PostgreSQL

PostgreSQL also known as Postgres, is a free and open-source relational database management system (RDBMS) emphasizing extensibility and SQL compliance. It was originally named POSTGRES. This database was developed at the University of California, Berkeley In 8 July 1996, the project was renamed to PostgreSQL to reflect its support for SQL. After a review in 2007, the development team decided to keep the name PostgreSQL and the alias Postgres.

It is available for Windows, Linux, UNIX, FreeBSD, and OpenBSD.

PostgreSQL evolved from *Ingres* project at the University of California, Berkeley. *Michael Stonebraker* is founder of PostgreSQL

It supports text, images, sounds, and video, and includes programming interfaces for C / C++, Java, Perl, Python, Ruby, Tcl and Open Database Connectivity (ODBC).

PostgreSQL supports a large part of the SQL standard and offers many modern features including the following –

- Complex SQL queries
- SQL Sub-selects
- Foreign keys
- Trigger
- Views
- Transactions
- Multiversion concurrency control (MVCC)
- Streaming Replication (as of 9.0)
- Hot Standby (as of 9.0)

SELECT

The SELECT statement has the following clauses:

- Select distinct rows using *DISTINCT* operator.
- Sort rows using *ORDER BY* clause.
- Filter rows using *WHERE* clause.
- Select a subset of rows from a table using *LIMIT* or *FETCH* clause.
- Group rows into groups using *GROUP BY* clause.
- Filter groups using *HAVING* clause.
- Join with other tables using joins such as *INNER JOIN, LEFT JOIN, FULL OUTER JOIN, CROSS JOIN* clauses.
- Perform set operations using *UNION, INTERSECT*, and *EXCEPT*.

SELECT statement that retrieves data from a single table.

```
SELECT
  select_list
FROM
  table_name;
```

If you specify a list of columns, you need to place a comma (,) between two columns to separate them. If you want to select data from all the columns of the table, you can use an asterisk (*)

Notice that we added a semicolon (;) at the end of the SELECT statement. The semicolon is not a part of the SQL statement. It is used to signal PostgreSQL the end of an SQL statement. The semicolon is also used to separate two SQL statements.

Note that the SQL keywords are case-insensitive. It means that SELECT is equivalent to select or Select.

The following example uses the SELECT statement to return full names and emails of all customers:

```
SELECT
  first_name || ' ' || last_name,
  email
FROM
  customer;
```

|| is a concatenation operator.

Column alias

A column alias allows you to assign a column or an expression in the select list of a SELECT statement a temporary name. The column alias exists temporarily during the execution of the query.

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

AS keyword is optional.

The main purpose of column aliases is to make the headings of the output of a query more meaningful.

If a column alias contains one or more spaces, you need to surround it with double quotes like this:

```
first_name || ' ' || last_name "full name"
```

ORDER BY

When you query data from a table, the SELECT statement returns rows in an unspecified order. To sort the rows of the result set, you use the ORDER BY clause in the SELECT statement.

The ORDER BY clause allows you to sort rows returned by a SELECT clause in ascending or descending order based on a sort expression.

Use the ASC option to sort rows in ascending order and the DESC option to sort rows in descending order. If you not using the ASC or DESC option, the ORDER BY uses ASC by default.

If column alias is available so use it with ASC and DESC option.

When you sort rows that contains NULL, you can specify the order of NULL with other non-null values by using the NULLS FIRST or NULLS LAST option of the ORDER BY clause:

DISTINCT

The DISTINCT clause is used in the SELECT statement to remove duplicate rows from a result set. The DISTINCT clause keeps one row for each group of duplicates. The DISTINCT clause can be applied to one or more columns in the select list of the SELECT statement. When we use multiple columns in query so distinct returns unique combination rows.

```
SELECT
  DISTINCT column1
FROM
  table_name;
```

if you have many coloumns in query but you want to use distinct on some coloumn so you can use DISTINCT ON. it return a single row for each distinct group defined by the ON clause.

WHERE

The WHERE clause appears right after the FROM clause of the SELECT statement. The WHERE clause uses the condition to filter the rows returned from the SELECT clause.

WHERE clause in the UPDATE and DELETE statement to specify rows to be updated or deleted. To form the condition in the WHERE clause, you use comparison and logical operators.

More: https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-where/

we can use where with equal (=) operator, AND operator, OR operator, IN operator, LIKE operator, BETWEEN operator, not equal(<>) operator.

LIMIT

PostgreSQL LIMIT is an optional clause of the SELECT statement that constrains the number of rows returned by the query.

```
SELECT
        title,
FROM
        Film
ORDER BY
        title,
LIMIT 10;
```

If row_count is zero, the query returns an empty set. In case row_count is NULL, the query returns the same result set as it does not have the LIMIT clause.

In case you want to skip a number of rows before returning the row_count rows, you use OFFSET clause placed after the LIMIT clause as the following statement: limit use after the order by.

```
SELECT select_list
FROM table_name
LIMIT row_count OFFSET row_to_skip;
```

Offset *row_to_skip* will skips rows from starting. If row_to_skip is zero, the statement will work like it doesn't have the OFFSET clause.

FETCH

The FETCH clause is functionally equivalent to the LIMIT clause. If you plan to make your application compatible with other database systems, you should use the FETCH clause because it follows the standard SQL.

The LIMIT clause is widely used by many relational database management systems such as MySQL, H2, and HSQLDB. However, the LIMIT clause is not a SQL-standard.

ROW is the synonym for ROWS, FIRST is the synonym for NEXT. So you can use any of them.

```
SELECT
   film_id,
   title
FROM
   film
ORDER BY
   title
FETCH FIRST 5 ROW ONLY;
```

IN operator

As I mentioned above in *where* section - IN operator use with *where* to check if a value matches any value in a list of values.

```
SELECT customer_id,
        rental_id,
        return_date
FROM
        rental
WHERE
        customer_id IN (1, 2)
ORDER BY
        return_date DESC;
```

NOT IN operator is just opposite than ***IN*** operator.

```
SELECT
        customer_id,
FROM
        rental
WHERE
        customer_id NOT IN (1, 2);
```

BETWEEN

BETWEEN operator use get values from a range of numbers. The values can be numbers, text, or dates. The BETWEEN operator is inclusive: begin and end values are included. *NOT BETWEEN* is just opposite of between operator.

```
SELECT
        customer_id,
        payment_id,
        amount
FROM
        payment
WHERE
        amount BETWEEN 8 AND 9;
```

LIKE

Suppose that you want to find a customer but you do not remember her name exactly. However, you just remember that her name begins with something like Jen.

Just use % or other wildcard character in place which you didn't remember.

```
SELECT
        first_name,
    last_name
FROM
        customer
WHERE
        first_name LIKE 'Jen%';
```

NOT LIKE is just opposite of LIKE operator

- Percent sign (%) matches any sequence of zero or more characters.
- Underscore sign (_) matches any single character.

% is an example of wildcard character others are: [], -, !, ?, *, #

If the pattern does not contain any wildcard character, the LIKE operator behaves like the equal (=) operator.

PostgreSQL supports the ILIKE operator that works like the LIKE operator. In addition, the ILIKE operator matches value case-insensitively. And NOT ILIKE is just opposite of ILIKE operator.

More: https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-like/

NULL

NULL means missing information or not applicable. NULL is not a value, so, you cannot compare it with any other values like numbers or strings. The comparison of NULL with a value will always result in NULL, which means an unknown result.

use null is condition is not a right way to use it because NULL is not equal to any value even itself. To check whether a value is NULL or not, you use the IS NULL operator instead. postgreSQL IS NOT NULL is just opposite of IS NULL

More: https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-is-null/

JOINS

Join is used to combine columns from one (self-join) or more tables based on the values of the columns between related tables. Join works typically the primary key columns of the first table and foreign key columns of the second table.

PostgreSQL supports inner join, left join, right join, full outer join, cross join, natural join, and a special kind of join called self-join.

At the top level there are mainly 3 types of joins:

- INNER
- OUTER
- CROSS

INNER JOIN - fetches data if present in both the tables.

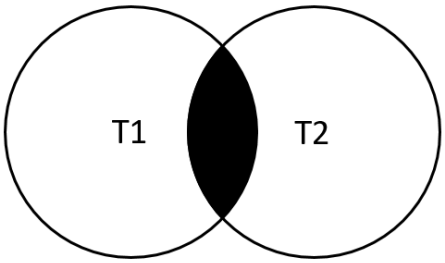
OUTER JOIN are of 3 types:

1. LEFT JOIN / LEFT OUTER JOIN - fetches data if present in the left table.
2. RIGHT JOIN / RIGHT OUTER JOIN - fetches data if present in the right table.
3. FULL JOIN / FULL OUTER JOIN - fetches data if present in either of the two tables.

CROSS JOIN, as the name suggests, does [n X m] that joins everything to everything. Similar to scenario where we simply lists the tables for joining (in the FROM clause of the SELECT statement), using commas to separate them.

INNER JOIN

The INNER JOIN keyword selects records that have matching values in both tables. If you didn't specify join name then database by default choose Inner join.



SELECT column_name(s)

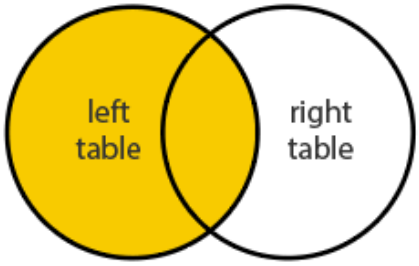
FROM table1

INNER JOIN table2

ON table1.column_name = table2.column_name;

LEFT JOIN

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2).



SELECT column_name(s)

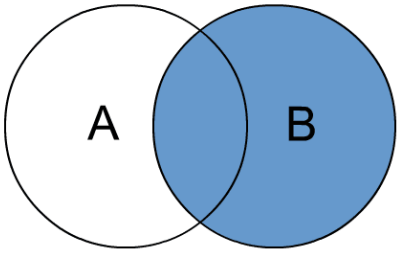
FROM table1

LEFT JOIN table2

ON table1.column_name = table2.column_name;

RIGHT JOIN

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1).



SELECT column_name(s)

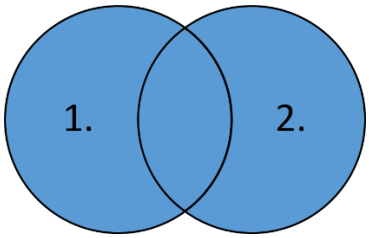
FROM table1

RIGHT JOIN table2

ON table1.column_name = table2.column_name;

FULL OUTER JOIN

The full outer join or full join returns a result set that contains all rows from both left and right tables.



SELECT column_name(s)

FROM table1

FULL OUTER JOIN table2

ON table1.column_name = table2.column_name

WHERE condition;

SELF JOIN

A self-join allows you to join a table to itself. A self-join is a join in which a table is joined with itself (which is also called Unary relationships), the self-join can be viewed as a join of two copies of the same table.

A self-join uses the inner join or left join clause. Because the query that uses the self-join references the same table, the table alias is used to assign different names to the same table within the query.

CROSS JOIN

A CROSS JOIN clause allows you to produce a Cartesian Product of rows in two or more tables.

Different from other join clauses such as LEFT JOIN or INNER JOIN, the CROSS JOIN clause does not have a join predicate.

Suppose you have to perform a CROSS JOIN of two tables T1 and T2.

If T1 has n rows and T2 has m rows, the result set will have nxm rows. For example, the T1 has 1,000 rows and T2 has 1,000 rows, the result set will have 1,000 x 1,000 = 1,000,000 rows.

The following illustrates the syntax of the CROSS JOIN syntax:

```
SELECT select_list
FROM T1
CROSS JOIN T2;
```

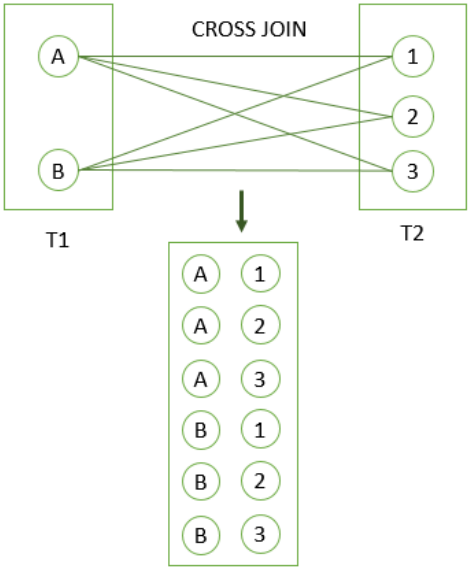


Table Aliases

Table aliases temporarily assign tables new names during the execution of a query.

```
table_name AS alias_name;
```

Similar to column aliases, the AS keyword is optional

GROUP BY

The GROUP BY clause divides the rows returned from the SELECT statement into groups. 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.

```
SELECT country, COUNT(*) AS number
FROM Customers
GROUP BY country;
```

HAVING

HAVING use for give condition in Query. HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

PostgreSQL evaluates the HAVING clause after the FROM, WHERE, GROUP BY, and before the SELECT, DISTINCT, ORDER BY and LIMIT clauses.

HAVING vs. WHERE

The WHERE clause allows you to filter rows based on a specified condition. However, the HAVING clause allows you to filter groups of rows according to a specified condition.

UNION

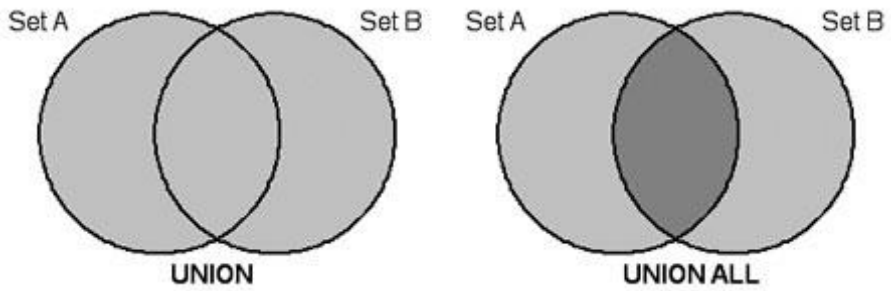
The UNION operator combines result sets of two or more SELECT statements into a single result set.

```
SELECT select_list_1
FROM table_expresssion_1
UNION
SELECT select_list_2
FROM table_expression_2
```

To using UNION. Please follow following terms

1. The number and the order of the columns in the select list of both queries must be the same.
2. The data types must be compatible.

The UNION operator removes all duplicate rows from the combined data set. To retain the duplicate rows, you use the the UNION ALL instead.

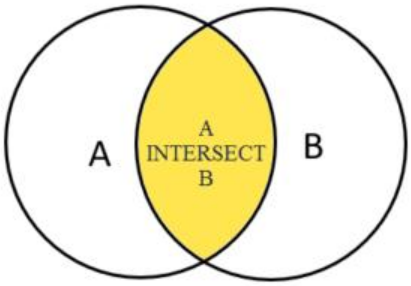


INTERSECT

PostgreSQL INTERSECT operator combines result sets of two or more SELECT statements into a single result set.

The INTERSECT operator returns any rows that are available in both result sets.

```
SELECT select_list
FROM A
INTERSECT
SELECT select_list
FROM B;
```

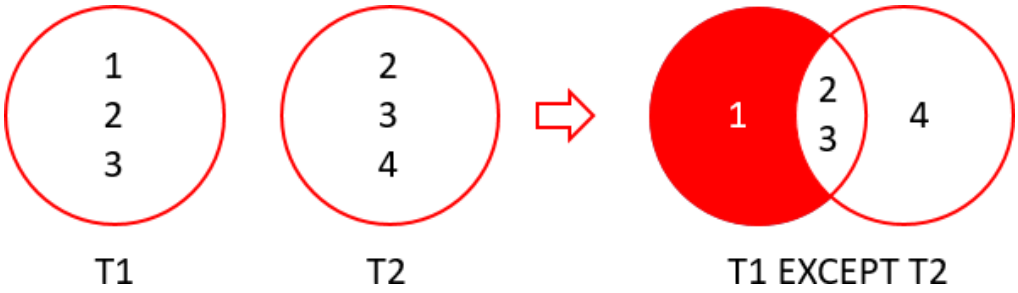


To using INTERSECT Please follow following terms.

1. The number and the order of the columns in the select list of both queries must be the same.
2. The data types must be compatible.

EXCEPT

EXCEPT clause/operator is used to combine two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement. This means EXCEPT returns only rows, which are not available in the second SELECT statement.




```
SELECT id, price FROM Books1
Except
SELECT id, price FROM Books2
```

GROUPING SETS

A grouping set is a set of columns by which you group by using the **GROUP BY** clause.

More: https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-grouping-sets/

CUBE

CUBE is a sub clause of the GROUP BY clause. The CUBE allows you to generate multiple grouping sets.

More: https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-cube/

ROLL-UP

ROLLUP is a sub clause of the GROUP BY clause that offers a shorthand for defining multiple grouping sets.

https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-rollup/

SUBQUERY

A SubQuery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

```
SELECT
        film_id,
        rental_rate
FROM
        film
WHERE
        rental_rate > (
                SELECT
                        AVG (rental_rate)
                FROM
                        film
        );
```

The query inside the brackets is called a subquery or an inner query. The query that contains the subquery is known as an outer query.

PostgreSQL data types

PostgreSQL supports the following data types:

1. Boolean
2. Character types such as char, varchar, and text.
3. Numeric types such as integer and floating-point number.
4. Temporal types such as date, time, timestamp, and interval
5. UUID for storing Universally Unique Identifiers
6. Array for storing array strings, numbers, etc.
7. JSON stores JSON data
8. hstore stores key-value pair
9. Special types such as network address and geometric data.