

[Documentation](#) → [PostgreSQL 13](#)Supported Versions: [Current \(13\)](#) / [12](#) / [11](#) / [10](#) / [9.6](#) / [9.5](#)Development Versions: [devel](#)Unsupported versions: [9.4](#) / [9.3](#) / [9.2](#) / [9.1](#) / [9.0](#) / [8.4](#) / [8.3](#) /[8.2](#) / [8.1](#) / [8.0](#) / [7.4](#) / [7.3](#) / [7.2](#) / [7.1](#)

Search the documentation for...



8.3. Character Types

Chapter 8. Data Types

[Prev](#)[Up](#)[Home](#)[Next](#)

8.3. Character Types

Table 8.4. Character Types

Name	Description
<code>character varying(n)</code> , <code>varchar(n)</code>	variable-length with limit
<code>character(n)</code> , <code>char(n)</code>	fixed-length, blank padded
<code>text</code>	variable unlimited length

[Table 8.4](#) shows the general-purpose character types available in PostgreSQL.

SQL defines two primary character types: `character varying(n)` and `character(n)`, where `n` is a positive integer. Both of these types can store strings up to `n` characters (not bytes) in length. An attempt to store a longer string into a column of these types will result in an error, unless the excess characters are all spaces, in which case the string will be truncated to the maximum length. (This somewhat bizarre exception is required by the SQL standard.) If the string to be stored is shorter than the declared length, values of type `character` will be space-padded; values of type `character varying` will simply store the shorter string.

If one explicitly casts a value to `character varying(n)` or `character(n)`, then an over-length value will be truncated to `n` characters without raising an error. (This too is required by the SQL standard.)

The notations `varchar(n)` and `char(n)` are aliases for `character varying(n)` and `character(n)`, respectively. `character` without length specifier is equivalent to `character(1)`. If `character varying` is used without length specifier, the type accepts strings of any size. The latter is a PostgreSQL extension.

In addition, PostgreSQL provides the `text` type, which stores strings of any length. Although the type `text` is not in the SQL standard, several other SQL database management systems have it as well.

Values of type `character` are physically padded with spaces to the specified width `n`, and are stored and displayed that way. However, trailing spaces are treated as semantically insignificant and disregarded when comparing two values of type `character`. In collations where whitespace is significant, this behavior can produce unexpected results; for example `SELECT 'a '::CHAR(2) collate "C" < E' a\n' ::CHAR(2)` returns true, even though C locale would consider a space to be greater than a newline. Trailing spaces are removed when converting a `character` value to one of the other string types. Note that trailing spaces *are* semantically significant in `character varying` and `text` values, and when using pattern matching, that is `LIKE` and regular expressions.

The storage requirement for a short string (up to 126 bytes) is 1 byte plus the actual string, which includes the space padding in the case of `character`. Longer strings have 4 bytes of overhead instead of 1. Long strings are compressed by the system automatically, so the physical requirement on disk might be less. Very long values are also stored in background tables so that they do not interfere with rapid access to shorter column values. In any case, the longest possible character string that can be stored is about 1 GB. (The maximum value that will be allowed for `n` in the data type declaration is less than that. It wouldn't be useful to change this because with multibyte character encodings the number of characters and bytes can be quite different. If you desire to store long strings with no specific upper limit, use `text` or `character varying` without a length specifier, rather than making up an arbitrary length limit.)

Tip

There is no performance difference among these three types, apart from increased storage space when using the blank-padded type, and a few extra CPU cycles to check the length when storing into a length-constrained column. While `character(n)` has performance advantages in some other database systems, there is no such advantage in PostgreSQL; in fact `character(n)` is usually the slowest of the three because of its additional storage costs. In most situations `text` or `character varying` should be used instead.

Refer to [Section 4.1.2.1](#) for information about the syntax of string literals, and to [Chapter 9](#) for information about available operators and functions. The database character set determines the character set used to store textual values; for more information on character set support, refer to [Section 23.3](#).

Example 8.1. Using the Character Types

```
CREATE TABLE test1 (a character(4));
INSERT INTO test1 VALUES ('ok');
SELECT a, char_length(a) FROM test1; -- (1)
```

a	char_length
ok	2

```
CREATE TABLE test2 (b varchar(5));
INSERT INTO test2 VALUES ('ok');
INSERT INTO test2 VALUES ('good ');
INSERT INTO test2 VALUES ('too long');
ERROR: value too long for type character varying(5)
INSERT INTO test2 VALUES ('too long'::varchar(5)); -- explicit truncation
SELECT b, char_length(b) FROM test2;
```

b	char_length
ok	2
good	5
too l	5

(1) The `char_length` function is discussed in [Section 9.4](#).

There are two other fixed-length character types in PostgreSQL, shown in [Table 8.5](#). The `name` type exists *only* for the storage of identifiers in the internal system catalogs and is not intended for use by the general user. Its length is currently defined as 64 bytes (63 usable characters plus terminator) but should be referenced using the constant `NAMEDATALEN` in C source code. The length is set at compile time (and is therefore adjustable for special uses); the default maximum length might change in a future release. The type `"char"` (note the quotes) is different from `char(1)` in that it only uses one byte of storage. It is internally used in the system catalogs as a simplistic enumeration type.

Table 8.5. Special Character Types

Name	Storage Size	Description
"char"	1 byte	single-byte internal type
name	64 bytes	internal type for object names

[Prev](#)

8.2. Monetary Types

[Up](#)

[Home](#)

[Next](#)

8.4. Binary Data Types

Submit correction

If you see anything in the documentation that is not correct, does not match your experience with the particular feature or requires further clarification, please use [this form](#) to report a documentation issue.



[Privacy Policy](#) | [Code of Conduct](#) | [About PostgreSQL](#) | [Contact](#)
 Copyright © 1996-2020 The PostgreSQL Global Development Group