

# Многоверсионность НОТ-обновления и самоочистка



## 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.

Почему очистки не хватает

НОТ-обновления

Самоочистка в таблице

Самоочистка в индексе

## Мертвые версии строк

приводят к разрастанию файлов

## Синхронизация индексов

при любом изменении строки таблицы нужно менять все индексы

## Расщепление индексных страниц

снижает производительность

увеличивает размер

При работе механизма многоверсионности образуются ненужные версии строк, что приводит к разрастанию таблиц и индексов и, как следствие, снижает производительность запросов.

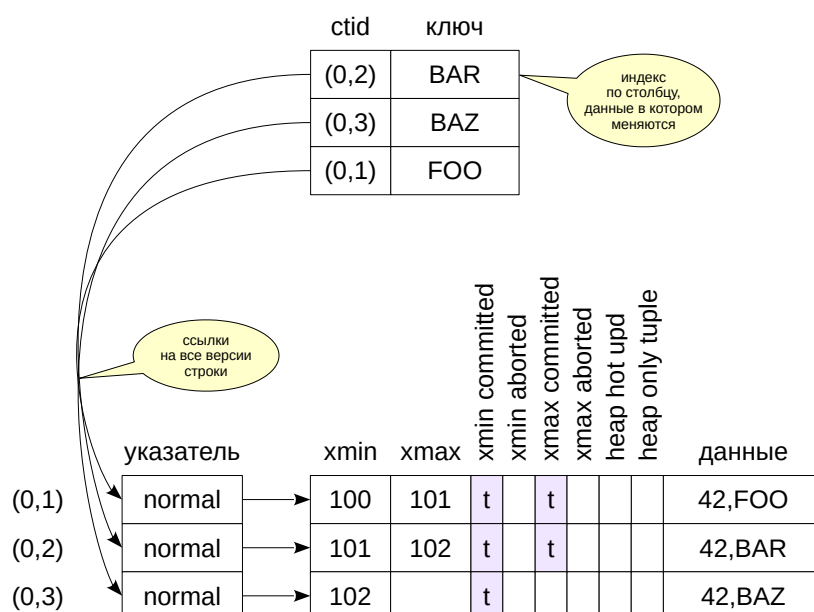
Если у таблицы есть индексы, при изменении таблицы приходится их синхронизировать. Это приводит к изменению большого числа страниц и также снижает производительность.

Проблема усугубляется особенностью реализации В-дерева в PostgreSQL: если на индексной странице недостаточно места для вставки новой записи, страница делится («расщепляется») на две и все данные перераспределяются между ними. Частые расщепления снижают производительность при вставке и изменении строк.

Более того, при удалении (или очистке) записей две индексные страницы не «склеиваются обратно» в одну. Из-за этого размер индекса может не уменьшиться даже при удалении существенной части данных.

Регулярная очистка решает проблему разрастания таблиц и индексов, однако требует значительных ресурсов. Поэтому даже при обычном, не связанном с очисткой, обращении к странице желательно удалять ненужные версии строк или индексные записи. А чтобы предотвратить избыточные обновления индексов при изменении таблицы, PostgreSQL объединяет версии строк в HOT-цепочки. Мы рассмотрим их на следующих слайдах.

# Обычное обновление



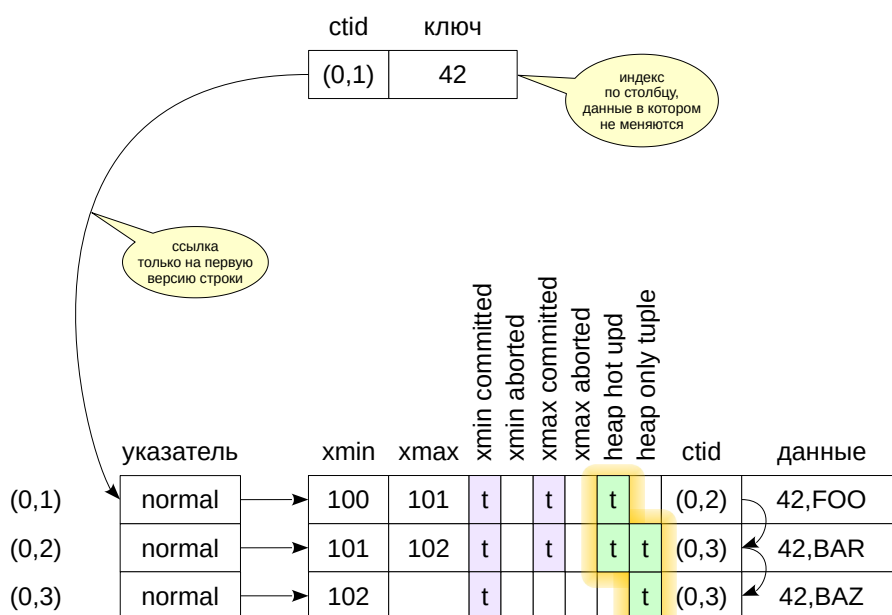
4

Напомним, что при обычном обновлении в индексе создаются ссылки на все версии строки, присутствующие в табличных страницах.

(Исключение составляет индекс BRIN, который не содержит ссылок на отдельные табличные строки; подробности устройства индексов различных типов рассматриваются в курсе QPT «Оптимизация запросов»).

Чем это плохо?

При любом изменении строки придется изменять все индексы, созданные для таблицы (даже если измененные поля не входят в индекс). Естественно, чем больше индексов создано на таблице, тем с большими сложностями приходится сталкиваться.



Если индекс создан по полю, значение которого не изменилось в результате обновления строки, можно не создавать в В-дереве дополнительную ссылку для того же значения ключа. Именно так работает оптимизация, называемая НОТ-обновлением — Heap-Only Tuple Update.

При таком обновлении в индексной странице находится лишь одна ссылка – на первую версию строки табличной страницы, а внутри табличной страницы организуется цепочка версий:

- строки, которые изменены и входят в цепочку, маркируются битом Heap Hot Updated;
- строки цепочки, на которые нет ссылок из индекса, маркируются битом Heap Only Tuple (то есть — «только табличная версия строки»);
- версии строк связаны в список с помощью поля ctid, входящего в заголовок версий.

Если при сканировании индекса PostgreSQL попадает в табличную страницу и обнаруживает версию, помеченную как Heap Hot Updated, он понимает, что надо пройти дальше по цепочке обновлений.

(Разумеется, для всех полученных таким образом версий строк проверяется видимость, прежде чем они будут возвращены клиенту.)

<https://git.postgresql.org/gitweb/?p=postgresql.git;a=blob;f=src/backend/access/heap/README.HOT;hb=HEAD>

Значения индексированных столбцов не должны изменяться

иначе придется добавить в индекс ссылку на новую версию строки, и версию нельзя будет пометить как «heap only»

Цепочка обновлений — только в пределах одной страницы

не требуется обращение к другим страницам,  
обход цепочки не ухудшает производительность

если в табличной странице не хватает места для новой версии,  
цепочка обрывается (как если бы оптимизация не работала)

место в странице можно зарезервировать, уменьшив параметр хранения таблицы *fillfactor* (100 % → 10 %)

Подчеркнем, что НОТ-обновления работают только в случае, если не изменяется *ни один ключ в индексах*. Иначе в каком-либо индексе появилась бы ссылка непосредственно на новую версию строки, что противоречит идее этой оптимизации. Только упомянутый выше индекс BRIN не препятствует НОТ-обновлению (в нем нет ссылок на табличные строки).

В том числе НОТ-обновления применяются и к таблицам, на которых нет вообще ни одного индекса: при обновлении любых полей такой таблицы будет строиться цепочка версий.

Оптимизация действует только в пределах одной страницы, поэтому дополнительный обход цепочки не требует обращения к другим страницам и не ухудшает производительность.

Однако если на странице не хватит свободного места, чтобы разместить новую версию строки, цепочка прервется. На версию строки, размещенную на другой странице, придется сделать и ссылку из индекса.

Поэтому при частых обновлениях неиндексированных полей может иметь смысл уменьшить параметр хранения *fillfactor*. Этот параметр определяет пороговый процент занятого на странице места, после которого вставка новых строк в эту страницу будет запрещена.

Значение по умолчанию – 100%, можно уменьшать до 10%.

Оставшееся место резервируется для обновлений: в этом случае новая версия строки может занять свободное место на той же странице.

(С другой стороны, чем выше *fillfactor*, тем компактнее располагаются записи и, соответственно, размер таблицы получается меньше.)

## НОТ-обновление

Создадим таблицу. Для простоты не будем создавать индекс: любое обновление будет НОТ-обновлением.

Каждая строка таблицы состоит из 2000 символов; если использовать только латинские буквы, то версия строки будет занимать 2000 байт плюс заголовок.

Параметр fillfactor установим в 75%, чтобы на страницу помещалось только три версии и одна была доступна для обновления.

```
=> CREATE DATABASE mvcc_hot;
```

```
CREATE DATABASE
```

```
=> \c mvcc_hot
```

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

```
=> CREATE TABLE t(
    s char(2000)
)
WITH (fillfactor = 75, autovacuum_enabled = off);
```

```
CREATE TABLE
```

Для изучения содержимого страницы используем расширение pageinspect.

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

```
CREATE EXTENSION
```

Для удобства создадим уже знакомое представление в немного более компактном виде и дополненное двумя полями:

```
=> 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
```

Вставим строку и обновим ее, чтобы создать новую версию:

```
=> INSERT INTO t(s) VALUES ('A');
```

```
INSERT 0 1
```

```
=> UPDATE t SET s = 'B';
```

```
UPDATE 1
```

Поскольку обновленный столбец не входит ни в какой индекс (в нашем случае ни одного индекса просто нет), в табличной странице появляется цепочка изменений:

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

ctid	state	xmin	xmax	hhu	hot	t_ctid
(0,1)	normal	747 (c)	748	t		(0,2)
(0,2)	normal	748	0 (a)		t	(0,2)

(2 rows)

- флаг hhu (Heap Hot Updated) показывает, что надо идти по цепочке ctid,
- флаг hot (Heap Only Tuple) показывает, что на данную версию строки нет ссылок из индексов.

При дальнейших изменениях цепочка будет расти (в пределах страницы):

=> **UPDATE** t **SET** s = 'C';

UPDATE 1

=> **SELECT** \* **FROM** t\_v;

ctid	state	xmin	xmax	hhu	hot	t_ctid
(0,1)	normal	747 (c)	748 (c)	t		(0,2)
(0,2)	normal	748 (c)	749	t	t	(0,3)
(0,3)	normal	749	0 (a)		t	(0,3)

(3 rows)



Выполняется при любом обращении к странице

если ранее выполненное обновление не нашло места  
для новой версии строки на этой же странице

если страница заполнена более чем на *fillfactor* или более чем на 90%

Действует в пределах одной табличной страницы

не освобождает указатели, на которые могут ссылаться индексы

не обновляет карту свободного пространства

не обновляет карту видимости

При обращении к странице — как при обновлении, так и при чтении — может происходить быстрая самоочистка, если PostgreSQL сочтет, что место на странице заканчивается. Должно выполняться одно из условий:

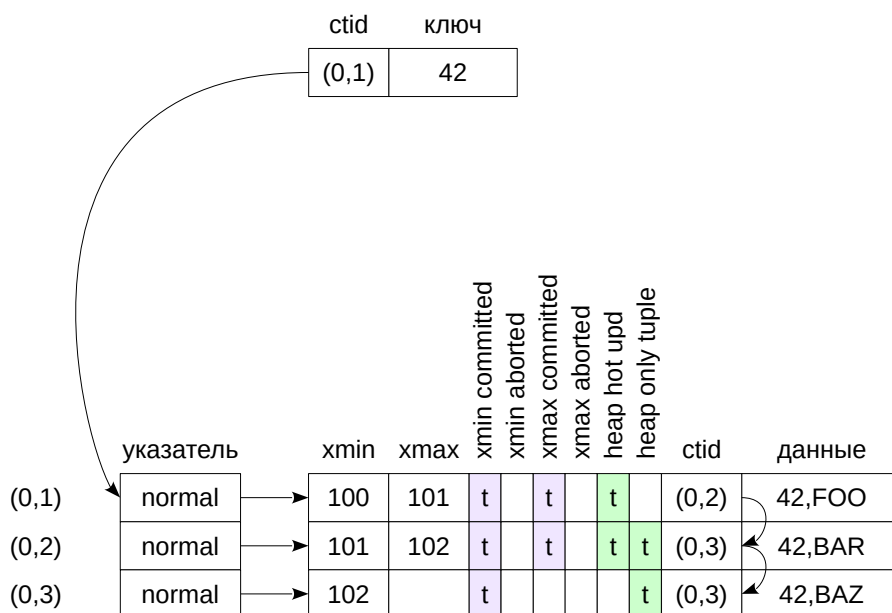
1. Ранее выполненное на этой странице обновление не обнаружило достаточного места, чтобы разместить здесь же новую версию строки и установило в заголовке страницы соответствующий признак, по которому при следующем обращении к ней срабатывает очистка.
2. Страница заполнена более чем на *fillfactor* или более чем на 90% — очистка сработает сразу.

Самоочистка убирает версии строк, не видимые ни в одном снимке (находящиеся за горизонтом базы данных), но работает строго в пределах одной табличной страницы. Указатели на версии строк не освобождаются, так как на них могут быть ссылки из индексов, а это уже другая страница — она не очищается.

Карта свободного пространства не обновляется из экономии ресурсов, а также из соображения, что освобожденное место лучше приберечь для обновлений, а не для вставок. Также не обновляется и карта видимости.

Тот факт, что страница может очищаться при чтении, означает, что запрос `SELECT` может вызвать изменение страниц. Это еще один такой случай, в дополнение к рассмотренному в теме «Страницы и версии строк» изменению битов-подсказок.

# До НОТ-самоочистки



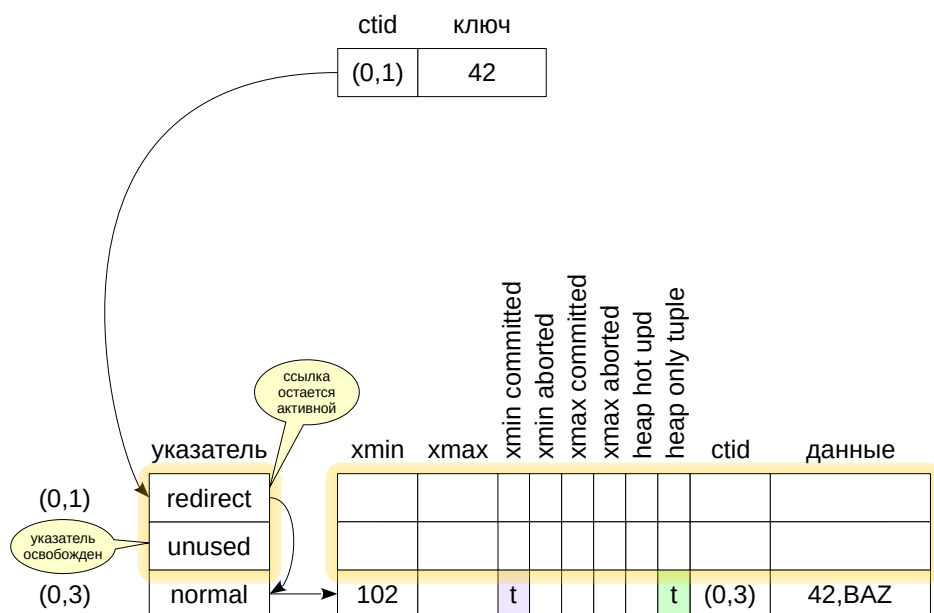
9

Частный, но важный случай самоочистки представляет собой очистка при НОТ-обновлениях.

На рисунке приведена ситуация до очистки. В таблице три версии одной и той же строки. На первую из них (0,1) ссылается индекс, остальные две (0,2) и (0,3) помечены как «только табличные».

Версии строк (0,1) и (0,2) неактуальны, не видны ни в одном снимке и могут быть удалены.

## После НОТ-самоочистки



10

Самоочистка удаляет неактуальные версии строк.

При НОТ-обновлениях в индексе может быть только одна ссылка на версию строки, представляющую собой «голову» НОТ-цепочки, которая поддерживается внутри одной табличной страницы. При любых изменениях версий строки указатель должен оставаться на своем месте и ссылаться на голову цепочки.

Поэтому применяется двойная адресация: для указателя, на который ссылается индекс — в данном случае (0,1), — используется статус «redirect», перенаправляющий на указатель нужной версии строки.

Указатель на вторую версию (0,2) больше не нужен. Он получает статус «unused» и будет использован при вставке какой-нибудь новой версии строки.

Все оставшиеся версии строк сдвигаются вместе так, чтобы свободное место на странице было представлено одним фрагментом.

Соответствующим образом изменяются и значения указателей.

Благодаря этому не возникает проблем с фрагментацией свободного места в странице.

## НОТ-самоочистка

Проверим, как работает самоочистка при НОТ-обновлениях. Обновим строку еще раз:

```
=> UPDATE t SET s = 'D';
```

UPDATE 1

В странице сейчас четыре версии строки:

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

ctid	state	xmin	xmax	hhu	hot	t_ctid
(0,1)	normal	747 (c)	748 (c)	t		(0,2)
(0,2)	normal	748 (c)	749 (c)	t	t	(0,3)
(0,3)	normal	749 (c)	750	t	t	(0,4)
(0,4)	normal	750	0 (a)		t	(0,4)

(4 rows)

На самом деле мы только что превысили порог fillfactor. 75% от размера страницы составляет 6144 байтов, а разница между значениями pagesize и upper равна 8192-64=8128:

```
=> SELECT lower, upper, pagesize FROM page_header(get_raw_page('t',0));
```

lower	upper	pagesize
40	64	8192

(1 row)

Проверим, что порог действительно превышен. Обновим строку еще раз:

```
=> UPDATE t SET s = 'E';
```

UPDATE 1

Какие изменения произойдут со страницей?

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

ctid	state	xmin	xmax	hhu	hot	t_ctid
(0,1)	redirect to 4					
(0,2)	normal	751	0 (a)		t	(0,2)
(0,3)	unused					
(0,4)	normal	750 (c)	751	t	t	(0,2)

(4 rows)

Все неактуальные версии строк (0,1), (0,2) и (0,3) были очищены; после этого новая версия строки была добавлена на освободившееся место.

Указатели на очищенные строки освобождены (имеют статус unused).

При этом указатель на первую версию остался на месте, но получил статус redirect. Проследите ссылки от этой головной версии до конца НОТ-цепочки.

Выполним обновление еще несколько раз:

```
=> UPDATE t SET s = 'F';
```

UPDATE 1

```
=> UPDATE t SET s = 'G';
```

UPDATE 1

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

ctid	state	xmin	xmax	hhu	hot	t_ctid
(0,1)	redirect to 4					
(0,2)	normal	751 (c)	752 (c)	t	t	(0,3)
(0,3)	normal	752 (c)	753	t	t	(0,5)
(0,4)	normal	750 (c)	751 (c)	t	t	(0,2)
(0,5)	normal	753	0 (a)		t	(0,5)

(5 rows)

Следующее обновление снова вызывает самоочистку:

```
=> UPDATE t SET s = 'H';
```

UPDATE 1

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

ctid	state	xmin	xmax	hhu	hot	t_ctid
(0,1)	redirect to 5					
(0,2)	normal	754	0 (a)		t	(0,2)
(0,3)	unused					
(0,4)	unused					
(0,5)	normal	753 (c)	754	t	t	(0,2)

(5 rows)

А теперь построим индекс по столбцу s и создадим вспомогательное представление, чтобы заглянуть в него:

```
=> CREATE INDEX t_s ON t(s);
```

CREATE INDEX

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

CREATE VIEW

При создании индекса перестройки данных в табличных страницах не происходит, HOT-цепочки сохраняются:

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

ctid	state	xmin	xmax	hhu	hot	t_ctid
(0,1)	redirect to 5					
(0,2)	normal	754 (c)	0 (a)		t	(0,2)
(0,3)	unused					
(0,4)	unused					
(0,5)	normal	753 (c)	754 (c)	t	t	(0,2)

(5 rows)

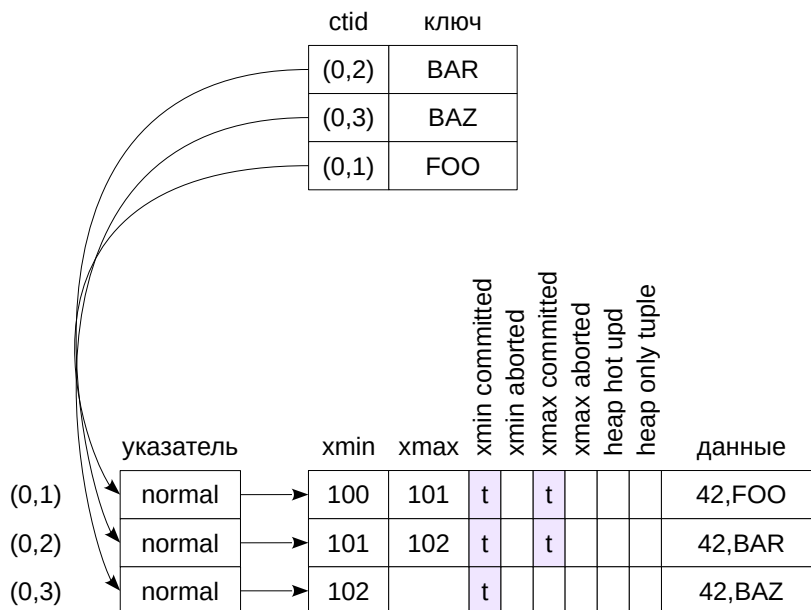
А из индекса ссылка ведет к указателю начала цепочки:

```
=> SELECT * FROM t_s_v;
```

itemoffset	ctid
1	(0,1)

(1 row)

## До самоочистки



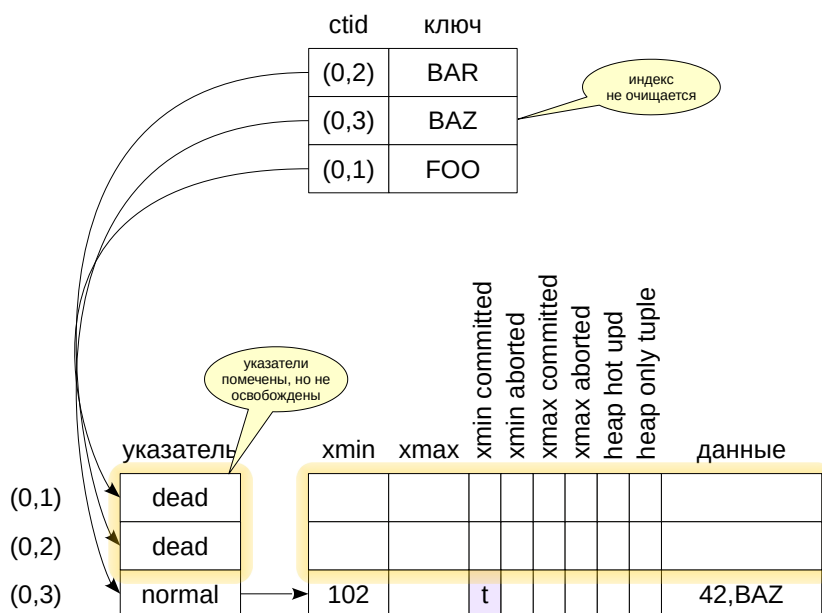
12

Разумеется, самоочистка работает и для версий, появившихся в результате обычного (не HOT) обновления.

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.

## После самоочистки



13

Если выполнено условие для срабатывания самоочистки, две неактуальные версии строки (0,1) и (0,2) могут быть ею удалены. Тогда указатели на удаленные версии получают статус «dead», а освободившееся место может быть использовано для вставки новых версий.

В отличие от ситуации с HOT-обновлениями, здесь нельзя освободить указатели на удаленные версии строк, поскольку на них существуют ссылки из индексной страницы. При индексном доступе PostgreSQL может получить (0,1) или (0,2) в качестве идентификатора версии строки, а затем попробовать пройти по этому идентификатору в табличную страницу, но благодаря статусу указателя обнаружит, что эта версия уже не существует.

Принципиально то, что самоочистка работает только в пределах одной табличной страницы и не очищает страницы индекса.

## Ненужные индексные записи

помечаются при выполнении запросов  
будут удалены, если в странице не хватает места

## Восходящее удаление индексных записей

если места все равно не хватает, проверяются версии строк

Если таблица имеет несколько индексов и обновляемый столбец входит хотя бы в один из них, HOT-обновление будет невозможно. При этом значения столбцов, входящих в другой индекс, могут не измениться, однако в такой индекс все равно придется добавить ссылку на новую версию строки (индекс изменяется физически, но не логически). Чтобы избежать частой очистки или перестройки индекса, можно заранее распознавать ненужные индексные записи и удалять их при недостатке места в индексной странице. В PostgreSQL есть два механизма, позволяющие это делать.

Если при выполнении запроса выясняется, что индексная запись ссылается на версию строки, которая не существует или находится за горизонтом, такая запись помечается как мертвая. Если не удастся разместить в странице очередную индексную запись, в первую очередь удаляются такие помеченные записи.

Если избежать расщепления таким образом не удастся, включается механизм «восходящего удаления кортежей» (bottom-up delete), добавленный в PostgreSQL 14. Для индексных записей с одинаковым значением ключа проверяются соответствующие версии строк. Если оказывается, что версия отсутствует или находится за горизонтом, индексная запись будет удалена. Проверка требует обращения к таблице, но это оказывается выгоднее, чем лишнее расщепление индексной страницы.



## Самоочистка индекса

Уменьшим длину строки таблицы, опустошим ее, вставим семь строк и обновим их:

```
=> ALTER TABLE t ALTER COLUMN s TYPE CHAR(500);
```

ALTER TABLE

```
=> TRUNCATE t;
```

TRUNCATE TABLE

```
=> INSERT INTO t SELECT random()::text FROM generate_series(1,7);
```

INSERT 0 7

```
=> UPDATE t SET s = random()::text;
```

UPDATE 7

Заглянем в постраничную статистику индекса с помощью функции из расширения pageinspect:

```
=> SELECT blkno, type, live_items, dead_items, avg_item_size, page_size, free_size
FROM bt_multi_page_stats('t_s',1,-1) \gx
```

```
-[ RECORD 1 ]-+-----
blkno       | 1
type        | l
live_items  | 14
dead_items  | 0
avg_item_size | 512
page_size   | 8192
free_size   | 924
```

В индексе сейчас всего одна листовая страница (она же корневая), в которой осталось около 900 байт свободного места. В ней находятся 14 индексных записей:

```
=> SELECT itemoffset, ctid, dead FROM bt_page_items('t_s',1);
```

```
itemoffset | ctid | dead
-----+-----+-----
1 | (0,10) | f
2 | (0,2) | f
3 | (0,12) | f
4 | (0,11) | f
5 | (0,8) | f
6 | (0,9) | f
7 | (0,14) | f
8 | (0,3) | f
9 | (0,4) | f
10 | (0,6) | f
11 | (0,13) | f
12 | (0,1) | f
13 | (0,7) | f
14 | (0,5) | f
(14 rows)
```

А вот строки, на которые они указывают:

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

ctid	state	xmin	xmax	hhu	hot	t_ctid
(0,1)	normal	759 (c)	760			(0,8)
(0,2)	normal	759 (c)	760			(0,9)
(0,3)	normal	759 (c)	760			(0,10)
(0,4)	normal	759 (c)	760			(0,11)
(0,5)	normal	759 (c)	760			(0,12)
(0,6)	normal	759 (c)	760			(0,13)
(0,7)	normal	759 (c)	760			(0,14)
(0,8)	normal	760	0 (a)			(0,8)
(0,9)	normal	760	0 (a)			(0,9)
(0,10)	normal	760	0 (a)			(0,10)
(0,11)	normal	760	0 (a)			(0,11)
(0,12)	normal	760	0 (a)			(0,12)
(0,13)	normal	760	0 (a)			(0,13)
(0,14)	normal	760	0 (a)			(0,14)

(14 rows)

Семь записей ссылаются на актуальные версии табличных строк, остальные семь — на мертвые, но получить эту информацию из индекса невозможно. Поэтому при следующем обновлении страница индекса расщепится.

Но если какой-то запрос обратится к таблице через индекс, ему придется пройти по ссылкам и узнать статусы табличных строк:

```
=> SET enable_seqscan = off; -- попросим обращаться к таблице через индекс
```

```
SET
```

```
=> SET enable_bitmapscan = off; -- причем непосредственно
```

```
SET
```

```
=> EXPLAIN (analyze) SELECT count(*) FROM t;
```

QUERY PLAN

```
-----
Aggregate (cost=48.84..48.85 rows=1 width=8) (actual time=0.060..0.061 rows=1 loops=1)
  -> Index Only Scan using t_s on t (cost=0.14..48.74 rows=40 width=0) (actual
time=0.042..0.046 rows=7 loops=1)
    Heap Fetches: 14
Planning Time: 0.060 ms
Execution Time: 0.088 ms
(5 rows)
```

Такой запрос помечает ненужные индексные записи как dead...

```
=> SELECT itemoffset, ctid, dead FROM bt_page_items('t_s',1);
```

itemoffset	ctid	dead
1	(0,10)	f
2	(0,2)	t
3	(0,12)	f
4	(0,11)	f
5	(0,8)	f
6	(0,9)	f
7	(0,14)	f
8	(0,3)	t
9	(0,4)	t
10	(0,6)	t
11	(0,13)	f
12	(0,1)	t
13	(0,7)	t
14	(0,5)	t

(14 rows)

... и если после этого выполнить обновление...

```
=> UPDATE t SET s = random()::text;
```

```
UPDATE 7
```

... то оно не приводит к расщеплению страницы, поскольку удастся освободить место для новых записей:

```
=> SELECT blkno, type, live_items, dead_items, avg_item_size, page_size, free_size
FROM bt_multi_page_stats('t_s',1,-1) \gx
```

```

-[ RECORD 1 ]--+-----
blkno      | 1
type       | l
live_items | 14
dead_items | 0
avg_item_size | 512
page_size  | 8192
free_size  | 924

```

## Восходящее удаление

Теперь добавим к таблице второй текстовый столбец и индекс по нему:

```
=> ALTER TABLE t ADD r CHAR(500);
```

```
ALTER TABLE
```

```
=> CREATE INDEX t_r ON t(r);
```

```
CREATE INDEX
```

Опять опустошим таблицу и вставим строки в таблицу:

```
=> TRUNCATE t;
```

```
TRUNCATE TABLE
```

```
=> INSERT INTO t SELECT random()::text, 1 FROM generate_series(1,12);
```

```
INSERT 0 12
```

Теперь, поскольку оба столбца проиндексированы, при обновлении HOT-цепочки строиться не будут. Если изменять только первый столбец, индекс по нему будет расти:

```
=> UPDATE t SET s = random()::text;
```

```
UPDATE 12
```

```
=> SELECT blkno, type, live_items, dead_items, avg_item_size, page_size, free_size
FROM bt_multi_page_stats('t_s',1,-1) \gx
```

```

-[ RECORD 1 ]--+-----
blkno      | 1
type       | l
live_items | 8
dead_items | 0
avg_item_size | 512
page_size  | 8192
free_size  | 4020
-[ RECORD 2 ]--+-----
blkno      | 2
type       | l
live_items | 7
dead_items | 0
avg_item_size | 512
page_size  | 8192
free_size  | 4536
-[ RECORD 3 ]--+-----
blkno      | 3
type       | r
live_items | 3
dead_items | 0
avg_item_size | 344
page_size  | 8192
free_size  | 7104
-[ RECORD 4 ]--+-----
blkno      | 4
type       | l
live_items | 11
dead_items | 0
avg_item_size | 512
page_size  | 8192
free_size  | 2472

```

Однако ключи второго индекса логически не изменяются, поэтому при недостатке места срабатывает механизм восходящего удаления и страница второго индекса расщепляться не будет:

```
=> SELECT blkno, type, live_items, dead_items, avg_item_size, page_size, free_size
FROM bt_multi_page_stats('t_r',1,-1) \gx
```

```
-[ RECORD 1 ]--+-  
blkno      | 1  
type       | l  
live_items | 10  
dead_items | 0  
avg_item_size | 521  
page_size  | 8192  
free_size  | 2892
```

Если изменяемый столбец не входит ни в один индекс и на странице есть место — применяется HOT-обновление

При недостатке места в странице таблицы или индекса автоматически выполняется самоочистка

1. Воспроизведите ситуацию самоочистки без участия HOT-обновлений. Проверьте содержимое табличной и индексной страниц с помощью расширения pageinspect.
2. Воспроизведите ситуацию HOT-обновления на таблице с индексом по некоторым полям.
3. Воспроизведите ситуацию HOT-обновления, при которой самоочистка не освобождает достаточно места на странице и новая версия создается на другой странице. Сколько записей будет в индексе в этом случае?

1. Запрос для индексной страницы появлялся в демонстрации к теме «Страницы и версии строк» этого модуля:

```
SELECT itemoffset, ctid  
FROM bt_page_items('имя-индекса',1);
```

## 1. Самоочистка

Создадим таблицу с двумя полями.

Параметр fillfactor установим в 75%, как в демонстрации.

```
=> CREATE DATABASE mvcc_hot;
```

CREATE DATABASE

```
=> \c mvcc_hot
```

You are now connected to database "mvcc\_hot" as user "student".

```
=> CREATE TABLE t(  
    id integer GENERATED ALWAYS AS IDENTITY,  
    s char(2000)  
)  
WITH (fillfactor = 75);
```

CREATE TABLE

Создадим индекс по столбцу s:

```
=> CREATE INDEX t_s ON t(s);
```

CREATE INDEX

Как обычно, используем расширение pageinspect.

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

CREATE EXTENSION

```
=> 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

Добавим строку и будем обновлять столбец s:

```
=> INSERT INTO t(s) VALUES ('A');
```

INSERT 0 1

```
=> UPDATE t SET s = 'B';
```

UPDATE 1

```
=> UPDATE t SET s = 'C';
```

UPDATE 1

```
=> UPDATE t SET s = 'D';
```

UPDATE 1

При этом в индексе для каждой из версий была сформирована запись:

```
=> SELECT itemoffset, ctid
FROM bt_page_items('t_s',1);
```

```
itemoffset | ctid
-----+-----
          1 | (0,1)
          2 | (0,2)
          3 | (0,3)
          4 | (0,4)
(4 rows)
```

Наличие индекса на обновляемом столбце привело к тому, что HOT-обновление не использовалось:

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

```
ctid | state | xmin | xmax | hhu | hot | t_ctid
-----+-----+-----+-----+-----+-----+-----
(0,1) | normal | 748 (c) | 749 (c) | | | (0,2)
(0,2) | normal | 749 (c) | 750 (c) | | | (0,3)
(0,3) | normal | 750 (c) | 751 | | | (0,4)
(0,4) | normal | 751 | 0 (a) | | | (0,4)
(4 rows)
```

Как и в демонстрации, еще одно обновление строки приводит к самоочистке:

```
=> UPDATE t SET s = 'E';
```

```
UPDATE 1
```

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

```
ctid | state | xmin | xmax | hhu | hot | t_ctid
-----+-----+-----+-----+-----+-----+-----
(0,1) | dead | | | | | 
(0,2) | dead | | | | | 
(0,3) | dead | | | | | 
(0,4) | normal | 751 (c) | 752 | | | (0,5)
(0,5) | normal | 752 | 0 (a) | | | (0,5)
(5 rows)
```

Все неактуальные версии строк (0,1), (0,2) и (0,3) очищены; после этого на освободившееся место добавлена новая версия строки (0,5).

Самоочистка не затрагивает индекс:

```
=> SELECT itemoffset, ctid
FROM bt_page_items('t_s',1);
```

```
itemoffset | ctid
-----+-----
          1 | (0,1)
          2 | (0,2)
          3 | (0,3)
          4 | (0,4)
          5 | (0,5)
(5 rows)
```

По этой причине указатели на очищенные строки в таблице не освобождены, а имеют статус dead.

## 2. HOT-обновление при наличии индекса

Удалим индекс по столбцу s и добавим другой, по столбцу id:

```
=> DROP INDEX t_s;
```

```
DROP INDEX
```

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

```
CREATE INDEX
```

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

```
CREATE VIEW
```

Опустошим таблицу и повторим команды, приводящие к самоочистке:



```

=> TRUNCATE t;

TRUNCATE TABLE

=> INSERT INTO t(s) VALUES ('A');

INSERT 0 1

=> UPDATE t SET s = 'B';

UPDATE 1

=> UPDATE t SET s = 'C';

UPDATE 1

=> UPDATE t SET s = 'D';

UPDATE 1

```

В индексной странице только одна ссылка на таблицу:

```

=> SELECT * FROM t_id_v;

 itemoffset | ctid
-----+-----
           1 | (0,1)
(1 row)

```

В табличной странице — версии строки, объединенные в список:

```

=> SELECT * FROM t_v;

 ctid | state | xmin | xmax | hhu | hot | t_ctid
-----+-----+-----+-----+-----+-----+-----
(0,1) | normal | 757 (c) | 758 (c) | t | | (0,2)
(0,2) | normal | 758 (c) | 759 (c) | t | t | (0,3)
(0,3) | normal | 759 (c) | 760 | t | t | (0,4)
(0,4) | normal | 760 | 0 (a) | | t | (0,4)
(4 rows)

```

Индекс на столбце не мешает HOT-обновлению, если этот столбец не обновляется.

### 3. Разрыв HOT-цепочки

```

| => \c mvcc_hot
|
| You are now connected to database "mvcc_hot" as user "student".
|
| => BEGIN ISOLATION LEVEL REPEATABLE READ;
|
| BEGIN
|
| => SELECT count(*) FROM t;
|
|    count
|-----
|         1
| (1 row)
|

```

Теперь выполняем обновление в первом сеансе:

```

=> UPDATE t SET s = 'I';

UPDATE 1

=> UPDATE t SET s = 'J';

UPDATE 1

=> UPDATE t SET s = 'K';

UPDATE 1

=> SELECT * FROM t_v;

```

ctid	state	xmin	xmax	hhu	hot	t_ctid
(0,1)	redirect to 4					
(0,2)	normal	761 (c)	762 (c)	t	t	(0,3)
(0,3)	normal	762 (c)	763	t	t	(0,5)
(0,4)	normal	760 (c)	761 (c)	t	t	(0,2)
(0,5)	normal	763	0 (a)		t	(0,5)

(5 rows)

Следующее обновление уже не сможет освободить место на странице:

```
=> UPDATE t SET s = 'L';
```

UPDATE 1

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

ctid	state	xmin	xmax	hhu	hot	t_ctid
(0,1)	redirect to 4					
(0,2)	normal	761 (c)	762 (c)	t	t	(0,3)
(0,3)	normal	762 (c)	763 (c)	t	t	(0,5)
(0,4)	normal	760 (c)	761 (c)	t	t	(0,2)
(0,5)	normal	763 (c)	764		t	(1,1)

(5 rows)

Видим ссылку (1,1), ведущую на страницу 1.

Представление pg\_stat\_all\_tables покажет количество измененных строк (в том числе с применением HOT), а также количество измененных строк с переходом на новую табличную страницу. Если это значение при работе с таблицей будет достаточно большим, то следует задуматься об уменьшении значения fillfactor.

```
=> SELECT relname, n_tup_upd, n_tup_hot_upd, n_tup_newpage_upd
FROM pg_stat_all_tables WHERE relname = 't';
```

relname	n_tup_upd	n_tup_hot_upd	n_tup_newpage_upd
t	11	6	1

(1 row)

Теперь в индексе две записи, каждая указывает на начало своей HOT-цепочки:

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

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

(2 rows)

```
=> COMMIT;
```

```
COMMIT
```