column `age` and 'dog' in the column `name`.

```
SELECT id, name, age  
FROM animal  
WHERE age >= 2 AND name =  
'dog';
```

5. Filtering Data Using Conditions Joined by OR Operator

If only one of the conditions needs to be met, you can use `OR`. In this example, we are looking for records with a value of 2 or more in the column `age` or 'dog' in the column `name`.

```
SELECT id, name, age
FROM animal
WHERE age >= 2 OR name =
'dog';
```

6. Using DISTINCT to Retrieve Non-Repeated Records

You can place `DISTINCT` after `SELECT` to retrieve only one of each type of record. In this example, we want to retrieve records from columns `name` and `color`. If the values from these columns are the same in more than one record (e.g., there is more than one yellow T-shirt in the table), the query returns only one of those records.

```
SELECT DISTINCT name,
color
FROM clothing;
```

7. Retrieving Data Without NULL in a Certain Column

If you want to retrieve data only from rows without `NULL` in a certain column, use `IS NOT NULL`. In this example, the value in the column `color` may not be `NULL`. So, only records with a value stored in the column `color` will be returned. A similar operator to `IS NOT NULL` is `IS NULL`, which checks if a value is equal to `NULL`.

```
SELECT name, color
FROM clothing
WHERE color IS NOT
NULL;
```

8. Sorting Data According to One Column

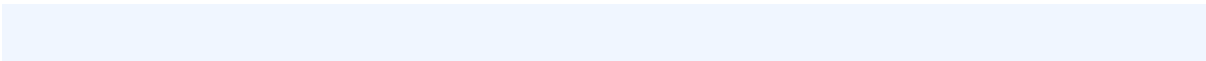
To sort data according to a column, place the column name after `ORDER BY`. The default sorting method is alphabetical, but you can also display rows in descending order by adding `DESC` after the name of the column. In this example, we want to sort the data in columns `id` and `name` according to the column `name`.

```
SELECT id, name
FROM animal
ORDER BY name;
```

9. Sorting Data According to More Than One Column

You can also sort data according to more than one column. In this example, the records are first sorted according to the column `name` in descending order and next according to the column `id` in ascending order. If the query finds records with the same name (e.g., all records with 'dog' in column `name`), it sorts these records in ascending order according to `id`.

```
SELECT id, name
FROM animal
ORDER BY name DESC,
id;
```



10. Searching for Values Matching a Certain Pattern

You can use `LIKE` to retrieve data that matches a certain pattern. In this example, we want to retrieve records from columns `id` and `name` that store a string containing the character "e" in the column `name` (e.g., records with names like **elephant**, **bee**, etc.)

```
SELECT id, name
FROM animal
WHERE name LIKE
'%e%';
```

11. Joining Values From Text Columns Into One String

The `CONCAT` function joins strings. In this example, the strings stored in the columns `category` and `name` are returned as a single column of strings with a space between the value in the column `category` and the value in the column `name`.

```
SELECT CONCAT(category, ' ',
name)
FROM tab;
```

12. Using Mathematical Operators

You can write queries to calculate values by using mathematical operators like "+", "-", "*", and "/." In this example, we want to calculate the discounted

price by subtracting the value in the column `discount` from the value in the column `price`.

```
SELECT price -  
discount  
FROM product;
```

13. Adding Data From Different Tables

You can join records from different tables using the operator `UNION ALL`. Remember that the records must be the same data type. In this example, we want to retrieve all rows with last names from the table `customer` and all rows with last names from the table `employee`. It will retrieve all last names, even if they are repeated. If we want to select all last names without repeats, we would use `UNION` instead of `UNION ALL`.

```
SELECT last_name FROM  
customer  
UNION ALL  
SELECT last_name FROM  
employee;
```

14. Finding the Intersection of Sets of Data

`INTERSECT` returns the intersection of two sets of data. In this example, we only want to retrieve the last names listed in both tables. To see what is different between the sets, use the operators `MINUS` or `EXCEPT`.

If you'd like to know more about set operators and see the visual explanation of this concept, [READ THIS ARTICLE](#).

```
SELECT last_name FROM  
customer  
INTERSECT  
SELECT last_name FROM  
employee;
```

15. Joining Data From Different Tables

You can join tables using `JOIN`, including `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, `FULL JOIN`, and `CROSS JOIN` (please see the courses listed at the end of this article for more information). In this example, we want to join data from the tables `customer` and `city`. `INNER JOIN` needs to come after `FROM` and the name of the first table, `customer`. After `INNER JOIN`, place the name of the second table, `city`. The records with data from both tables are matched by `ON` with the condition to join. The records in the table `city` are matched to the records from the table `customer` if they have the same value in the column `id` in the table `customer` and in the column `customer_id` in the table `city`.

```
SELECT customer.last_name,  
city.name  
FROM customer  
INNER JOIN city  
    ON customer.id =  
city.customer_id;
```

16. Using Aliases of Tables and Columns

If you join tables, it is a good idea to use aliases for table and column names. In this example, we want to join two tables, `customer` and `city`, and name them `c` and `t`, respectively. We define these new names in `FROM` or `JOIN`, using `AS`. Similarly, we rename the columns `last_name` in the table `customer` and `name` in the table `city` as `lname` and `city`, respectively.

```
SELECT c.last_name AS lname, t.name AS
city
FROM customer AS c
INNER JOIN city AS t
    ON c.id = t.customer_id;
```

17. Counting the Number of Rows in a Table

`COUNT` counts the number of rows. In this example, it returns the number of values from the column `id` stored in the table `product` (the number of all products).

```
SELECT COUNT(id)
FROM product;
```


18. Calculating the Average of the Values in a Column

You can calculate the average of the values in a column using `AVG`. In this example, the query returns the average price of all products in the table `product`.

```
SELECT AVG(price)
FROM product;
```

19. Calculating the Sum of the Values in a Column

`SUM` calculates the sum of the values in a column. In this example, it returns the value of all of the products.

```
SELECT SUM(price)
FROM product;
```

20. Finding the Minimum Value in a Column

You can find the minimum value stored in a column using `MIN`. In this example, the query returns the minimum price among the products.

```
SELECT MIN(price)
FROM product;
```

21. Finding the Maximum Value in a Column

You can find the maximum value stored in a column using `MAX`. In this example, the query returns the maximum price among the products.

```
SELECT MAX(price)
FROM product;
```

22. Calculating the Aggregate Value for Groups of Records

`GROUP BY` puts rows into groups to calculate a value. In this example, we use `COUNT` to calculate the number of rows (the number of products) in each group (`category`). The columns in `SELECT` have to put in the `GROUP BY` clause. `GROUP BY` can be used in the same way with other aggregate functions like `MAX`, `MIN`, `AVG`, and `SUM`.

```
SELECT category,
COUNT(id)
FROM product
GROUP BY category;
```

23. Filtering Rows Using Aggregate Functions

You can filter records after calculating values for each group using `HAVING`. In this example, we want to retrieve categories with an average price of products less than 56.50.

```
SELECT category,  
AVG(price)  
FROM product  
GROUP BY category  
HAVING AVG(price) < 56.50;
```

24. Removing Data From a Table

`DELETE FROM` removes all data from a table. In this example, we want to delete all data from the table `product`.

```
DELETE FROM  
product;
```

25. Removing Records Meeting a Certain Condition From a Table

You can remove records meeting a certain condition using `WHERE`. In this example, we want to remove records from the table `product` with `id` equal to 5.

```
DELETE FROM  
product  
WHERE id = 5;
```

26. Inserting Data Into a Table

You can add a new record to a table using `INSERT INTO`. After `INSERT INTO`, put the name of the table and then in brackets the names of the columns of

the table. After that, put `VALUES` and then in the brackets the values for the columns. In this example, we want to insert 25 into `id`, 'sofa' into `name`, and 'furniture' into `category` in the table `product`.

```
INSERT INTO product(id, name,  
category)  
VALUES(25, 'sofa', 'furniture');
```

27. Updating a Column in a Table

`UPDATE` allows you to modify data in the records. After `UPDATE`, put the name of the table, then `SET`, and then the name of the column to modify with "=" and new value to insert. This query modifies all values in the column. In this example, we want to change all values in the column `company` to 'ABC'.

```
UPDATE product SET company =  
'ABC';
```

28. Updating a Column by Filtering Records

However, if you don't want to change all values in a column, you can add `WHERE` with a condition. In the condition, you can specify which records to modify. In this example, we want to change values in the column `name` to 'armchair' only for records with `id=25`.

```
UPDATE product  
SET name =  
'armchair'  
WHERE id = 25;
```

29. Creating a Table

You can create a table using `CREATE TABLE`. After `CREATE TABLE`, put the name of the table and define in brackets the names of the columns and their data types. In this example, we want to create the table `tab` with two columns: `id` with integer as the data type and `name` limited to a maximum of 50 characters.

```
CREATE TABLE tab(id int, name  
varchar(50));
```

30. Deleting a Table

You can delete a table using `DROP TABLE`. Simply put the name of the table you want to delete after `DROP TABLE`. In this example, we want to delete the table `tab`.

```
DROP TABLE tab;
```

Summary

The queries described above are the most commonly used by both beginners and professionals. These queries you need to create or drop a table, insert data into a table, update records or remove data from a table.