Authors: Egor Rogov, Pavel Luzanov

Feedback
Send feedback, comments and suggestions to:
edu@postgrespro.ru

Topics

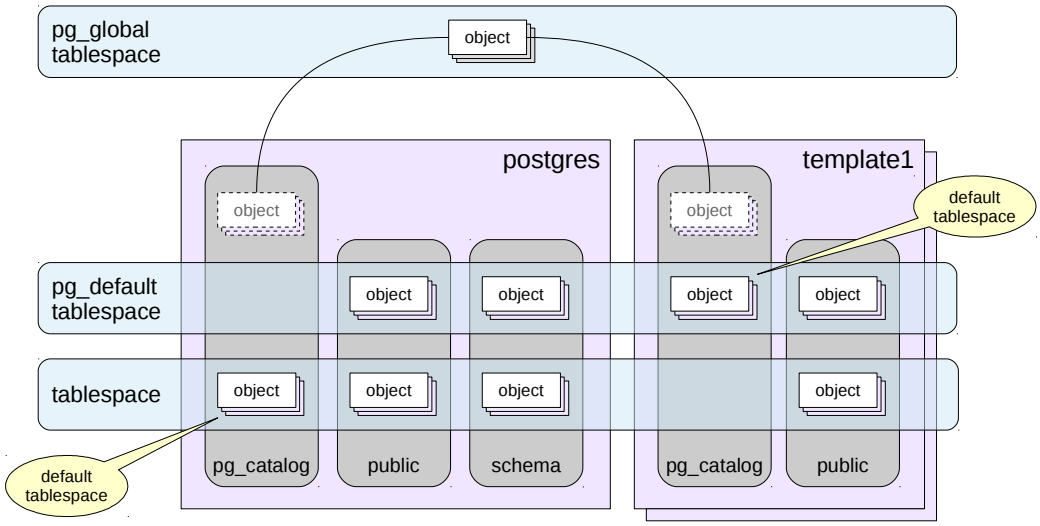
Tablespaces and directories

Files and data pages

Forks: main, visibility map, free space map

TOAST

Tablespaces



Tablespaces (TS) are used to organize the physical storage of data and determine the location of the data in the file system.

For example, you can create one TS on slow disks for archived data, and another on fast disks for hot data.

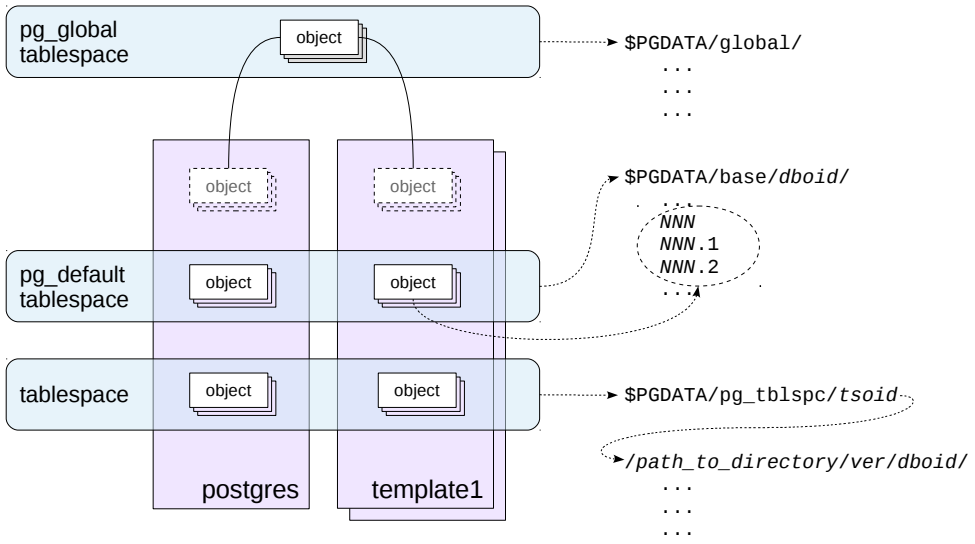
During cluster initialization, two TSs are created: `pg_default` and `pg_global`.

The same TS can be used by different databases, and one database can store data in several TSs.

In addition, each database has a so-called «default TS», in which all objects are created, unless explicitly specified otherwise. The objects of the system catalog are also stored in the default TS. Initially, the `pg_default` TS is used as the «default TS» (thus the name), but you can set another one.

The `pg_global` TS is special: it stores all cluster-level objects of the system catalog.

Directories, Files, Pages



In essence, a tablespace is an indication of the directory in which the data is located. Standard tablespaces `pg_global` and `pg_default` are always mapped to `$PGDATA/global/` and `$PGDATA/base/`, respectively. When creating a custom TS, an arbitrary directory is specified; for its own convenience, PostgreSQL creates a symbolic link to it in the `$PGDATA/pg_tblspc/` directory.

Inside the `$PGDATA/base/` directory, the data is further divided into subdirectories according to databases (this is not required for `$PGDATA/global/`, as the data in it refer to the cluster as a whole).

Inside the user TS directory, another nesting level appears: the PostgreSQL server version. This is done for the convenience of upgrading the server to another major version.

Objects themselves are stored in files inside these directories, each object is in separate set of files.

Each file (sometimes called a *segment*) takes no more than 1 GB, so each object can correspond to several files. 1 GB limit can be changed when building the server. It is important to consider the impact of a potentially large number of files on the file system performance.

All segments are logically divided into pages, usually 8 KB each (the size can be changed for the entire cluster when building the server). Pages of different objects (such as tables or indexes) are read from the disk in exactly the same way through the common buffer cache manager.

https://postgrespro.com/docs/postgresql/11/storage-file-layout

Forks

Main

- actual data

Visibility map (vm)

- pages containing only tuples known to be visible in all snapshots
- used to optimize vacuuming and speed up index-only access
- exists only for tables

Free space map (fsm)

- free space in pages after vacuuming
- used when inserting new tuples

Usually each object is represented by several forks. Each fork consists of a set of files (segments).

The main fork is the actual data: table row versions or index rows.

Visibility map (vm) fork is a bitmap which keeps track of pages that contain only tuples that are known to be visible in all data snapshots. In other words, these are pages that have not been changed for sufficiently long time to be completely vacuumed from non-actual row versions.

The visibility map is used to optimize vacuuming (tracked pages are not visited by vacuum process) and to speed up index-only access. Versioning information is stored only for tables, but not for indexes (therefore indexes do not have a visibility map). Having obtained from the index a pointer to the row version, PostgreSQL needs to read the table page to check this row version visibility. But if the index itself already has all the columns necessary for the query, and at the same time the page is tracked in the visibility map, then the table page access can be skipped.

Free space map (fsm) fork keeps track of available space within the pages. Free space is gained by vacuuming dead rows. The map is used when inserting new row versions to quickly find a suitable page.

TOAST

Row version must fit one page

- compress some attributes
- or move some attributes to an external TOAST table
- or move compressed values

TOAST table

- pg_toast schema
- supported by its own index
- large values are broken up into chunks smaller than a page
- only read when referring to the large value
- separate versioning
- works transparently for the application

Any row version must fit entirely in one page, PostgreSQL does not allow a row version to span multiple pages. For large rows versions, TOAST (The Oversized Attributes Storage Technique) is used. It involves several strategies. A large value can be compressed so that the row version fits on a page. Another approach is to move a large value to a separate TOAST table. Both approaches can be applied.

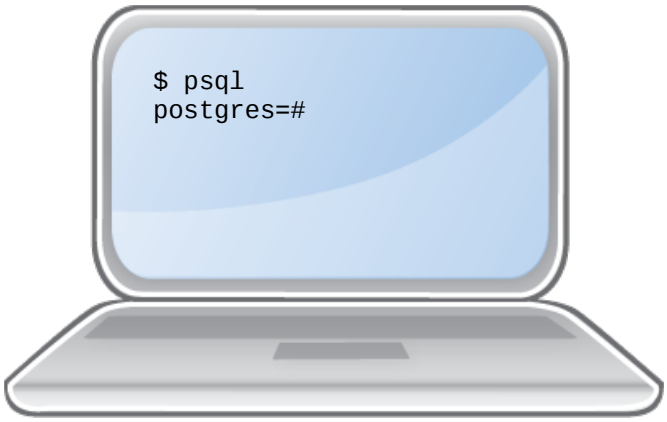
For each main table, if necessary, a separate TOAST table is created (and a special index for it). Such tables and indexes are located in `pg_toast` schema and therefore are usually not visible.

The row versions in the TOAST table should also fit on one page, so the large values are broken up into a number of smaller chunks. PostgreSQL transparently sticks together these chunks to form the required value for the application.

TOAST table is used only when referring to the large value. In addition, TOAST table has its own versioning: for example, if an UPDATE command does not affect the large attribute, the new row version will link to the same value in the TOAST table, which saves space.

https://postgrespro.com/docs/postgresql/11/storage-toast

Demonstration



Summary

Physically

- data is stored in tablespaces (directories)
- each object is represented by several forks
- each fork consists of one or more files (segments)

Tablespaces are managed by the administrator

Layers, files, and TOAST are PostgreSQL internals

Practice

1. Create a new database and connect to it.
2. Create tablespace ts.
3. Create table t in tablespace ts and add a few rows to it.
4. Calculate the volume occupied by the database, table and tablespaces ts and pg_default.
5. Move the table to the pg_default tablespace.
6. How have the tablespaces volumes changed?
7. Delete tablespace ts.