

Многоверсионность Очистка



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

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.

Обычная очистка и схема ее работы

Регулирование нагрузки

Анализ

Полная очистка и аналоги

Выполняется командой VACUUM

не конфликтует с обычной активностью в системе

Обрабатывает таблицу и все ее индексы

очищает ненужные версии строк в табличных страницах
(пропуская страницы, уже отмеченные в карте видимости)

очищает индексные записи, ссылающиеся на очищенные версии строк
освобождает указатели

обновляет карту свободного пространства

обновляет карту видимости

возможна отдельная очистка основной таблицы и TOAST-части

Самоочистка таблицы выполняется быстро, но не решает всех задач. Она работает только в пределах одной табличной страницы и не затрагивает индексы.

Очистка, выполняемая командой VACUUM, обрабатывает всю таблицу, включая все созданные на ней индексы. При этом удаляются и ненужные версии строк, и указатели на них в начале страницы.

Во время работы очистки таблица может использоваться обычным образом и для чтения, и для изменения данных. Но одновременное выполнение ряда команд (например CREATE INDEX, ALTER TABLE и некоторых других) будет невозможно.

Очистка пропускает страницы таблицы, в которых нет ненужных версий строк. Для этого используется карта видимости (visibility map), где отмечаются страницы, заведомо содержащие только актуальные версии. Сама карта при этом тоже обновляется. Если нужно, чтобы очистка обработала все страницы, следует включить параметр DISABLE_PAGE_SKIPPING.

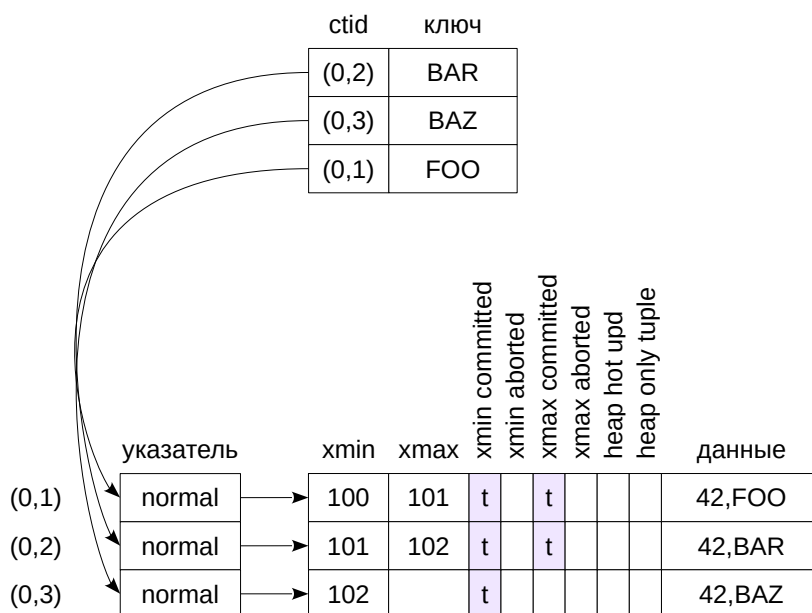
В процессе работы обновляется также и карта свободного пространства (free space map), в которой отражается наличие свободного места в страницах.

Можно отказаться от очистки основной таблицы или ее TOAST-части, выключив параметры PROCESS_MAIN или PROCESS_TOAST (по умолчанию оба включены).

<https://postgrespro.ru/docs/postgresql/16/sql-vacuum>

<https://postgrespro.ru/docs/postgresql/16/routine-vacuuming>

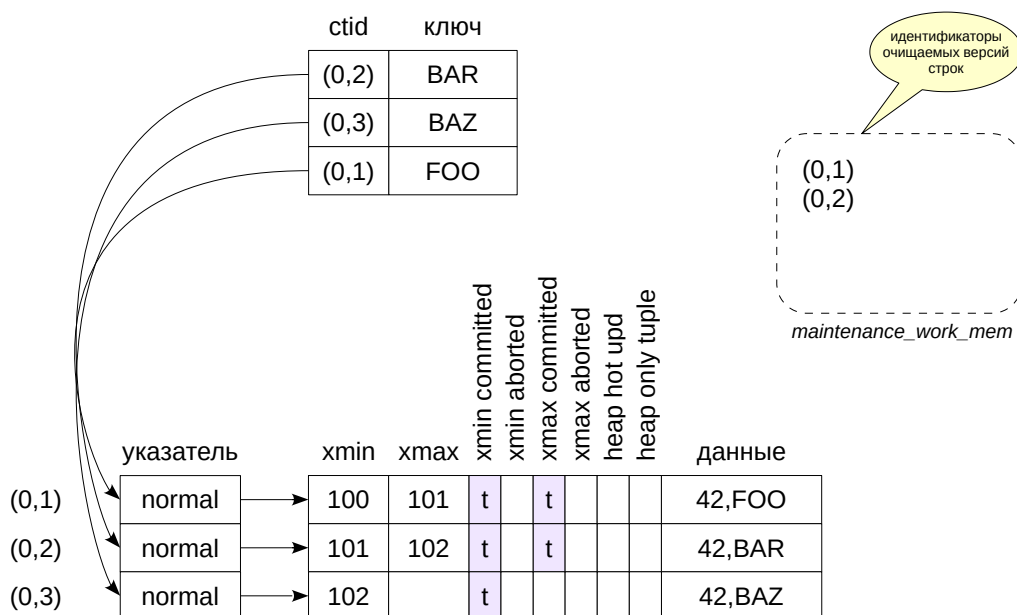
До очистки



На рисунке приведена ситуация до очистки. В табличной странице три версии одной строки. Две из них неактуальны, не видны ни в одном снимке и могут быть удалены.

В индексе есть три ссылки: на каждую из версий строки.

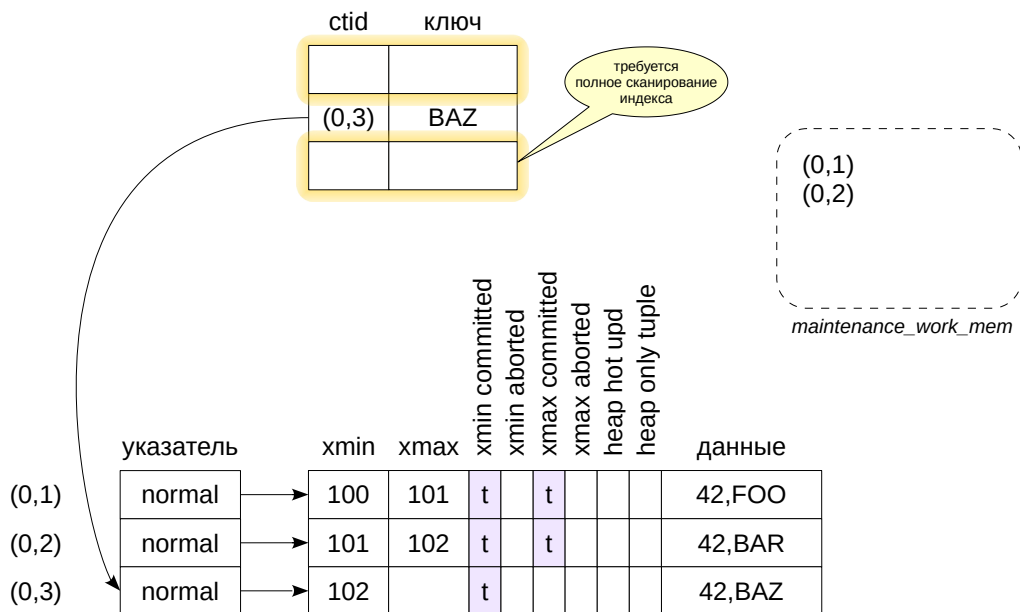
1. Сканирование таблицы



Сначала VACUUM выполняет сканирование таблицы (пропуская те страницы, которые помечены в карте видимости).

В прочитанных страницах процесс определяет ненужные версии строк и записывает их идентификаторы в специальный массив. Для этого в локальной памяти процесса VACUUM выделяется фрагмент размером *maintenance_work_mem* (или меньший, если его заведомо будет достаточно). Значение этого параметра по умолчанию — 64 Мбайт. Отметим, что эта память выделяется сразу в полном объеме, а не по мере необходимости.

2. Очистка индексов

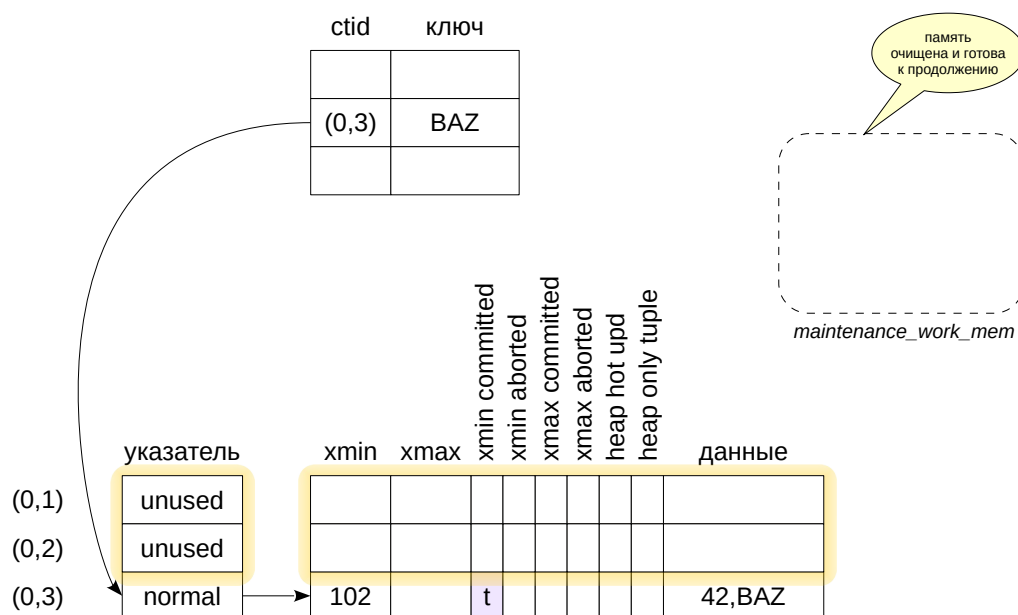


Когда выделенная под массив память заканчивается (или если мы уже дошли до конца таблицы), выполняется очистка индексов.

Для этого *каждый* из индексов, созданных на таблице, *полностью сканируется* в поисках записей, которые ссылаются на очищаемые версии строк. Найденные записи вычищаются из индексных страниц.

Начиная с PostgreSQL 14, очистка пропускает сканирование индексов, если это малоэффективно (когда количество страниц, содержащих мертвые строки, не превышает 2% от общего количества страниц таблицы). Но если параметр `INDEX_CLEANUP` выставлен в значение `on`, то `VACUUM` будет безусловно удалять ненужные записи из индексов.

3. Очистка таблицы



7

После этого необходимые табличные страницы повторно читаются, чтобы очистить в них ненужные версии строк и освободить указатели, на которые больше нет ссылок из индексов.

Если на первом проходе таблица не была просканирована полностью, VACUUM очищает массив в памяти и продолжает работу с этой точки.

Таким образом, если при очистке удаляется так много версий строк, что их идентификаторы не помещаются в память размером *maintenance_work_mem*, все индексы будут полностью сканироваться несколько раз. На больших таблицах это может занимать существенное время и заметно нагружать систему. В таком случае можно либо распределить нагрузку во времени, вызывая очистку чаще (чтобы за каждый раз очищалось не очень большое количество версий строк), либо выделить больше памяти, чтобы исключить повторную обработку индексов.

В конце очистки выполняется этап усечения таблицы. Если в самом конце файла образовалось достаточно много пустых страниц, они «откусываются» и освободившееся место возвращается операционной системе. Это действие требует установки исключительной блокировки. Если такая блокировка вызывает проблемы, этап усечения можно отключить параметром хранения *vacuum_truncate* или явно: VACUUM (TRUNCATE off). Подробнее см. модуль «Блокировки».

VACUUM VERBOSE

Представление `pg_stat_progress_vacuum`

- полный размер таблицы
- число прочитанных страниц и число очищенных страниц
- количество уже завершенных циклов очистки индексов
- число идентификаторов версий строк, помещающихся в память и текущее число идентификаторов в памяти
- текущая фаза очистки

8

Если очистка выполняется долго, может потребоваться узнать текущее состояние дел.

Для этого можно вызывать `VACUUM` с указанием `VERBOSE` — на консоль будет выводиться информация о ходе выполнения.

Кроме того, есть представление `pg_stat_progress_vacuum`, которое содержит всю информацию о выполняющейся в данный момент очистке. Ход очистки таблицы можно отслеживать, сравнивая число очищенных страниц (`heap_blks_vacuumed`) с общим количеством (`heap_blks_total`). Ход очистки индексов детально не отображается, но основное внимание надо обращать на количество циклов очистки (`index_vacuum_count`) — значение больше 1 означает, что памяти *maintenance_work_mem* не хватило для того, чтобы завершить очистку за один проход.

<https://postgrespro.ru/docs/postgresql/16/progress-reporting>

Обычная очистка

Создадим таблицу и индекс.

```
=> CREATE DATABASE mvcc_vacuum;
```

CREATE DATABASE

```
=> \c mvcc_vacuum
```

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

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

CREATE TABLE

```
=> CREATE INDEX t_id ON t(id);
```

CREATE INDEX

Как обычно, используем расширение pageinspect:

```
=> CREATE EXTENSION pageinspect;
```

CREATE EXTENSION

А также представление для табличной страницы:

```
=> CREATE VIEW t_v AS
SELECT '(0,'||lp||')' AS ctid,
       CASE lp_flags
         WHEN 0 THEN 'unused'
         WHEN 1 THEN 'normal'
         WHEN 2 THEN 'redirect to '||lp_off
         WHEN 3 THEN 'dead'
       END AS state,
       t_xmin || CASE
         WHEN (t_infomask & 256) > 0 THEN ' (c)'
         WHEN (t_infomask & 512) > 0 THEN ' (a)'
         ELSE ''
       END AS xmin,
       t_xmax || CASE
         WHEN (t_infomask & 1024) > 0 THEN ' (c)'
         WHEN (t_infomask & 2048) > 0 THEN ' (a)'
         ELSE ''
       END AS xmax,
       CASE WHEN (t_infomask2 & 16384) > 0 THEN 't' END AS hhu,
       CASE WHEN (t_infomask2 & 32768) > 0 THEN 't' END AS hot,
       t_ctid
FROM heap_page_items(get_raw_page('t',0))
ORDER BY lp;
```

CREATE VIEW

И представление для индекса:

```
=> CREATE VIEW t_id_v AS
SELECT itemoffset,
       ctid
FROM bt_page_items('t_id',1);
```

CREATE VIEW

Вставим строку в таблицу и обновим ее:

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

INSERT 0 1

```
=> UPDATE t SET id = 2;
```

UPDATE 1

Теперь, чтобы еще раз напомнить про понятие горизонта, откроем в другом сеансе транзакцию с активным снимком данных.

```
| => \c mvcc_vacuum
```

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

```
| => BEGIN ISOLATION LEVEL REPEATABLE READ;
```

```

| BEGIN
|
| => SELECT * FROM t;
|
|   id
|   ----
|   2
| (1 row)

```

Горизонт базы данных определяется этим снимком:

```

| => SELECT backend_xmin FROM pg_stat_activity WHERE pid = pg_backend_pid();
|
| backend_xmin
| -----
|          751
| (1 row)

```

Снова обновим строку.

```

=> UPDATE t SET id = 3;

```

```

UPDATE 1

```

Сейчас в таблице три версии строки:

```

=> SELECT * FROM t_v;

```

ctid	state	xmin	xmax	hhu	hot	t_ctid
(0,1)	normal	749 (c)	750 (c)			(0,2)
(0,2)	normal	750 (c)	751			(0,3)
(0,3)	normal	751	0 (a)			(0,3)

(3 rows)

Выполним теперь очистку.

```

=> VACUUM t;

```

```

VACUUM

```

Как изменится табличная страница?

```

=> SELECT * FROM t_v;

```

ctid	state	xmin	xmax	hhu	hot	t_ctid
(0,1)	unused					
(0,2)	normal	750 (c)	751 (c)			(0,3)
(0,3)	normal	751 (c)	0 (a)			(0,3)

(3 rows)

Очистка освободила одну версию строки, а вторая осталась без изменений, так как параллельная транзакция до сих пор не завершена и ее снимок активен.

В индексе — два указателя на оставшиеся версии:

```

=> SELECT * FROM t_id_v;

```

itemoffset	ctid
1	(0,2)
2	(0,3)

(2 rows)

Можно попросить очистку рассказать о том, что происходит:

```

=> VACUUM VERBOSE t;

```

```

INFO: vacuuming "mvcc_vacuum.public.t"
INFO: finished vacuuming "mvcc_vacuum.public.t": index scans: 0
pages: 0 removed, 1 remain, 1 scanned (100.00% of total)
tuples: 0 removed, 2 remain, 1 are dead but not yet removable
removable cutoff: 751, which was 1 XIDs old when operation ended
frozen: 0 pages from table (0.00% of total) had 0 tuples frozen
index scan not needed: 0 pages from table (0.00% of total) had 0 dead item identifiers
removed
avg read rate: 0.000 MB/s, avg write rate: 0.000 MB/s
buffer usage: 10 hits, 0 misses, 0 dirtied
WAL usage: 0 records, 0 full page images, 0 bytes
system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
VACUUM

```

Обратите внимание:

- tuples: 0 removed, 2 remain, 1 are dead but not yet removable,
- index scan not needed
- removable cutoff показывает текущий горизонт

Теперь завершим параллельную транзакцию и снова вызовем очистку.

```
=> COMMIT;
```

```
COMMIT
```

```
=> VACUUM VERBOSE t;
```

```

INFO: vacuuming "mvcc_vacuum.public.t"
INFO: finished vacuuming "mvcc_vacuum.public.t": index scans: 1
pages: 0 removed, 1 remain, 1 scanned (100.00% of total)
tuples: 1 removed, 1 remain, 0 are dead but not yet removable
removable cutoff: 752, which was 0 XIDs old when operation ended
new relfrozenxid: 751, which is 1 XIDs ahead of previous value
frozen: 0 pages from table (0.00% of total) had 0 tuples frozen
index scan needed: 1 pages from table (100.00% of total) had 1 dead item identifiers
removed
index "t_id": pages: 2 in total, 0 newly deleted, 0 currently deleted, 0 reusable
avg read rate: 0.000 MB/s, avg write rate: 101.461 MB/s
buffer usage: 16 hits, 0 misses, 1 dirtied
WAL usage: 6 records, 1 full page images, 8791 bytes
system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
VACUUM

```

Горизонт сдвинулся вперед и позволил очистке удалить мертвую строку — tuples: 1 removed. Теперь в странице осталась только последняя актуальная версия строки:

```
=> SELECT * FROM t_v;
```

ctid	state	xmin	xmax	hhu	hot	t_ctid
(0,1)	unused					
(0,2)	unused					
(0,3)	normal	751 (c)	0 (a)			(0,3)

(3 rows)

В индексе также осталась только одна запись:

```
=> SELECT * FROM t_id_v;
```

itemoffset	ctid
1	(0,3)

(1 row)

Процесс чередует работу и ожидание

примерно `vacuum_cost_limit` условных единиц работы,
затем засыпает на `vacuum_cost_delay` мс

Настройки

`vacuum_cost_limit` = 200

`vacuum_cost_delay` = 0 ms

стоимость обработки:

`vacuum_cost_page_hit` = 1 — страница нашлась в кеше

`vacuum_cost_page_miss` = 2 — страница прочитана с диска

`vacuum_cost_page_dirty` = 20 — чистая страница стала грязной

10

Поскольку процесс очистки интенсивно работает с таблицами и индексами, может оказаться необходимым распределить действия во времени и тем самым сгладить пики нагрузки. Для этого существует возможность выполнять очистку порциями, чередуя работу и ожидание.

Размер порции `vacuum_cost_limit` указывается в условных единицах. Три других параметра, указанных на слайде, определяют стоимость обработки одной страницы в этих единицах в зависимости от необходимости разных действий. В лучшем случае стоимость будет равна `vacuum_cost_page_hit` (если страница нашлась в кеше и либо не изменилась, либо уже была грязной); в худшем случае — `vacuum_cost_page_miss` + `vacuum_cost_page_dirty` (если страница была прочитана с диска и изменилась в результате очистки).

Настройка по умолчанию фактически отключает механизм регулирования нагрузки, так как время ожидания выставлено в ноль. Это сделано из тех соображений, что если администратору пришлось запускать VACUUM вручную, он, скорее всего, хочет выполнить очистку как можно быстрее.

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

Сканирование и очистка таблицы — последовательно

Очистка индексов может выполняться параллельно

каждый индекс — только одним рабочим процессом

размер индекса должен превышать *min_parallel_index_scan_size*

количество процессов ограничено *max_parallel_maintenance_workers*
и, если указано, значением `VACUUM (PARALLEL n)`

Этапы сканирования и очистки таблицы всегда выполняются последовательно одним процессом.

Этап очистки индексов может выполняться в параллельном режиме. Это происходит, если на таблице создано несколько (больше одного) достаточно больших индексов: размер индекса должен превышать значение параметра *min_parallel_index_scan_size* (512 Кбайт по умолчанию). Тогда для каждого подходящего индекса запускается отдельный рабочий процесс.

Каждый индекс обрабатывается только одним рабочим процессом, то есть несколько процессов не могут очищать один и тот же индекс.

Количество процессов ограничено сверху значением параметра *max_parallel_maintenance_workers* и может быть дополнительно ограничено явным указанием степени параллелизма при вызове команды `VACUUM (PARALLEL n)`.

Выполняется при VACUUM ANALYZE или ANALYZE

не конфликтует с обычной активностью в системе

Собирает статистику для планировщика

Еще одна задача, которую обычно совмещают с очисткой, — анализ, то есть сбор статистической информации для планировщика запросов. Анализируется число строк в таблице, распределение данных по столбцам и т. п. Подробно сбор статистики рассматривается в курсе QPT «Оптимизация запросов» .

Вручную анализ выполняется командой ANALYZE (только анализ) или VACUUM ANALYZE (и очистка, и анализ).

Как и в случае очистки, обработка происходит в фоновом режиме и не мешает обычному использованию таблиц и индексов.

<https://postgrespro.ru/docs/postgresql/16/sql-analyze>

Анализ

Создадим таблицу с большим количеством одинаковых строк и проиндексируем ее:

```
=> CREATE TABLE tt(s) AS SELECT 'F00' FROM generate_series(1,1_000_000) AS g;
```

```
SELECT 1000000
```

```
=> CREATE INDEX ON tt(s);
```

```
CREATE INDEX
```

Планировщик ничего не знает про данные и выбирает индексный доступ, хотя читать придется всю таблицу:

```
=> EXPLAIN (costs off) SELECT * FROM tt WHERE s = 'F00';
```

```
QUERY PLAN
```

```
-----  
Bitmap Heap Scan on tt  
  Recheck Cond: (s = 'F00'::text)  
    -> Bitmap Index Scan on tt_s_idx  
          Index Cond: (s = 'F00'::text)  
(4 rows)
```

При анализе собирается статистика по случайной выборке строк:

```
=> ANALYZE VERBOSE tt;
```

```
INFO:  analyzing "public.tt"  
INFO:  "tt": scanned 4480 of 4480 pages, containing 1000000 live rows and 0 dead rows;  
30000 rows in sample, 1000000 estimated total rows  
ANALYZE
```

Статистика сохраняется в системном каталоге. После этого планировщик знает, что во всех строках находится одно и то же значение, и перестает использовать индекс:

```
=> EXPLAIN (costs off) SELECT * FROM tt WHERE s = 'F00';
```

```
QUERY PLAN
```

```
-----  
Seq Scan on tt  
  Filter: (s = 'F00'::text)  
(2 rows)
```

Выполняется командой `VACUUM FULL`

не совместима ни с какими операциями над таблицей, включая чтение

Полностью перестраивает таблицу и все ее индексы

при работе потребуется дополнительное место для новых файлов

освобожденное место возвращается операционной системе

выполняется дольше, чем обычная очистка

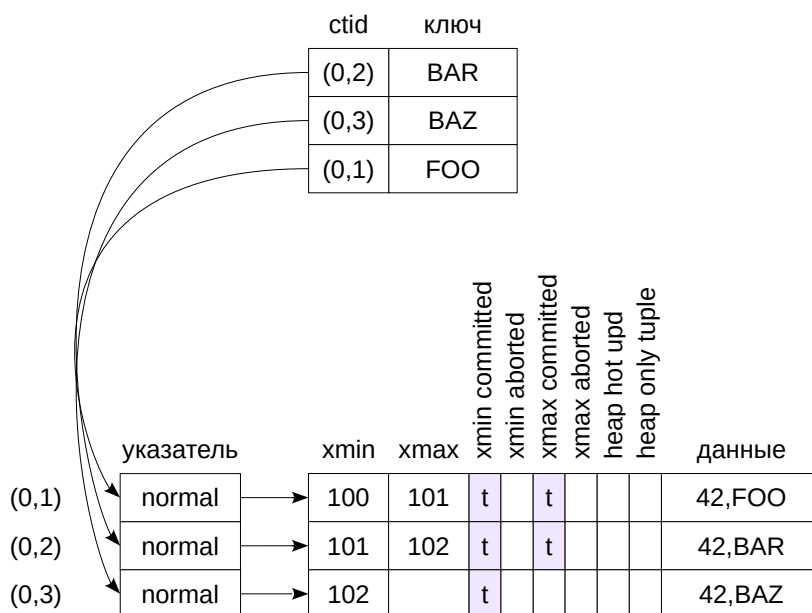
Обычная очистка не решает всех задач по освобождению места. Если таблица или индекс сильно выросли в размерах, то очистка не приведет к сокращению числа страниц (и к уменьшению файлов). Вместо этого внутри существующих страниц появятся «дыры», которые могут быть использованы для вставки новых строк или изменения существующих. Единственное исключение составляют полностью очищенные страницы, находящиеся в конце файла — такие страницы «откусываются» и освободившееся место возвращается операционной системе.

Если размер файлов превышает некие разумные пределы, можно выполнить полную очистку. При этом таблица и все ее индексы перестраиваются полностью с нуля, а данные упаковываются максимально компактно (разумеется, с учетом *fillfactor*).

При перестройке PostgreSQL последовательно перестраивает сначала таблицу, а затем и каждый из индексов. Для них создаются новые файлы, а старые удаляются. Следует учитывать, что в процессе работы на диске потребуется дополнительное место.

Полная очистка не предполагает регулярного использования, так как полностью блокирует всякую работу с таблицей (включая и выполнение запросов к ней) на все время своей работы. На активно используемой системе это может быть неприемлемым; в таком случае может помочь расширение `pg_repack` (https://github.com/reorg/pg_repack), не входящее в поставку.

До полной очистки

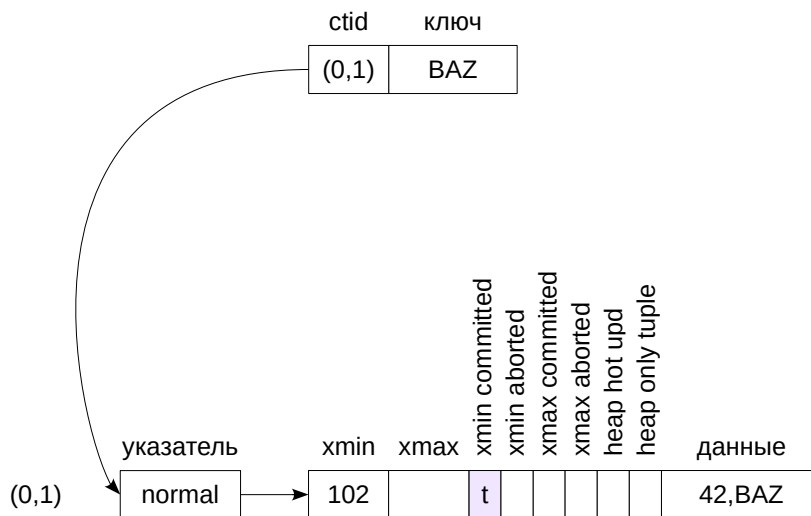


15

На рисунке приведена ситуация до полной очистки. В табличной странице три версии одной строки. Две из них неактуальны, не видны ни в одном снимке и могут быть удалены.

В индексе есть три ссылки на каждую из версий строки.

После полной очистки



После выполнения полной очистки содержимое таблицы и индекса полностью перестроено и физически находится в других файлах. При этом OID таблицы в системном каталоге сохраняется.

Полная очистка

Файлы, занимаемые таблицей и индексом:

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

pg_relation_filepath	pg_relation_filepath
base/16390/16391	base/16390/16394

(1 row)

Вызываем полную очистку.

```
=> VACUUM FULL VERBOSE t;
```

```
INFO: vacuuming "public.t"
INFO: "public.t": found 0 removable, 1 nonremovable row versions in 1 pages
DETAIL: 0 dead row versions cannot be removed yet.
CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.01 s.
VACUUM
```

Таблица и индекс теперь полностью перестроены:

```
=> SELECT * FROM t_v;
```

ctid	state	xmin	xmax	hhu	hot	t_ctid
(0,1)	normal	751 (c)	0 (a)			(0,1)

(1 row)

```
=> SELECT * FROM t_id_v;
```

itemoffset	ctid
1	(0,1)

(1 row)

Имена файлов поменялись:

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

pg_relation_filepath	pg_relation_filepath
base/16390/16455	base/16390/16458

(1 row)

CLUSTER

полностью перестраивает таблицу и все ее индексы
дополнительно физически упорядочивает версии строк
в соответствии с одним из индексов

REINDEX

полностью перестраивает индексы

TRUNCATE

«опустошает» таблицу

Особенности

все команды полностью блокируют работу с таблицей
все команды создают новые файлы для данных

Есть несколько команд, которые работают, используя схожий механизм. Все они полностью блокируют работу с таблицей, все они удаляют старые файлы данных и создают новые.

Команда CLUSTER во всем аналогична VACUUM FULL, но дополнительно физически упорядочивает версии строк в соответствии с одним из индексов. Это дает планировщику возможность более эффективно использовать индексный доступ в некоторых случаях. Однако надо понимать, что кластеризация не поддерживается: при последующих изменениях таблицы физический порядок версий строк будет нарушаться.

Команда REINDEX перестраивает индексы, не трогая при этом таблицу. Фактически, VACUUM FULL использует эту команду для перестроения индексов. При этом читать таблицу формально не запрещается, однако планировщик попытается получить блокировку перестраиваемого индекса, чтобы рассмотреть все планы, и до окончания работы команды прочитать таблицу фактически не удастся. Вариант REINDEX CONCURRENTLY работает дольше, но не блокирует индекс и не мешает чтению и обновлению данных.

Команда TRUNCATE логически работает так же, как и DELETE — удаляет все табличные строки. Но DELETE, как уже было рассмотрено, помечает версии строк как удаленные, что требует дальнейшей очистки. TRUNCATE же просто создает новый, чистый файл. Это работает быстрее, но надо учитывать, что TRUNCATE заблокирует работу с таблицей на все время до конца транзакции.

Обычная очистка освобождает место в страницах

- выполняется в фоновом режиме

- может потребоваться несколько раз сканировать индексы

Анализ собирает статистику для планировщика

- выполняется в фоновом режиме

- можно совместить с очисткой

Полная очистка перестраивает таблицу и индексы

- блокирует работу с таблицей на все время выполнения

1. Создайте большую таблицу с индексом. Временно уменьшите значение *maintenance_work_mem* так, чтобы потребовалось несколько проходов для очистки индекса. Проконтролируйте, запуская `VACUUM VERBOSE`.
2. Удалите из большой таблицы 90% случайных строк и проверьте, как изменился объем, который таблица занимает на диске, после выполнения обычной очистки.
3. Повторите п. 2 с полной очисткой.

1. Чтобы исключить срабатывание автоматической очистки (рассматривается в следующей теме), при создании таблицы укажите параметр хранения `autovacuum_enabled`:

```
CREATE TABLE ... WITH (autovacuum_enabled = off);
```

1. Сканирование индексов при очистке

Создадим таблицу с данными и индекс. Параметр `autovacuum_enabled` выключен, чтобы не срабатывала автоматическая очистка.

```
=> CREATE DATABASE mvcc_vacuum;
```

```
CREATE DATABASE
```

```
=> \c mvcc_vacuum
```

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

```
=> CREATE TABLE t(id integer) WITH (autovacuum_enabled = off);
```

```
CREATE TABLE
```

```
=> INSERT INTO t SELECT gen.id FROM generate_series(1,1_000_000) gen(id);
```

```
INSERT 0 1000000
```

```
=> CREATE INDEX t_id ON t(id);
```

```
CREATE INDEX
```

Уменьшаем размер памяти, выделяемой под массив идентификаторов:

```
=> ALTER SYSTEM SET maintenance_work_mem = '1MB';
```

```
ALTER SYSTEM
```

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

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

Обновляем все строки:

```
=> UPDATE t SET id = id + 1;
```

```
UPDATE 1000000
```

Запускаем очистку. Через небольшое время в другом сеансе обратимся к `pg_stat_progress_vacuum`.

```
=> VACUUM VERBOSE t;
```

```
| => \c mvcc_vacuum
```

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

```
| => SELECT * FROM pg_stat_progress_vacuum \gx
```

```
| -[ RECORD 1 ]-----+-----
| pid          | 46018
| datid        | 16390
| datname      | mvcc_vacuum
| relid        | 16391
| phase        | vacuuming indexes
| heap_blks_total | 8850
| heap_blks_scanned | 3088
| heap_blks_vacuumed | 2315
| index_vacuum_count | 3
| max_dead_tuples  | 174761
| num_dead_tuples  | 174472
```

```
INFO: vacuuming "mvcc_vacuum.public.t"
INFO: finished vacuuming "mvcc_vacuum.public.t": index scans: 6
pages: 0 removed, 8850 remain, 8850 scanned (100.00% of total)
tuples: 1000000 removed, 1000000 remain, 0 are dead but not yet removable
removable cutoff: 748, which was 0 XIDs old when operation ended
new relfrozenxid: 747, which is 3 XIDs ahead of previous value
frozen: 0 pages from table (0.00% of total) had 0 tuples frozen
index scan needed: 4425 pages from table (50.00% of total) had 1000000 dead item
identifiers removed
index "t_id": pages: 5486 in total, 0 newly deleted, 0 currently deleted, 0 reusable
avg read rate: 0.000 MB/s, avg write rate: 0.060 MB/s
buffer usage: 55073 hits, 0 misses, 6 dirtied
WAL usage: 23177 records, 6 full page images, 10055108 bytes
system usage: CPU: user: 0.46 s, system: 0.06 s, elapsed: 0.78 s
VACUUM
```

Восстановим значение измененного параметра.

```
=> ALTER SYSTEM RESET maintenance_work_mem;
```

```
ALTER SYSTEM
```

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

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

2. Очистка большого количества строк

Текущий размер файла данных:

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

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

Удалим 90% случайных строк. Случайность важна, чтобы в каждой странице остались какие-нибудь неудаленные строки — в противном случае очистка имеет шанс уменьшить размер файла.

```
=> DELETE FROM t WHERE random() < 0.9;
```

```
DELETE 899954
```

Объем после очистки:

```
=> VACUUM t;
```

```
VACUUM
```

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

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

Объем не изменился.

3. Полная очистка большого количества строк

Заново наполним таблицу.

```
=> TRUNCATE t;
```

```
TRUNCATE TABLE
```

```
=> INSERT INTO t SELECT gen.id FROM generate_series(1,1_000_000) gen(id);
```

```
INSERT 0 1000000
```

Текущий размер файла данных:

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



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

Обратите внимание, что в прошлый раз размер был примерно в два раза больше. Вторая половина была занята версиями строк, которые создала команда UPDATE.

Объем после удаления и полной очистки:

```
=> DELETE FROM t WHERE random() < 0.9;
```

```
DELETE 899492
```

```
=> VACUUM FULL t;
```

```
VACUUM
```

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

```
pg_size_pretty
-----
3560 kB
(1 row)
```

Объем уменьшился на 90%.