Postgres PROFESSIONAL

Страницы и версии строк

Темы

Структура страницы

Устройство версий строк

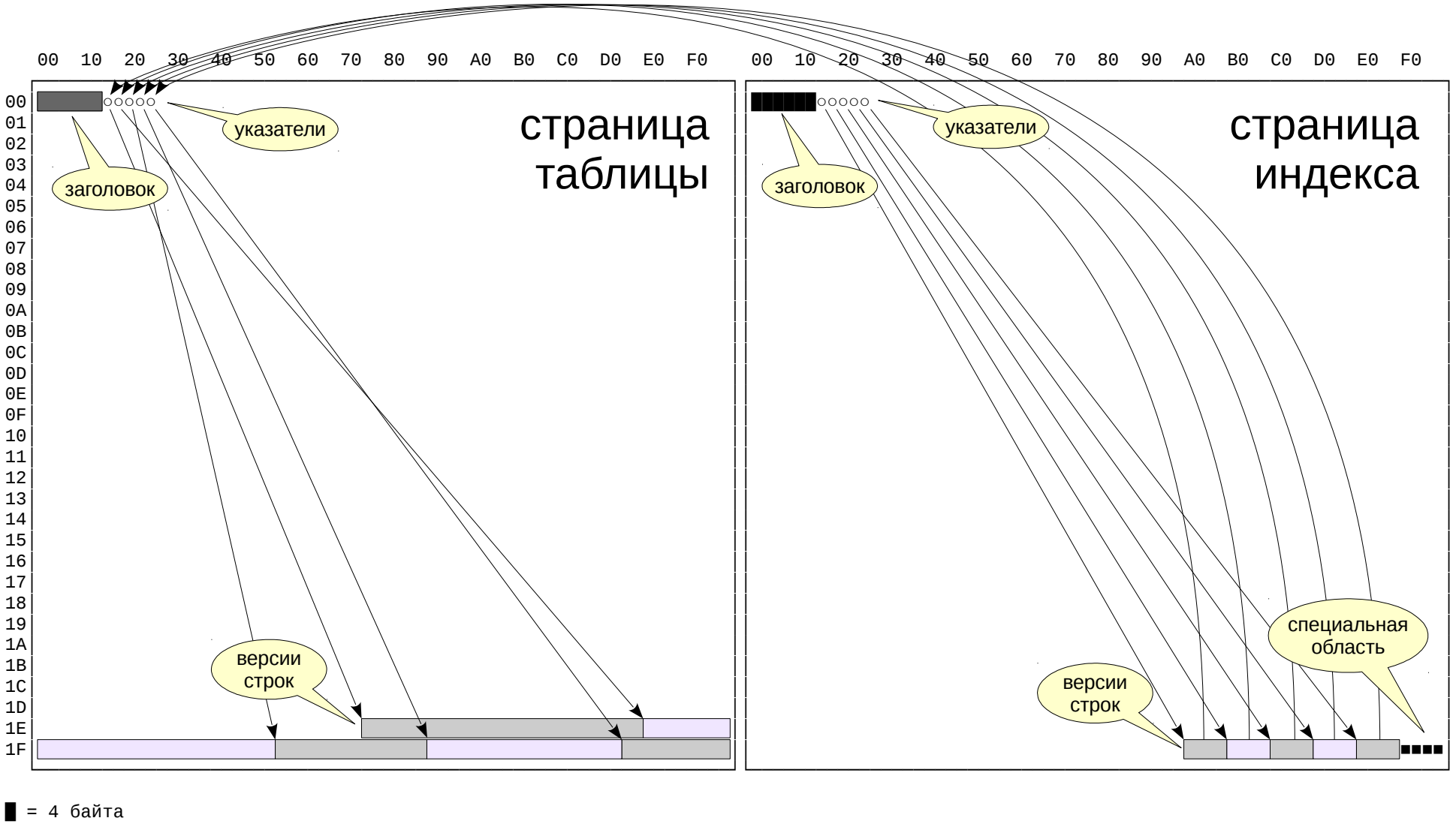
Как работают вставка, обновление и удаление строк

Как работают фиксация и откат изменений

HOT-обновления

Точка сохранения и вложенные транзакции

Страницы


00 10 20 30 40 50 60 70 80 90 A0 B0 C0 D0 E0 F0
00 10 20 30 40 50 60 70 80 90 A0 B0 C0 D0 E0 F0
00
01
02
03
04
05
06
07
08
09
0A
0B
0C
0D
0E
0F
10
11
12
13
14
15
16
17
18
19
1A
1B
1C
1D
1E
1F
страница таблицы
заголовок
указатели
версии строк
страница индекса
заголовок
указатели
специальная область
версии строк
= 4 байта

Версии строк в таблице

Массив указателей

offset, len	указатель на версию строки и ее длина
flags	статус версии строки

Версии строк

xmin	номер транзакции, создавшей версию
xmax	номер транзакции, удалившей версию
infomask	информационные биты
ctid	указатель на след. версию той же строки
данные	

Каждая версия строки целиком помещается в странице

- часть полей может быть сжата
- часть полей может быть отправлена во внешнее TOAST-хранилище

TOAST

Служебная таблица

для каждой обычной таблицы (pg_toast_*N*), поддержана индексом

«длинные» значения порезаны на части, помещающиеся на страницу

используется прозрачно для приложения

читается только при обращении к «длинному» значению

собственная версионность, не зависящая от основной таблицы

Настройка

`ALTER TABLE` *`таблица`* `ALTER` *`столбец`* `SET STORAGE` *`стратегия`*

`PLAIN`	запрещает и сжатие, и внешнее хранение
`EXTENDED`	разрешает сжатие и внешнее хранение
`EXTERNAL`	разрешает только внешнее хранение
`MAIN`	в первую очередь сжатие

Строки в индексе

Указатели

offset, len	указатель на версию строки и ее длина
flags	статус версии строки

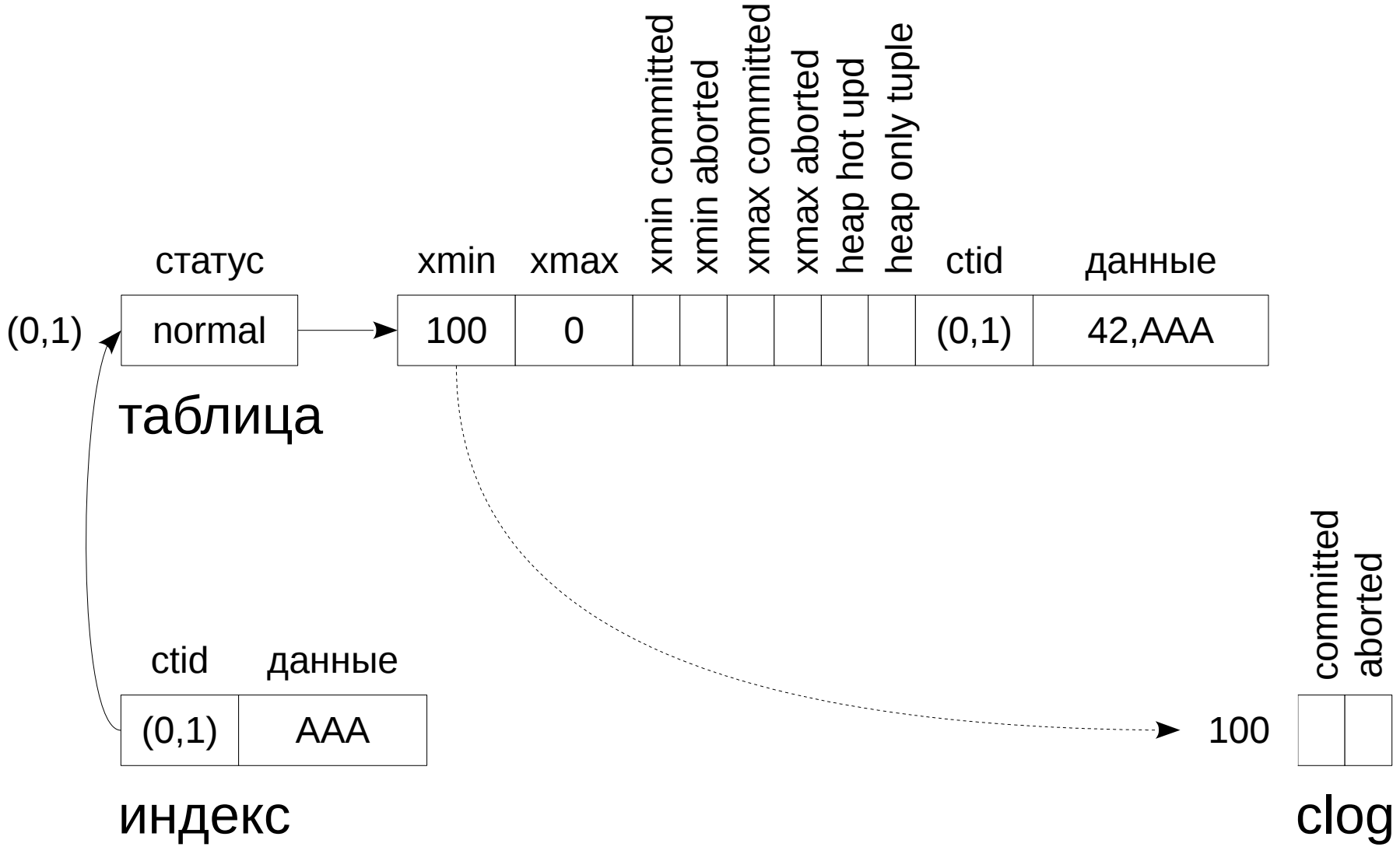
Строки

ctid	указатель на версию строки в таблице
данные	значения ключей индексирования
...	

Хранимые данные зависят от типа индекса

Нет информации о версионности

Вставка


статус
xmin
xmax
xmin committed
xmin aborted
xmax committed
xmax aborted
heap hot upd
heap only tuple
ctid
данные
(0,1)
normal
100
0
(0,1)
42,AAA
таблица
ctid
данные
(0,1)
AAA
индекс
100
committed
aborted
clog

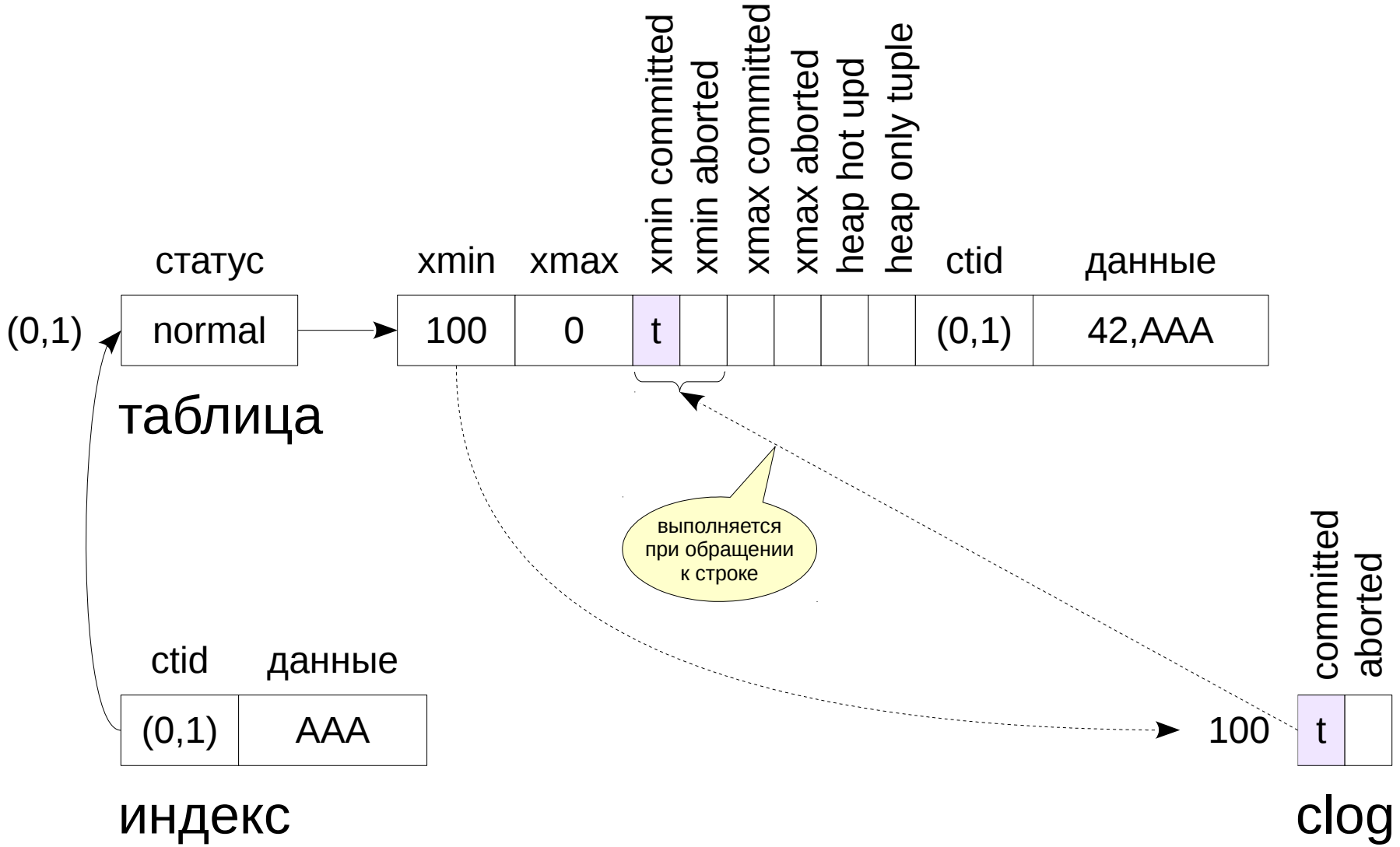
Вставка

Вставка в страницу, только если процент занятого места не превышает fillfactor

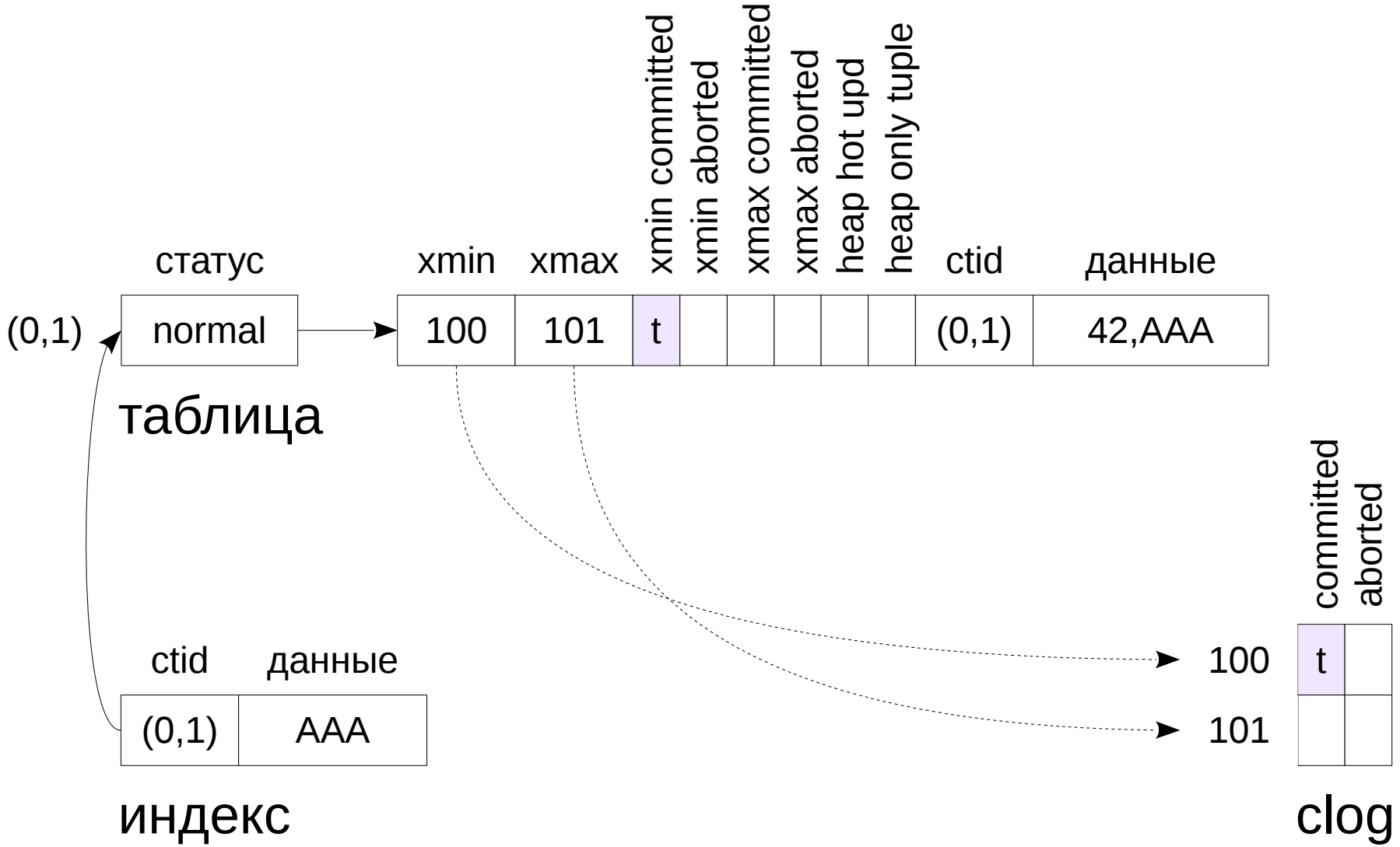
```
CREATE TABLE ( ... )
WITH (fillfactor = значение);
```

меньше (от 10%)	*больше (до 100%)*
плотность ниже	плотность выше
таблица больше	таблица меньше
рекомендуется при частых обновлениях	рекомендуется при статичных данных

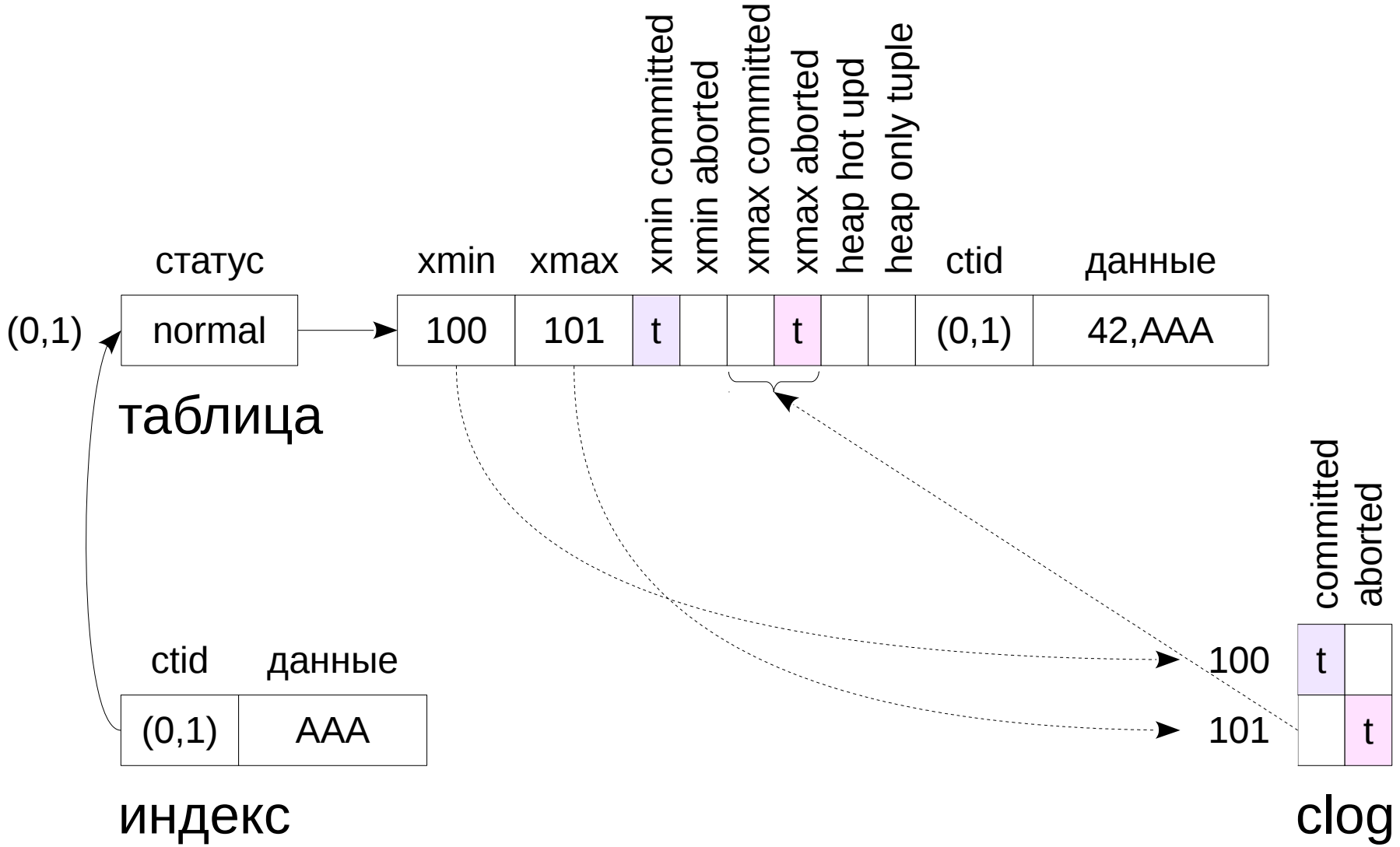
Фиксация изменений



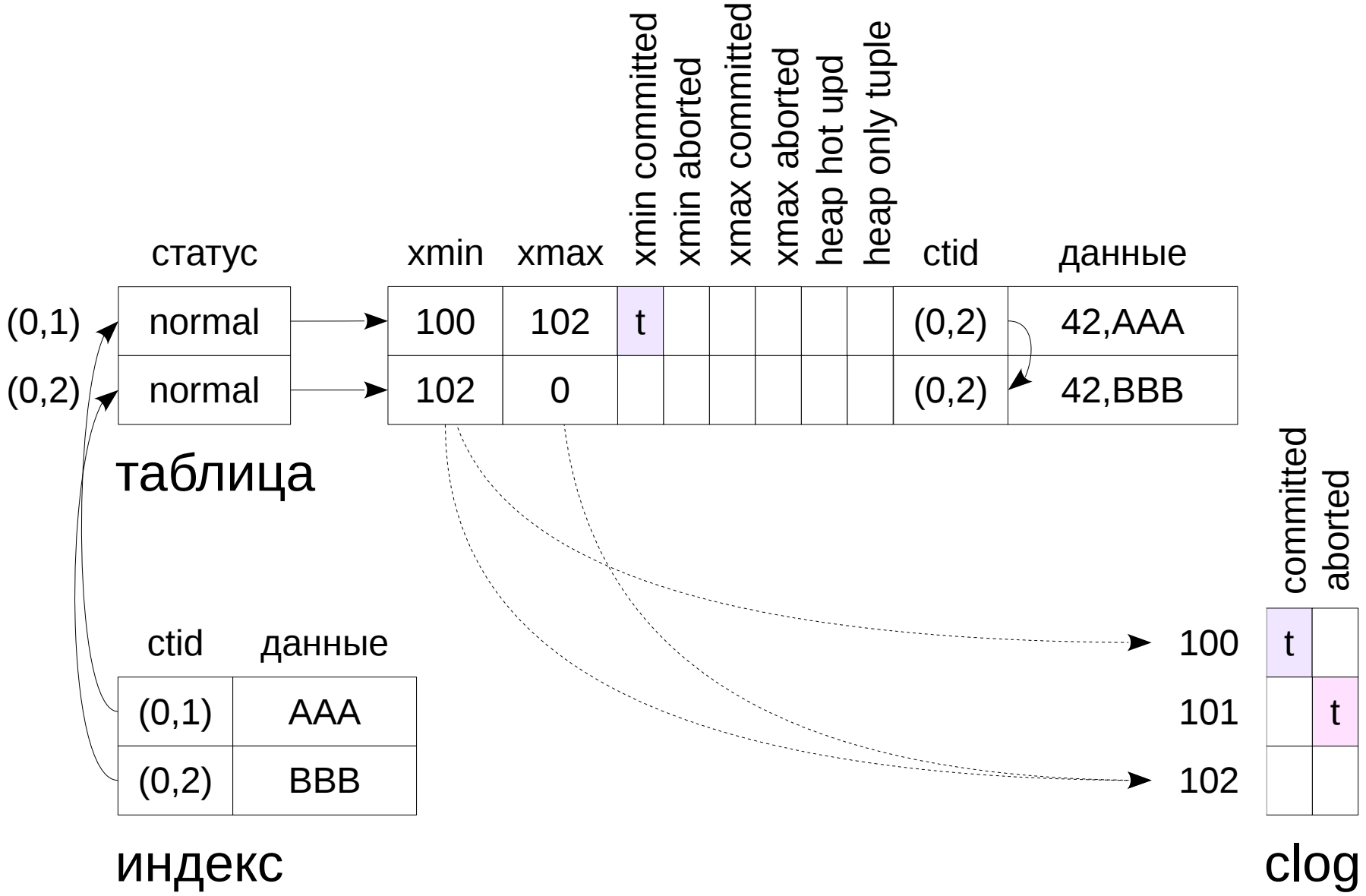
Удаление


статус
xmin
xmax
xmin committed
xmin aborted
xmax committed
xmax aborted
heap hot upd
heap only tuple
ctid
данные
(0,1)
normal
100
101
t
(0,1)
42,AAA
таблица
committed
aborted
ctid
данные
(0,1)
AAA
индекс
100
101
t
clog

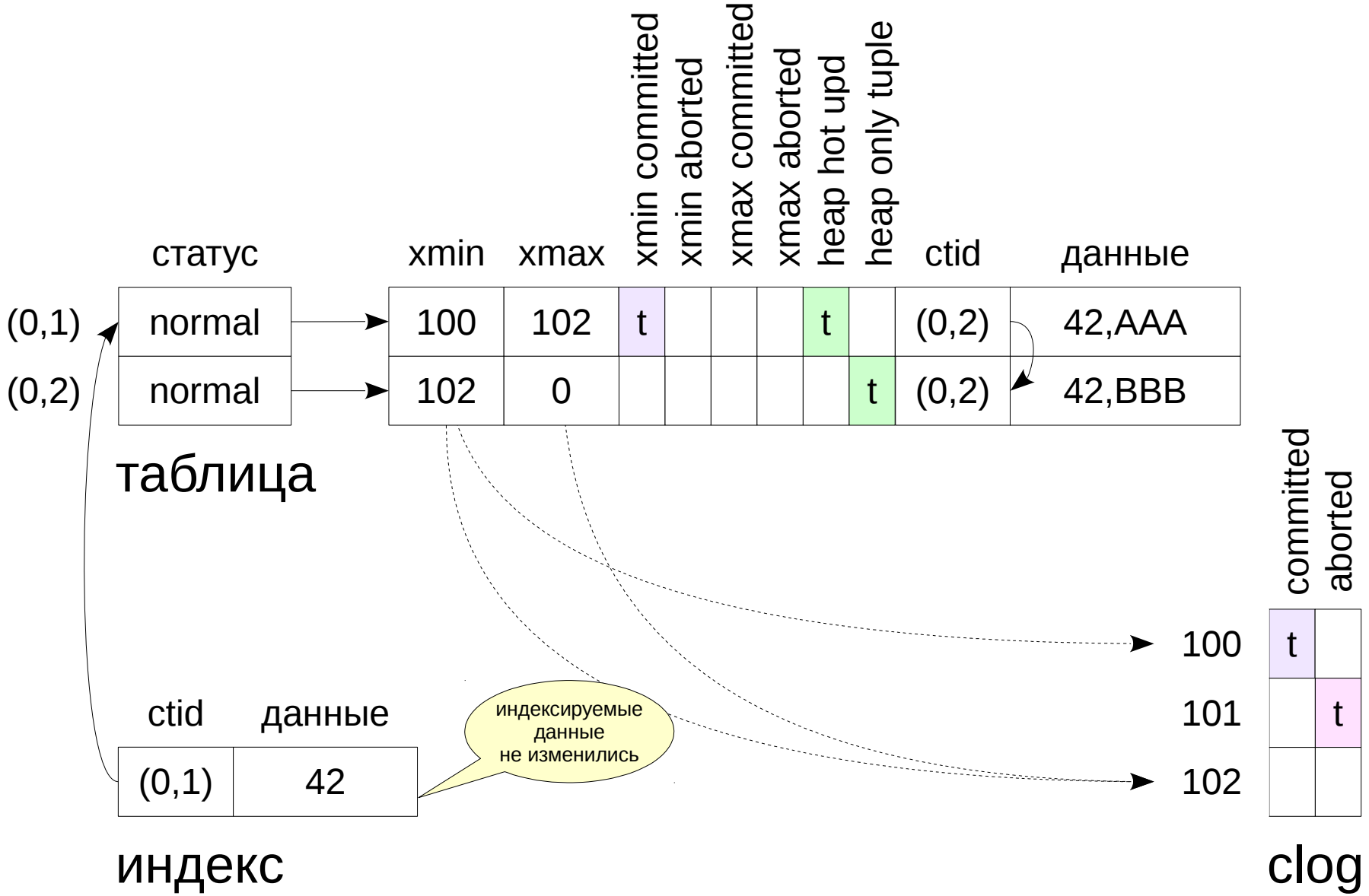
Откат изменений



Обновление


статус
xmin
xmax
xmin committed
xmin aborted
xmax committed
xmax aborted
heap hot upd
heap only tuple
ctid
данные
(0,1)
normal
100
102
t
(0,2)
42,AAA
(0,2)
normal
102
0
(0,2)
42,BBB
таблица
ctid
данные
(0,1)
AAA
(0,2)
BBB
индекс
committed
aborted
100
t
101
t
102
clog

HOT-обновление

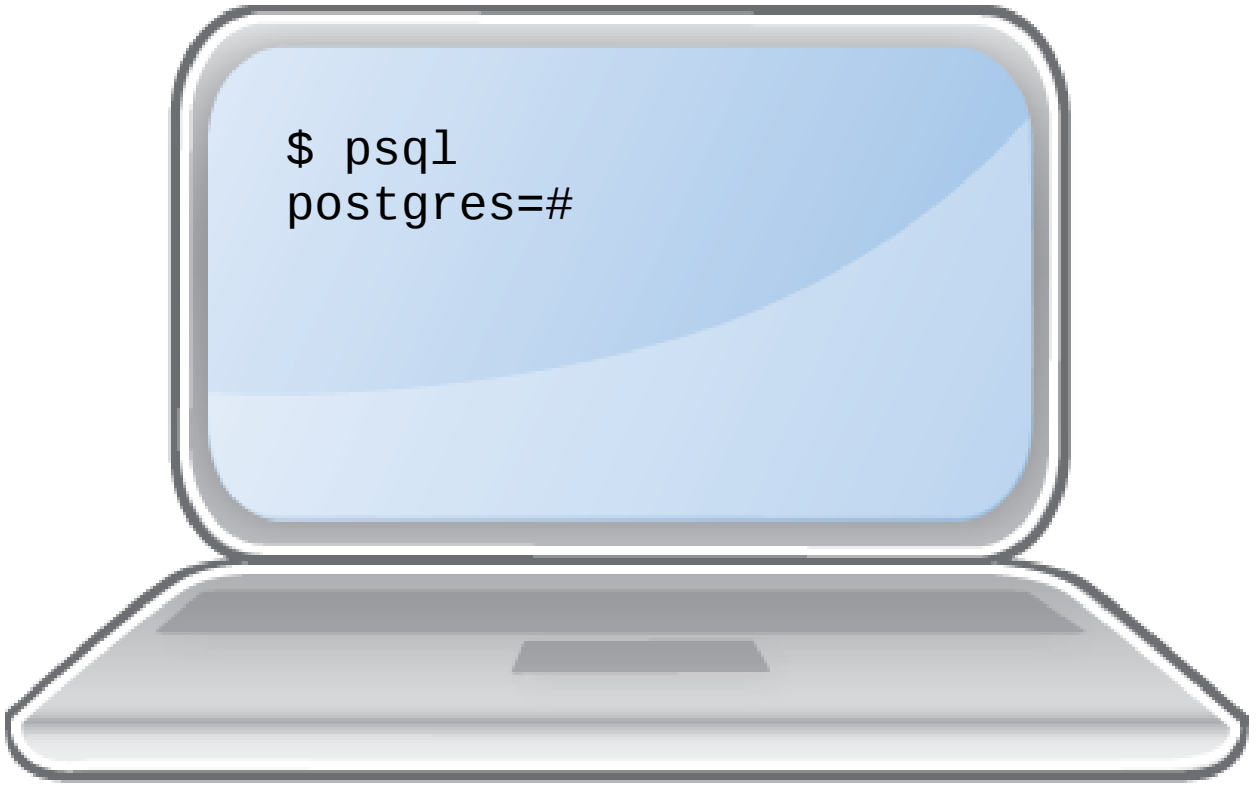


HOT-обновление

Цепочка обновлений строится в пределах одной страницы

- не требуется обращение к другим блокам,
  не ухудшается производительность
- если на странице не хватает места, цепочка обрывается
  (как если бы оптимизация не работала)
- уменьшение fillfactor увеличивает шансы на оптимизацию

Демонстрация



Точка сохранения

```
BEGIN;
  insert into t(n) values (42);
SAVEPOINT SP;
  delete from t;
ROLLBACK TO SP;
  update t set n=n+1;
COMMIT;
```

Как откатить только часть изменений транзакции?

Вложенные транзакции

Собственный номер

Собственный статус в CLOG

- конечный статус зависит от статуса основной транзакции
- не то же самое, что автономные транзакции!

Информация хранится на диске

- каталог $PGDATA/pg_subtrans/
- данные кэшируются в буферах (аналогично CLOG)

Вложенные транзакции

```
BEGIN;                                                    txid=100
  insert into t(n) values (42);
```

```
SAVEPOINT SP;                                             txid=101
  delete from t;
ROLLBACK TO SP;
```

```
  update t set n=n+1;                                     txid=102
```

```
COMMIT;
```

Итоги

В странице могут храниться несколько версий строки, ограниченные номерами транзакций

Только табличные страницы содержат информацию о версионности

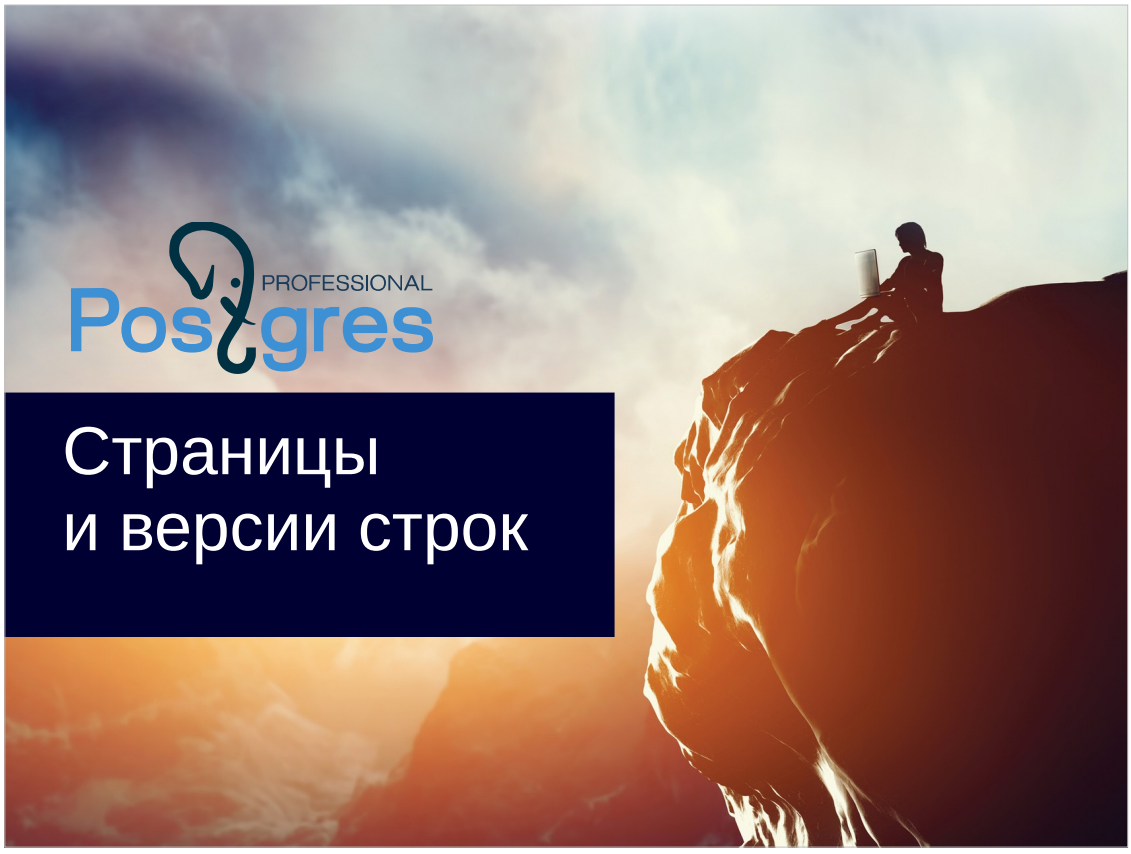
Фиксация и откат выполняются одинаково быстро

Для точек сохранения используются вложенные транзакции

Практика

1. Создать базу данных DB3, в ней установить расширение pageinspect и создать таблицу.
2. Внутри одной транзакции:
   – вставить строку в таблицу;
   – создать точку сохранения;
   – вставить в таблицу еще одну строку;
   – откатить изменения до точки сохранения;
   – вставить еще одну строку;
   – зафиксировать изменения.

   При этом после каждой вставки:
   – вывести номер текущей транзакции;
   – проконтролировать таблицу с помощью псевдостолбцов xmin и xmax;
   – проконтролировать содержимое страницы с помощью pageinspect.
3. Объяснить полученные результаты.



Темы

Структура страницы

Устройство версий строк

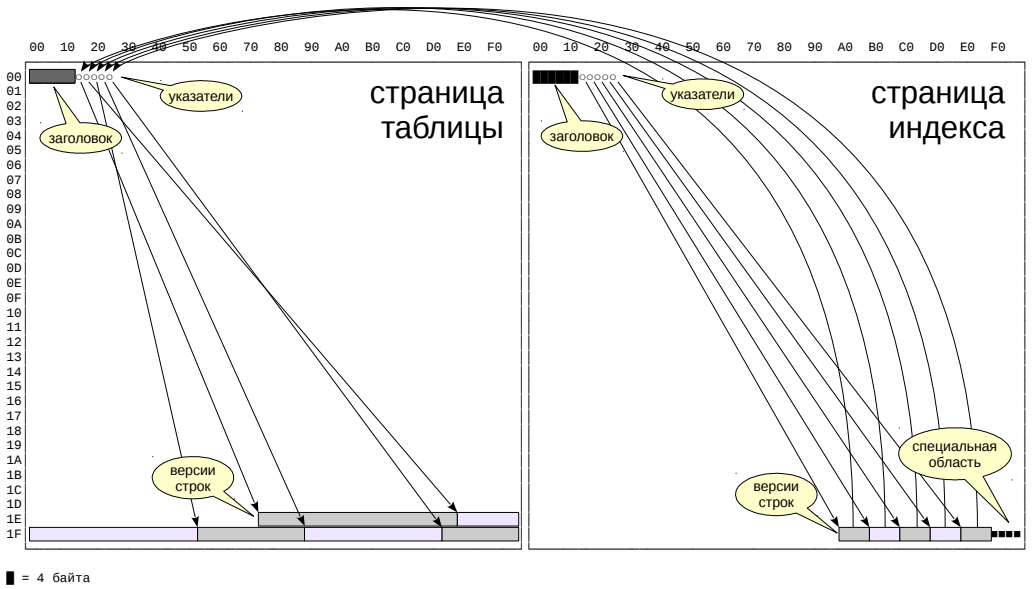
Как работают вставка, обновление и удаление строк

Как работают фиксация и откат изменений

HOT-обновления

Точка сохранения и вложенные транзакции

Страницы



Размер страницы составляет 8 килобайт. Это значение увеличить изменить (вплоть до 32 килобайт), но только при сборке.

И таблицы, и индексы, и большинство других объектов, которые в PostgreSQL обозначаются по-английски термином relation, используют одинаковую структуру страниц, чтобы пользоваться общим буферным кэшем. В начале страницы идет заголовок (24 байта), содержащий общие сведения и размер следующих областей: указателей, свободного пространства, версий строк и специальной области.

«Версия строки» называется по-английски tuple; если это не нарушает однозначности, мы будем сокращать название до просто «строки».

Указатели имеют фиксированный размер (четыре байта) и составляют массив, позиция в котором определяет идентификатор строки (tuple id, tid). Указатели ссылаются на собственно строки (tuple), которые расположены в конце блока. Такая косвенная адресация удобна тем, что во-первых, позволяет найти строку, не перебирая все содержимое блока (строки имеют разную длину), а во-вторых, позволяют перемещать строку внутри блока, не нарушая внешних ссылок из индексов.

Между указателями и версиями строк находится свободное место.
В конце блока может находиться специальная область, которая используется в некоторых индексных страницах.

Версии строк в таблице

Массив указателей

offset, len	указатель на версию строки и ее длина
flags	статус версии строки

Версии строк

xmin	номер транзакции, создавшей версию
xmax	номер транзакции, удалившей версию
infomask	информационные биты
ctid	указатель на след. версию той же строки
данные	

Каждая версия строки целиком помещается в странице

- часть полей может быть сжата
- часть полей может быть отправлена во внешнее TOAST-хранилище

Страница содержит массив указателей на версии строк.

Каждый указатель содержит ссылку на строку, а также несколько бит, определяющих статус этой строки.

Версии строк в табличных страницах (heap pages по-английски) кроме собственно данных, имеют также заголовок. Заголовок, помимо прочего, содержит следующие важные поля:

- xmin и xmax определяют видимость данной версии строки в терминах начального и конечного номеров транзакций;

- infomask содержит ряд битов, определяющих свойства данной версии. Часть этих битов служит «подсказками» для оптимизации;

- ctid является ссылкой на следующую, более новую, версию той же строки.

При этом каждая версия строки всегда помещается внутрь одной страницы. Если версия строки имеет большой размер, PostgresSQL попробует сжать часть полей. Если не получается за счет этого достаточно уменьшить размер строки, она отправляется во внешнее хранилище TOAST.

TOAST

Служебная таблица

для каждой обычной таблицы (pg_toast_*N*), поддержана индексом

«длинные» значения порезаны на части, помещающиеся на страницу

используется прозрачно для приложения

читается только при обращении к «длинному» значению

собственная версионность, не зависящая от основной таблицы

Настройка

`ALTER TABLE` *`таблица`* `ALTER` *`столбец`* `SET STORAGE` *`стратегия`*

`PLAIN`	запрещает и сжатие, и внешнее хранение
`EXTENDED`	разрешает сжатие и внешнее хранение
`EXTERNAL`	разрешает только внешнее хранение
`MAIN`	в первую очередь сжатие

TOAST — The Oversized Attributes Storage Technique — технология хранения «длинных» значений в отдельных служебных таблицах. Для каждой основной таблицы при необходимости создается отдельная toast-таблица (и к ней специальный индекс).

Версии строк в toast-таблице тоже должны помещаться на одну страницу. Поэтому «длинные» значения хранятся порезенными на части. Из этих частей PostgreSQL прозрачно для приложения «склеивает» необходимое значение. Обычными средствами toast-таблица не видна.

Преимущество такого похода состоит в том, что toast-таблица используется только при обращении к «длинному» значению (еще один повод не писать «select *» в серьезном коде). Кроме того, для toast-таблицы поддерживается своя версионность. То есть если обновление данных не затрагивает «длинное» значение, новая версия строки будет ссылаться на то же самое значение в toast-таблице — это экономит место.

Недостаток же состоит в том, что при обращении к «длинному» значению приходится обращаться не к одной таблице, а к двум.

Стратегию использования toast-хранилища можно настраивать на уровне каждого столбца. По умолчанию для потенциально длинных значений используется EXTENDED (сначала сжатие, потом внешнее хранение).

http://www.postgresql.org/docs/9.5/static/storage-toast.html

Строки в индексе

Указатели

offset, len	указатель на версию строки и ее длина
flags	статус версии строки

Строки

ctid	указатель на версию строки в таблице
данные	значения ключей индексирования
...	

Хранимые данные зависят от типа индекса

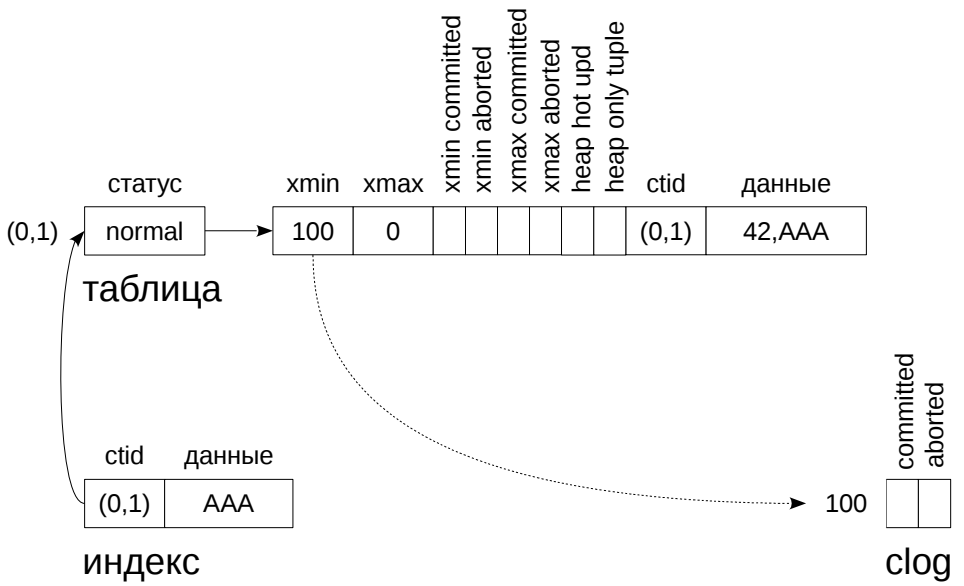
Нет информации о версионности

Информация в индексной странице зависит от типа индекса. Тем не менее, обычно имеется массив указателей и строки (как в табличной странице). Кроме того, в конце страницы отводится место под специальные данные.

Строки в индексах могут иметь очень разную структуру в зависимости от типа индекса. Обычно они содержат как минимум значение ключа индексирования и ссылку (ctid) на соответствующую строку таблицы.

Важный момент состоит в том, что индекс не содержит информации о версионности. Прочитав строку индекса, невозможно определить видимость этой строки, не заглянув в табличную страницу (для оптимизации служит карта видимости).

Вставка



Рассмотрим, как выполняются операции со строками на низком уровне, и начнем со вставки.

В нашем примере предполагается таблица с двумя столбцами (числовой и текстовый); по текстовому полю создан индекс b-tree.

При вставке строки в табличной странице появится указатель с номером 1, ссылающийся на первую и единственную версию строки. Номера на рисунке имеют вид (0,1): здесь 0 — номер блока, а 1 — порядковый номер указателя.

В версии строки поле xmin заполнено номером текущей транзакции (100 в нашем примере). Поскольку изменения еще не фиксировались, в CLOG соответствующая запись (два бита) заполнена нулями — признак активной транзакции.

Поле ctid версии строки ссылается на эту же строку. Это означает, что более новой версии не существует.

В индексной странице также создается указатель с номером 1, который ссылается на версию строки, которая, в свою очередь, ссылается на первую строку в табличной странице. Чтобы не загромождать рисунок, указатель и строка объединены.

Вставка

Вставка в страницу, только если процент занятого места не превышает fillfactor

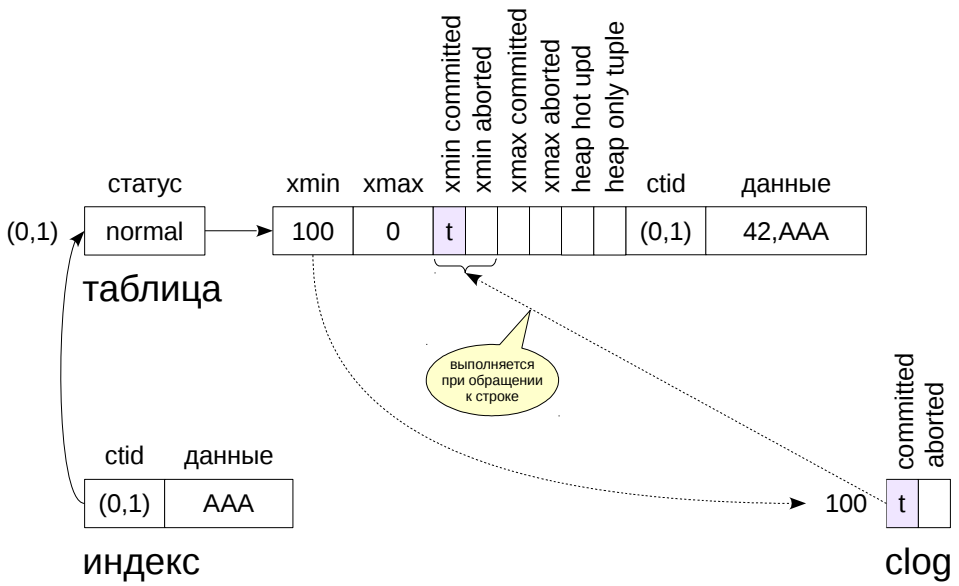
```
CREATE TABLE ( ... )
WITH (fillfactor = значение);
```

меньше (от 10%)	*больше (до 100%)*
плотность ниже	плотность выше
таблица больше	таблица меньше
рекомендуется при частых обновлениях	рекомендуется при статичных данных

Вставка строки произойдет в страницу, только если процент занятого места в ней не превышает fillfactor (по умолчанию он равен 100%).

Чем выше fillfactor, тем компактнее располагаются записи и, соответственно, размер таблицы получается меньше. Однако при обновлении строк на данной странице может не найтись свободного места и новая версия будет помещена в другую страницу (что менее эффективно).

Фиксация изменений



При фиксации изменений в CLOG для данной транзакции выставляется признак committed. Это единственная операция (не считая, конечно, журнала упреждающей записи), которая необходима.

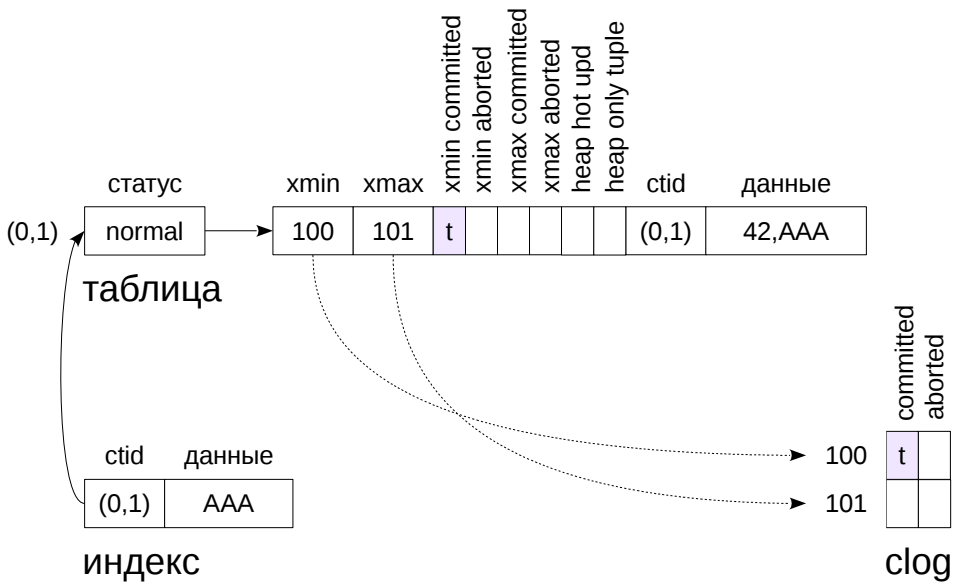
Когда какая-либо другая транзакция обратится к этой табличной странице, ей придется ответить на вопрос: была ли транзакция 100 зафиксирована или откачена (то есть, иными словами, существует ли строка или нет)? Для этого ей придется свериться с CLOG.

Поскольку выполнять такую проверку каждый раз накладно, выясненный статус транзакции записывается в биты-подсказки xmin committed и xmin aborted. Если один из этих битов установлен, то состояние транзакции xmin считается известным и следующей транзакции уже не придется обращаться к CLOG.

Почему эти биты не устанавливаются той транзакцией, которая выполняла вставку? Дело в том, что в момент времени, когда судьба транзакции определится (выполнена фиксация или откат), будет совершенно непонятно, какие именно строки в каких именно страницах транзакция успела поменять. Кроме того, часть этих страниц может быть вытеснена из буферного кэша на диск; читать их заново, чтобы изменить биты, означало бы существенно замедлить фиксацию.

Обратная сторона отложенного изменения состоит в том, что любая транзакция (даже выполняющая простое чтение — select) может неожиданно породить большой объем журналов (при установленном параметре wal_log_hints или при наличии в странице контрольных сумм) и вызвать вытеснение грязных буферов из кэша, если перед ней массово изменялись данные.

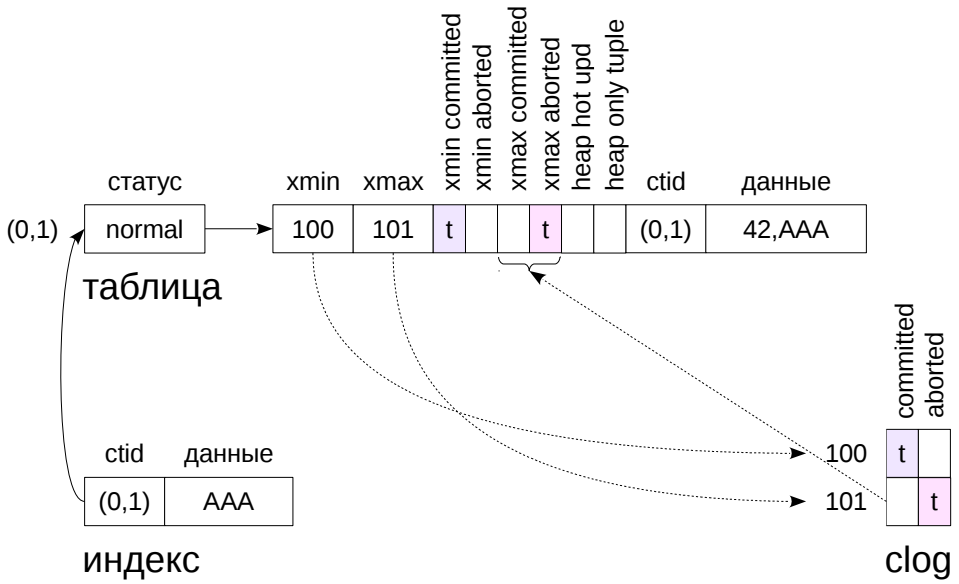
Удаление



При удалении строки в поле xmax записывается номер текущей удаляющей транзакции.

Больше ничего не происходит.

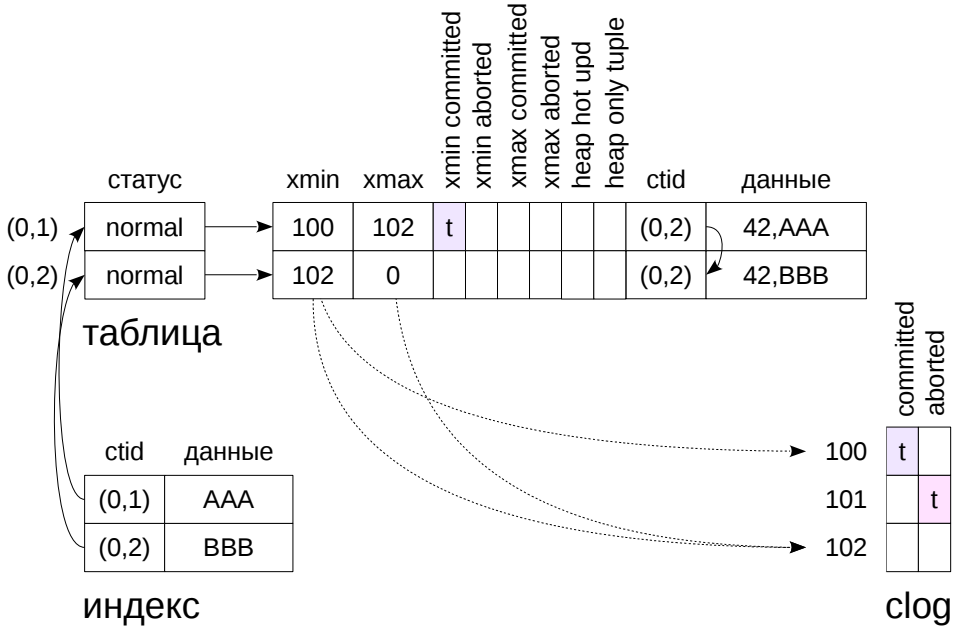
Откат изменений



Откат изменений работает аналогично фиксации, только в CLOG для транзакции выставляется бит aborted. Он выполняется так же быстро, как и фиксация.

Номер откаченной транзакции остается в поле xmax — его можно было бы стереть, но в этом нет смысла. При обращении к странице статус транзакции будет перенесен в версию строки в биты xmax committed и xmax aborted.

Обновление



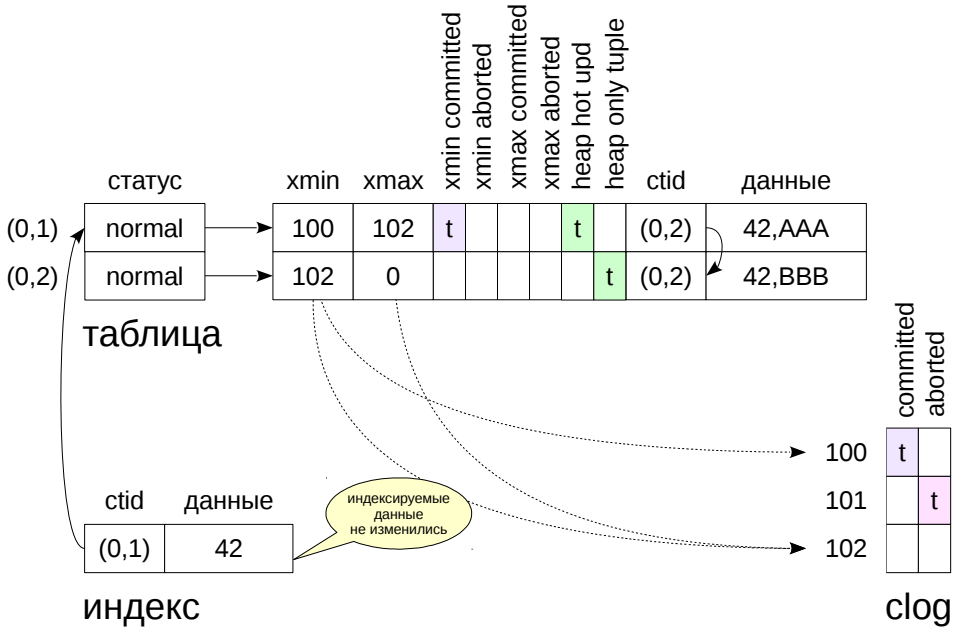
Обновление работает так, как будто сначала выполнялось удаление старой версии строки, а затем вставка новой.

Старая версия помечается номером текущей транзакции в поле xmax. Обратите внимание, что новое значение 102 записалось поверх старого 101, так как транзакция 101 была откачена. Кроме того, биты xmax committed и xmax aborted сброшены в ноль, так как статус текущей транзакции еще не известен.

Первая версия строки ссылается теперь на вторую (поле ctid), как на более новую.

В индексной странице появляется второй указатель и вторая строка, ссылающаяся на вторую версию в табличной странице.

HOT-обновление



Что, если индекс создан по полю, значение которого не изменилось в результате обновления строки? В этом случае нет смысла создавать дополнительный узел b-tree, содержащий то же самое, не измененное, значение. Для того, чтобы не создавать лишние узлы (которые потом все равно пришлось бы удалять из страницы), сделана важная оптимизация Heap-Only Tuple Update.

Дополнительный узел индекса не создается, в индексной странице по-прежнему будет одна строка, ссылающаяся на первую строку 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;

- поддерживается обычная связь версий строк через поле ctid.

Таким образом, попадая в табличную страницу из индекса и обнаруживая версию, помеченную как heap hot updated, PostgreSQL понимает, что надо также пройти по всей цепочке обновлений.

HOT-обновление

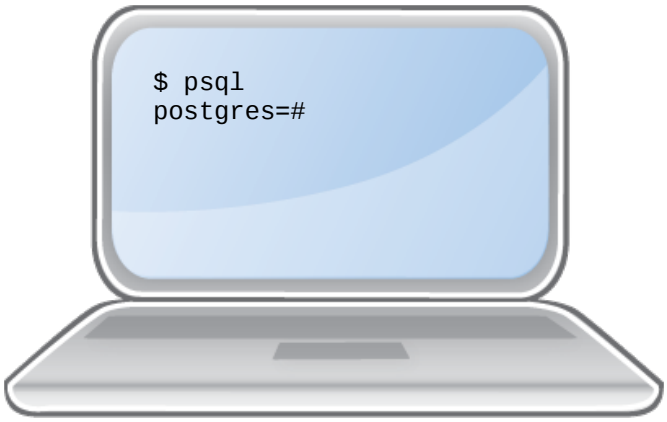
Цепочка обновлений строится в пределах одной страницы

- не требуется обращение к другим блокам,
  не ухудшается производительность
- если на странице не хватает места, цепочка обрывается
  (как если бы оптимизация не работала)
- уменьшение fillfactor увеличивает шансы на оптимизацию

Оптимизация действует только в пределах одного блока, поэтому дополнительный обход цепочки не требует обращения к другим блокам и не ухудшает производительность.

Однако если на странице не хватит свободного места, чтобы разместить новую версию строки, цепочка прервется. Поэтому при частых обновлениях имеет смысл уменьшать fillfactor таблицы.

Демонстрация



Точка сохранения

```
BEGIN;
  insert into t(n) values (42);
SAVEPOINT SP;
  delete from t;
ROLLBACK TO SP;
  update t set n=n+1;
COMMIT;
```

Как откатить только часть изменений транзакции?

Тонкий момент представляют функционал точек сохранения, позволяющий откатить часть текущей транзакции. Это не укладывается в приведенную выше схему, поскольку физически никакие данные не откатываются, лишь изменяется статус всей транзакции.

Вложенные транзакции

Собственный номер

Собственный статус в CLOG

- конечный статус зависит от статуса основной транзакции
- не то же самое, что автономные транзакции!

Информация хранится на диске

- каталог $PGDATA/pg_subtrans/
- данные кэшируются в буферах (аналогично CLOG)

Для этого случая применяется механизм вложенных (не путать с автономными) транзакций.

Вложенные транзакции порождаются командой savepoint и rollback to savepoint и имеют свой номер (больший, чем номер основной транзакции). Статус вложенных транзакций записывается обычным образом в CLOG, однако финальный статус зависит от статуса основной транзакции: если она откачена, то откатываются и все вложенные транзакции.

Информация о вложенности транзакций хранится в каталоге $PGDATA/pg_subtrans/. Обращение к файлам происходит через буферы, организованные так же, как и буферы CLOG.

Вложенные транзакции

```
BEGIN;                                          txid=100
  insert into t(n) values (42);
SAVEPOINT SP;                                   txid=101
  delete from t;
ROLLBACK TO SP;
  update t set n=n+1;                           txid=102
COMMIT;
```


В приведенном примере первая вложенная транзакция (101) будет создана при выполнении команды savepoint.

Встретив команду rollback to savepoint, PostgreSQL откатывает транзакцию 101 и порождает новую вложенную транзакцию (102). Новая транзакция нужна, так как в коде снова может встретиться rollback to savepoint и снова потребуется выполнить частичный откат.

Встретив команду commit, PostgreSQL фиксирует транзакции 102 и 100, при этом 101 остается откаченной.

Итоги

В странице могут храниться несколько версий строки, ограниченные номерами транзакций

Только табличные страницы содержат информацию о версионности

Фиксация и откат выполняются одинаково быстро

Для точек сохранения используются вложенные транзакции

Практика

1. Создать базу данных DB3, в ней установить расширение pageinspect и создать таблицу.
2. Внутри одной транзакции:
   – вставить строку в таблицу;
   – создать точку сохранения;
   – вставить в таблицу еще одну строку;
   – откатить изменения до точки сохранения;
   – вставить еще одну строку;
   – зафиксировать изменения.

   При этом после каждой вставки:
   – вывести номер текущей транзакции;
   – проконтролировать таблицу с помощью псевдостолбцов xmin и xmax;
   – проконтролировать содержимое страницы с помощью pageinspect.
3. Объяснить полученные результаты.