

Авторы: Егор Рогов, Павел Лузанов



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

# Темы

Устройство буферного кеша

Алгоритм вытеснения

Журнал предзаписи

Контрольная точка

# Буферный кеш

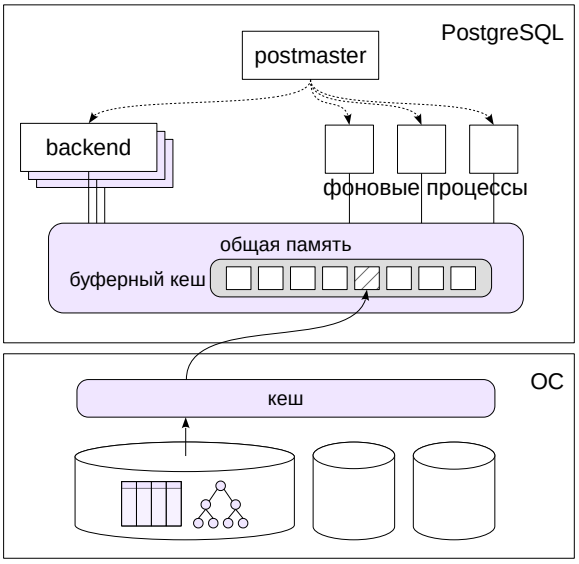
Массив буферов

страница данных (8 Кбайт)

доп. информация

Блокировки в памяти

для совместного доступа






Буферный кеш используется для сглаживания скорости работы оперативной памяти и дисков. Он состоит из массива буферов, которые содержат страницы данных и дополнительную информацию (например, имя файла и положение страницы внутри этого файла).

Размер страницы обычно составляет 8 Кбайт (его можно изменить, но только при сборке PostgreSQL, и обычно в этом нет смысла).

Любая работа со страницами данных проходит через буферный кеш. Если какой-либо процесс собирается работать со страницей, он в первую очередь пытается найти ее в кеше. Если там страницы нет, процесс обращается к операционной системе с просьбой прочитать эту страницу и помещает ее в буферный кеш. (Обратите внимание, что ОС может прочитать страницу с диска, а может обнаружить ее в собственном кеше.)

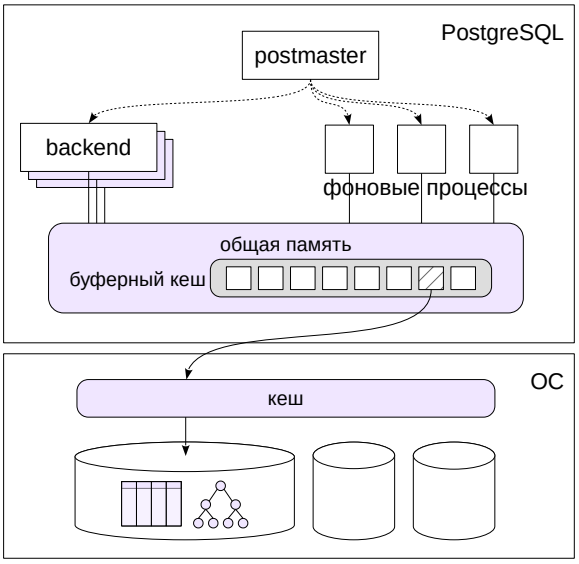
После того, как странице записана в буферный кеш, к ней можно обращаться многократно без накладных расходов на вызовы ОС.

Однако буферный кеш, как и другие структуры общей памяти, защищен блокировками для управления одновременным доступа. Хотя блокировки и реализованы эффективно, доступ к буферному кешу далеко не так быстр, как простое обращение к оперативной памяти. Поэтому в общем случае чем меньше данных читает и изменяет запрос, тем быстрее он будет работать.

# Вытеснение

Вытеснение редко используемых страниц

- «грязный» буфер записывается на диск
- на освободившееся место читается другая страница






Размер буферного кеша обычно не так велик, чтобы база данных помещалась в него целиком. Его ограничивают и доступная оперативная память, и возрастающие при его увеличении накладные расходы. Поэтому при чтении очередной страницы рано или поздно окажется, что место в буферном кеше закончилось. В этом случае применяется *вытеснение* страниц.

Алгоритм вытеснения выбирает в кеше страницу, которая в последнее время использовалась реже других, и заменяет ее новой. Если выбранная страница изменялась, то ее предварительно надо записать на диск, чтобы не потерять изменения (буфер, содержащий измененную страницу, называется «грязным»).

Такой алгоритм вытеснения называется LRU — Least Recently Used. Он сохраняет в кеше данные, с которыми происходит активная работа. Таких «горячих» данных обычно не так много, и при достаточном объеме буферного кеша получается существенно сократить количество обращений к ОС (и дисковых операций).

### Влияние буферного кеша на выполнение запросов

Создадим базу данных и в ней таблицу:

```
=> CREATE DATABASE arch_wal_overview;

CREATE DATABASE

=> \c arch_wal_overview

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

=> CREATE TABLE t(n integer);

CREATE TABLE
```

Заполним таблицу некоторым количеством строк:

```
=> INSERT INTO t SELECT id FROM generate_series(1,100000) AS id;

INSERT 0 100000

=> VACUUM ANALYZE t;

VACUUM
```

Теперь перезапустим сервер, чтобы содержимое буферного кеша сбросилось.

```
=> \q

student$ sudo pg_ctlcluster 12 main restart

student$ psql arch_wal_overview
```

Сравним, что происходит при первом и при втором выполнении одного и того же запроса. В этом курсе мы не рассматриваем подробно планы запросов, но иногда будем в них заглядывать. Сейчас мы воспользуемся командой EXPLAIN ANALYZE, которая выполняет запрос и выводит не только план выполнения, но и дополнительную информацию:

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

                QUERY PLAN
-------------------------------------------
 Seq Scan on t (actual rows=100000 loops=1)
   Buffers: shared read=443
 Planning Time: 0.163 ms
 Execution Time: 10.643 ms
(4 rows)
```

Строка «Buffers: shared» показывает использование буферного кеша.

- read — количество буферов, в которые пришлось прочитать страницы с диска.

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

                QUERY PLAN
-------------------------------------------
 Seq Scan on t (actual rows=100000 loops=1)
   Buffers: shared hit=443
 Planning Time: 0.031 ms
 Execution Time: 10.557 ms
(4 rows)
```

- hit — количество буферов, в которых нашлись нужные для запроса страницы.

Обратите внимание, что во второй раз уменьшилось и время выполнения запроса, и время его планирования (потому что таблицы системного каталога тоже кешируются).

# Журнал (WAL)

Проблема: при сбое теряются данные из оперативной памяти, не записанные на диск

Журнал

- поток информации о выполняемых действиях, позволяющий повторно выполнить потерянные при сбое операции
- запись попадает на диск раньше, чем измененные данные

Журнал защищает

- страницы таблиц, индексов и других объектов
- статус транзакций (xact)

Журнал не защищает

- временные и нежурналируемые таблицы



Наличие буферного кеша (и других буферов в оперативной памяти) увеличивает производительность, но уменьшает надежность. В случае сбоя в СУБД содержимое буферного кеша потеряется. Если сбой произойдет в операционной системе или на аппаратном уровне, то пропадет содержимое и буферов ОС (но с этим справляется сама операционная система).

Для обеспечения надежности PostgreSQL использует журналирование. При выполнении любой операции формируется запись, содержащая минимально необходимую информацию для того, чтобы операцию можно было выполнить повторно. Такая запись должна попасть на диск (или другой энергонезависимый накопитель) раньше, чем будут записаны изменяемые операцией данные (поэтому журнал и называется *журналом предзаписи*, write-ahead log).

Журнал защищает все объекты, работа с которыми ведется в оперативной памяти: таблицы, индексы и другие объекты, статус транзакций.

В журнал не попадают данные о *временных таблицах* (такие таблицы доступны только создавшему из пользователю и только на время сеанса или транзакции) и о *нежурналируемых таблицах* (такие таблицы ничем не отличаются от обычных, кроме того, что не защищены журналом). В случае сбоя такие таблицы просто очищаются. Смысл их существования в том, что работа с ними происходит быстрее.

https://postgrespro.ru/docs/postgresql/12/wal-intro

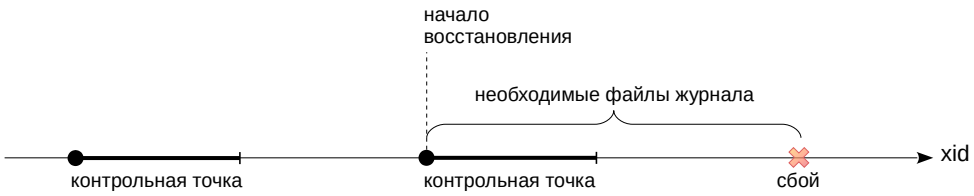
# Контрольная точка

Периодический сброс всех грязных буферов на диск

- гарантирует попадание на диск всех изменений до контрольной точки
- ограничивает размер журнала, необходимого для восстановления

Восстановление при сбое

- начинается с последней контрольной точки
- последовательно проигрываются записи, если изменений нет на диске






Когда сервер PostgreSQL запускается после сбоя, он входит в режим восстановления. На диске в это время находится несогласованная информация: одни страницы были записаны в одно время, другие — в другое.

Чтобы восстановить согласованность, PostgreSQL читает журнал WAL и последовательно проигрывает каждую журнальную запись, если соответствующее изменение не попало на диск. Таким образом восстанавливаются все транзакции, кроме тех, запись о фиксации которых не успела попасть в журнал (такие транзакции обрываются).

Однако объем журнала за время работы сервера мог бы достигнуть гигантских размеров. Хранить его целиком и целиком просматривать при сбое совершенно не реально. Поэтому СУБД периодически выполняет *контрольную точку:* принудительно сбрасывает на диск все грязные буферы (включая буферы xact, в которых хранится состояние транзакций). Это гарантирует, что изменения всех транзакций до момента контрольной точки находятся на диске.

Контрольная точка может занимать много времени, и это нормально. Собственно «точка», о которой мы говорим как о моменте времени — это начало процесса. Но точка считается выполненной только после того, как записаны *все* грязные буферы, которые имелись на момент начала процесса.

Восстановление после сбоя начинается с ближайшей контрольной точки, что позволяет хранить только файлы журнала, записанные с момента последней пройденной контрольной точки.

### Восстановление при помощи журнала

Файлы журнала хранятся в отдельном каталоге; они не являются частью какой-либо базы данных. В этот каталог можно заглянуть в файловой системе, а можно воспользоваться функцией:

```
=> SELECT * FROM pg_ls_waldir() ORDER BY name;
```

```
           name           |   size   |      modification
--------------------------+----------+------------------------
 000000010000000000000001 | 16777216 | 2021-01-12 20:30:39+03
 000000010000000000000002 | 16777216 | 2021-01-12 20:30:39+03
(2 rows)
```

PostgreSQL удаляет файлы, не требующиеся для восстановления, по мере необходимости после выполнения контрольной точки.

---

Внесем какие-нибудь изменения:

```
=> DELETE FROM t;
```

```
DELETE 100000
```

```
=> INSERT INTO t(n) VALUES (0);
```

```
INSERT 0 1
```

Измененные табличные страницы находятся в буферном кеше, но еще не записаны на диск. Сымитируем сбой системы, остановив ее в режиме immediate. При обычной остановке сервер выполняет контрольную точку, чтобы записать все грязные страницы на диск, но в режиме immediate сервер выключается без контрольной точки.

```
student$ sudo pg_ctlcluster 12 main stop -m immediate --skip-systemctl-redirect
```

```
student$ sudo pg_ctlcluster 12 main start
```

При старте происходит восстановление согласованности данных с помощью журнала. Проверим:

```
student$ psql arch_wal_overview
```

```
=> SELECT * FROM t;
```

```
 n
---
 0
(1 row)
```

Все изменения были восстановлены.

# Производительность

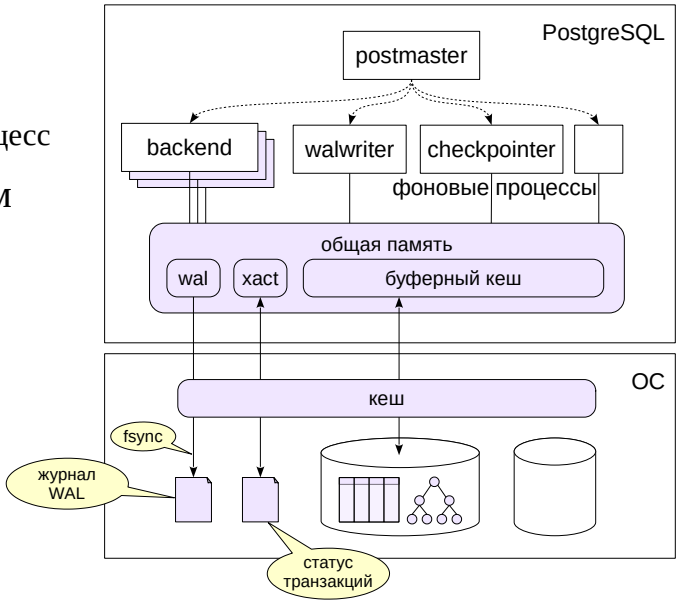
Postgres PROFESSIONAL

Синхронный режим

- запись при фиксации
- обслуживающий процесс

Асинхронный режим

- фоновая запись
- walwriter






Механизм журналирования более эффективен, чем работа напрямую с диском без буферного кеша. Во-первых, размер журнальных записей меньше, чем размер целой страницы данных; во-вторых, журнал записывается строго последовательно (и обычно не читается, пока не случится сбой), с чем вполне справляются простые HDD-диски.

На эффективность можно также влиять настройкой. Если запись происходит сразу (синхронно), то гарантируется, что зафиксированная транзакция не пропадет. Но запись — довольно дорогая операция, в течение которой обслуживающий процесс, выполняющий фиксацию, вынужден ждать. Чтобы журнальная запись не «застряла» в кеше операционной системы, выполняется вызов fsync: PostgreSQL полагается на то, что это гарантирует попадание данных на энергонезависимый носитель.

Поэтому есть и режим отложенной (асинхронной) записи. В этом случае записи пишутся фоновым процессом walwriter постепенно, с небольшой задержкой. Надежность уменьшается, зато производительность увеличивается. В этом случае после сбоя также гарантируется восстановление согласованности, но некоторые последние зафиксированные транзакции могут пропасть.

# Итоги

Буферный кеш существенно ускоряет работу,
уменьшая число дисковых операций

Надежность обеспечивается журналированием

Размер журнала ограничен благодаря контрольным точкам

Журнал удобен и используется во многих случаях

- для восстановления после сбоя
- при резервном копировании
- для репликации между серверами

# Практика

1. Проверьте, как используется буферный кеш в случае обновления одной строки в обычной и во *временной* таблице. Попробуйте объяснить отличие.
2. Создайте *нежурналируемую* таблицу и вставьте в нее несколько строк. Сымитируйте сбой системы, остановив сервер в режиме immediate, как в демонстрации. Запустите сервер и проверьте, что произошло с таблицей. Найдите в журнале сообщений сервера упоминание о восстановлении после сбоя.



1. Временные таблицы выглядят так же, как обычные, но время их жизни — текущий сеанс. Такая таблица видна тоже только в текущем сеансе.

Воспользуйтесь командой

```
EXPLAIN (analyze, buffers, costs off, timing off)
```

как было показано в демонстрации.

2. Останов в режиме immediate выполняется так:

```
sudo pg_ctlcluster 12 main stop -m immediate --skip-systemctl-redirect
```

Ключ --skip-systemctl-redirect нужен здесь из-за того, что используется PostgreSQL, установленный в Ubuntu из пакета. Он управляется командой pg_ctlcluster, которая вызывает утилиту systemctl, которая «теряет» режим по пути к pg_ctl. Ключ позволяет обойтись без systemctl и передать информацию pg_ctl.

## 1. Использование кеша для обычных и временных таблиц

```
=> CREATE DATABASE arch_wal_overview;
```

```
CREATE DATABASE
```

```
=> \c arch_wal_overview
```

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

Создадим обычную таблицу с одной строкой...

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

```
CREATE TABLE
```

```
=> INSERT INTO t(n) VALUES (1);
```

```
INSERT 0 1
```

...и такую же временную таблицу.

```
=> CREATE TEMPORARY TABLE tt(n integer);
```

```
CREATE TABLE
```

```
=> INSERT INTO tt(n) VALUES (1);
```

```
INSERT 0 1
```

Обновление строки в обычной таблице:

```
=> EXPLAIN (analyze, buffers, costs off, timing off)
UPDATE t SET n = n + 1;
```

```
                QUERY PLAN
-------------------------------------------
 Update on t (actual rows=0 loops=1)
   Buffers: shared hit=2
   ->  Seq Scan on t (actual rows=1 loops=1)
         Buffers: shared hit=1
 Planning Time: 0.166 ms
 Execution Time: 0.071 ms
(6 rows)
```

- При сканировании таблицы (Seq Scan) страница была найдена в буферном кеше (shared hit=1).
- При обновлении потребовалось прочитать две страницы (shared hit=2). Вторая страница относится к карте видимости (с которой мы познакомимся в модуле «Организация данных»).

Обновление строки во временной таблице:

```
=> EXPLAIN (analyze, buffers, costs off, timing off)
UPDATE tt SET n = n + 1;
```

```
                 QUERY PLAN
--------------------------------------------
 Update on tt (actual rows=0 loops=1)
   Buffers: local hit=2
   ->  Seq Scan on tt (actual rows=1 loops=1)
         Buffers: local hit=1
 Planning Time: 0.057 ms
 Execution Time: 0.037 ms
(6 rows)
```

Отличие в том, что вместо общего буферного кеша, расположенного в разделяемой памяти сервера, используется локальный кеш текущего сеанса (local).

## 2. Нежурналируемые таблицы при сбое

Создаем нежурналируемую таблицу:

```
=> CREATE UNLOGGED TABLE u(s text);
```

```
CREATE TABLE
```

```
=> INSERT INTO u VALUES ('Привет!');
```

```
INSERT 0 1
```

Имитируем сбой:

```
student$ sudo pg_ctlcluster 12 main stop -m immediate --skip-systemctl-redirect
```

Запускаем сервер:

```
student$ sudo pg_ctlcluster 12 main start
```

```
student$ psql arch_wal_overview
```

```
=> SELECT * FROM u;
```

```
 s
---
(0 rows)
```

Таблица на месте, но она пуста. Содержимое нежурналируемых таблиц не восстанавливается при сбое; вместо этого такие таблицы «обнуляются».

Проверим журнал сообщений:

```
student$ tail -n 5 /var/log/postgresql/postgresql-12-main.log
```

```
2021-01-12 20:35:23.637 MSK [196593] LOG:  database system was not properly shut down; automatic recovery in progress
2021-01-12 20:35:23.641 MSK [196593] LOG:  redo starts at 0/3ACA9118
2021-01-12 20:35:23.642 MSK [196593] LOG:  invalid record length at 0/3ACC8298: wanted 24, got 0
2021-01-12 20:35:23.642 MSK [196593] LOG:  redo done at 0/3ACC8270
2021-01-12 20:35:23.685 MSK [196592] LOG:  database system is ready to accept connections
```

Здесь мы видим, что при запуске был установлен факт аварийного завершения и было произведено автоматическое восстановление.