











- - Introduction to databases
 - [What are databases?](#)
 - [Comparing database types: how database types evolved to meet different needs](#)
 - [Introduction to database schemas](#)
 -
 -
 - Data Modeling
 - [Intro \(Don't Panic\)](#)
 - [Know Your Problem Space](#)
 - [Tables, Tuples, Types](#)
 - [Correctness and Constraints](#)
 - [Making Connections](#)
 - [Functional Units](#)
 - [In Vivo: Information Ecosystems](#)
 -
 - Database types
 -  [Relational Databases](#)
 -
 - PostgreSQL
 - [The benefits of PostgreSQL](#)
 - [Getting to know PostgreSQL](#)
 - [5 ways to host PostgreSQL databases](#)
 - [Setting up a local PostgreSQL database](#)
 - [Connecting to PostgreSQL databases](#)
 -  [Authentication and authorization](#)
 - [How to create and delete databases and tables in PostgreSQL](#)
 - [An introduction to PostgreSQL data types](#)
 - [An introduction to PostgreSQL column and table constraints](#)
 -  [Inserting and modifying data](#)
 -  [Reading and querying data](#)
 -  [Short guides](#)
 - [How to use single and double quotes in PostgreSQL](#)
 -
 -
 - MySQL
 - [5 ways to host MySQL databases](#)
 - [Setting up a local MySQL database](#)
 - [Connecting to MySQL databases](#)
 -  [Authentication and authorization](#)
 - [How to create and delete databases and tables in MySQL](#)
 - [An introduction to MySQL data types](#)
 - [An introduction to MySQL column and table constraints](#)
 -  [Inserting and modifying data](#)
 -  [Reading and querying data](#)
 -
 - SQLite
 - [Setting up a local SQLite database](#)
 - [Importing and exporting data in SQLite](#)
 -
 - Microsoft SQL Server
 - [Setting up a local SQL Server database](#)
 -
 - Database tools | SQL, MySQL, Postgres | Prisma's Data Guide
 - [Top 11 Node.js ORMs, Query Builders & Database Libraries in 2021](#)
 - [Top 8 TypeScript ORMs, Query Builders, & Database Libraries: Evaluating Type Safety](#)
 -
 - Managing databases | Prisma's Data Guide
 - [Troubleshooting Database Outages and Connection Issues](#)
 - [How to Spot Bottlenecks in Performance](#)
 - [Syncing Development Databases Between Team Members](#)
 -
 - -

PostgreSQL / Short guides

How to use single and double quotes in PostgreSQL

CONTENT

- [Introduction](#)
- [Double quotes](#)
- [Single quotes](#)
- [Additional Examples](#)
- [Conclusion](#)

Share on



Introduction

Single and double quotation marks are used within PostgreSQL for different purposes. When getting started working with these databases, it can be difficult to understand the differences between these two types of quotes and how to use them correctly.

In this guide, we'll take a look at how PostgreSQL interprets both single and double quotes. We'll talk about the side effects of using various quotes and provide examples of scenarios where each are used.

Double quotes

In PostgreSQL, double quotes (like "a red dog") are always used to denote *delimited identifiers*. In this context, an *identifier* is the name of an object within PostgreSQL, such as a table name or a column name. Delimited identifiers are identifiers that have a specifically marked beginning and end.

For example, to select all of the information from a `customer` table, you could type the following. Here, the table name is encapsulated in double quotes.

```
SELECT * FROM "customer";
```

While double quotes indicate an identifier, not all identifiers use double quotes. For examples like the above, it is much more common to see the identifier unquoted entirely:

```
SELECT * FROM customer;
```

Quoting Identifiers and the problem of case sensitivity

While the two formats used above both work correctly for a `customer` table, there *are* important differences.

Unquoted identifiers (like the second version) are *case insensitive*. This means that PostgreSQL will recognize `customer`, `Customer`, and `CUSTOMER` as the same object.

However, quoted identifiers are *case sensitive*. This leads to PostgreSQL treating `"CUSTOMER"` and `"customer"` as entirely different objects.

This difference allows you to create identifiers that would otherwise not be legal within PostgreSQL. For instance, if you need to create a column with a period in it, you would need to use double quotes so that PostgreSQL interprets it correctly.

However, keep in mind that this can lead to usability issues if not used carefully. For example, suppose you use double quotes to preserve upper-case characters in the identifier when creating an object. From then on, you will be required to use double quotes to match that case every time you reference it.

Use double quotes sparingly for better compatibility, especially when creating objects. If you want to use double quotes, keep in mind that the case problem does not arise if you use double quotes with fully lower-cased identifiers.

Single quotes

Single quotes, on the other hand, are used to indicate that a token is a string. This is used in many different contexts throughout PostgreSQL.

In general, if an item is a string, it needs to be surrounded by single quotation marks. Keep in mind that when creating and referencing objects, identifiers must be represented by unquoted or double quoted text.

For example, here we use single quotes to insert a string into a `text` field within a database:

```
INSERT INTO my_table(text) VALUES ('hello there!');
```

If we wanted to, we could optionally use double quotes around the identifiers, like this:

```
INSERT INTO "my_table"("text") VALUES ('hello there!');
```

The two statements above are the same, assuming that both `my_table` and the `text` column were unquoted or lower-case when created.

If you need to include a single quote *within* your string, you can do so by instead inserting two sequential single quotes (Two single quotes, not a double quote).

For example, you could insert another string with an embedded single quote by typing:

```
INSERT INTO my_table(text) VALUES ('How's it going?');
```

Single quoted strings are the appropriate means of assigning or checking the value of strings.

Additional Examples

Here, we'll go over a few more examples to help clarify why different parts of an SQL statement use different quoting methods.

Creating a new role with a password

First, we can look at a role creation statement:

```
CREATE ROLE "user1" WITH LOGIN PASSWORD 'secretpassword';
```

The statement has two quoted components:

- `user1` is in double quotes because it will reference a role, which is an identifier.
- `secretpassword` is a string that will exist in a table column. It is therefore a string value and needs single quotations.

Checking if your current user has privileges necessary to manage roles

The next query determines whether the role the user is currently signed in as has the privileges to manage roles within the database cluster:

```
SELECT 'Yes' AS "Can I manage roles?" FROM pg_roles WHERE rolname = :'USER' AND (rolsuper OR rolcreatorole);
```

There are a few different quoting patterns in use here:

- `Yes` is in single quotes because it's a value that will be printed within the context of a column value.

- `Can I manage roles?` is in double quotes because it will be the name of the column in the constructed table, and is therefore an identifier.
- `USER` is in single quotes because we are checking the value of a string.
- The `:'USER'` syntax is a special format used to interpolate the `psql USER` variable while placing the resulting value in single quotes.

Conclusion

In this guide, we took a look at both single and double quoting in PostgreSQL. Double quotes are used to indicate identifiers within the database, which are objects like tables, column names, and roles. In contrast, single quotes are used to indicate string literals.

Learning how to correctly use quotes in PostgreSQL, as well as the implications of different quotation choices, will help you avoid frustrating mistakes. While the quotation rules may not correspond to other systems you may be familiar with, they are useful once you understand their distinct purposes.

Previous [←](#)

[Using joins to combine data from different tables in PostgreSQL](#)

Next

[5 ways to host MySQL databases](#)

[→](#)

[Edit this page on GitHub](#)

Get notified of new articles

Sign up to get notified by email when new content is added to Prisma's Data Guide.

Subscribe [→](#)



Prisma's Data Guide

A growing library of articles focused on making databases more approachable.

Made with [♥](#) by [Prisma](#)