

Search Documentation: [Home](#) → [Documentation](#) → [Manuals](#) → [PostgreSQL 9.3](#)This page in other versions: [9.2](#) / [9.3](#) / [9.4](#) / [9.5](#) / [current \(9.6\)](#) | Development versions: [devel](#) | Unsupported versions: [7.1](#) / [7.2](#) / [7.3](#) / [7.4](#) / [8.0](#) / [8.1](#) / [8.2](#) / [8.3](#) / [8.4](#) / [9.0](#) / [9.1](#)[PostgreSQL 9.3.15 Documentation](#)[Prev](#) [Up](#) [Next](#)

CREATE SEQUENCE

Name

CREATE SEQUENCE -- define a new sequence generator

Synopsis

```
CREATE [ TEMPORARY | TEMP ] SEQUENCE name [ INCREMENT [ BY ] increment ]
      [ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]
      [ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE ]
      [ OWNED BY { table_name.column_name | NONE } ]
```

Description

CREATE SEQUENCE creates a new sequence number generator. This involves creating and initializing a new special single-row table with the name name. The generator will be owned by the user issuing the command.

If a schema name is given then the sequence is created in the specified schema. Otherwise it is created in the current schema. Temporary sequences exist in a special schema, so a schema name cannot be given when creating a temporary sequence. The sequence name must be distinct from the name of any other sequence, table, index, view, or foreign table in the same schema.

After a sequence is created, you use the functions `nextval`, `currval`, and `setval` to operate on the sequence. These functions are documented in [Section 9.16](#).

Although you cannot update a sequence directly, you can use a query like:

```
SELECT * FROM name;
```

to examine the parameters and current state of a sequence. In particular, the `last_value` field of the sequence shows the last value allocated by any session. (Of course, this value might be obsolete by the time it's printed, if other sessions are actively doing `nextval` calls.)

Parameters

TEMPORARY or TEMP

If specified, the sequence object is created only for this session, and is automatically dropped on session exit. Existing permanent sequences with the same name are not visible (in this session) while the temporary sequence exists, unless they are referenced with schema-qualified names.

name

The name (optionally schema-qualified) of the sequence to be created.

increment

The optional clause `INCREMENT BY increment` specifies which value is added to the current sequence value to create a new value. A positive value will make an ascending sequence, a negative one a descending sequence. The default value is 1.

minvalue

`NO MINVALUE`

The optional clause `MINVALUE minvalue` determines the minimum value a sequence can generate. If this clause is not supplied or `NO MINVALUE` is specified, then defaults will be used. The defaults are 1 and $-2^{63}-1$ for ascending and descending sequences, respectively.

maxvalue

`NO MAXVALUE`

The optional clause `MAXVALUE maxvalue` determines the maximum value for the sequence. If this clause is not supplied or `NO MAXVALUE` is specified, then default values will be used. The defaults are $2^{63}-1$ and -1 for ascending and descending sequences, respectively.

start

The optional clause `START WITH start` allows the sequence to begin anywhere. The default starting value is `minvalue` for ascending sequences and `maxvalue` for descending ones.

cache

The optional clause `CACHE cache` specifies how many sequence numbers are to be preallocated and stored in memory for faster access. The minimum value is 1 (only one value can be generated at a time, i.e., no cache), and this is also the default.

CYCLE

`NO CYCLE`

The `CYCLE` option allows the sequence to wrap around when the `maxvalue` or `minvalue` has been reached by an ascending or descending sequence respectively. If the limit is reached, the next number generated will be the `minvalue` or `maxvalue`, respectively.

If `NO CYCLE` is specified, any calls to `nextval` after the sequence has reached its maximum value will return an error. If neither `CYCLE` or `NO CYCLE` are specified, `NO CYCLE` is the default.

`OWNED BY table_name.column_name`

`OWNED BY NONE`

The `OWNED BY` option causes the sequence to be associated with a specific table column, such that if that column (or its whole table) is dropped, the sequence will be automatically dropped as well. The specified

table must have the same owner and be in the same schema as the sequence. `OWNED BY NONE`, the default, specifies that there is no such association.

Notes

Use `DROP SEQUENCE` to remove a sequence.

Sequences are based on `bigint` arithmetic, so the range cannot exceed the range of an eight-byte integer (-9223372036854775808 to 9223372036854775807).

Unexpected results might be obtained if a `cache` setting greater than one is used for a sequence object that will be used concurrently by multiple sessions. Each session will allocate and cache successive sequence values during one access to the sequence object and increase the sequence object's `last_value` accordingly. Then, the next `cache-1` uses of `nextval` within that session simply return the preallocated values without touching the sequence object. So, any numbers allocated but not used within a session will be lost when that session ends, resulting in "holes" in the sequence.

Furthermore, although multiple sessions are guaranteed to allocate distinct sequence values, the values might be generated out of sequence when all the sessions are considered. For example, with a `cache` setting of 10, session A might reserve values 1..10 and return `nextval=1`, then session B might reserve values 11..20 and return `nextval=11` before session A has generated `nextval=2`. Thus, with a `cache` setting of one it is safe to assume that `nextval` values are generated sequentially; with a `cache` setting greater than one you should only assume that the `nextval` values are all distinct, not that they are generated purely sequentially. Also, `last_value` will reflect the latest value reserved by any session, whether or not it has yet been returned by `nextval`.

Another consideration is that a `setval` executed on such a sequence will not be noticed by other sessions until they have used up any preallocated values they have cached.

Examples

Create an ascending sequence called `serial`, starting at 101:

```
CREATE SEQUENCE serial START 101;
```

Select the next number from this sequence:

```
SELECT nextval('serial');
```

```
nextval
-----
101
```

Select the next number from this sequence:

```
SELECT nextval('serial');
```

```
nextval
-----
102
```

Use this sequence in an `INSERT` command:

```
INSERT INTO distributors VALUES (nextval('serial'), 'nothing');
```

Update the sequence value after a `COPY FROM`:

```
BEGIN;  
COPY distributors FROM 'input_file';  
SELECT setval('serial', max(id)) FROM distributors;  
END;
```

Compatibility

CREATE SEQUENCE conforms to the SQL standard, with the following exceptions:

- The standard's AS <data type> expression is not supported.
- Obtaining the next value is done using the nextval() function instead of the standard's NEXT VALUE FOR expression.
- The OWNED BY clause is a PostgreSQL extension.

See Also

[ALTER SEQUENCE](#), [DROP SEQUENCE](#)

Prev	Home	Next
CREATE SCHEMA	Up	CREATE SERVER

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](#) | [About PostgreSQL](#)

Copyright © 1996-2016 The PostgreSQL Global Development Group