

# Журналирование Буферный кеш



## Авторские права

© Postgres Professional, 2016–2025

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](mailto:edu@postgrespro.ru)

## Отказ от ответственности

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или косвенным, 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.

Устройство и использование буферного кеша

Механизм вытеснения страниц

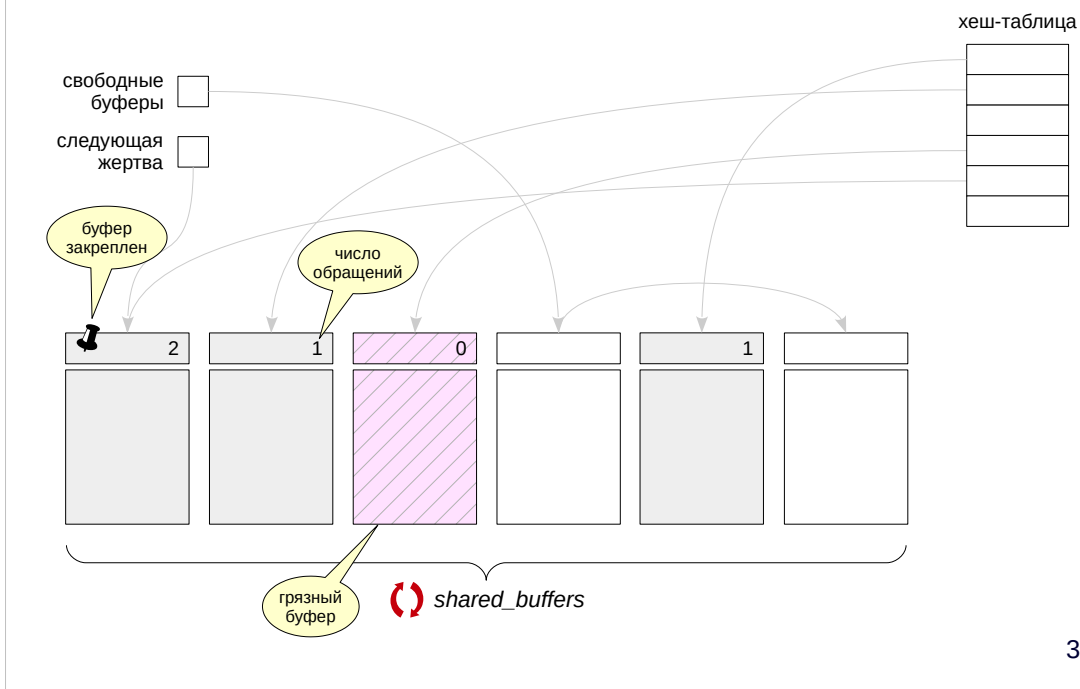
Массовое вытеснение и буферные кольца

Настройка размера кеша

Локальный кеш для временных таблиц

Прогрев кеша

# Буферный кеш



3

Задача буферного кеша — сглаживать разницу в производительности двух типов памяти: оперативной (быстрая, но мало) и дисковой (медленная, но много). Чтобы работать с данными — читать или изменять, — процессы читают страницы в буферный кеш. Пока страница находится в кеше, мы экономим на обращениях к диску.

Буферный кеш располагается в общей памяти сервера и представляет собой массив буферов. Каждый буфер состоит из места под одну страницу данных и заголовка. Заголовок содержит, в числе прочего:

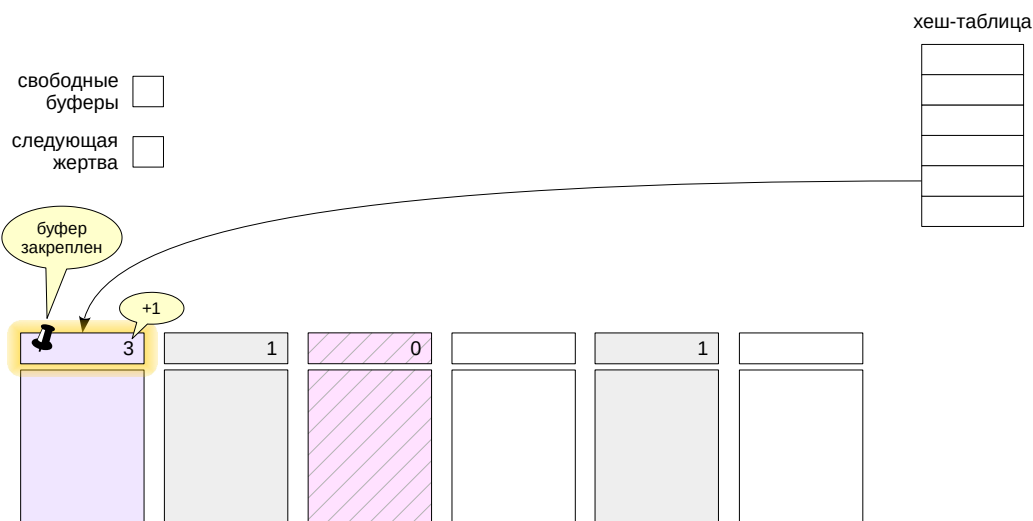
- расположение страницы на диске (файл и номер страницы в нем);
- число обращений к буферу (счетчик `usage count`, увеличивается каждый раз, когда процесс читает или изменяет буфер);
- признак того, что данные на странице изменились и рано или поздно должны быть записаны на диск (такой буфер называют грязным);
- признак закрепления (счетчик `pin count` количества процессов, работающих в данный момент с содержимым буфера).

Размер буферного кеша задается параметром `shared_buffers`. Его изменение требует перезапуска сервера.

Изначально кеш содержит пустые буферы, и все они связаны в список свободных буферов. Смысл указателя на «следующую жертву» станет ясен чуть позже.

Чтобы быстро находить нужную страницу в кеше, используется хеш-таблица.

# Страница в кеше



4

Когда процессу требуется прочитать страницу, он сначала пытается найти ее в буферном кеше с помощью хеш-таблицы.

Ключом хеширования служит файл и номер страницы внутри файла; получив номер буфера, процесс быстро проверяет, действительно ли он содержит нужную страницу. Как и в любой хеш-таблице, здесь возможны коллизии; в таком случае процессу придется проверить несколько страниц.

Если нужная страница найдена в кеше, процесс должен «закрепить» буфер (увеличить счетчик pin count) и увеличить число обращений (счетчик usage count). Буфер могут закреплять несколько процессов одновременно, поэтому используется именно счетчик, а не логический признак.

Пока буфер закреплен (значение pin count больше нуля), считается, что буфер используется и его содержимое не должно «радикально» измениться. Например, в странице может появиться новая версия строки — это никому не мешает благодаря многоверсионности и правилам видимости. Но в закрепленный буфер не может быть прочитана другая страница.

## Страница в кеше

```
=> CREATE DATABASE wal_buffercache;
```

```
CREATE DATABASE
```

```
=> \c wal_buffercache
```

You are now connected to database "wal\_buffercache" as user "student".

Создадим таблицу с одной строкой:

```
=> CREATE TABLE test(  
    t text  
)  
WITH (autovacuum_enabled = off);
```

```
CREATE TABLE
```

```
=> INSERT INTO test VALUES ('a row');
```

```
INSERT 0 1
```

Информацию о содержимом буферного кеша и его использовании можно получить с помощью расширения pg\_buffercache, включающего в себя одноименное представление и дополнительные функции:

```
=> CREATE EXTENSION pg_buffercache;
```

```
CREATE EXTENSION
```

Создадим для удобства представление, расшифровывающее некоторые столбцы.

Условие на базу данных необходимо, так как в буферном кеше содержатся данные всего кластера. Расшифровать мы можем только информацию из той БД, к которой подключены. Глобальные объекты считаются принадлежащими БД с нулевым OID.

```
=> CREATE VIEW pg_buffercache_v AS  
SELECT bufferid,  
    (SELECT c.relname  
     FROM pg_class c  
     WHERE pg_relation_filenode(c.oid) = b.relfilenode  
    ) relname,  
    CASE relforknumber  
        WHEN 0 THEN 'main'  
        WHEN 1 THEN 'fsm'  
        WHEN 2 THEN 'vm'  
    END relfork,  
    relblocknumber,  
    isdirty,  
    usagecount  
FROM pg_buffercache b  
WHERE b.reldatabase IN (  
    0, (SELECT oid FROM pg_database WHERE datname = current_database())  
)  
AND b.usagecount IS NOT NULL;
```

```
CREATE VIEW
```

В буферном кеше уже находятся страницы таблицы; они появились при выполнении вставки:

```
=> SELECT * FROM pg_buffercache_v WHERE relname = 'test';
```

bufferid	relname	relfork	relblocknumber	isdirty	usagecount
1845	test	main	0	t	1

(1 row)

Команда EXPLAIN с указанием analyze и buffers показывает использование буферного кеша при выполнении запроса:

```
=> EXPLAIN (analyze,buffers, costs off, timing off, summary off)  
SELECT * FROM test;
```

# QUERY PLAN

```
-----  
Seq Scan on test (actual rows=1 loops=1)  
  Buffers: shared hit=1  
Planning:  
  Buffers: shared hit=4  
(4 rows)
```

Строка «Buffers: shared hit=1» говорит о том, что страница была найдена в кеше.

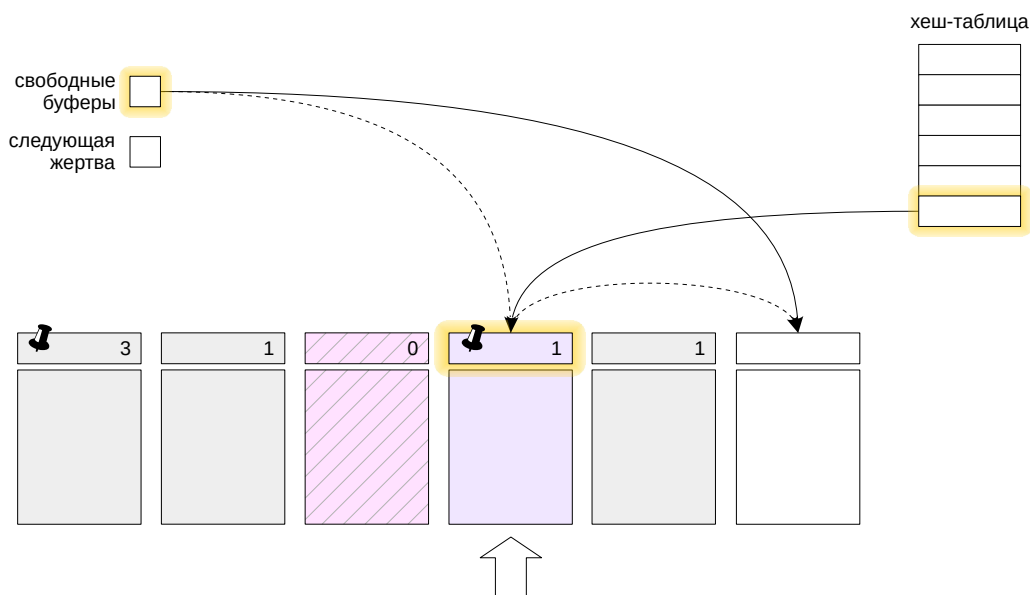
После выполнения запроса счетчик использования увеличился на единицу:

=> **SELECT \* FROM pg\_buffercache\_v WHERE relname = 'test';**

bufferid	relname	relfork	relblocknumber	isdirty	usagecount
1845	test	main	0	t	2

(1 row)

# Чтение в свободный буфер



Может получиться так, что необходимая страница не будет найдена в кеше. В этом случае ее необходимо считать с диска в какой-либо буфер.

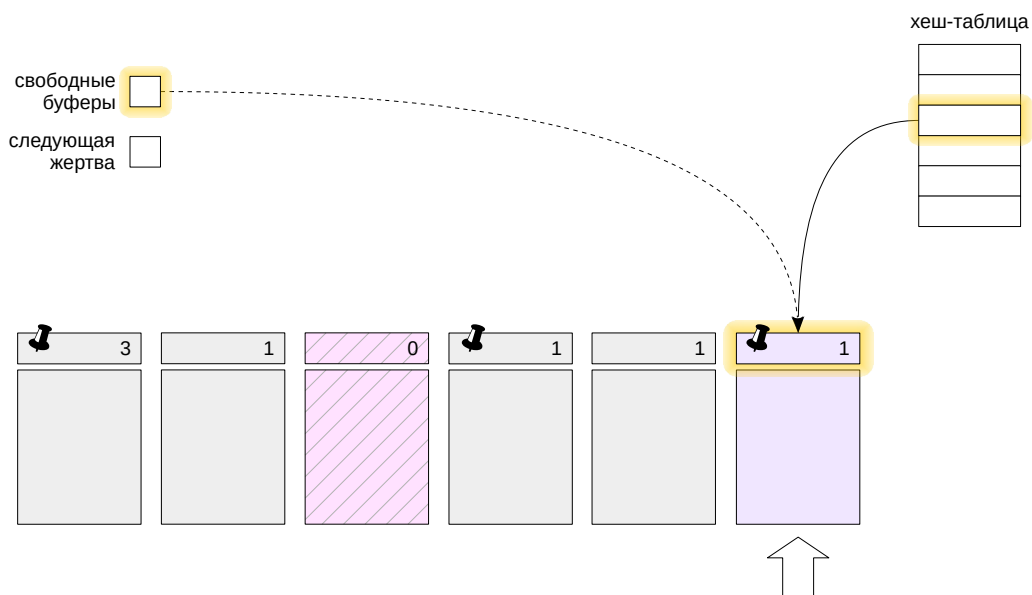
В первую очередь поиск подходящего буфера будет проходить по списку свободных буферов, если он не пуст.

Найденный по указателю буфер закрепляется, в него читается необходимая страница, число обращений устанавливается в единицу.

Указатель на свободный буфер передвигается на следующий по списку буфер, если он есть.

Кроме того, ссылку на загруженную страницу необходимо прописать в хеш-таблицу, чтобы в дальнейшем ее можно было найти.

# Чтение в свободный буфер



В конце концов будет заполнен и последний свободный буфер.

Теперь ссылка на свободный буфер пуста — в дальнейшем, чтобы прочитать новую страницу в любой занятый буфер, придется из этого буфера вытеснить страницу, которая там находится.

Буферы возвращаются в список свободных при удалении или опустошении таблицы или при отсечении части хвостовых страниц при очистке — то есть в случаях, когда страница в буфере не заменяется другой, а просто исчезает.



## Чтение в свободный буфер

Перезагрузим сервер, чтобы сбросить буферный кеш.

```
student$ sudo pg_ctlcluster 16 main restart
```

```
student$ psql wal_buffers
```

В кеше есть свободные буферы:

```
=> SELECT count(*) FROM pg_buffers WHERE usagecount IS NULL;

count
-----
16204
(1 row)
```

Выполним запрос к таблице:

```
=> EXPLAIN (analyze,buffers, costs off, timing off, summary off)
SELECT * FROM test;
```

```
QUERY PLAN
-----
Seq Scan on test (actual rows=1 loops=1)
  Buffers: shared read=1
Planning:
  Buffers: shared hit=12 read=7
(4 rows)
```

Строка «Buffers: shared read=1» показывает, что страницу пришлось прочитать с диска. Вот она:

```
=> SELECT * FROM pg_buffers_v WHERE relname = 'test';

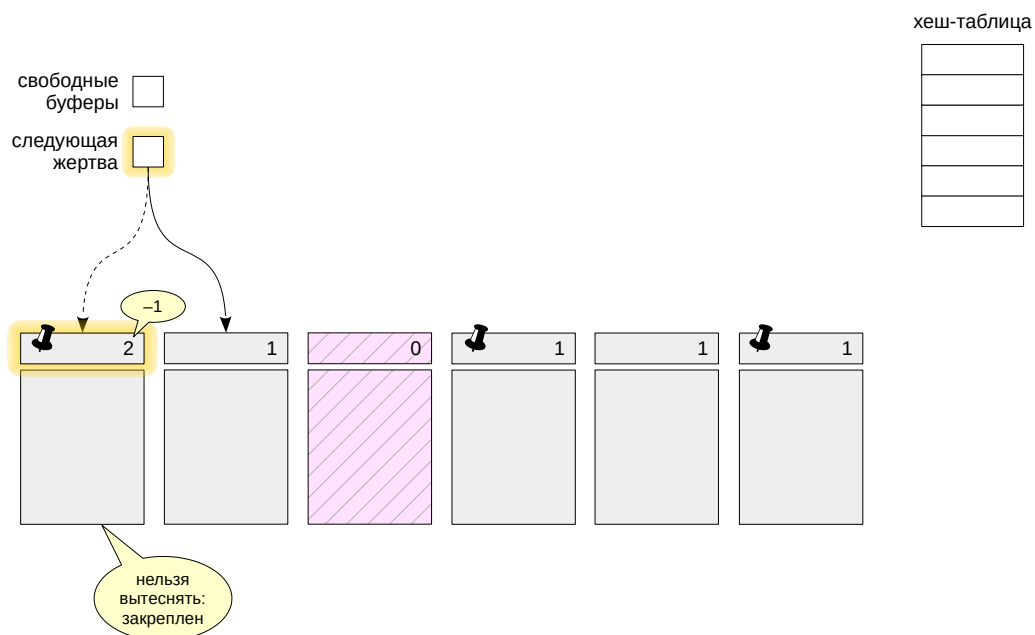
bufferid | relname | relfork | relblocknumber | isdirty | usagecount
-----+-----+-----+-----+-----+-----
189 | test | main | 0 | f | 1
(1 row)
```

Количество свободных буферов уменьшилось:

```
=> SELECT count(*) FROM pg_buffers WHERE usagecount IS NULL;

count
-----
16152
(1 row)
```

В кеш попала не только прочитанная страница, но и страницы таблиц системного каталога, которые потребовались планировщику.

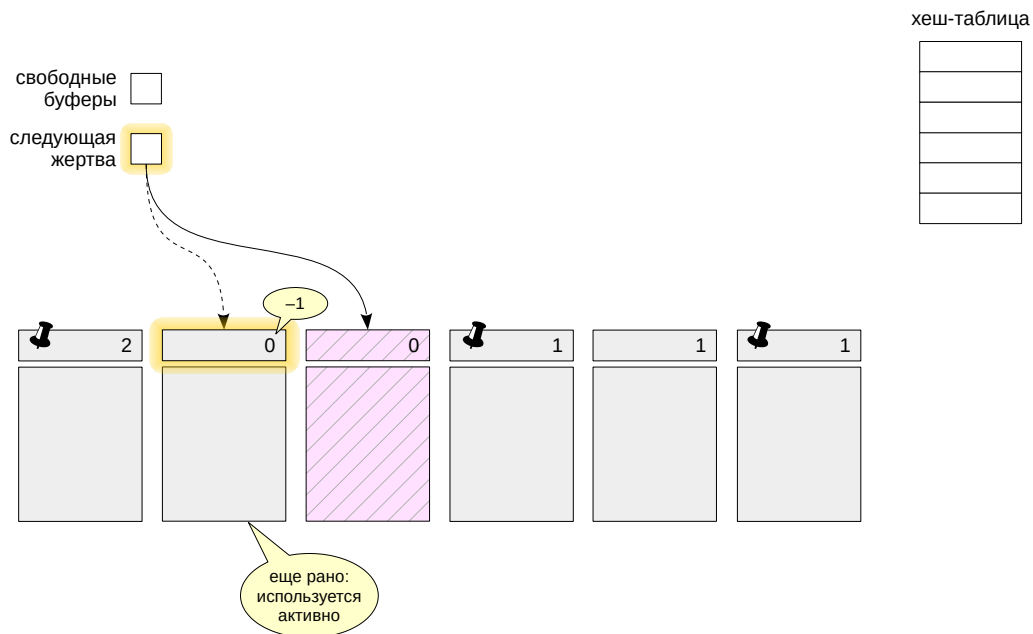


Механизм вытеснения использует указатель на следующую «жертву».

Алгоритм состоит в том, чтобы перебирать по кругу все буферы, уменьшая их счетчики обращений (usage count). Выбирается первый же буфер, у которого значение счетчика равно нулю и который в принципе может быть занят новой страницей. По-английски этот алгоритм называется clock-sweeper: аналогия с часовой стрелкой, которая пробегает циферблат по кругу.

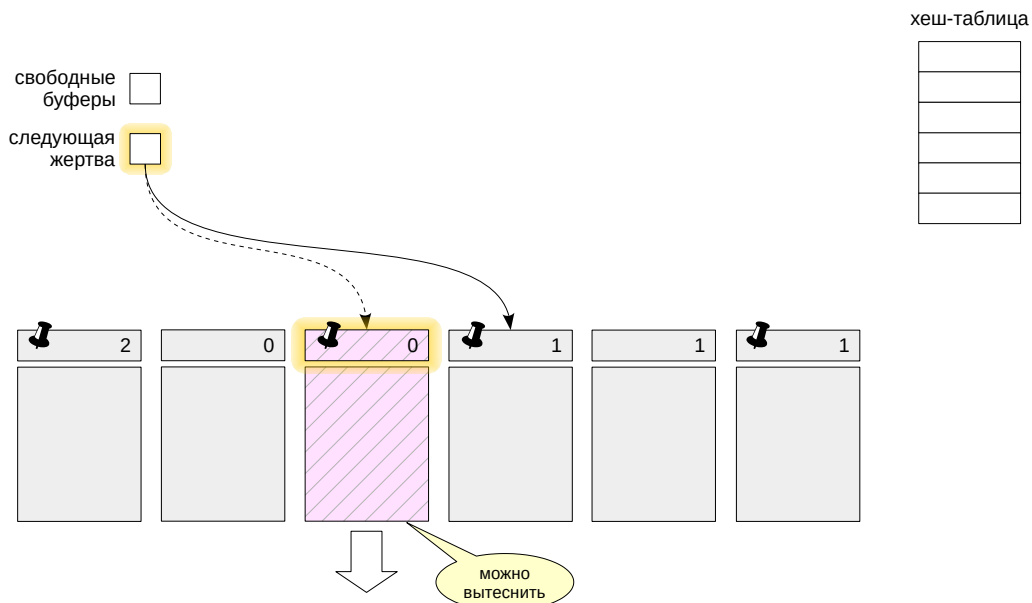
Чем больше значение счетчика у буфера (то есть чем чаще он используется), тем больше у него шансов задержаться в кеше. Максимальное значение счетчика обращений ограничено числом 5, чтобы избежать «наматывания кругов» при больших значениях.

В нашем примере процесс сначала обращается к буферу по указателю. Буфер закреплен (то есть используется каким-то процессом), и поэтому страница не может быть вытеснена из него. Тем не менее мы уменьшаем значение счетчика обращений на единицу и переходим к следующему буферу.



Следующий буфер не закреплен, однако его счетчик обращений больше нуля.

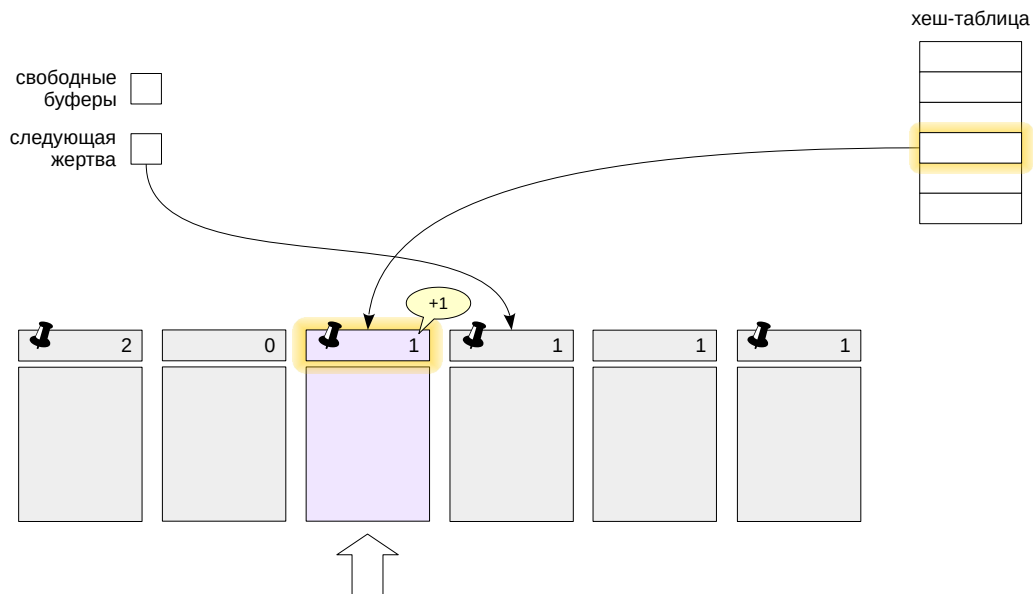
Мы уменьшаем счетчик на единицу и даем странице шанс остаться в кеше.



Наконец, мы доходим до незакрепленного буфера с нулевым счетчиком обращений. Страница из этого буфера и будет вытеснена.

Однако в нашем примере найденный буфер оказался грязным — он содержит измененные данные. Поэтому сначала страницу требуется сохранить на диск. Для этого буфер закрепляется (чтобы показать остальным процессам, что он используется), после чего страница записывается на диск.

Это не очень хорошая ситуация: процессу, который собирался прочитать страницу, придется ждать, пока «чужие» данные будут записаны. Этот эффект сглаживается процессами контрольной точки и фоновой записи, которые рассмотрены в теме «Контрольная точка».



После того как страница из грязного буфера записана на диск, в освободившийся буфер ранее рассмотренным образом читается новая страница.

Ссылка на следующую «жертву» уже указывает на следующий буфер, а у только что загруженного есть время нарастить счетчик обращений, пока указатель не обойдет по кругу весь буферный кеш и не вернется вновь.

## Настройка

 `shared_buffers = 128MB`



## Буферный кеш должен содержать «активные» данные

при меньшем размере постоянно вытесняются полезные страницы  
при большем размере бессмысленно растут накладные расходы  
начальное приближение — 1/4 ОЗУ

## Нужно учитывать двойное кеширование

если страницы нет в кеше СУБД, она может оказаться в кеше ОС  
алгоритм вытеснения ОС не учитывает специфики базы данных

Размер кеша устанавливается параметром `shared_buffers`. Значение по умолчанию — 128 МБ — сильно занижено.

Как выбрать подходящее значение? Даже самая большая база имеет ограниченный набор данных, с которыми ведется активная работа. В идеале этот набор (плюс некоторое место для «одноразовых» данных) должен помещаться в буферный кеш.

При меньшем размере кеша активно используемые страницы будут постоянно вытеснять друг друга, создавая избыточный ввод-вывод. Признаком такой ситуации может служить то, что все страницы в кеше имеют большое число обращений (`usage count`).

Однако при большом размере кеша будут расти накладные расходы на его поддержание.

В качестве первого приближения можно взять 1/4 оперативной памяти, но надо понимать, что оптимальное значение зависит не от размера ОЗУ, а от конкретных данных и нагрузки.

Не следует забывать и о том, что PostgreSQL работает с диском через операционную систему и, таким образом, происходит двойное кеширование: страницы попадают как в буферный кеш, так и в кеш ОС. Таким образом, «непопадание» в буферный кеш не всегда приводит к необходимости реального ввода-вывода. Но стратегия вытеснения ОС не учитывает специфики баз данных.

<https://postgrespro.ru/docs/postgresql/16/runtime-config-resource#RUNTIME-CONFIG-RESOURCE-MEMORY>

## Настройка размера кеша

Используя расширение `pg_buffercache`, можно наблюдать за состоянием кеша под разными углами.

Кроме информации о том, как представлена в кеше та или иная страница, можно, например, посмотреть распределение буферов по их «популярности».

```
=> SELECT usage_count, buffers AS count FROM pg_buffercache_usage_counts();
```

usage_count	count
0	16150
1	46
2	34
3	18
4	5
5	131

(6 rows)

Функция `pg_buffercache_usage_counts()` возвращает набор строк со сводной информацией о состоянии всех общих буферов, агрегированных по возможным значениям счетчика использования.

Как и представление `pg_buffercache`, функция не использует блокировки менеджера буферов, поэтому при параллельной работе в базе данных возможна незначительная погрешность результатов функции.

Посмотрим, какая доля каких таблиц закеширована (и насколько активно используются эти данные):

```
=> SELECT c.relname,
       count(*) blocks,
       round( 100.0 * 8192 * count(*) / pg_table_size(c.oid) ) "% of rel",
       round( 100.0 * 8192 * count(*) FILTER (WHERE b.usagcount > 3) / pg_table_size(c.oid) ) "% hot"
FROM pg_buffercache b
JOIN pg_class c ON pg_relation_filenode(c.oid) = b.relfilenode
WHERE b.reldatabase IN (
    0, (SELECT oid FROM pg_database WHERE datname = current_database())
)
AND b.usagcount is not null
GROUP BY c.relname, c.oid
ORDER BY 2 DESC
LIMIT 10;
```

relname	blocks	% of rel	% hot
pg_attribute	31	50	47
pg_proc	14	13	2
pg_class	14	78	78
pg_operator	11	61	39
pg_proc_oid_index	9	82	27
pg_attribute_relid_attnum_index	8	73	55
pg_proc_proname_args_nsp_index	7	22	3
pg_amproc	5	56	11
pg_amop	5	45	27
pg_operator_oprname_l_r_n_index	5	83	33

(10 rows)

А сводную информацию о буферном кеше нам покажет другая функция из расширения:

```
=> SELECT * FROM pg_buffercache_summary() \gx
```

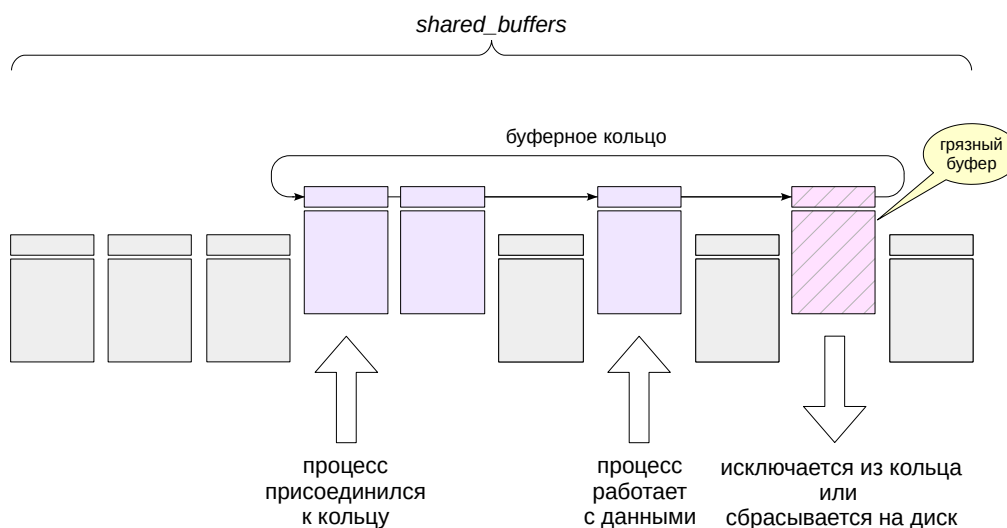
```
-[ RECORD 1 ]--+-----
buffers_used   | 265
buffers_unused | 16119
buffers_dirty  | 6
buffers_pinned | 0
usagecount_avg | 3.709433962264151
```

Подобные запросы могут подсказать, насколько активно используется буферный кеш и дать пищу для размышлений о том, стоит ли увеличивать или уменьшать его размер. Надо учитывать, что такие запросы надо повторять несколько раз: цифры будут меняться в определенных пределах, а набор результатов, полученный для всех буферов, в целом может оказаться несогласованным.

Информацию, подобную той, что возвращают функции `pg_buffercache_summary()` и `pg_buffercache_usage_counts()`, можно получить и с помощью представления `pg_buffercache`, однако использование функций менее затратно, что удобно для использования в системах мониторинга.







При операциях, выполняющих массовое чтение или запись данных, есть опасность вытеснения полезных страниц из буферного кеша «одноразовыми» данными. Чтобы этого не происходило, для таких операций используются *буферные кольца* — выделенные небольшие части буферного кеша. Вытеснение действует в этом случае только в пределах кольца, и остальные данные в кеше не страдают.

Если в процессе работы в кольце появляются грязные буферы, то они будут отключены от кольца или записаны на диск.

## Последовательное чтение

32 страницы  
грязный буфер исключается из кольца

## Очистка

`vacuum_buffer_usage_limit = 32`  
грязный буфер вытесняется на диск

## Массовая запись

$\leq 2048$  страниц  
грязный буфер вытесняется на диск

При последовательном чтении (sequential scan) таблиц используется кольцо размером 32 страницы. Если при работе с этими данными появляются грязные буферы (в результате простановки битов-подсказок или при выполнении UPDATE), они отключаются от кольца, возвращаются в основной кеш и вытесняются затем уже на общих основаниях. А вместо отключенного буфера в кольцо из кеша подключается другой буфер. Такая стратегия рассчитана на то, что данные в основном читаются, а не изменяются.

Если в процессе чтения таблицы другому процессу тоже потребуются эти данные, он подключается к уже имеющемуся буферному кольцу, а потом отдельно дочитывает страницы, которых в кольце не было.

Для работы очистки используется кольцо, размер которого задается параметром `vacuum_buffer_usage_limit`. По умолчанию его значение невелико (32 страницы), так как замедление очистки обычно не столь критично, как замедление пользовательских процессов (исключение — защитный режим: в нем очистка может использовать весь кеш).

Для массовых операций записи, таких как COPY или CREATE TABLE AS SELECT, кольцо имеет достаточно большой размер (обычно 2048 страниц, но не больше одной восьмой всего буферного кеша).

## Массовое вытеснение

Для полного чтения таблиц, размер которых превышает четверть буферного кеша, используется буферное кольцо. Размер кеша (в страницах):

```
=> SELECT setting FROM pg_settings WHERE name='shared_buffers';

setting
-----
16384
(1 row)
```

Добавим строк в таблицу:

```
=> INSERT INTO test SELECT repeat('A',1000) FROM generate_series(1,30_000);
```

```
INSERT 0 30000
```

```
=> ANALYZE test;
```

```
ANALYZE
```

```
=> SELECT relpages FROM pg_class WHERE relname = 'test';
```

```
relpages
-----
4286
(1 row)
```

Перезагрузим сервер, чтобы сбросить буферный кеш.

```
student$ sudo pg_ctlcluster 16 main restart
```

```
student$ psql wal_buffers
```

И сбросим общую статистику ввода-вывода:

```
=> SELECT pg_stat_reset_shared('io');
```

```
pg_stat_reset_shared
-----
(1 row)
```

Прочитаем данные из таблицы:

```
=> EXPLAIN (analyze,buffers, costs off, timing off, summary off)
SELECT count(*) FROM test;
```

```
QUERY PLAN
-----
Aggregate (actual rows=1 loops=1)
  Buffers: shared read=4286
    -> Seq Scan on test (actual rows=30001 loops=1)
        Buffers: shared read=4286
Planning:
  Buffers: shared hit=14 read=8
(6 rows)
```

Были прочитаны все страницы с данными, но сколько буферов кеша ими занято?

```
=> SELECT count(*) FROM pg_buffers WHERE relname = 'test';
```

```
count
-----
32
(1 row)
```

При сканировании большой таблицы размер кольца — 32 буфера.

---

В 16-й версии PostgreSQL появилось представление `pg_stat_io`, которое показывает статистику ввода/вывода с точки зрения СУБД (кеширование на уровне ОС в нем не учитывается).

```
=> SELECT context, reads, hits, reuses FROM pg_stat_io
WHERE backend_type = 'client backend' AND
```

```
    object = 'relation' AND
    context IN ('normal', 'bulkread');
```

context	reads	hits	reuses
bulkread	4286	0	4254
normal	92	4211	

(2 rows)

Строка с контекстом normal показывает операции чтения-записи в общей части буферного кеша, а bulkread — в буферных кольцах.

- reads — количество чтений из файла,
- hits — сколько раз страница нашлась в кеше,
- reuses — число повторных использований буферов в буферном кольце.

Разность между reads и reuses в строке bulkread равна размеру кольца (32).

А теперь сбросим статистику и изменим все строки таблицы. При этом буферы со страницами будут отсоединяться от буферного кольца и оставаться в кеше:

```
=> SELECT pg_stat_reset_shared('io');
```

```
pg_stat_reset_shared
-----
(1 row)
```

```
=> EXPLAIN (analyze,buffers, costs off, timing off, summary off)
UPDATE test SET t = t || '!';
```

#### QUERY PLAN

```
-----
Update on test (actual rows=0 loops=1)
  Buffers: shared hit=98604 read=4256 dirtied=8574 written=4312
  -> Seq Scan on test (actual rows=30001 loops=1)
        Buffers: shared hit=32 read=4254 written=25
Planning:
  Buffers: shared hit=4 read=2 dirtied=1
(6 rows)
```

Сейчас в кеше находятся все или почти все страницы таблицы, количество которых удвоилось из-за появления новых версий строк:

```
=> SELECT relfork, count(*) FROM pg_buffercache_v WHERE relname = 'test' GROUP BY relfork;
```

relfork	count
fsm	3
main	8547

(2 rows)

Еще раз заглянем в статистику.

```
=> SELECT context, reads, hits, reuses FROM pg_stat_io
WHERE backend_type = 'client backend' AND
```

```
    object = 'relation' AND
    context IN ('normal', 'bulkread');
```

context	reads	hits	reuses
bulkread	4254	32	25
normal	10	220917	

(2 rows)

Таблица еще раз полностью сканируется через буферное кольцо. При этом 32 страницы остались в кольце от предыдущего сканирования, а остальные пришлось прочитать заново. Измененные страницы одна за другой передаются в общий кеш, обращения к ним учитываются в строке normal. При этом обновление каждой из 30 тысяч строк таблицы требует нескольких обращений к страницам, поэтому значение в поле hits довольно велико.

## Для временных таблиц

видны только одному сеансу — нет смысла использовать общий кеш  
существуют в пределах сеанса — не жалко потерять при сбое

## Особенности

не требуются блокировки  
память выделяется по необходимости в пределах *temp\_buffers*  
обычный алгоритм вытеснения  
механизм буферных колец не используется

Исключением из общего правила являются временные таблицы. Поскольку временные данные видны только одному процессу, им нечего делать в общем буферном кеше. Более того, временные данные существуют только в рамках одного сеанса, так что их не нужно защищать от сбоя.

Для временных данных используется облегченный локальный кеш.

Поскольку локальный кеш доступен только одному процессу, для него не требуются блокировки. Память выделяется по мере необходимости (в пределах, заданных параметром *temp\_buffers*), ведь временные таблицы используются далеко не во всех сеансах. В локальном кеше используется обычный алгоритм вытеснения.

Временные таблицы на физическом уровне хранятся схожим с обычными таблицами образом. Параметр *temp\_tablespace* задает табличные пространства, в которых будут создаваться временные объекты (временные таблицы и индексы временных таблиц), когда в команде CREATE табличное пространство не указывается явно.

Значение параметра *temp\_tablespace* может содержать список имен табличных пространств. Когда этот список содержит больше одного имени, PostgreSQL выбирает из него случайный элемент при создании каждого временного объекта; однако при создании последующих объектов внутри транзакции табличные пространства перебираются последовательно. Если же в списке оказывается пустая строка (значение по умолчанию), PostgreSQL будет использовать табличное пространство по умолчанию для текущей базы данных.

## Временные таблицы

Создадим временную таблицу с одной строкой:

```
=> CREATE TEMP TABLE test_tmp(  
    t text  
);
```

CREATE TABLE

```
=> INSERT INTO test_tmp VALUES ('a row');
```

INSERT 0 1

В плане выполнения запроса обращение к локальному кешу выглядит как «Buffers: local»:

```
=> EXPLAIN (analyze,buffers,costs off,timing off,summary off)  
SELECT * FROM test_tmp;
```

QUERY PLAN

```
-----  
Seq Scan on test_tmp (actual rows=1 loops=1)  
  Buffers: local hit=1  
Planning:  
  Buffers: shared hit=4  
(4 rows)
```

```
=> SELECT context, hits, extends FROM pg_stat_io  
WHERE backend_type = 'client backend' AND  
      object = 'temp relation' AND context = 'normal';
```

```
context | hits | extends  
-----+-----+-----  
normal  |    1 |        1  
(1 row)
```

Видим, что вставка строки привела к расширению таблицы (extends).

## Расширение pg\_prewarm

- ручной прогрев кеша операционной системы
- ручной прогрев буферного кеша
- автоматический прогрев буферного кеша при перезапуске сервера

После перезапуска сервера буферный кеш пуст и должно пройти некоторое время, чтобы он «прогрелся», то есть набрал актуальные активно используемые данные. Иногда может оказаться полезным сразу прочитать в кеш данные определенных таблиц.

Расширение `pg_prewarm` позволяет в ручном режиме прочитать некоторые таблицы либо в кеш операционной системы, либо и в буферный кеш СУБД тоже.

Расширение также позволяет в автоматическом режиме восстановить содержимое буферного кеша после перезапуска сервера. Для этого необходимо подключить библиотеку расширения, добавив ее в параметр `shared_preload_libraries`.

<https://postgrespro.ru/docs/postgresql/16/pgprewarm>

## Прогрев кеша

Рассмотрим самый простой сценарий использования расширения для прогрева кеша.

```
=> CREATE EXTENSION pg_prewarm;
```

```
CREATE EXTENSION
```

В очередной раз перезапустим сервер, чтобы в кеше не было данных таблицы test:

```
student$ sudo pg_ctlcluster 16 main restart
```

```
student$ psql wal_buffers
```

```
=> SELECT count(*) FROM pg_buffers WHERE relname = 'test';
```

```
count
-----
      0
(1 row)
```

Вызов функции pg\_prewarm без дополнительных параметров полностью считывает основной слой указанной таблицы в буферный кеш:

```
=> SELECT pg_prewarm('test');
```

```
pg_prewarm
-----
      8572
(1 row)
```

```
=> SELECT relname, count(*) FROM pg_buffers WHERE relname = 'test' GROUP BY relname;
```

```
relname | count
-----+-----
main    | 8572
(1 row)
```



Вся работа с данными происходит через буферный кеш

Редко используемые страницы вытесняются,  
часто используемые — остаются

Буферный кеш сокращает объем ввода-вывода,  
но приводит к необходимости журналирования

1. Создайте таблицу и вставьте в нее некоторое количество строк. Определите, сколько занимает таблица
  - а) страниц на диске,
  - б) буферов в кеше.
2. Узнайте количество грязных буферов в кеше на текущий момент. Выполните контрольную точку командой СНЕСКРОИТ. Сколько грязных буферов осталось теперь?
3. Подключите библиотеку расширения pg\_prewarm и проверьте, что после перезапуска сервера содержимое буферного кеша восстанавливается.

Чтобы проверить содержимое буферного кеша, используйте расширение pg\_buffercache:

<https://postgrespro.ru/docs/postgresql/16/pgbuffercache.html>

## 1. Таблица в кеше

Создадим таблицу:

```
=> CREATE DATABASE wal_buffercache;
```

```
CREATE DATABASE
```

```
=> \c wal_buffercache
```

You are now connected to database "wal\_buffercache" as user "student".

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

```
CREATE TABLE
```

```
=> INSERT INTO t SELECT 1 FROM generate_series(1,10_000);
```

```
INSERT 0 10000
```

Выполним VACUUM — это обеспечит нам наличие всех слоев:

```
=> VACUUM t;
```

```
VACUUM
```

Сколько страниц на диске занимает таблица?

```
=> SELECT setting FROM pg_settings WHERE name = 'block_size';
```

```
setting
-----
8192
(1 row)
```

```
=> SELECT pg_table_size('t') / 8192;
```

```
?column?
-----
49
(1 row)
```

Из них основной слой:

```
=> SELECT pg_relation_size('t','main') / 8192;
```

```
?column?
-----
45
(1 row)
```

Карта свободного пространства:

```
=> SELECT pg_relation_size('t','fsm') / 8192;
```

```
?column?
-----
3
(1 row)
```

И карта видимости:

```
=> SELECT pg_relation_size('t','vm') / 8192;
```

```
?column?
-----
1
(1 row)
```

Сколько буферов в кеше занимает таблица?

```
=> CREATE EXTENSION pg_buffercache;
```

```
CREATE EXTENSION
```

```
=> SELECT CASE relforknumber
        WHEN 0 THEN 'main'
        WHEN 1 THEN 'fsm'
        WHEN 2 THEN 'vm'
    END relfork,
    count(*)
FROM pg_buffercache b,
    pg_class c
WHERE b.reldatabase = (
    SELECT oid FROM pg_database WHERE datname = current_database()
)
AND c.oid = 't'::regclass
AND b.relfilenode = c.relfilenode
GROUP BY 1;
```

```
relfork | count
-----+-----
fsm      |      3
main     |     45
vm       |      1
(3 rows)
```

## 2. Грязные буферы в кеше

```
=> SELECT buffers_dirty FROM pg_buffercache_summary();

buffers_dirty
-----
          981
(1 row)
```

Выполним контрольную точку:

```
=> CHECKPOINT;

CHECKPOINT

=> SELECT buffers_dirty FROM pg_buffercache_summary();

buffers_dirty
-----
           0
(1 row)
```

Грязных буферов не осталось. Подробнее о контрольной точке рассказывается в отдельной теме.

## 3. Автоматический прогрев кеша

Подключим библиотеку pg\_prewarm и перезапустим сервер.

```
=> ALTER SYSTEM SET shared_preload_libraries = 'pg_prewarm';

ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 16 main restart
```

```
student$ psql wal_buffers
```

Теперь отдельный фоновый процесс будет сбрасывать на диск список страниц, находящихся в буферном кеше, раз в pg\_prewarm.autoprewarm\_interval единиц времени.

```
student$ ps -o pid,command --ppid $(sudo head -n 1 /var/lib/postgresql/16/main/postmaster.pid) | grep prewarm

175436 postgres: 16/main: autoprewarm leader
```

Прочитаем таблицу t в буферный кеш:

```
=> CREATE EXTENSION pg_prewarm;
```

```
CREATE EXTENSION
```

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

```
pg_prewarm
-----
         45
(1 row)
```

```
=> SELECT count(*)
FROM pg_buffercache
WHERE relfilenode = pg_relation_filenode('t'::regclass);

count
-----
      45
(1 row)
```

Можно либо подождать, либо сбросить список страниц вручную:

```
=> SELECT autoprewarm_dump_now();

autoprewarm_dump_now
-----
                  299
(1 row)
```

В файл записываются идентификаторы базы данных, табличного пространства и файла, номер слоя и номер блока:

```
student$ sudo head -n 10 /var/lib/postgresql/16/main/autoprewarm.blocks
```

```
<<299>>
0,1664,1262,0,0
0,1664,1260,0,0
1,1663,1259,0,0
1,1663,1259,0,1
1,1663,1259,0,2
1,1663,1259,0,3
1,1663,1249,0,0
1,1663,1249,0,1
1,1663,1249,0,2
```

Снова перезапустим сервер.

```
student$ sudo pg_ctlcluster 16 main restart
```

```
student$ psql wal_buffercache
```

После запуска фоновый процесс прочитает в буферный кеш все страницы, указанные в файле.

```
=> SELECT count(*)
FROM pg_buffercache
WHERE relfilenode = pg_relation_filenode('t'::regclass);

count
-----
      45
(1 row)
```

Вернемся к настройкам по умолчанию.

```
=> ALTER SYSTEM RESET shared_preload_libraries;
```

```
ALTER SYSTEM
```

```
student$ sudo pg_ctlcluster 16 main restart
```