

Data Organization

Physical Structure



Copyright

© Postgres Professional, 2017–2025

Authors Egor Rogov, Pavel Luzanov, Ilya Bashtanov, Igor Gnatyuk

Translated by Liudmila Mantrova, Alexander Meleshko, Elena Sharafutdinova

Photo by: Oleg Bartunov (Phu monastery, Bhrikuti summit, Nepal)

Use of course materials

Non-commercial use of course materials (presentations, demonstrations) is allowed without restrictions. Commercial use is possible only with the written permission of Postgres Professional. It is prohibited to make changes to the course materials.

Feedback

Please send your feedback, comments and suggestions to:

edu@postgrespro.ru

Disclaimer

Postgres Professional assumes no responsibility for any damages and losses, including loss of income, caused by direct or indirect, intentional or accidental use of course materials. Postgres Professional company specifically disclaims any warranties on course materials. Course materials are provided “as is,” and Postgres Professional company has no obligations to provide maintenance, support, updates, enhancements, or modifications.

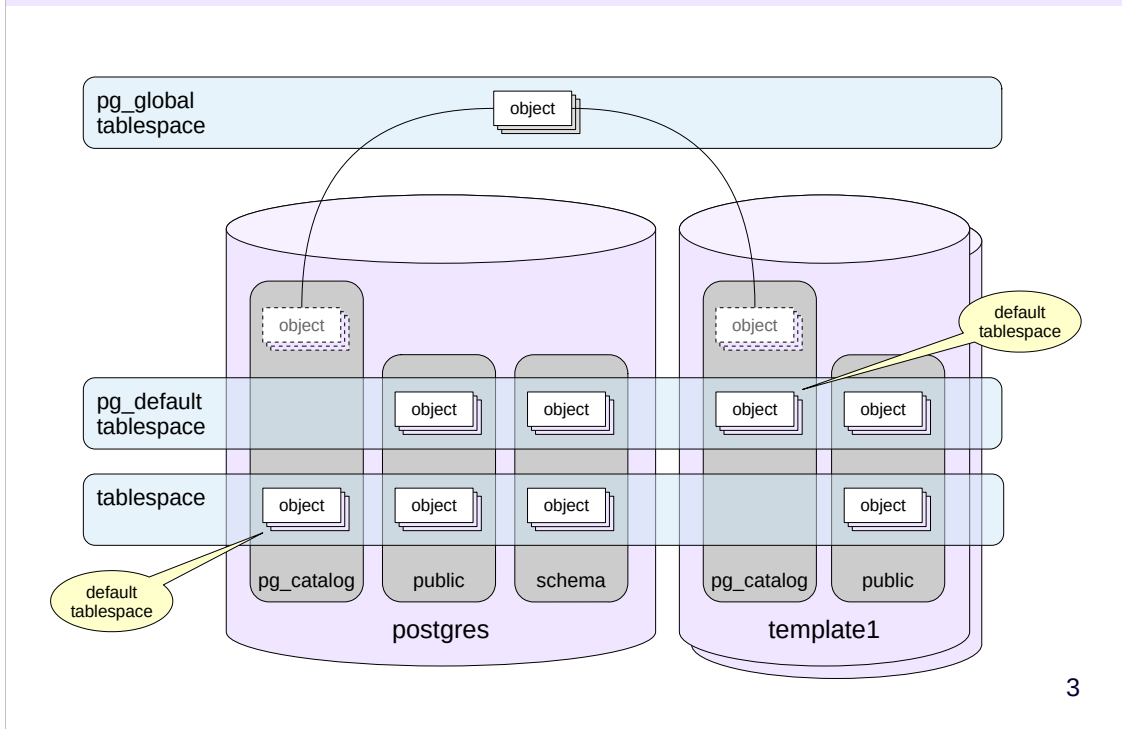
Tablespaces and Catalogs

Files and Pages

Forks: Main, Visibility Map, Free Space Map

TOAST

Tablespaces



Tablespaces are used to organize the physical storage of data and determine the location of data in the file system. For example, one tablespace can be mapped to slow disks for archived data, and another on fast disks with frequent activity.

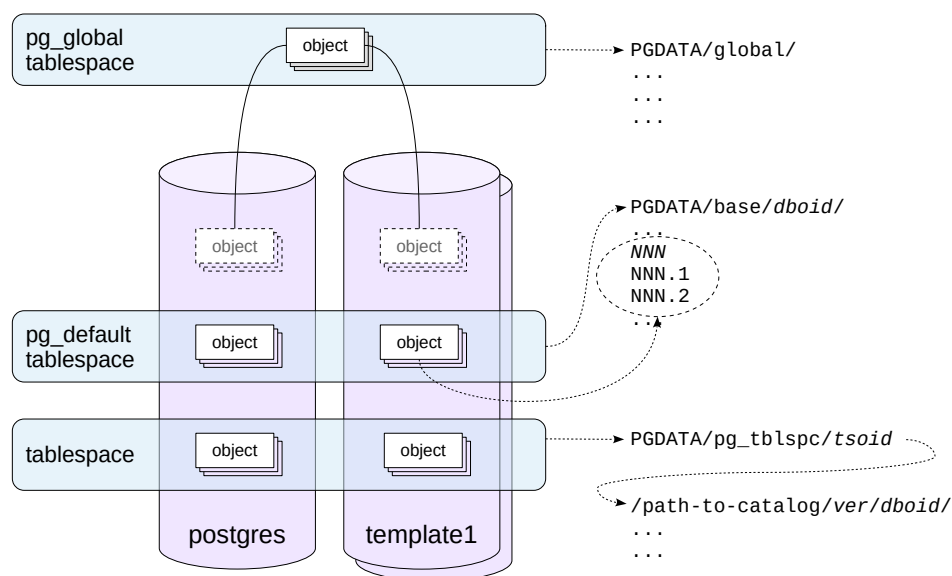
On cluster initialization, two tablespaces are created: `pg_default` and `pg_global`.

A tablespace can be used by multiple databases, and a database can use multiple tablespaces at once.

Each database has a default tablespace where all database objects are created (unless explicitly set otherwise). System catalog objects are also stored there. Databases will use the `pg_default` tablespace as their default, unless another one is set by the user.

The `pg_global` tablespace is special as it stores only those objects that are shared by the whole cluster.

Catalogs



4

Essentially, a tablespace is a reference to the directory in which the data is located. The standard tablespaces `pg_global` and `pg_default` are always located in `PGDATA/global/` and `PGDATA/base/`, respectively. When a custom tablespace is created, an arbitrary directory can be specified. For (internal) convenience, PostgreSQL also creates a symbolic link to the directory in `PGDATA/pg_tblspc/`.

The `PGDATA/base/` directory comprises different subdirectories for each database (unlike `PGDATA/global/`, which stores data referring to the whole cluster).

Inside a custom tablespace directory, there is another level of subdirectories for different PostgreSQL server versions. This is helpful during server version upgrade.

Finally, these directories are where the actual objects are stored, one or more files per object.

Each such file, called a *segment*, takes up no more than 1GB by default (this size can be changed when building the server). This is also why several files can correspond to each object. It is necessary to take into account the impact of a potentially large number of files on the file system used.

<https://postgrespro.com/docs/postgresql/16/storage-file-layout>

Using Tablespaces

Initially, a cluster contains two tablespaces. You can learn about them from the system catalog:

```
=> SELECT spcname FROM pg_tablespace;
```

```
   spcname
-----
pg_default
pg_global
(2 rows)
```

Naturally, this is one of the cluster's global tables.

A command in psql that does a similar thing:

```
=> \db
```

```
      List of tablespaces
   Name   | Owner   | Location
-----+-----+-----
pg_default | postgres |
pg_global  | postgres |
(2 rows)
```

A new tablespace needs an empty directory owned by the same OS user who has started the database server:

```
=> \! sudo mkdir /var/lib/postgresql/ts_dir
```

Change the directory owner:

```
=> \! sudo chown postgres /var/lib/postgresql/ts_dir
```

The tablespace can be created now:

```
=> CREATE TABLESPACE ts LOCATION '/var/lib/postgresql/ts_dir';
```

CREATE TABLESPACE

```
=> \db
```

```
      List of tablespaces
   Name   | Owner   | Location
-----+-----+-----
pg_default | postgres |
pg_global  | postgres |
ts         | student  | /var/lib/postgresql/ts_dir
(3 rows)
```

When creating a database, you can set a tablespace to be used by default:

```
=> CREATE DATABASE data_physical TABLESPACE ts;
```

CREATE DATABASE

```
=> \c data_physical
```

You are now connected to database "data_physical" as user "student".

All new database objects will be created in this tablespace by default.

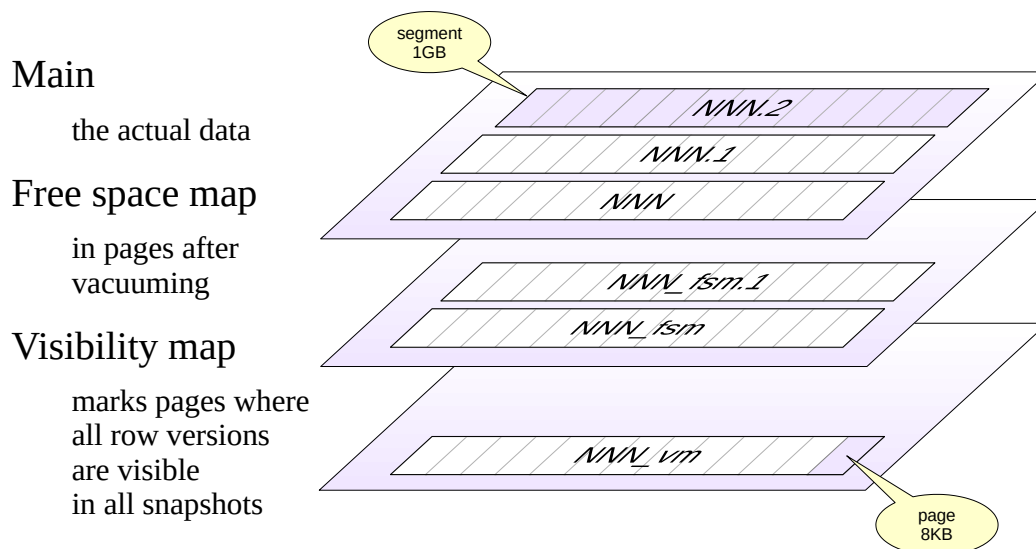
```
=> CREATE TABLE t(id integer PRIMARY KEY, s text);
```

CREATE TABLE

```
=> INSERT INTO t(id, s)
    SELECT id, id::text FROM generate_series(1,100_000) id;
```

INSERT 0 100000

Forks and Files



6

Usually, each object has several corresponding *forks*. Each fork is a set of *segments* (that is, one or more files). The segment files that make up the forks are logically divided into *pages*, usually 8KB each (this size can be set for the entire cluster only when building the server). The file pages of all forks are read from disk in a completely uniform way through a shared buffer cache mechanism. The forks themselves contain different information and have different internal page structure.

The **main** fork contains the actual data, such as table row versions and index records.

The *vm* fork is the **visibility (bit)map**. It marks pages containing only the current row versions visible in all data snapshots. In other words, these pages have not been changed for so long that they have been completely vacuumed of obsolete tuples.

The visibility map is used to optimize vacuuming (marked pages do not need to be vacuumed) and to speed up index access. Row versioning information exists only for tables, but not for indexes. Therefore, after getting a link to the required row version from an index, you first need to check its visibility by reading the corresponding table page. But if the index itself already contains all the necessary columns, and the page is marked in the visibility map, then reading the table page can be skipped.

The *fsm* fork is the **free space map**. It marks the available space inside the pages, which is created, for example, during vacuuming. This map is used when inserting new row versions in order to quickly find the appropriate page.

Forks and Files

Run vacuum to ensure that all forks exist in the new table:

```
=> VACUUM t;
```

VACUUM

You can check where an object's files are stored:

```
=> SELECT pg_relation_filepath('t');

      pg_relation_filepath
-----
pg_tblspc/16386/PG_16_202307071/16387/16388
(1 row)
```

Let's check the files (names and sizes in bytes):

```
student$ sudo bash -c 'cd /var/lib/postgresql/16/main/pg_tblspc/16386/PG_16_202307071/16387; ls -l 16388*'
-rw-r--r-- 1 postgres postgres 4423680 Nov 27 14:19 16388
-rw-r--r-- 1 postgres postgres  24576 Nov 27 14:19 16388_fsm
-rw-r--r-- 1 postgres postgres   8192 Nov 27 14:19 16388_vm
```

The files represent the three forks: main, free space map, and visibility map.

Objects can be moved between tablespaces, but (unlike with schemas) this involves physically moving the data around.

```
=> ALTER TABLE t SET TABLESPACE pg_default;
```

ALTER TABLE

```
=> SELECT pg_relation_filepath('t');

      pg_relation_filepath
-----
base/16387/16395
(1 row)
```

Size of Objects

There are several ways to find the size of a database and objects in it.

```
=> SELECT pg_database_size('data_physical');

      pg_database_size
-----
14422499
(1 row)
```

Prettify the number for readability:

```
=> SELECT pg_size_pretty(pg_database_size('data_physical'));

      pg_size_pretty
-----
14 MB
(1 row)
```

Total table size (with all indexes):

```
=> SELECT pg_size_pretty(pg_total_relation_size('t'));

      pg_size_pretty
-----
6568 kB
(1 row)
```

Table size (separately):

```
=> SELECT pg_size_pretty(pg_table_size('t'));
```

```
pg_size_pretty
-----
4360 kB
(1 row)
```

Index size (separately):

```
=> SELECT pg_size_pretty(pg_indexes_size('t'));
```

```
pg_size_pretty
-----
2208 kB
(1 row)
```

You can get the size of each individual fork, for example:

```
=> SELECT pg_size_pretty(pg_relation_size('t','main'));
```

```
pg_size_pretty
-----
4320 kB
(1 row)
```

Another function shows the size of a tablespace on the disk:

```
=> SELECT pg_size_pretty(pg_tablespace_size('ts'));
```

```
pg_size_pretty
-----
9740 kB
(1 row)
```


A row version must fit into one page

- some of the attributes can be compressed
- or moved into a separate TOAST table
- or both compressed and moved

TOAST table

- pg_toast schema
- supported by its own index
- oversized attributes are divided into parts smaller than a page
- accessed by querying a corresponding oversized attribute
- native versioning
- works transparently for the application

Any row version in PostgreSQL must fit entirely into one page. Oversized row versions are stored using TOAST, The Oversized Attributes Storage Technique. It implies several strategies. Firstly, the oversized attribute can be compressed so that the row version fits into the page. If this fails, the attribute can be moved into a separate service table. Both approaches can be applied at the same time.

Any primary table can have a separate TOAST table (with a dedicated index) created for it, if necessary. Such tables and indexes are located in a separate pg_toast schema and therefore are usually hidden.

The row versions in the TOAST table must also fit into one page each, so longer values are split into multiple chunks and transparently “glued” together by PostgreSQL when the application demands.

TOAST tables are used only when oversized values are accessed. The tables have their own row versioning. Whenever a data update in the main table does not modify the oversized value in the TOAST table, the new row version in the main table will refer to the same old TOAST value, saving disk space.

<https://postgrespro.com/docs/postgresql/16/storage-toast>

TOAST

Add a very long string to the table:

```
=> INSERT INTO t(id, s)
SELECT 0, string_agg(id::text, '.') FROM generate_series(1,5000) AS id;
```

```
INSERT 0 1
```

Will the table size change?

```
=> SELECT pg_size_pretty(pg_table_size('t'));
```

```
pg_size_pretty
-----
4416 kB
(1 row)
```

Yes. What about the main fork size?

```
=> SELECT pg_size_pretty(pg_relation_size('t','main'));
```

```
pg_size_pretty
-----
4320 kB
(1 row)
```

No.

Since the tuple won't fit into a single page, the value of s will be sliced and put into a toast table. You can find this table in the system catalog (using the regclass type to get the table name from its oid):

```
=> SELECT oid, reltoastrelid::regclass::text FROM pg_class WHERE relname='t';
```

```
oid | reltoastrelid
-----+-----
16388 | pg_toast.pg_toast_16388
(1 row)
```

The character string is sliced, and PostgreSQL will reassemble it as needed:

```
=> SELECT chunk_id, chunk_seq, left(chunk_data::text,45) AS chunk_data
FROM pg_toast.pg_toast_16388;
```

chunk_id	chunk_seq	chunk_data
16398	0	\x545d000000312e322e332e342e00352e362e372e382
16398	1	\x392e353161ff31002e3531322e353133002e3531342
16398	2	\xe215e216e217e218abe219e11a30e11b30e11c30e11
16398	3	\x11f4aa3611f43611f43611f43611f4aa3611f43611f
16398	4	\xf43211f4325511f43211f43211f43211f4325511f43
16398	5	\xf43811f43811f43811f4385511f43811f43811f4381
16398	6	\x11f4aa3411f43411f43411f43411f4aa3411f43411f
16398	7	\x0132370132aa370132370132370132370132aa38013
16398	8	\xf43611f43611f43611f4365511f43611f43611f4361

(9 rows)

Finally, remove the database.

```
=> \c postgres
```

You are now connected to database "postgres" as user "student".

```
=> DROP DATABASE data_physical;
```

```
DROP DATABASE
```

Since there are no objects left in the tablespace, it too can be dropped:

```
=> DROP TABLESPACE ts;
```

```
DROP TABLESPACE
```

```
=> \! sudo rmdir /var/lib/postgresql/ts_dir
```


On the physical level

- data is distributed across tablespaces (directories)
- an object comprises several forks
- a fork consists of one or more segments

Tablespaces are managed by the administrator

Forks, files, and TOAST are all PostgreSQL internals

1. Create a new database and connect to it. Create a tablespace *ts*. Create a table *t* in the *ts* tablespace and add several rows to it.
2. Calculate the space occupied by the database, table, and tablespaces *ts* and *pg_default*.
3. Move the table to the *pg_default* tablespace.
How has the space used by tablespaces changed?
4. Delete the tablespace *ts*.

1. Tablespaces and Tables

Create a database:

```
=> CREATE DATABASE data_physical;
```

```
CREATE DATABASE
```

```
=> \c data_physical
```

You are now connected to database "data_physical" as user "student".

Tablespace:

```
student$ sudo mkdir /var/lib/postgresql/ts_dir
```

```
student$ sudo chown postgres /var/lib/postgresql/ts_dir
```

```
=> CREATE TABLESPACE ts LOCATION '/var/lib/postgresql/ts_dir';
```

```
CREATE TABLESPACE
```

Create a table:

```
=> CREATE TABLE t(n integer) TABLESPACE ts;
```

```
CREATE TABLE
```

```
=> INSERT INTO t SELECT 1 FROM generate_series(1,1000);
```

```
INSERT 0 1000
```

2. Data Size

Database size:

```
=> SELECT pg_size_pretty(pg_database_size('data_physical')) AS db_size;
```

```
db_size
-----
7580 kB
(1 row)
```

Table size:

```
=> SELECT pg_size_pretty(pg_total_relation_size('t')) AS t_size;
```

```
t_size
-----
64 kB
(1 row)
```

Tablespace size:

```
=> SELECT
    pg_size_pretty(pg_tablespace_size('pg_default')) AS pg_default_size,
    pg_size_pretty(pg_tablespace_size('ts')) AS ts_size;
```

```
pg_default_size | ts_size
-----+-----
36 MB           | 68 kB
(1 row)
```

The size of the tablespace is larger than the size of the table by 4KB due to how Linux calculates the size of the directory contents.

3. Moving a Table

Move the table:

```
=> ALTER TABLE t SET TABLESPACE pg_default;
```

```
ALTER TABLE
```

New size of tablespaces:

=> **SELECT**

```
pg_size_pretty(pg_tablespace_size('pg_default')) AS pg_default_size,  
pg_size_pretty(pg_tablespace_size('ts')) AS ts_size;
```

pg_default_size	ts_size
36 MB	4096 bytes

(1 row)

4. Removing a Tablespace

Remove the tablespace...

=> **DROP TABLESPACE** ts;

DROP TABLESPACE

...and the directory where its data was stored:

```
student$ sudo rm -rf /var/lib/postgresql/ts_dir
```

=> \c postgres

You are now connected to database "postgres" as user "student".

=> **DROP DATABASE** data_physical;

DROP DATABASE