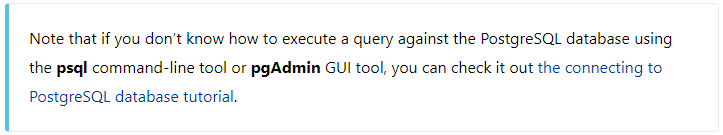
PostgreSQL SELECT



One of the most common tasks, when you work with the database, is to query data from tables by using the SELECT statement.

The SELECT statement is one of the most complex statements in PostgreSQL. It has many clauses that you can use to form a flexible query.

Because of its complexity, we will break it down into many shorter and easy-to-understand tutorials so that you can learn about each clause faster.

The SELECTstatement has the following clauses:

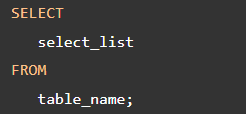
* Select distinct rows using [DISTINCT](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-select-distinct/) operator.
* Sort rows using[ORDER BY](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-order-by/) clause.
* Filter rows using [WHERE](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-where/) clause.
* Select a subset of rows from a table using [LIMIT](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-limit/) or [FETCH](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-fetch/) clause.
* Group rows into groups using [GROUP BY](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-group-by/) clause.
* Filter groups using [HAVING](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-having/) clause.
* Join with other tables using [joins](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-joins/) such as [INNER JOIN](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-inner-join/), [LEFT JOIN](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-left-join/), [FULL OUTER JOIN](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-full-outer-join/), [CROSS JOIN](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-cross-join/) clauses.
* Perform set operations using [UNION](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-union/), [INTERSECT](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-intersect/), and [EXCEPT](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-tutorial/postgresql-except/).

In this tutorial, you are going to focus on the SELECTand FROMclauses.

## **PostgreSQL SELECT statement syntax**

Let’s start with the basic form of the SELECT statement that retrieves data from a single table.

The following illustrates the syntax of the SELECT statement:

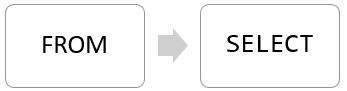


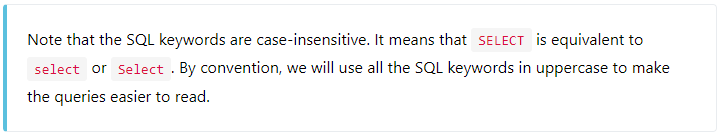
Let’s examine the SELECTstatement in more detail:

* First, specify a select list that can be a column or a list of columns in a table from which you want to retrieve data. 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 (\*) shorthand instead of specifying all the column names. The select list may also contain expressions or literal values.
* Second, specify the name of the table from which you want to query data after the FROM keyword.

The FROM clause is optional. If you do not query data from any table, you can omit the FROM clause in the SELECT statement.

PostgreSQL evaluates the FROM clause before the SELECT clause in the SELECT statement:

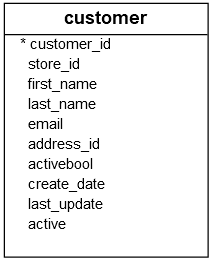




## **PostgreSQL SELECT examples**

Let’s take a look at some examples of using PostgreSQL SELECT statement.

We will use the following customer table in the [sampledatabase](https://www.postgresqltutorial.com/postgresql-sample-database/) for the demonstration.



### **1) Using PostgreSQL SELECT statement to query data from one column example**

This example uses the SELECT statement to find the first names of all customers from the customer table:

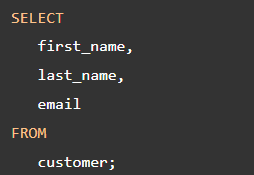


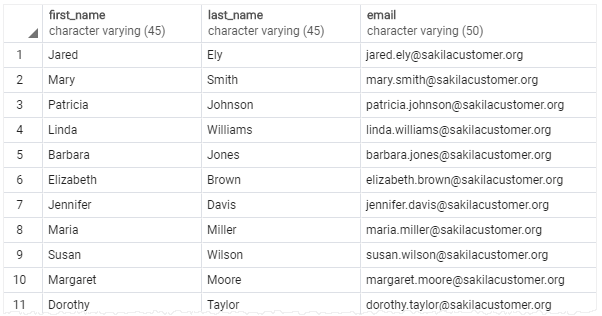


Notice that we added a semicolon (;) at the end of the SELECT statement. The semicolon is not a part of the SQL statement. It is used to signal PostgreSQL the end of an SQL statement. The semicolon is also used to separate two SQL statements.

### **2) Using PostgreSQL SELECT statement to query data from multiple columns example**

Suppose you just want to know the first name, last name and email of customers, you can specify these column names in the SELECT clause as shown in the following query:

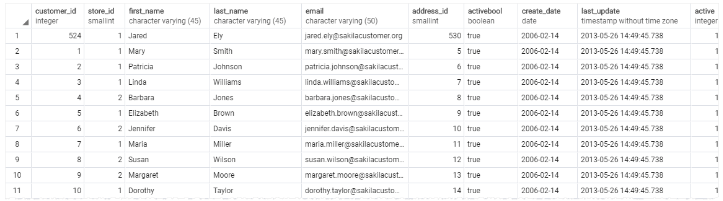




### **3) Using PostgreSQL SELECT statement to query data from all columns of a table example**

The following query uses the SELECT statement to select data from all columns of the customer table:





In this example, we used an asterisk (\*) in the SELECT clause, which is a shorthand for all columns. Instead of listing all columns in the SELECT clause, we just used the asterisk (\*) to save some typing.

However, it is not a good practice to use the asterisk (\*) in the SELECT statement when you embed SQL statements in the application code like [Python](https://www.postgresqltutorial.com/postgresql-python/), [Java](https://www.postgresqltutorial.com/postgresql-jdbc/), Node.js, or [PHP](https://www.postgresqltutorial.com/postgresql-php/) due to the following reasons:

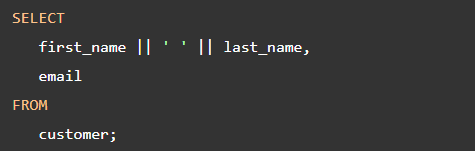
1. Database performance. Suppose you have a table with many columns and a lot of data, the SELECT statement with the asterisk (\*) shorthand will select data from all the columns of the table, which may not be necessary to the application.
2. Application performance. Retrieving unnecessary data from the database increases the traffic between the database server and application server. In consequence, your applications may be slower to respond and less scalable.

Because of these reasons, it is a good practice to explicitly specify the column names in the SELECT clause whenever possible to get only necessary data from the database.

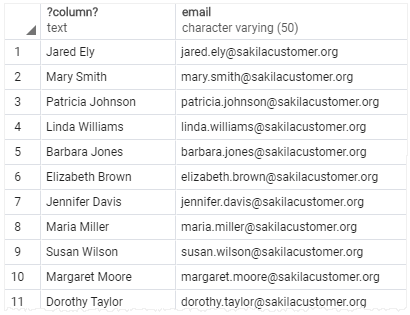
And you should only use the asterisk (\*) shorthand for the ad-hoc queries that examine data from the database.

### **4) Using PostgreSQL SELECT statement with expressions example**

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



Output:



In this example, we used the [concatenation operator](https://www.postgresqltutorial.com/postgresql-concat-function/) || to concatenate the first name, space, and last name of every customer.

You will learn how to use [column aliases](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-column-alias/) to assign expressions more meaningful names in the next tutorial.

### **5) Using PostgreSQL SELECT statement with expressions example**

The following example uses the SELECT statement with an expression. It omits the FROM clause:



Here is the output:



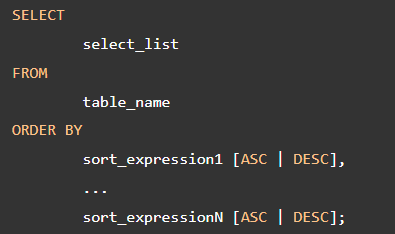
# PostgreSQL ORDER BY

## **Introduction to PostgreSQL ORDER BY clause**

When you query data from a table, the [SELECT](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-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](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-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.

The following illustrates the syntax of the ORDER BY clause:



In this syntax:

* First, specify a sort expression, which can be a column or an expression, that you want to sort after the ORDER BY keywords. If you want to sort the result set based on multiple columns or expressions, you need to place a comma (,) between two columns or expressions to separate them.
* Second, you use the ASC option to sort rows in ascending order and the DESC option to sort rows in descending order. If you omit the ASC or DESC option, the ORDER BY uses ASC by default.

PostgreSQL evaluates the clauses in the SELECT statment in the following order: FROM, SELECT, and ORDER BY:



Due to the order of evaluation, if you have a column alias in the SELECT clause, you can use it in the ORDER BY clause.

Let’s take some examples of using the PostgreSQL ORDER BY clause.

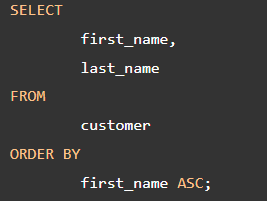
## **PostgreSQL ORDER BY examples**

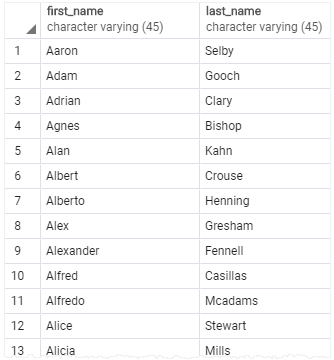
We will use the customer table in the [sample database](https://www.postgresqltutorial.com/postgresql-sample-database/) for the demonstration.

### 

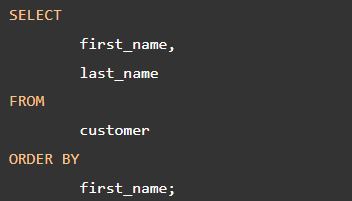
### **1) Using PostgreSQL ORDER BY clause to sort rows by one column**

The following query uses the ORDER BY clause to sort customers by their first names in ascending order:



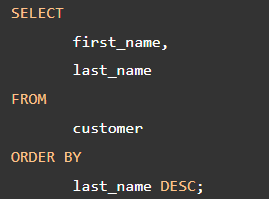


Since the ASC option is the default, you can omit it in the ORDER BY clause like this:



### **2) Using PostgreSQL ORDER BY clause to sort rows by one column in descending order**

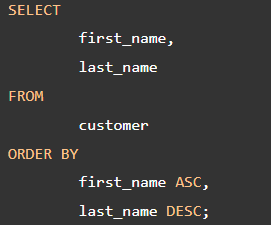
The following statement selects the first name and last name from the customer table and sorts the rows by values in the last name column in descending order:

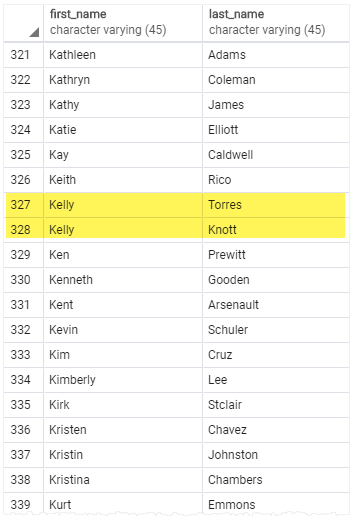




### **3) Using PostgreSQL ORDER BY clause to sort rows by multiple columns**

The following statement selects the first name and last name from the customer table and sorts the rows by the first name in ascending order and last name in descending order:





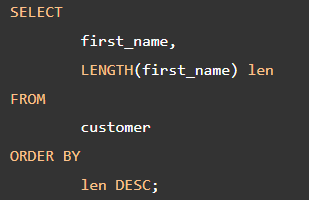
In this example, the ORDER BY clause sorts rows by values in the first name column first. And then it sorts the sorted rows by values in the last name column.

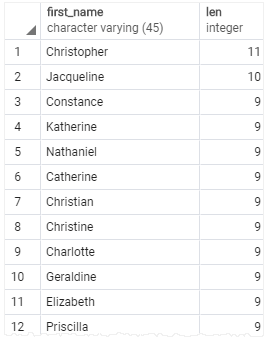
As you can see clearly from the output, two customers with the same first name Kelly have the last name sorted in descending order.

### **4) Using PostgreSQL ORDER BY clause to sort rows by expressions**

The [LENGTH()](https://www.postgresqltutorial.com/postgresql-length-function/) function accepts a string and returns the length of that string.

The following statement selects the first names and their lengths. It sorts the rows by the lengths of the first names:





Because the ORDER BY clause is evaluated after the SELECT clause, the column alias len is available and can be used in the ORDER BY clause.

## **PostgreSQL ORDER BY clause and NULL**

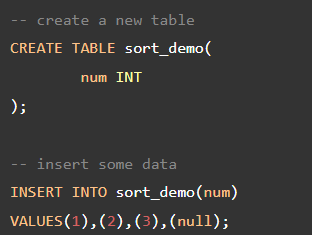
In the database world, NULL is a marker that indicates the missing data or the data is unknown at the time of recording.

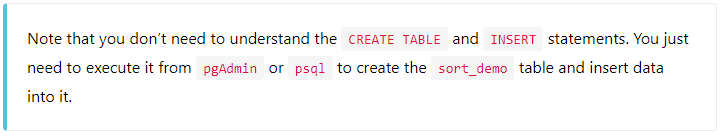
When you sort rows that contains NULL, you can specify the order of NULL with other non-null values by using the NULLS FIRST or NULLS LAST option of the ORDER BY clause:



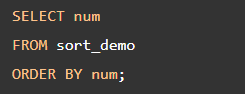
The NULLS FIRST option places NULL before other non-null values and the NULL LAST option places NULL after other non-null values.

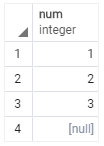
Let’s [create a table](https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-create-table/) for the demonstration.





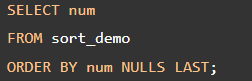
The following query returns data from the sort\_demo table:



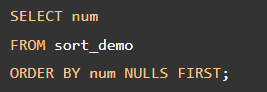


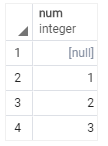
In this example, the ORDER BY clause sorts values in the num column of the sort\_demo table in ascending order. It places NULL after other values.

So if you use the ASC option, the ORDER BY clause uses the NULLS LAST option by default. Therefore, the following query returns the same result:

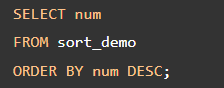


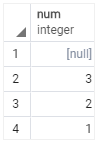
To place NULL before other non-null values, you use the NULLS FIRST option:





The following statement sorts values in the num column of the sort\_demo table in descending order:





As you can see clearly from the output, the ORDER BY clause with the DESC option uses the NULLS FIRST by default.

To reverse the order, you can use the NULLS LAST option:

