Авторы: Алексей Береснев, Илья Баштанов, Павел Толмачев

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

Темы

Возможность и польза сжатия

Реализация сжатия в Postgres Pro Enterprise

Использование CFS

Объем данных и степень сжатия

Фрагментация и сбор мусора

Возможности и польза

Данные БД обычно хорошо сжимаются

- текстовые данные
- дублирование

Преимущества сжатия

- уменьшение объема хранения
- экономия ресурсов ввода-вывода
- последовательное чтение и запись

Сжатие при записи грязного буфера

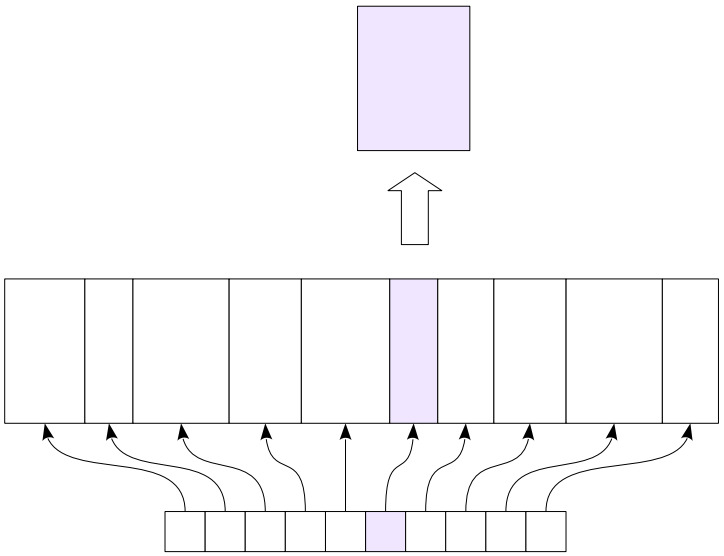
- в буферном кеше нужны несжатые данные

В базах данных обычно хранятся большие объемы текста и повторяющейся информации. Поэтому для большинства баз сжатие может быть довольно эффективным и позволяет сократить объем хранимых данных в несколько раз. PostgreSQL сжимает данные в TOAST-таблицах, но небольшие текстовые поля, умещающиеся в странице, не сжимаются. Сжатие может быть полезно не только для таблиц, но и для индексов по текстовым ключам или для индексов с большим количеством повторяющихся значений.

Помимо очевидного плюса — экономии места, сжатие может также увеличить быстродействие системы по двум причинам. Во-первых, сокращается объем дискового ввода-вывода. Во-вторых, улучшается кучность хранения, что увеличивает вероятность последовательного чтения.

Работать со сжатыми данными в общей памяти сервера (в буферном кеше) непроизводительно, поэтому логично выполнять сжатие страницы при сохранении грязного буфера и разжимать ее при чтении в буферный кеш.

Реализация сжатия

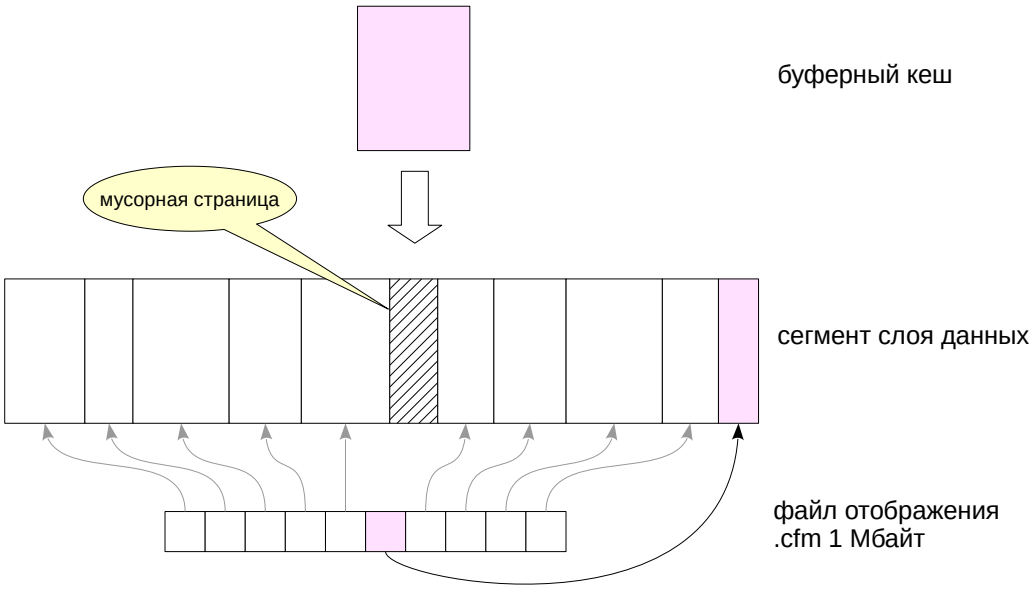


Буферный кеш для эффективности использует страницы фиксированного размера (обычно 8 Кбайт), а сжатые страницы имеют разный размер. При сохранении страница может не уместиться на прежнее место, поэтому вводится дополнительный уровень адресации — отображение страниц.

Как и в несжатых табличных пространствах, данные хранятся в файлах-сегментах, и для каждого из них имеется своя карта отображения страниц. Карта имеет размер 1 Мбайт (32 байта на заголовок + по 8 байт на каждый из 131072 указателей) и поэтому обычно умещается в оперативной памяти.

https://postgrespro.ru/docs/enterprise/13/cfs-implementation

Реализация сжатия



При записи (в частности, при выполнении контрольной точки) страница, содержащаяся в грязном буфере, сжимается в соответствии с настройками сжатия и добавляется в конец сегмента, а в указатель в файле отображения заносится новая позиция страницы. После этого область файла данных, в которой раньше находилась страница, становится ненужной. Такие мусорные страницы будут накапливаться, что приведет к увеличению фрагментации хранимых данных.

Заметим, что если изменилось значительное число страниц отношения, перед определением степени фрагментации имеет смысл выполнить контрольную точку, чтобы все грязные буферы сбросились на диск.

Использование CFS

```
CREATE TABLESPACE ...
    WITH (compression=on)
    WITH (compression=zstd|pglz|zlib|lz4)
```

Системные отношения не сжимаются

Проверка контрольной суммы — при чтении в буфер

pg_checksums и pg_basebackup не проверяют

Сжатие можно включить только на уровне табличного пространства и только в момент его создания, при этом нужно задать алгоритм сжатия. Изменить факт и алгоритм сжатия в дальнейшем нельзя.

Поддерживаются библиотеки zstd, pglz, zlib, lz4. По умолчанию используется алгоритм zstd.

Системные отношения никогда не сжимаются — для соответствующих сегментов не создаются файлы отображения и механизм сжатия не используется.

Проверка контрольных сумм для сжатых страниц невозможна, она происходит только при чтении страницы в буферный кеш, а утилиты pg_checksums и pg_basebackup контрольные суммы страниц не проверяют.

https://postgrespro.ru/docs/enterprise/13/cfs-usage

Настройка табличного пространства

Убедимся, что сжатие поддерживается:

```
=> SELECT cfs_version();
```

```
       cfs_version
-------------------------
 0.54 pglz,zlib,lz4,zstd
(1 row)
```

Создадим табличное пространство со сжатием.

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

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

```
=> CREATE TABLESPACE cts LOCATION '/var/lib/postgresql/ts_dir' WITH (compression=on);
```

```
CREATE TABLESPACE
```

```
=> \db+ cts
```

```
                                        List of tablespaces
 Name |  Owner  |          Location          | Access privileges |     Options      |  Size   | Description
------+---------+----------------------------+-------------------+------------------+---------+-------------
 cts  | student | /var/lib/postgresql/ts_dir |                   | {compression=on} | 4 bytes |
(1 row)
```

Объем данных

Стандартные функции

pg_relation_size *pg_table_size* *pg_indexes_size* *pg_total_relation_size*	работают корректно
pg_database_size	+ файлы отображения
pg_tablespace_size	+ файлы отображения + pg_compression

Стандартные функции *pg_*_size*, возвращающие занимаемый на диске объем, корректно работают со сжатыми табличными пространствами.

В объеме базы данных учитываются файлы отображения, а в объеме табличного пространства — файлы отображения и служебный файл pg_compression.

https://postgrespro.ru/docs/enterprise/13/functions-admin#FUNCTIONS-COMPRESSION-CONTROL

Степень сжатия

Параметры

cfs_level = 1	0 — без сжатия, 1–100 – степень сжатия
cfs_compress_temp_relations = off	

Функции

cfs_estimate	прогноз степени сжатия (среднее по 10 первым страницам)
cfs_compression_ratio	фактическое сжатие по всем сегментам

Уровень сжатия настраивается параметром *cfs_level*. Значение 0 отключает сжатие, значение 1 (по умолчанию) соответствует максимальной скорости, значения от 2 до 100 определяют уровень сжатия. Максимальное значение зависит от используемого алгоритма.

Временные таблицы по умолчанию не сжимаются. Параметр *cfs_compress_temp_relations* позволяет включить их сжатие, это может быть полезно при большом объеме временных таблиц.

Функция *cfs_estimate* оценивает степень сжатия, анализируя десять первых страниц таблицы. Функция *cfs_compression_ratio* возвращает фактическую степень сжатия таблицы. Степень сжатия – это отношение объема несжатых данных к объему сжатых, в данном случае объем страниц в буферном кеше делится на объем сжатых страниц, сохраненных в файле.

https://postgrespro.ru/docs/enterprise/13/runtime-config-cfs

https://postgrespro.ru/docs/enterprise/13/functions-admin#FUNCTIONS-COMPRESSION-CONTROL

Сжатая таблица

В новой базе данных таблицы и индексы будут по умолчанию располагаться в сжатом табличном пространстве.

```
=> CREATE DATABASE cfs TABLESPACE cts;
```

```
CREATE DATABASE
```

```
=> \c cfs
```

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

Создадим таблицу с минимальным fillfactor, отключив автоочистку:

```
=> CREATE TABLE test (s CHAR(500))
WITH (fillfactor=10, autovacuum_enabled=off);
```

```
CREATE TABLE
```

Вставим 100000 строк со случайной буквой, в этом случае версии строк будут достаточно короткими и не попадут в TOAST-хранилище:

```
=> INSERT INTO test
SELECT (ascii('a') + (random()*25)::int)::char s
FROM generate_series(1,1e5);
```

```
INSERT 0 100000
```

```
=> ANALYZE;
```

```
ANALYZE
```

Сколько места в оперативной памяти занимает основной слой таблицы?

```
=> SELECT reltuples, relpages,
  pg_size_pretty(relpages * 8192::numeric)
FROM pg_class
WHERE relname='test';
```

```
 reltuples | relpages | pg_size_pretty
-----------+----------+----------------
    100000 |   100000 | 781 MB
(1 row)
```

А сколько на диске? Чтобы измерение было корректным, предварительно выполним контрольную точку.

```
=> CHECKPOINT;
```

```
CHECKPOINT
```

```
=> SELECT pg_size_pretty(pg_relation_size('test','main'));
```

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

Прогнозируемая (по первым 10 страницам) и фактическая степени сжатия:

```
=> SELECT cfs_estimate('test'), cfs_compression_ratio('test');
```

```
    cfs_estimate    | cfs_compression_ratio
--------------------+-----------------------
 30.389682491024946 |     75.06816798115378
(1 row)
```

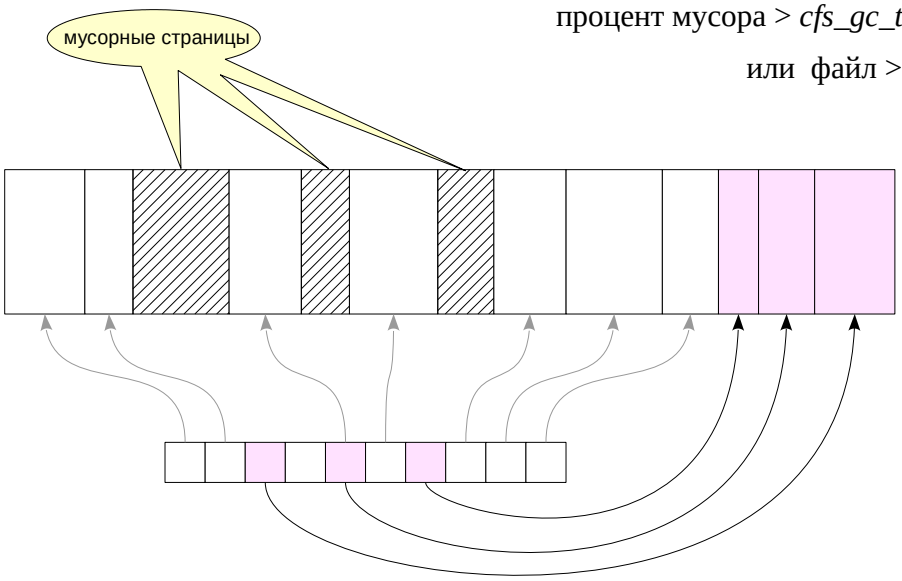
Как видим, наша таблица сжалась в десятки раз.

Системный каталог тоже находится в пространстве по умолчанию (cts), но системные таблицы никогда не сжимаются.

```
=> SELECT relname, reltablespace, relpages * 8192 buffers,
  pg_relation_size(oid,'main') file
FROM pg_class
WHERE relname = 'pg_class';
```

```
 relname  | reltablespace | buffers |  file
----------+---------------+---------+--------
 pg_class |             0 |  122880 | 122880
(1 row)
```

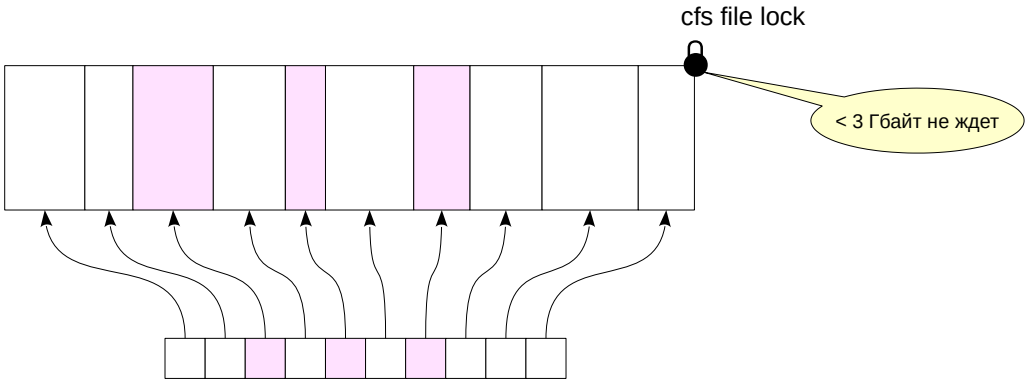
Фрагментация



Чтобы бороться с фрагментацией, в Postgres Pro Enterprise реализован механизм сбора мусора.

Файл данных подлежит обработке, если суммарный размер мусорных страниц превышает *cfs_gc_threshold* процентов от размера файла или размер файла больше 2 Гбайт. Одновременно с файлом данных обрабатывается соответствующий файл отображения.

Сбор мусора



На время обработки файл данных полностью блокируется — ставится легкая блокировка cfs file lock в исключительном режиме, она запрещает обслуживающим процессам обращаться к файлу. Если не удается получить блокировку немедленно, а размер файла менее 3 Гбайт, процедура сбора мусора его пропустит, а если файл имеет больший размер — будет ждать получения блокировки.

Затем нужные страницы копируются в новый файл данных и строится новый файл отображения. По окончании обработки старые файлы удаляются, а новые переименовываются; это происходит атомарно.

Обслуживающие процессы устанавливают на файл ту же блокировку, но в разделяемом режиме, чтобы процесс сбора мусора дождался окончания работы с файлом, прежде чем начинать дефрагментацию.

Сбор мусора

Настройки

cfs_gc = on
cfs_gc_workers = 1
cfs_gc_threshold = 50
cfs_gc_delay = 0
cfs_gc_period = 5000ms

Запуск

cfs_start_gc(*число-процессов*), 0 — синхронно
cfs_gc_relation(*отношение*)

Сбор мусора выполняется одним или несколькими процессами в ручном или фоновом режиме.

По умолчанию фоновый сбор мусора включен (*cfs_gc* = on) и использует один рабочий процесс (*cfs_gc_workers* = 1).

Если выключить фоновый сбор (*cfs_gc* = off) и задать параметр *cfs_gc_workers* = 0, можно вызовом функции cfs_start_gc(0) запустить сбор мусора синхронно, а вызовом cfs_start_gc(*N*) — асинхронно *N* процессами. Функция cfs_gc_relation собирает мусор в файлах определенного отношения, возвращая число обработанных файлов.

Параметр *cfs_gc_threshold* задает минимальный процент мусора, при котором файл размером до 2 Гбайт подлежит обработке (файлы больше 2 Гбайт обрабатываются всегда). Эти условия действуют независимо от способа запуска.

После обработки каждого файла процесс приостанавливается на *cfs_gc_delay* миллисекунд. По окончании обработки всех файлов процесс бездействует *cfs_gc_period* миллисекунд.

При ручном запуске можно задать все упомянутые параметры на уровне сеанса, чтобы, например, обработать файлы с меньшим уровнем фрагментации или сделать обработку более интенсивной.

Сбор мусора

Мониторинг

- cfs_fragmentation (*отношение*)
- cfs_gc_activity_scanned_files
- cfs_gc_activity_processed_files
- cfs_gc_activity_processed_pages
- cfs_gc_activity_processed_bytes

Для мониторинга процессов сбора мусора имеется несколько функций. Статистика накапливается с момента запуска сервера.

Важно, чтобы сбор мусора выполнялся регулярно, иначе файлы отношений будут неограниченно расти, что приведет к падению производительности, а в дальнейшем к недостатку свободного места на диске. Поэтому рекомендуется не отключать фоновый сбор и при необходимости настроить его частоту и производительность.

https://postgrespro.ru/docs/enterprise/13/functions-admin#FUNCTIONS-COMPRESSION-CONTROL

Сбор мусора

Сейчас таблица немного фрагментирована:

```
=> SELECT pg_size_pretty(pg_table_size('test')), cfs_fragmentation('test');
```

```
 pg_size_pretty |  cfs_fragmentation
----------------+--------------------
 11 MB          | 0.31560144154341324
(1 row)
```

Посмотрим, что будет, если отключить сбор мусора.

```
=> ALTER SYSTEM SET cfs_gc = off;
```

```
ALTER SYSTEM
```

```
=> SELECT pg_reload_conf();
```

```
 pg_reload_conf
----------------
 t
(1 row)
```

Изменим все строки таблицы.

```
=> UPDATE test SET s = s;
```

```
UPDATE 100000
```

```
=> CHECKPOINT;
```

```
CHECKPOINT
```

```
=> SELECT pg_size_pretty(pg_table_size('test')), cfs_fragmentation('test');
```

```
 pg_size_pretty | cfs_fragmentation
----------------+--------------------
 20 MB          | 0.5301870083158641
(1 row)
```

Объем хранения вырос вдвое, фрагментация около 50%.

При дальнейшем изменении таблицы рост продолжится.

```
=> UPDATE test SET s = s;
```

```
UPDATE 100000
```

```
=> CHECKPOINT;
```

```
CHECKPOINT
```

```
=> SELECT pg_size_pretty(pg_table_size('test')), cfs_fragmentation('test');
```

```
 pg_size_pretty | cfs_fragmentation
----------------+--------------------
 30 MB          | 0.6646340313550497
(1 row)
```

```
=> UPDATE test SET s = s;
```

```
UPDATE 100000
```

```
=> CHECKPOINT;
```

```
CHECKPOINT
```

```
=> SELECT pg_size_pretty(pg_table_size('test')), cfs_fragmentation('test');
```

```
 pg_size_pretty | cfs_fragmentation
----------------+--------------------
 40 MB          | 0.7493495657952584
(1 row)
```

```
=> UPDATE test SET s = s;
```

```
UPDATE 100000

=> CHECKPOINT;

CHECKPOINT

=> SELECT pg_size_pretty(pg_table_size('test')), cfs_fragmentation('test');

 pg_size_pretty | cfs_fragmentation
----------------+--------------------
 49 MB          | 0.8002196880053908
(1 row)
```

Поэтому в реальной системе мусор необходимо собирать.

Попробуем собрать мусор в таблице вручную, задав минимальный порог на уровне сеанса, чтобы были обработаны все сегменты:

```
=> SET cfs_gc_threshold = 0;

SET

=> SELECT cfs_gc_relation('test');

 cfs_gc_relation
-----------------
               1
(1 row)
```

Неиспользуемые страницы удалены, размер файла уменьшился:

```
=> SELECT pg_size_pretty(pg_table_size('test')), cfs_fragmentation('test');

 pg_size_pretty | cfs_fragmentation
----------------+-------------------
 10 MB          |                 0
(1 row)
```

Обновим таблицу и выполним контрольную точку, чтобы сжатые страницы записались в файл.

```
=> UPDATE test SET s = s;

UPDATE 100000

=> CHECKPOINT;

CHECKPOINT

=> SELECT pg_size_pretty(pg_table_size('test')), cfs_fragmentation('test');

 pg_size_pretty | cfs_fragmentation
----------------+--------------------
 20 MB          | 0.4999224751062721
(1 row)
```

Объем опять вырос вдвое, фрагментация около 50%.

Теперь вернем автоматический сбор мусора и сбросим порог, который мы задавали на уровне сеанса.

```
=> ALTER SYSTEM RESET cfs_gc;

ALTER SYSTEM

=> SELECT pg_reload_conf();

 pg_reload_conf
----------------
 t
(1 row)


=> RESET cfs_gc_threshold; SHOW cfs_gc_threshold;

RESET
 cfs_gc_threshold
------------------
 50
(1 row)
```

Если еще раз изменить все строки таблицы, доля мусора превысит порог по умолчанию (50%).

```
=> UPDATE test SET s = s;

UPDATE 100000

=> CHECKPOINT;

CHECKPOINT
```

Сбор мусора выполнится автоматически (немного подождем).

```
=> SELECT pg_size_pretty(pg_table_size('test')), cfs_fragmentation('test');

 pg_size_pretty | cfs_fragmentation
----------------+-------------------
 10 MB          |                 0
(1 row)
```

Итоги

Сжатие настраивается на уровне табличных пространств

Можно выбирать алгоритм и управлять степенью сжатия

Встроенный механизм сбора мусора сокращает объем хранимых данных

Практика

1. Создайте сжатое табличное пространство, использующее алгоритм zstd с максимальной степенью сжатия 14.
2. Поместите копии таблиц демобазы в сжатое табличное пространство. Насколько уменьшился их размер?
3. Удалите сжатое пространство.

1. Сжатое табличное пространство

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

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

```
=> CREATE TABLESPACE cts LOCATION '/var/lib/postgresql/ts_dir' WITH (compression=zstd);
```

```
CREATE TABLESPACE
```

```
=> ALTER SYSTEM SET cfs_level=14;
```

```
ALTER SYSTEM
```

```
=> SELECT pg_reload_conf();
```

```
 pg_reload_conf
----------------
 t
(1 row)
```

```
=> \db+ cts
```

```
                                        List of tablespaces
 Name |  Owner  |          Location          | Access privileges |      Options       |  Size   | Description
------+---------+----------------------------+-------------------+--------------------+---------+-------------
 cts  | student | /var/lib/postgresql/ts_dir |                   | {compression=zstd} | 4 bytes |
(1 row)
```

2. База данных

```
=> CREATE DATABASE cfs TEMPLATE template0 TABLESPACE cts;
```

```
CREATE DATABASE
```

```
=> \c cfs
```

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

```
=> CREATE SCHEMA bookings;
```

```
CREATE SCHEMA
```

Загружаем определение таблицы и ее строки в новую базу:

```
student$ /opt/pgpro/ent-13/bin/pg_dump -d demo --table=tickets --section=pre-data --section=data | psql -d cfs
```

```
SET
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
SET
SET
SET
CREATE TABLE
ALTER TABLE
COMMENT
COMMENT
COMMENT
COMMENT
COMMENT
COMMENT
COPY 366733
```

Размер таблицы и степень сжатия:

```
=> ANALYZE bookings.tickets;

ANALYZE

=> SELECT relname, relpages,
  pg_size_pretty(relpages*8192::numeric) original,
  pg_size_pretty(pg_table_size(oid)) compressed,
  cfs_compression_ratio(oid) ratio
FROM pg_class
WHERE relname = 'tickets'
;

 relname | relpages | original | compressed |       ratio
---------+----------+----------+------------+--------------------
 tickets |     6159 | 48 MB    | 23 MB      | 2.0975552973214233
(1 row)
```

Удаление табличного пространства

```
=> \c student

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

=> DROP DATABASE cfs;

DROP DATABASE

=> DROP TABLESPACE cts;

DROP TABLESPACE

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