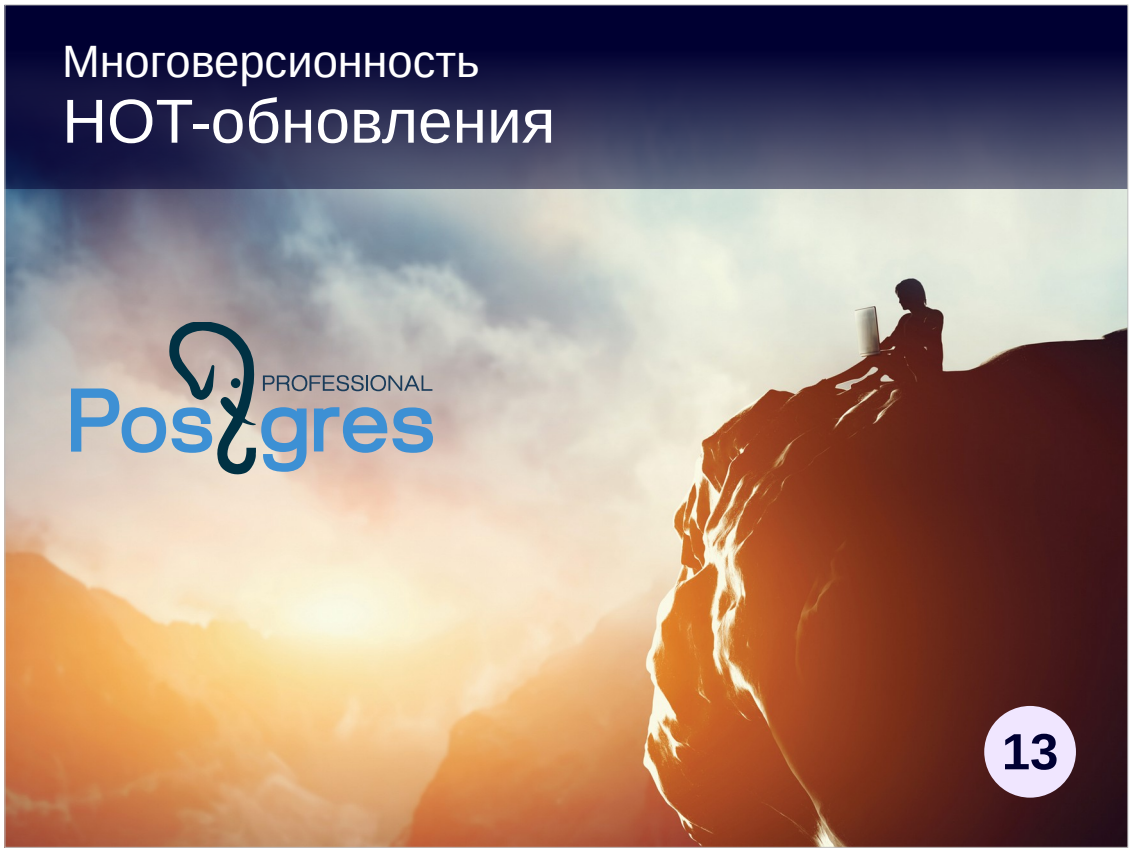


Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

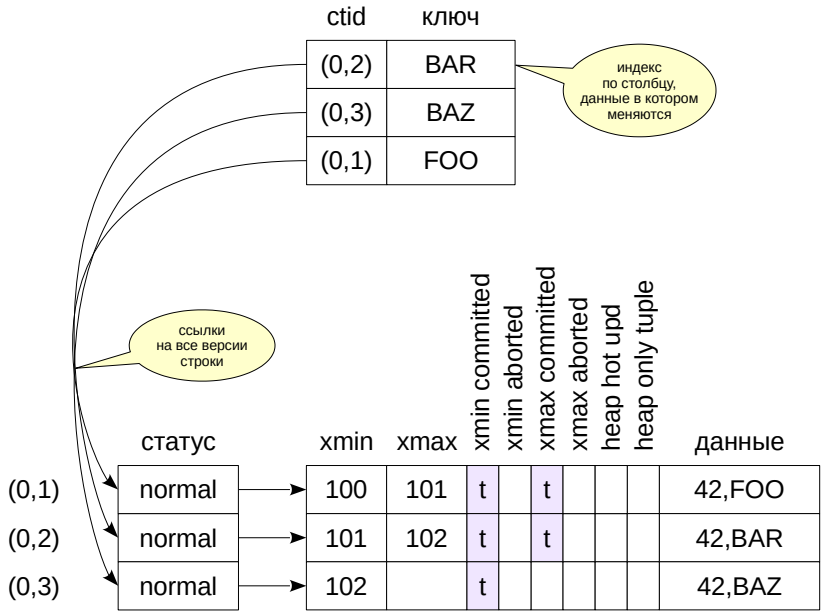
Обратная связь
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Темы

HOT-обновления

Внутристраничная очистка

Обычное обновление



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

Проблемы обновления

При любом обновлении строки надо изменять индекс

- страдает производительность вставок и изменений

В индексе накапливаются ссылки на неактуальные версии

- размер индекса растет, требуется очистка

Все сложности умножаются на количество индексов, построенных по таблице

Чем это плохо?

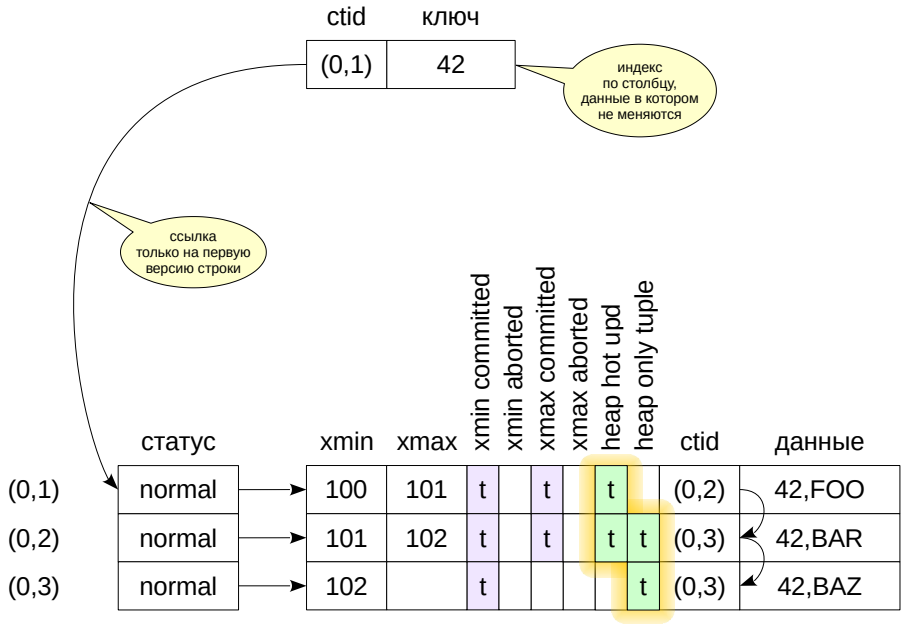
Во-первых, при любом изменении строки приходится обновлять все индексы, созданные для таблицы (даже если измененные поля не входят в индекс). Очевидно, это снижает производительность.

Во-вторых, в индексах накапливаются ссылки на исторические версии строки, которые потом приходится очищать.

Более того, есть особенность реализации B-дерева в PostgreSQL. Если на индексной странице недостаточно места для вставки новой записи, страница делится на две и все данные перераспределяются между ними. Однако при удалении (или очистке) записей две индексные страницы не «склеиваются» в одну. Из-за этого размер индекса может не уменьшиться даже при удалении существенной части данных.

Естественно, чем больше индексов создано на таблице, тем с большими сложностями приходится сталкиваться.

HOT-обновление



Однако если значение поля, по которому создан индекс, не изменилось в результате обновления строки, то нет смысла создавать дополнительную запись в B-дереве, содержащую то же самое значение ключа. Именно так работает оптимизация, называемая HOT-обновлением — Heap-Only Tuple Update.

При таком обновлении в индексной странице находится лишь одна запись, ссылающаяся 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, 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

HOT-обновление

Значения индексированных столбцов не должны измениться

- иначе придется добавить индексную запись, ссылающуюся 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 HOT-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.

В том числе HOT-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 уменьшать параметр хранения *fillfactor*. Этот параметр определяет пороговый процент занятого на странице места, после которого вставка новых строк в эту страницу будет запрещена. Оставшееся место остается зарезервированным для обновлений: при обновлении 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.)

HOT-обновление

Создадим таблицу. Для простоты не будем создавать индекс: любое обновление будет HOT-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 вставлялись только три версии, и еще 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);
```

```
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 | 1204299 (c) | 1204300 | t   |     | (0,2)
 (0,2) | normal | 1204300     | 0 (a)   |     | t   | (0,2)
(2 rows)
```

- флаг Heap Hot Updated показывает, что надо идти по цепочке ctid,
- флаг 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 | 1204299 (c) | 1204300 (c) | t   |     | (0,2)
 (0,2) | normal | 1204300 (c) | 1204301     | t   | t   | (0,3)
 (0,3) | normal | 1204301     | 0 (a)       |     | t   | (0,3)
(3 rows)
```

Внутристраничная очистка

Выполняется при любом обращении к странице

- если ранее выполненное обновление не нашло места для новой версии строки на этой же странице
- если страница заполнена больше, чем на *fillfactor*

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 внутристраничная очистка, если PostgreSQL сочтет, что место на странице заканчивается. Есть два критерия:

1. Ранее выполненное на этой странице обновление не обнаружило достаточного места, чтобы разместить новую версию строки на той же странице.

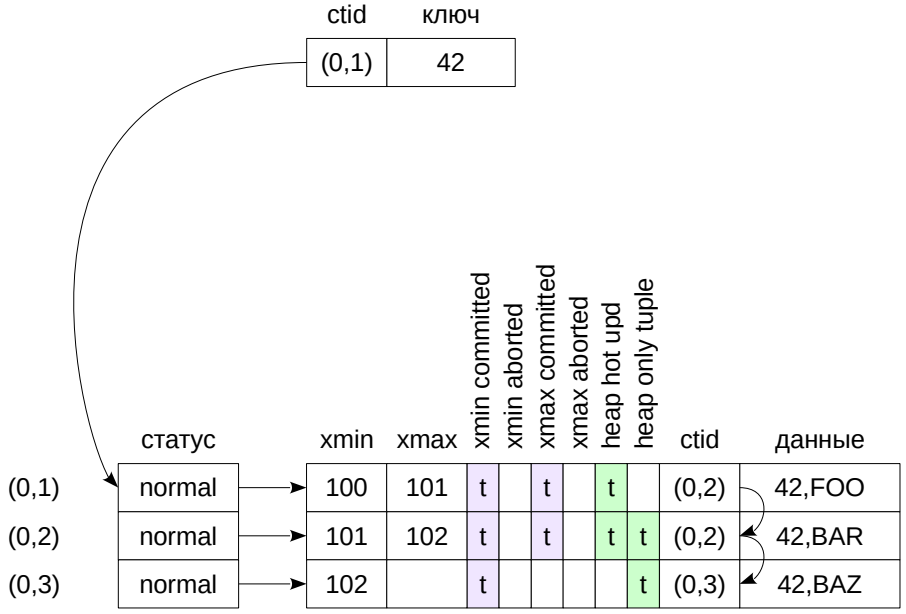
2. Страница заполнена больше, чем на *fillfactor*.

Внутристраничная очистка 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 HOT-очистки

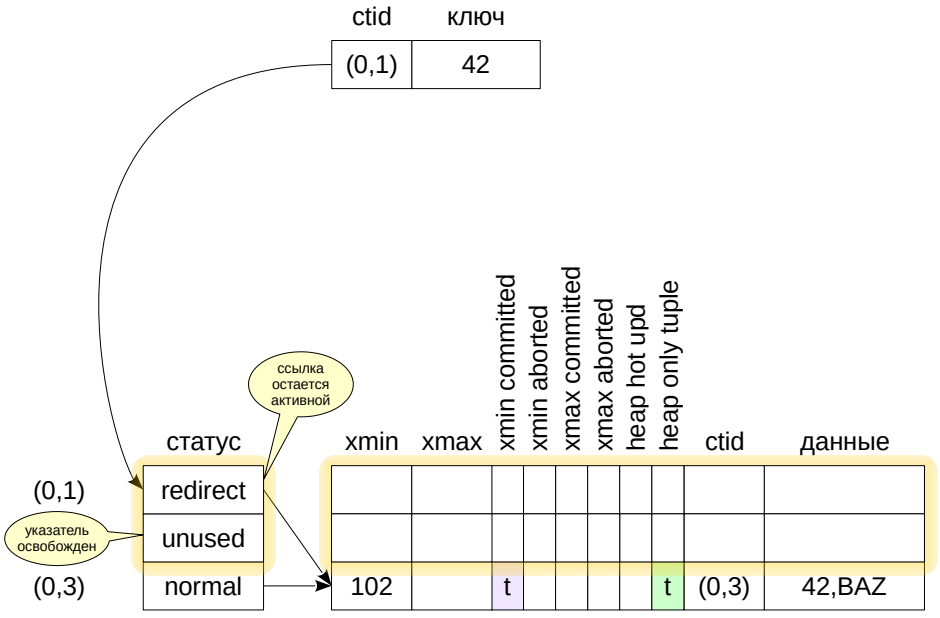


Частный, но важный случай внутристраничной очистки представляет собой очистка при HOT-обновлениях.

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 HOT-очистки



После срабатывания внутристраничной очистки неактуальные версии строк удаляются.

При HOT-обновлениях в индексе может быть только одна ссылка на версию строки, представляющую собой «голову» HOT-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 нужную версию 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.

Внутристраничная HOT-очистка

Проверим, как работает внутристраничная очистка при HOT-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 | 1204299 (c) | 1204300 (c) | t   |     | (0,2)
 (0,2) | normal | 1204300 (c) | 1204301 (c) | t   | t   | (0,3)
 (0,3) | normal | 1204301 (c) | 1204302     | t   | t   | (0,4)
 (0,4) | normal | 1204302     | 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        | 1204303     | 0 (a)   |     | t   | (0,2)
 (0,3) | unused        |             |         |     |     |
 (0,4) | normal        | 1204302 (c) | 1204303 | 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 первая версия осталась на месте, но получила статус redirect. Проследите ссылки от этой головной версии до конца HOT-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        | 1204303 (c) | 1204304 (c) | t   | t   | (0,3)
 (0,3) | normal        | 1204304 (c) | 1204305     | t   | t   | (0,5)
 (0,4) | normal        | 1204302 (c) | 1204303 (c) | t   | t   | (0,2)
 (0,5) | normal        | 1204305     | 0 (a)       |     | t   | (0,5)
(5 rows)
```

Следующее обновление снова вызывает очистку:

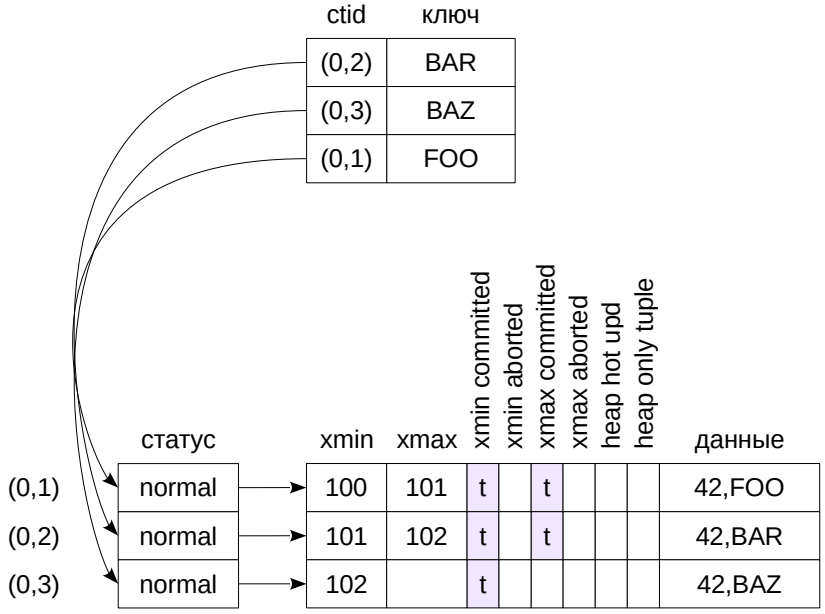
```
=> UPDATE t SET s = 'H';

UPDATE 1

=> SELECT * FROM t_v;

 ctid  |     state     |    xmin     |  xmax   | hhu | hot | t_ctid
-------+---------------+-------------+---------+-----+-----+--------
 (0,1) | redirect to 5 |             |         |     |     |
 (0,2) | normal        | 1204306     | 0 (a)   |     | t   | (0,2)
 (0,3) | unused        |             |         |     |     |
 (0,4) | unused        |             |         |     |     |
 (0,5) | normal        | 1204305 (c) | 1204306 | t   | t   | (0,2)
(5 rows)
```

До очистки

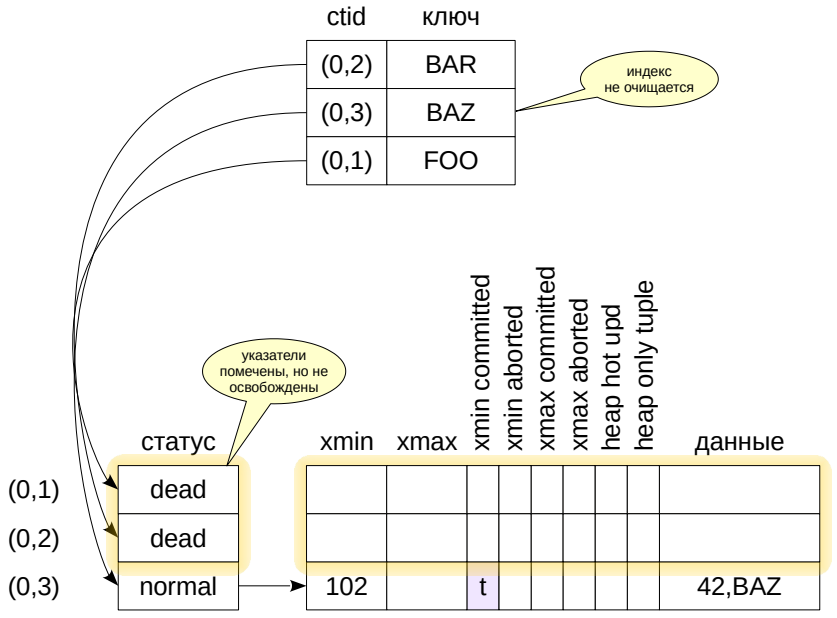


Разумеется, внутристраничная очистка работает и для версий, появившихся в результате обычного обновления.

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 очистки



Если выполнены условия срабатывания внутристраничной очистки, две неактуальные версии строки (0,1) и (0,2) могут быть удалены. Указатели на удаленные версии получает статус «dead», а освободившееся место может быть использовано для вставки новой версии.

В отличие от ситуации с HOT-обновлениями, здесь нельзя освободить указатели на удаленные версии строк, поскольку на них существует ссылки из индексной страницы. При индексном доступе PostgreSQL может получить (0,1) или (0,2) в качестве идентификатора версии строки, попробует получить саму строку из табличной страницы, но благодаря статусу указателя обнаружит, что эта версия уже не существует.

(На рисунке не показаны указатели на индексные строки. На самом деле, в первый раз обнаружив отсутствие версии табличной строки, PostgreSQL изменит и статус указателя в индексной странице, чтобы повторно не обращаться к табличной странице.)

Принципиально то, что внутристраничная очистка работает только в пределах одной табличной страницы и не очищает индексные страницы.

Итоги

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

При удобном случае автоматически выполняется быстрая внутристраничная очистка

При частых обновлениях можно подумать об уменьшении *fillfactor*

Практика

1. Воспроизведите ситуацию внутристраничной очистки без участия HOT-обновлений.
   Проверяйте 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 внутристраничная очистка не освобождает достаточно места на странице и новая версия создается на другой странице. Сколько строк будет в индексе в этом случае?

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

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

1. Внутристраничная очистка

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
```

Также создадим представление, чтобы заглядывать в индекс:

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

```
CREATE VIEW
```

Добавим строку и будем обновлять столбец s, как в демонстрации:

```
=> 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
```

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

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

```
 ctid  | state  |    xmin     |    xmax     | hhu | hot | t_ctid
-------+--------+-------------+-------------+-----+-----+--------
 (0,1) | normal | 1259247 (c) | 1259248 (c) |     |     | (0,2)
 (0,2) | normal | 1259248 (c) | 1259249 (c) |     |     | (0,3)
 (0,3) | normal | 1259249 (c) | 1259250     |     |     | (0,4)
 (0,4) | normal | 1259250     | 0 (a)       |     |     | (0,4)
(4 rows)
```

Как и в демонстрации, еще одно обновление строки приводит к внутристраничной очистке.:

```
=> 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 | 1259250 (c) | 1259251 |     |     | (0,5)
 (0,5) | normal | 1259251     | 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).

Указатели на очищенные строки 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 внутристраничной очистке:

```
=> 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 | 1259256 (c) | 1259257 (c) | t   |     | (0,2)
 (0,2) | normal | 1259257 (c) | 1259258 (c) | t   | t   | (0,3)
 (0,3) | normal | 1259258 (c) | 1259259     | t   | t   | (0,4)
 (0,4) | normal | 1259259     | 0 (a)       |     | t   | (0,4)
(4 rows)
```

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

Разрыв HOT-цепочки

Теперь, чтобы HOT-очистка не смогла освободить место, начнем параллельную транзакцию и построим в ней снимок данных.

```
=> \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        | 1259260 (c) | 1259261 (c) | t   | t   | (0,3)
 (0,3) | normal        | 1259261 (c) | 1259262     | t   | t   | (0,5)
 (0,4) | normal        | 1259259 (c) | 1259260 (c) | t   | t   | (0,2)
 (0,5) | normal        | 1259262     | 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        | 1259260 (c) | 1259261 (c) | t   | t   | (0,3)
 (0,3) | normal        | 1259261 (c) | 1259262 (c) | t   | t   | (0,5)
 (0,4) | normal        | 1259259 (c) | 1259260 (c) | t   | t   | (0,2)
 (0,5) | normal        | 1259262 (c) | 1259263     |     | t   | (1,1)
(5 rows)
```

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

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

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

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

```
=> COMMIT;
```

```
COMMIT
```