Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов, Игорь Гнатюк
Фото: Олег Бартунов (монастырь Пху и пик Бхрикути, Непал)

Обратная связь
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Темы

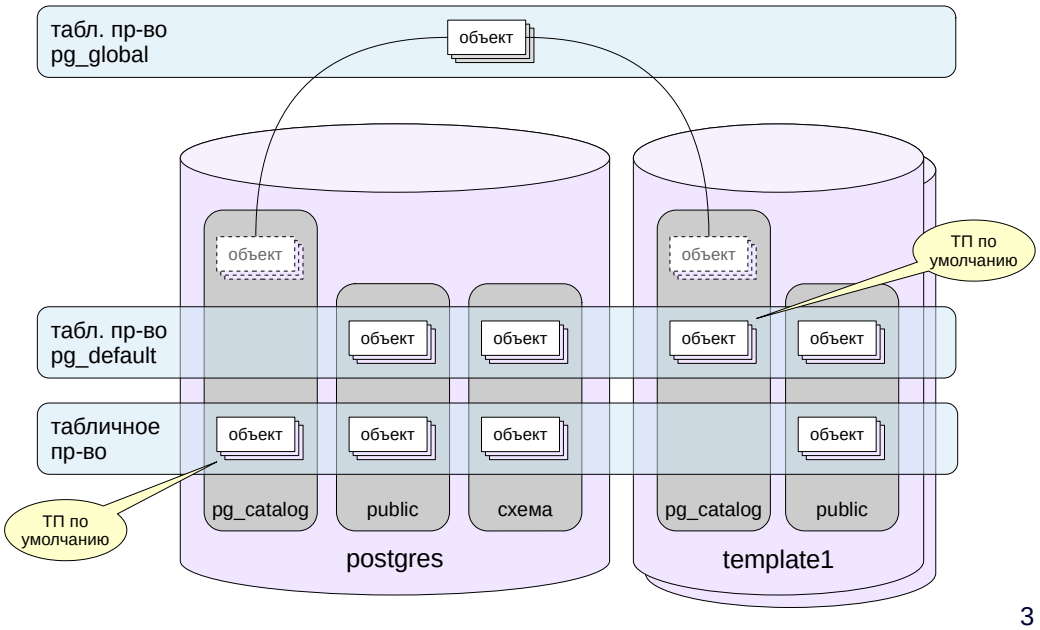
Табличные пространства и каталоги

Файлы и страницы данных

Слои: данные, карты видимости и свободного пространства

Технология TOAST

Табличные пространства



Табличные пространства (ТП) служат для организации физического хранения данных и определяют расположение данных в файловой системе.

Например, можно создать одно ТП на медленных дисках для архивных данных, а другое – на быстрых дисках для данных, с которыми идет активная работа.

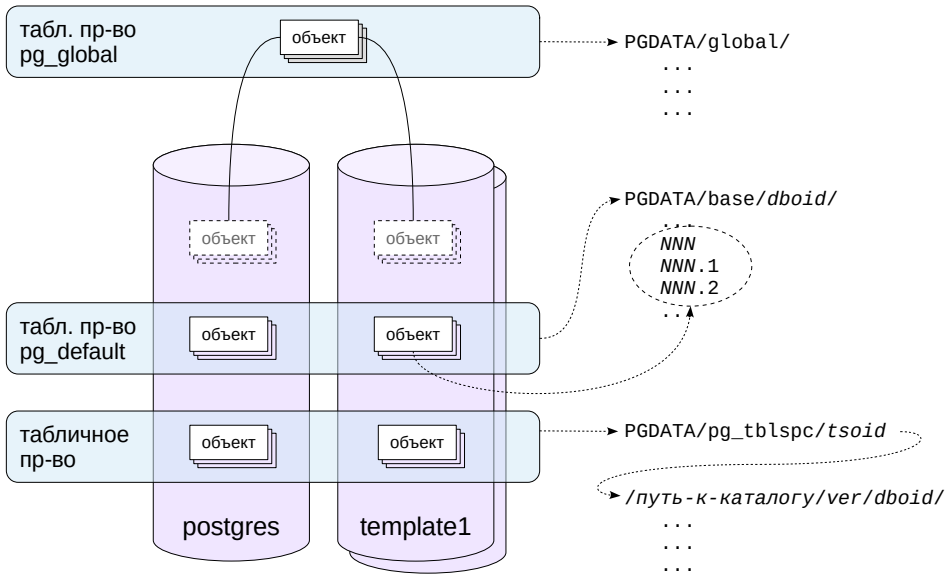
При инициализации кластера создаются два ТП: pg_default и pg_global.

Одно и то же ТП может использоваться разными базами данных, а одна база данных может хранить данные в нескольких ТП.

При этом у каждой БД есть так называемое «ТП по умолчанию», в котором создаются все объекты базы, если явно не указать иное. В этом же ТП хранятся и объекты системного каталога. Изначально в качестве «ТП по умолчанию» используется ТП pg_default, но можно установить и другое.

ТП pg_global особенное: в нем хранятся те объекты системного каталога, которые являются общими для кластера.

Каталоги



По сути, табличное пространство — это указание на каталог, в котором располагаются данные. Стандартные ТП pg_global и pg_default всегда находятся в PGDATA/global/ и PGDATA/base/ соответственно. При создании пользовательского ТП указывается произвольный каталог; для собственного удобства PostgreSQL создает на него символьную ссылку в каталоге PGDATA/pg_tblspc/.

Внутри каталога PGDATA/base/ данные дополнительно разложены по подкаталогам баз данных (для PGDATA/global/ это не требуется, так как данные в нем относятся к кластеру в целом).

Внутри каталога пользовательского ТП появляется еще один уровень вложенности: версия сервера PostgreSQL. Это сделано для удобства обновления сервера на другую версию.

Собственно объекты хранятся в файлах внутри этих каталогов — каждый объект размещается в одном или нескольких файлах.

Каждый такой файл, называемый *сегментом*, занимает по умолчанию не более 1 Гбайт (этот размер можно изменить при сборке сервера). В том числе поэтому каждому объекту может соответствовать несколько файлов. Необходимо учитывать влияние потенциально большого количества файлов на используемую файловую систему.

https://postgrespro.ru/docs/postgresql/16/storage-file-layout

Использование табличных пространств

Изначально в кластере присутствуют два табличных пространства. Информация о них содержится в системном каталоге:

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

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

Конечно, это одна из глобальных для всего кластера таблиц.

Аналогичная команда psql:

```
=> \db
```

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

Чтобы создать новое табличное пространство, надо сначала подготовить пустой каталог, владельцем которого является пользователь ОС, запускающий сервер СУБД:

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

Команда выполнялась прямо из psql; мы можем проверить результат выполнения средствами psql — прочитав значения его переменных:

```
=> \echo Ошибка была? :SHELL_ERROR. Код возврата команды? :SHELL_EXIT_CODE.
```

```
Ошибка была? false. Код возврата команды? 0.
```

Сделаем владельцем созданного каталога пользователя ОС postgres и проверим, что команда выполнилась:

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

```
=> \echo Ошибка? :SHELL_ERROR. Код возврата? :SHELL_EXIT_CODE.
```

```
Ошибка? false. Код возврата? 0.
```

Теперь выполняем саму команду из psql, указывая каталог:

```
=> 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)
```

При создании базы данных можно указать табличное пространство по умолчанию:

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

```
CREATE DATABASE
```

```
=> \c data_physical
```

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

Это означает, что все объекты базы по умолчанию будут создаваться в этом табличном пространстве.

```
=> 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;
```

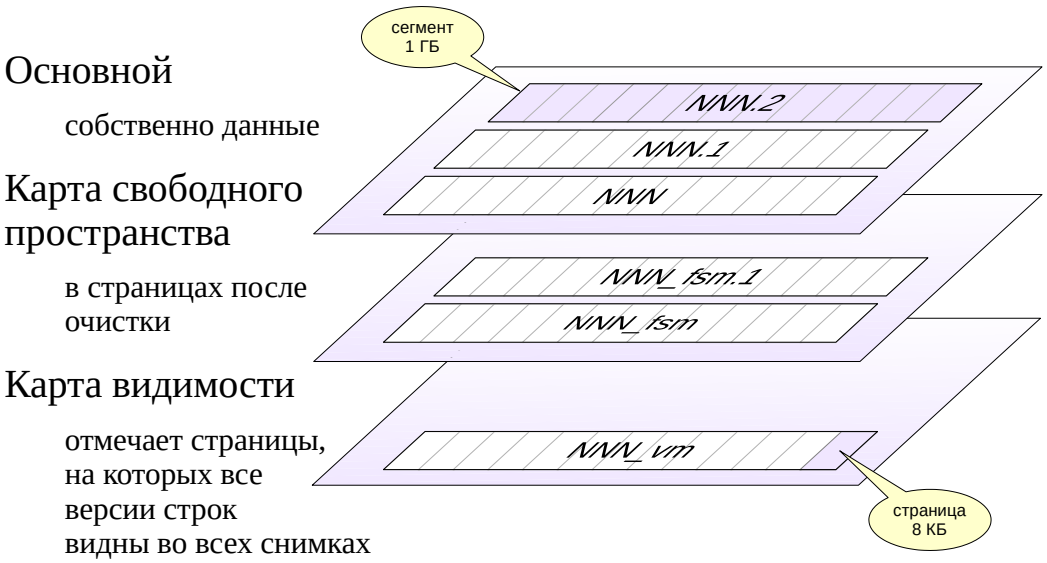
```
INSERT 0 100000
```

Очистка обеспечит нам создание всех слоев таблицы (про это ниже):

```
=> VACUUM t;
```

```
VACUUM
```

Слои и файлы



Обычно каждому объекту соответствует несколько *слоев* (forks). Каждый слой — это набор *сегментов* (то есть один или несколько файлов). Файлы-сегменты, из которых состоят слои, логически разбиты на *страницы*, обычно по 8 Кбайт (этот размер можно установить для всего кластера только при сборке сервера). Страницы файлов всех слоев объектов считываются с диска совершенно однотипно через общий механизм буферного кеша. Сами слои содержат разную информацию и имеют разное внутреннее устройство страниц.

Основной слой — это собственно данные: версии строк таблиц или строки индексов.

Слой vm (visibility map) — битовая **карта видимости**. В ней отмечены страницы, содержащие только актуальные версии строк, видимые во всех снимках данных. Иными словами, это страницы, которые давно не изменялись и успели полностью очиститься от неактуальных версий.

Карта видимости применяется для оптимизации очистки (отмеченные страницы не нуждаются в очистке) и для ускорения индексного доступа. Информация о версионности строк хранится только для таблиц, но не для индексов. Поэтому, получив из индекса ссылку на версию строки, нужно проверить ее видимость, обратившись к табличной странице. Но если в самом индексе уже есть все нужные столбцы, а страница отмечена в карте видимости, то к таблице можно не обращаться.

Слой fsm (free space map) — **карта свободного пространства**. В ней отмечено доступное место внутри страниц, образующееся, например, при работе очистки. Эта карта используется при вставке новых версий строк для того, чтобы быстро найти подходящую страницу.

Слои и файлы

Узнать расположение файлов, составляющих собой объект, можно так:

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

```
            pg_relation_filepath
-----------------------------------------
 pg_tblspc/49162/PG_16_202307071/49163/49164
(1 row)
```

Посмотрим на сами файлы (имя и размер в байтах):

```
student$ sudo bash -c 'cd /var/lib/postgresql/16/main/pg_tblspc/49162/PG_16_202307071/49163; ls -l 49164*'
```

```
-rw------- 1 postgres postgres 4423680 янв 10 19:53 49164
-rw------- 1 postgres postgres   24576 янв 10 19:53 49164_fsm
-rw------- 1 postgres postgres    8192 янв 10 19:53 49164_vm
```

Видно, что они относятся к трем слоям: основному, fsm и vm.

Объекты можно перемещать между табличными пространствами, но (в отличие от схем) это приводит к физическому перемещению данных:

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

```
ALTER TABLE
```

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

```
 pg_relation_filepath
----------------------
 base/49163/49171
(1 row)
```

Размер объектов

Узнать размер, занимаемый базой данных и объектами в ней, можно с помощью ряда функций.

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

```
 pg_database_size
------------------
         14488035
(1 row)
```

Для упрощения восприятия можно вывести число в отформатированном виде:

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

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

Полный размер таблицы (вместе со всеми индексами):

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

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

А также отдельно размер таблицы...

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

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

...и индексов:

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

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

При желании можно узнать и размер отдельных слоев таблицы, например:

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

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

Объем, который занимает на диске табличное пространство, показывает другая функция:

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

```
 pg_size_pretty
----------------
 9804 kB
(1 row)
```

TOAST

Версия строки должна помещаться на одну страницу

- можно сжать часть атрибутов,
- или вынести в отдельную TOAST-таблицу,
- или сжать и вынести одновременно

TOAST-таблица

- схема pg_toast
- поддержана собственным индексом
- «длинные» атрибуты разделены на части размером меньше страницы
- читается только при обращении к «длинному» атрибуту
- собственная версионность
- работает прозрачно для приложения

Любая версия строки в PostgreSQL должна целиком помещаться на одну страницу. Для «длинных» версий строк применяется технология TOAST — The Oversized Attributes Storage Technique. Она подразумевает несколько стратегий. Подходящий «длинный» атрибут может быть сжат так, чтобы версия строки поместилась на страницу. Если это не получается, атрибут может быть отправлен в отдельную служебную таблицу. Могут применяться и оба подхода.

Для каждой основной таблицы при необходимости создается отдельная toast-таблица (и к ней специальный индекс). Такие таблицы и индексы располагаются в отдельной схеме pg_toast и поэтому обычно не видны.

Версии строк в toast-таблице тоже должны помещаться на одну страницу, поэтому «длинные» значения хранятся порезанными на части. Из этих частей PostgreSQL прозрачно для приложения «склеивает» необходимое значение.

Toast-таблица используется только при обращении к «длинному» значению. Кроме того, для toast-таблицы поддерживается своя версионность: если обновление данных не затрагивает «длинное» значение, новая версия строки будет ссылаться на то же самое значение в toast-таблице — это экономит место.

https://postgrespro.ru/docs/postgresql/16/storage-toast

TOAST

Добавим в таблицу очень длинную строку:

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

INSERT 0 1
```

Изменится ли размер таблицы?

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

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

Да. А размер основного слоя, в котором хранятся данные?

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

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

Нет.

Поскольку строка не помещается в одну страницу, значение атрибута s будет разрезано на части и помещено в отдельную toast-таблицу. Ее можно отыскать в системном каталоге (мы используем тип regclass, чтобы преобразовать oid в имя отношения):

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

  oid  |      reltoastrelid
-------+-------------------------
 49164 | pg_toast.pg_toast_49164
(1 row)
```

Наша строка хранится по частям, из которых PostgreSQL при необходимости склеивает полное значение:

```
=> SELECT chunk_id, chunk_seq, left(chunk_data::text,45) AS chuck_data
FROM pg_toast.pg_toast_49164;

 chunk_id | chunk_seq |                  chuck_data
----------+-----------+----------------------------------------------
    49174 |         0 | \x545d000000312e322e332e342e00352e362e372e382
    49174 |         1 | \x392e353161ff31002e3531322e353133002e3531342
    49174 |         2 | \xe215e216e217e218abe219e11a30e11b30e11c30e11
    49174 |         3 | \x11f4aa3611f43611f43611f43611f4aa3611f43611f
    49174 |         4 | \xf43211f4325511f43211f43211f43211f4325511f43
    49174 |         5 | \xf43811f43811f43811f4385511f43811f43811f4381
    49174 |         6 | \x11f4aa3411f43411f43411f43411f4aa3411f43411f
    49174 |         7 | \x0132370132aa370132370132370132370132aa38013
    49174 |         8 | \xf43611f43611f43611f4365511f43611f43611f4361
(9 rows)
```

В заключение удалим базу данных.

```
=> \c postgres

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

=> DROP DATABASE data_physical;

DROP DATABASE
```

После того, как в табличном пространстве не осталось объектов, можно удалить и его:

```
=> DROP TABLESPACE ts;

DROP TABLESPACE
```

Итоги

Физически

- данные распределены по табличным пространствам (каталогам)
- объект представлен несколькими слоями
- каждый слой состоит из одного или нескольких файлов-сегментов

Табличными пространствами управляет администратор

Слои, файлы, TOAST — внутренняя кухня PostgreSQL

Практика

1. Создайте новую базу данных и подключитесь к ней.
   Создайте табличное пространство ts.
   Создайте таблицу t в табличном пространстве ts
   и добавьте в нее несколько строк.
2. Вычислите объем, занимаемый базой данных, таблицей
   и табличными пространствами ts и pg_default.
3. Перенесите таблицу в табличное пространство pg_default.
   Как изменился объем табличных пространств?
4. Удалите табличное пространство ts.

1. Табличные пространства и таблица

Создаем базу данных:

```
=> CREATE DATABASE data_physical;

CREATE DATABASE

=> \c data_physical

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

Табличное пространство:

```
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
```

Создаем таблицу:

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

CREATE TABLE

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

INSERT 0 1000
```

2. Размер данных

Объем базы данных:

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

 db_size
---------
 7644 kB
(1 row)
```

Размер таблицы:

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

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

Объем табличных пространств:

```
=> 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
-----------------+---------
 44 MB           | 68 kB
(1 row)
```

Размер табличного пространства больше размера таблицы на 4kB из-за особенностей вычисления размера содержимого каталога в Linux.

3. Перенос таблицы

Перенесем таблицу:

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

ALTER TABLE
```

Новый объем табличных пространств:

```
=> 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
-----------------+------------
 44 MB           | 4096 bytes
(1 row)
```

4. Удаление табличного пространства

Удаляем табличное пространство:

```
=> DROP TABLESPACE ts;

DROP TABLESPACE
```

и каталог, где были размещены его данные:

```
postgres$ rm -rf /var/lib/postgresql/ts_dir
```