

Организация данных Физическая структура



Авторские права

© Postgres Professional, 2017–2024

Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов, Игорь Гнатюк

Фото: Олег Бартунов (монастырь Пху и пик Бхрикути, Непал)

Использование материалов курса

Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

Обратная связь

Отзывы, замечания и предложения направляйте по адресу:

edu@postgrespro.ru

Отказ от ответственности

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или непрямым, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Темы



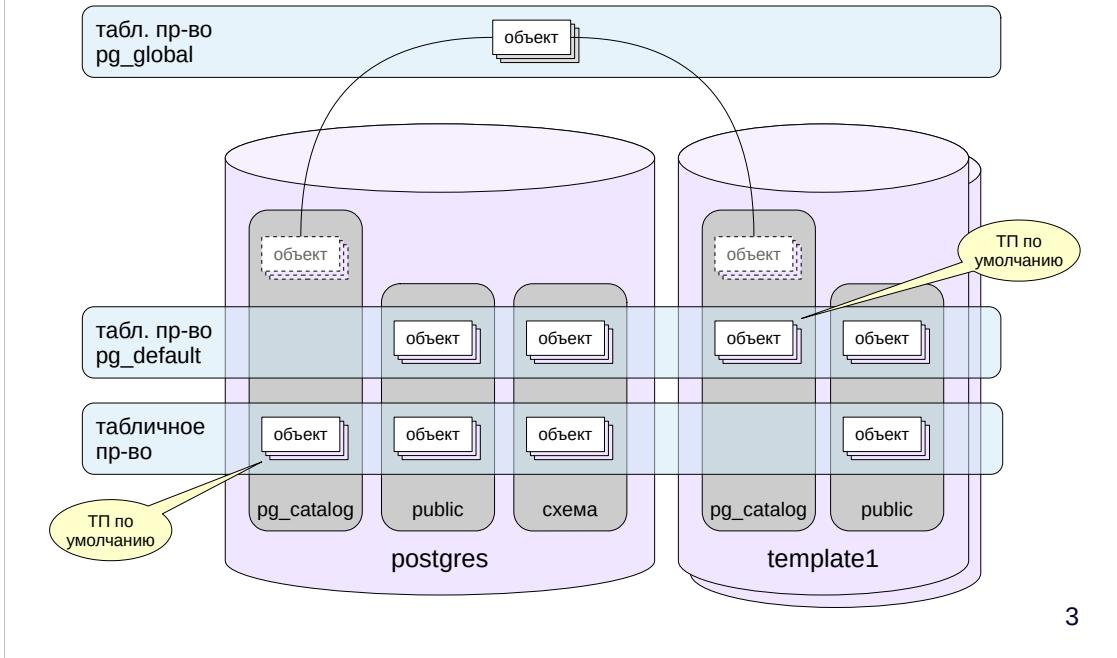
Табличные пространства и каталоги

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.

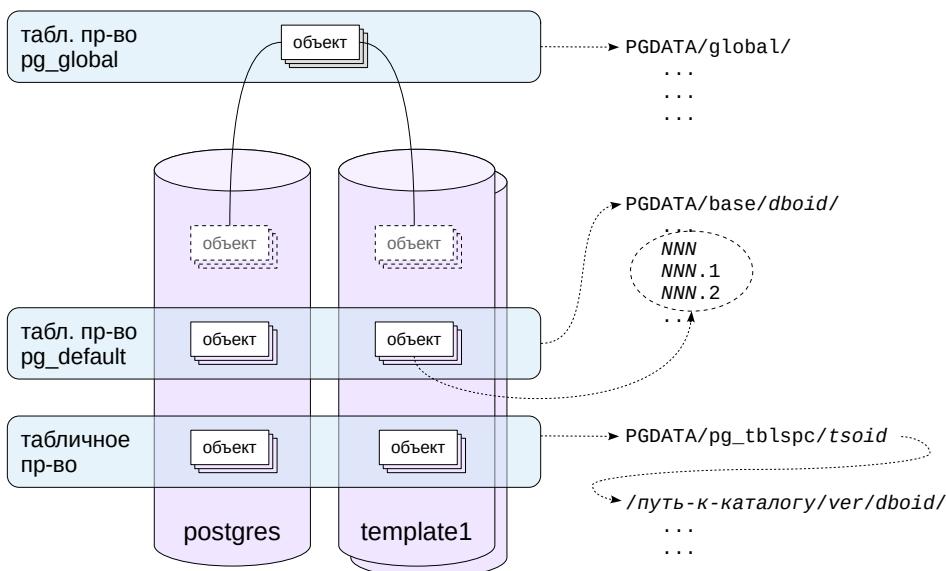
При инициализации кластера создаются два ТП: pg_default и pg_global.

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



4

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)
```

Для нового табличного пространства нужен пустой каталог, владельцем которого является пользователь ОС, запускающий сервер СУБД:

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

Сменим владельца каталога:

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

Теперь можем выполнить команду создания табличного пространства:

```
=> 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
```

Слои и файлы



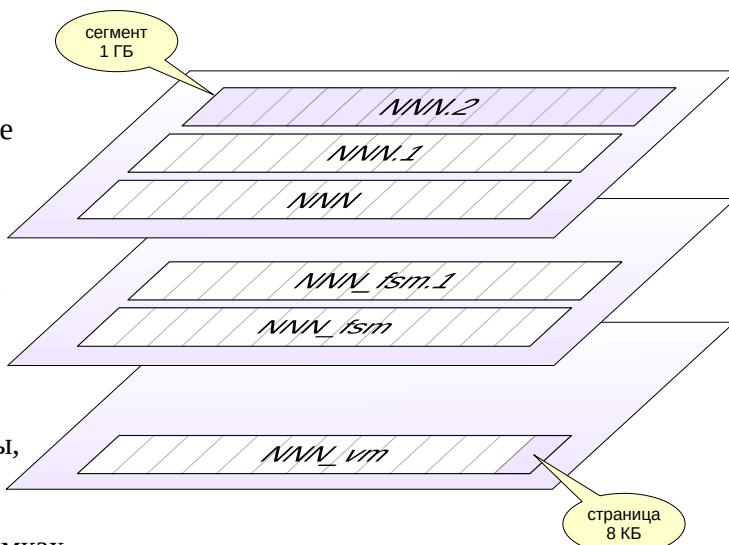
Основной
собственно данные

Карта свободного
пространства

в страницах после
очистки

Карта видимости

отмечает страницы,
на которых все
версии строк
видны во всех снимках



6

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 объектовчитываются 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, которые содержат только актуальные версии строк, причем эти версии видны во всех снимках данных. Такие страницы давно не изменились 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

Очистка обеспечит нам создание всех слоев таблицы:

```
=> VACUUM t;
```

```
VACUUM
```

Узнать расположение файлов, из которых состоит объект, можно так:

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

```
pg_relation_filepath
-----
pg_tblspc/16390/PG_16_202307071/16391/16392
(1 row)
```

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

```
student$ sudo bash -c 'cd /var/lib/postgresql/16/main/pg_tblspc/16390/PG_16_202307071/16391; ls -l
16392*'
```

```
-rw----- 1 postgres postgres 4423680 июн 24 04:31 16392
-rw----- 1 postgres postgres    24576 июн 24 04:31 16392_fsm
-rw----- 1 postgres postgres     8192 июн 24 04:31 16392_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/16391/16399
(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)
```

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

8

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 версия строки 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
-----+
16392 | pg_toast.pg_toast_16392
(1 row)
```

Символьная 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_16392;

chunk_id | chunk_seq | chuck_data
-----+-----+-----+
16402 | 0 | \x545d000000312e322e332e342e00352e362e372e382
16402 | 1 | \x392e353161ff31002e3531322e353133002e3531342
16402 | 2 | \xe215e216e217e218abe219e11a30e11b30e11c30e11
16402 | 3 | \x11f4aa3611f43611f43611f43611f4aa3611f43611f
16402 | 4 | \xf43211f4325511f43211f43211f4325511f43
16402 | 5 | \xf43811f43811f43811f4385511f43811f43811f4381
16402 | 6 | \x11f4aa3411f43411f43411f43411f4aa3411f43411f
16402 | 7 | \x0132370132aa370132370132370132370132aa38013
16402 | 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
=> \! sudo rmdir /var/lib/postgresql/ts_dir
```


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

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.