

# Многоверсионность Страницы и версии строк



## 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:

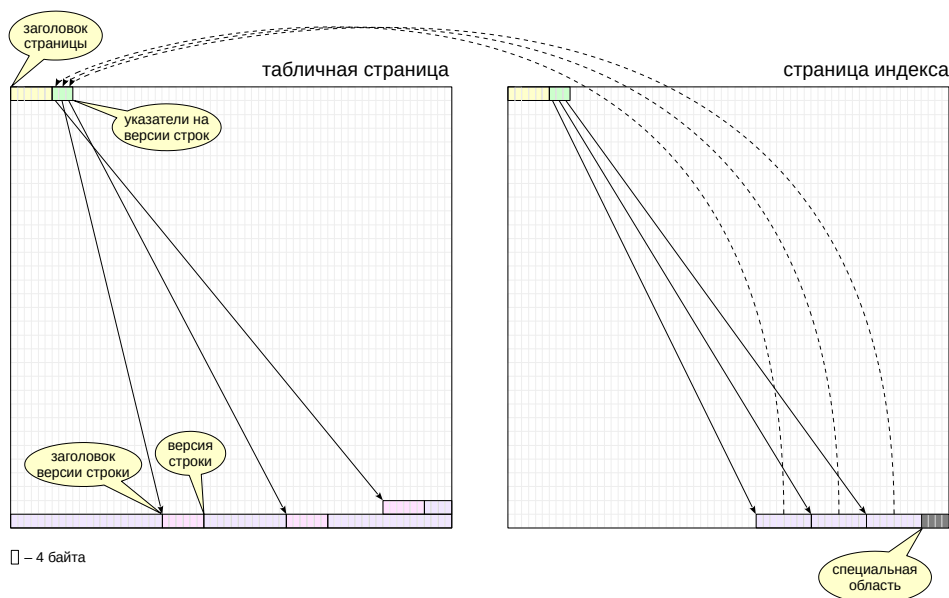
[edu@postgrespro.ru](mailto:edu@postgrespro.ru)

## 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.

Структура страниц и версий строк  
Как работают операции над данными  
Вложенные транзакции

# Структура страниц



3

Размер страницы составляет 8 Кбайт. Это значение можно увеличить (вплоть до 32 Кбайт), но только при сборке. И таблицы, и индексы, и большинство других объектов, которые в PostgreSQL обозначаются термином *relation*, используют одинаковую структуру страниц, чтобы пользоваться общим буферным кешем. В начале страницы идет **заголовок** (24 байта), содержащий общие сведения о странице и размер ее областей: указателей, свободного пространства, версий строк и специальной области.

**Версии строк** содержат те самые данные, которые мы храним в таблицах и других объектах БД, плюс заголовок. «Версия строки» по-английски называется *tuple*; иногда мы будем говорить просто «строка».

**Указатели** имеют фиксированный размер (4 байта) и составляют массив, позиция в котором определяет идентификатор строки (*tuple id*, *tid*). Указатели ссылаются на версии строк (*tuple*), расположенные в конце блока. Такая косвенная адресация удобна тем, что во-первых, позволяет найти нужную строку, не перебирая все содержимое блока (строки имеют разную длину), а во-вторых, позволяет перемещать строку внутри блока, не ломая ссылки из индексов

Между указателями и версиями строк находится **свободное место**.

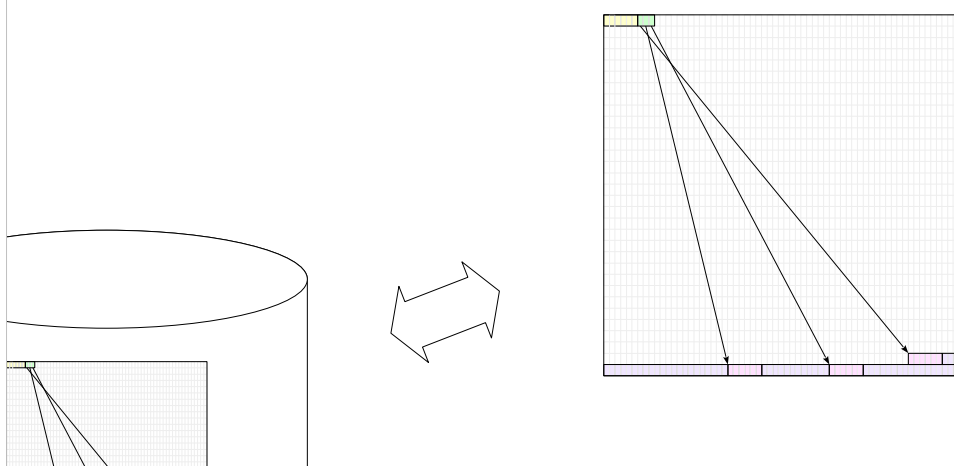
Для некоторых типов индексов нужно хранить служебную информацию; для этого может использоваться нулевая страница, а также **специальная область** в конце каждой страницы.

<https://postgrespro.ru/docs/postgresql/16/storage-page-layout>

## Страницы читаются в оперативную память «как есть»

данные не переносятся между разными платформами

между полями данных возможны пропуски из-за выравнивания



4

Формат данных на диске полностью совпадает с представлением данных в оперативной памяти. Страница записывается в файл и читается в буферный кеш «как есть», без преобразований.

Поэтому файлы данных на одной платформе (разрядность, порядок байтов и т. п.) оказываются несовместимыми с другими платформами.

Кроме того, многие архитектуры предусматривают выравнивание данных по границам машинных слов. Например, на 32-битной системе x86 целые числа (тип `integer`, занимает 4 байта) будут выровнены по границе 4-байтных слов, как и числа с плавающей точкой двойной точности (тип `double precision`, 8 байт). А в 64-битной системе значения `double precision` будут выровнены по границе 8-байтных слов.

Из-за этого размер табличной строки зависит от порядка расположения и типов полей. Обычно этот эффект не сильно заметен, но в некоторых случаях он может привести к существенному увеличению размера.

Например, если располагать поля типов `boolean` и `integer` попеременно, между ними, как правило, будет пропадать 3 байта.

<https://pgconf.ru/media/2016/05/13/tuple-internals-ru.pdf>

## Структура страниц

Для изучения структуры и содержания страниц предназначено расширение pageinspect.

```
=> CREATE DATABASE mvcc_tuples;
```

```
CREATE DATABASE
```

```
=> \c mvcc_tuples
```

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

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

```
CREATE EXTENSION
```

Границы областей страницы записаны в ее заголовке. Возьмем для примера нулевую страницу одной из таблиц системного каталога:

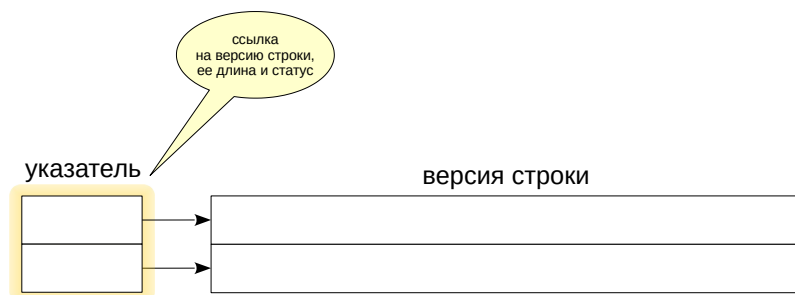
```
=> SELECT lower, upper, special, pagesize
FROM page_header(get_raw_page('pg_class',0));
```

lower	upper	special	pagesize
212	7744	8192	8192

(1 row)

Области занимают следующие диапазоны адресов:

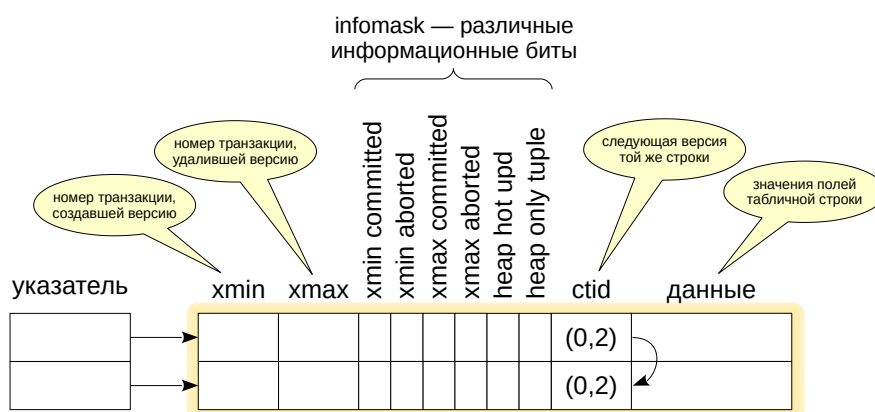
- 0 — начало заголовка страницы и указатели на версии строк,
- lower — начало свободного места,
- upper — начало данных (версий строк),
- special — начало спец. данных (только для индексов),
- pagesize — конец страницы.



Страница содержит массив указателей на версии строк.

Каждый указатель (занимающий 4 байта) содержит:

- ссылку на версию строки;
- длину этой версии строки (для удобства);
- несколько бит, определяющих статус версии строки.

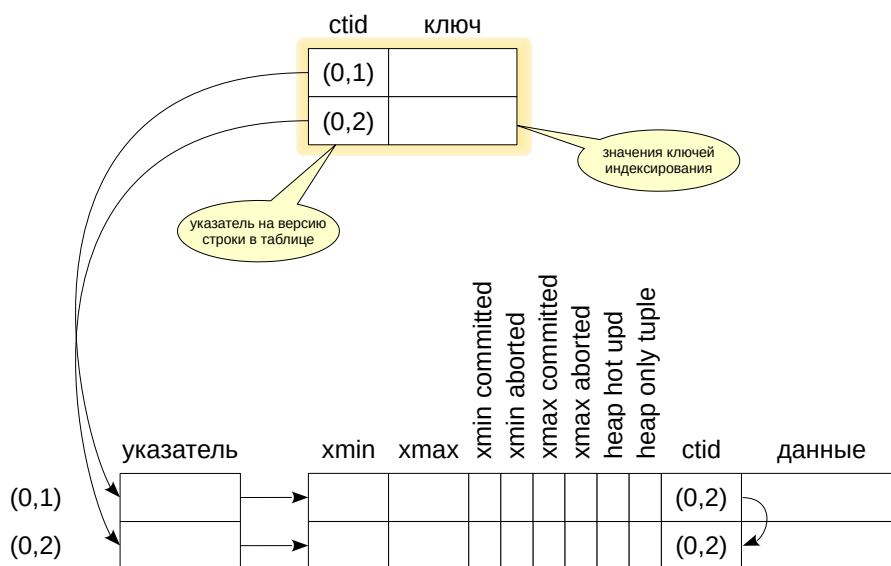


Версии строк (tuples) в табличных страницах (heap pages) кроме собственно данных также имеют заголовок. Этот заголовок, помимо прочего, содержит следующие важные поля:

- **xmin** и **xmax** определяют видимость данной версии строки в терминах начального и конечного номеров транзакций. Номера транзакций (TransactionId или xid) последовательно выбираются для транзакций из глобального счетчика, который используется всеми базами данных в рамках кластера PostgreSQL.
- **infomask** содержит ряд битов, определяющих свойства данной версии. На рисунке показаны основные из них, но далеко не все. Часть показанных битов будет рассмотрена в этой теме, часть — в других темах этого модуля.
- **ctid** является ссылкой на следующую, более новую, версию той же строки. У самой новой, *актуальной*, версии строки ctid ссылается на саму эту версию. Такие ссылки используются не всегда, мы рассмотрим их в теме «HOT-обновления и самоочистка».

Заголовок версии строки на табличной странице составляет 23 байта (или больше: в него включается битовая карта неопределенных значений).

Напомним, что каждая версия строки всегда целиком помещается внутри одной страницы. Если версия строки имеет большой размер, PostgreSQL попытается сжать часть полей или вынести часть полей во внешнее TOAST-хранилище (это рассматривается в модуле «Организация данных» курса DBA1).



Информация в индексной странице сильно зависит от типа индекса. И даже у одного типа индекса бывают разные виды страниц. Например, у В-дерева есть страница с метаданными и «обычные» страницы.

Тем не менее, обычно в странице имеется массив указателей и строки (так же, как и в табличной странице). Во избежание путаницы мы будем называть индексные строки *записями*. Кроме того, в конце индексной страницы отводится место под специальные данные.

Сами индексные записи тоже могут иметь очень разную структуру в зависимости от типа индекса. Например, для В-дерева записи, относящиеся к листовым страницам, содержат значение ключа индексирования и идентификатор (`ctid`) строки таблицы (подробно структура В-дерева и другие индексы разбираются в учебном курсе QRT «Оптимизация запросов»).

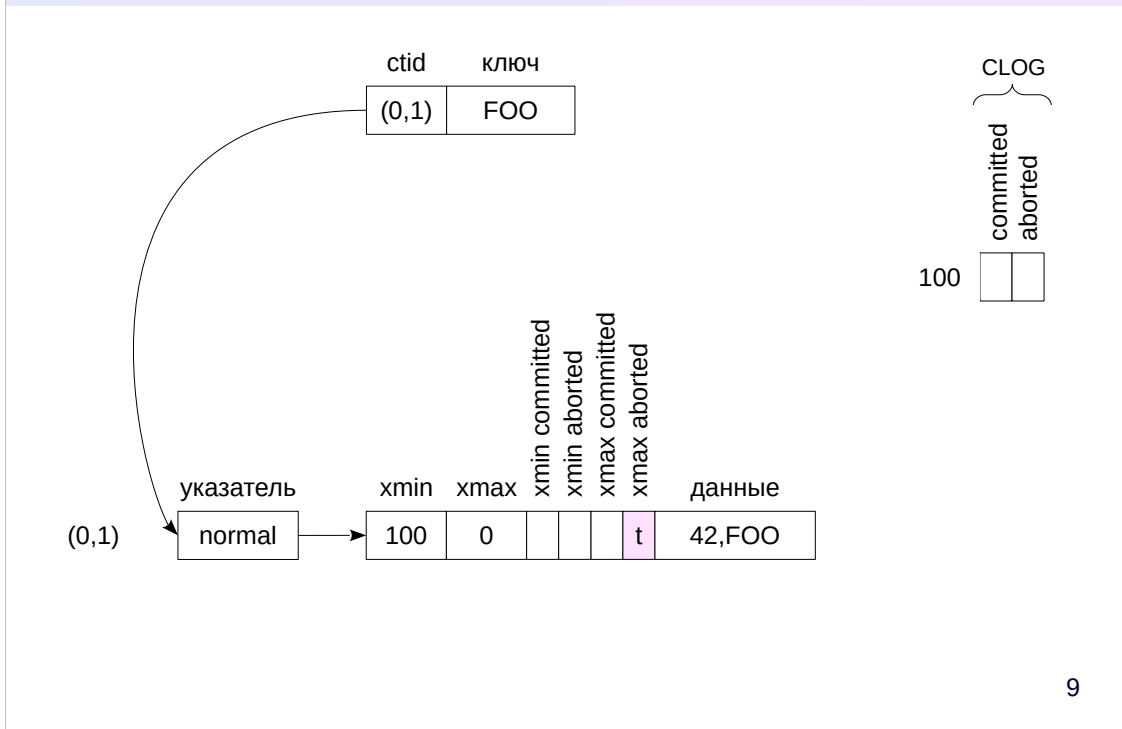
Устройство индексов других типов может быть разным, но, как правило, страницы индекса все равно содержат ссылки на версии строк.

Идентификаторы `ctid` имеют вид  $(x,y)$ : здесь  $x$  — номер страницы,  $y$  — порядковый номер указателя в массиве. Для удобства мы будем показывать их слева от указателей на табличные версии строк.

Никакой индекс не содержит информацию о версионности (в нем нет полей `xmin` и `xmax`). Прочитав только индексную запись, невозможно определить видимость строки, на которую она ссылается: приходится обращаться к табличной странице или карте видимости.

На рисунке показаны записи листовых страниц обычного индекса — В-дерева. Для простоты указатели на эти записи опущены.





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 В-дерево.

При вставке строки в табличной странице появится указатель с номером 1, ссылающийся на первую и единственную версию строки. В версии строки поле xmin заполнено номером текущей транзакции (100 в нашем примере).

В журнале статусов транзакций (CLOG) хранятся состояния всех транзакций (начиная с некоторой), для каждой транзакции отведено два бита. Эти данные хранятся в каталоге PGDATA/pg\_xact, а несколько наиболее актуальных страниц кешируются в разделяемой памяти сервера. Поскольку в нашем примере транзакция 100 еще активна, оба бита пока сброшены.

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

Поле xmax заполнено фиктивным номером 0, поскольку данная версия строки не удалена и является актуальной. Транзакции не будут обращать внимание на этот номер, поскольку установлен бит xmax aborted.

## Вставка

Создадим таблицу и индекс:

```
=> CREATE TABLE t(  
    n integer,  
    s text  
);  
  
CREATE TABLE  
  
=> CREATE INDEX t_s on t(s);  
  
CREATE INDEX
```

Для удобства создадим представление, которое с помощью расширения pageinspect покажет интересующую нас информацию о версиях строк из нулевой страницы таблицы:

```
=> 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 as xmin,  
    t_xmax as xmax,  
    CASE WHEN (t_infomask & 256) > 0 THEN 't' END AS xmin_c,  
    CASE WHEN (t_infomask & 512) > 0 THEN 't' END AS xmin_a,  
    CASE WHEN (t_infomask & 1024) > 0 THEN 't' END AS xmax_c,  
    CASE WHEN (t_infomask & 2048) > 0 THEN 't' END AS xmax_a  
FROM heap_page_items(get_raw_page('t',0))  
ORDER BY lp;  
  
CREATE VIEW
```

Также создадим представление, чтобы заглянуть в индекс. Нулевая страница индекса содержит метаданные, поэтому смотрим в первую:

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

Вставим одну строку, предварительно начав транзакцию.

```
=> BEGIN;  
  
BEGIN  
  
=> INSERT INTO t VALUES (42, 'F00');  
  
INSERT 0 1
```

Вот номер нашей текущей транзакции и ее статус:

```
=> SELECT pg_current_xact_id(); -- txid_current() до версии 13  
  
pg_current_xact_id  
-----  
749  
(1 row)  
  
=> SELECT pg_xact_status('749');  
  
pg_xact_status  
-----  
in progress  
(1 row)
```

Вот что содержится в табличной странице:

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

ctid	state	xmin	xmax	xmin_c	xmin_a	xmax_c	xmax_a
(0,1)	normal	749	0				t

(1 row)

Похожую, но существенно менее детальную информацию можно получить и из самой таблицы, используя псевдостолбцы ctid, xmin и xmax:

=> **SELECT** ctid, xmin, xmax, \* **FROM** t;

ctid	xmin	xmax	n	s
(0,1)	749	0	42	F00

(1 row)

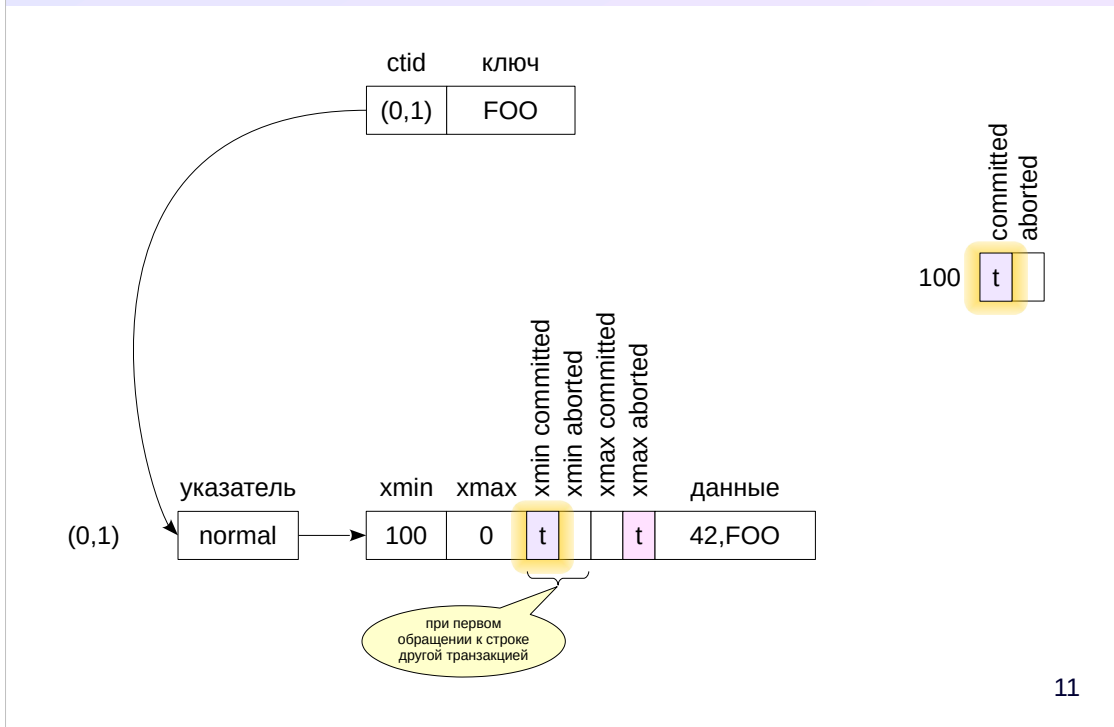
В индексной странице видим один указатель на единственную строку таблицы:

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

itemoffset	ctid
1	(0,1)

(1 row)

# Фиксация изменений



11

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

Когда какая-либо другая транзакция обратится к этой табличной странице, ей придется ответить на вопросы:

- 1) завершилась ли транзакция 100 (надо проверить список активных транзакций — такая структура поддерживается в общей памяти сервера),
- 2) а если завершилась, то фиксацией или отменой (свериться с CLOG).

Поскольку выполнять проверку по CLOG каждый раз накладно, выясненный однажды 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? В момент, когда транзакция фиксируется или отменяется, уже непонятно, какие именно строки в каких именно страницах транзакция успела поменять. Кроме того, часть этих страниц может быть вытеснена из буферного кеша на диск; читать их заново, чтобы изменить биты, означало бы существенно замедлить фиксацию.

Обратная сторона состоит в том, что любая транзакция (даже выполняющая простое чтение — SELECT) может загрязнить данные в буферном кеше и породить новые журнальные записи.

## Фиксация изменений

Выполним фиксацию:

```
=> COMMIT;
```

COMMIT

Что изменилось в странице?

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

ctid	state	xmin	xmax	xmin_c	xmin_a	xmax_c	xmax_a
(0,1)	normal	749	0				t

(1 row)

Ничего, так как единственная операция, которая выполняется при фиксации — запись статуса транзакции в CLOG.

```
=> SELECT pg_xact_status('749');
```

```
pg_xact_status
-----
committed
(1 row)
```

Информация о статусах транзакций хранится в подкаталоге pg\_xact каталога PGDATA и кешируется в общей памяти. Начиная с PostgreSQL 13, статистику использования кешей, в том числе кеша статусов транзакций, показывает представление pg\_stat\_slru:

```
=> SELECT name, blks_hit, blks_read, blks_written
FROM pg_stat_slru WHERE name = 'Xact';
```

name	blks_hit	blks_read	blks_written
Xact	3841	4	7

(1 row)

Транзакция, первой обратившаяся к странице, должна будет определить статус транзакции xmin. Этот статус будет записан в информационные биты:

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

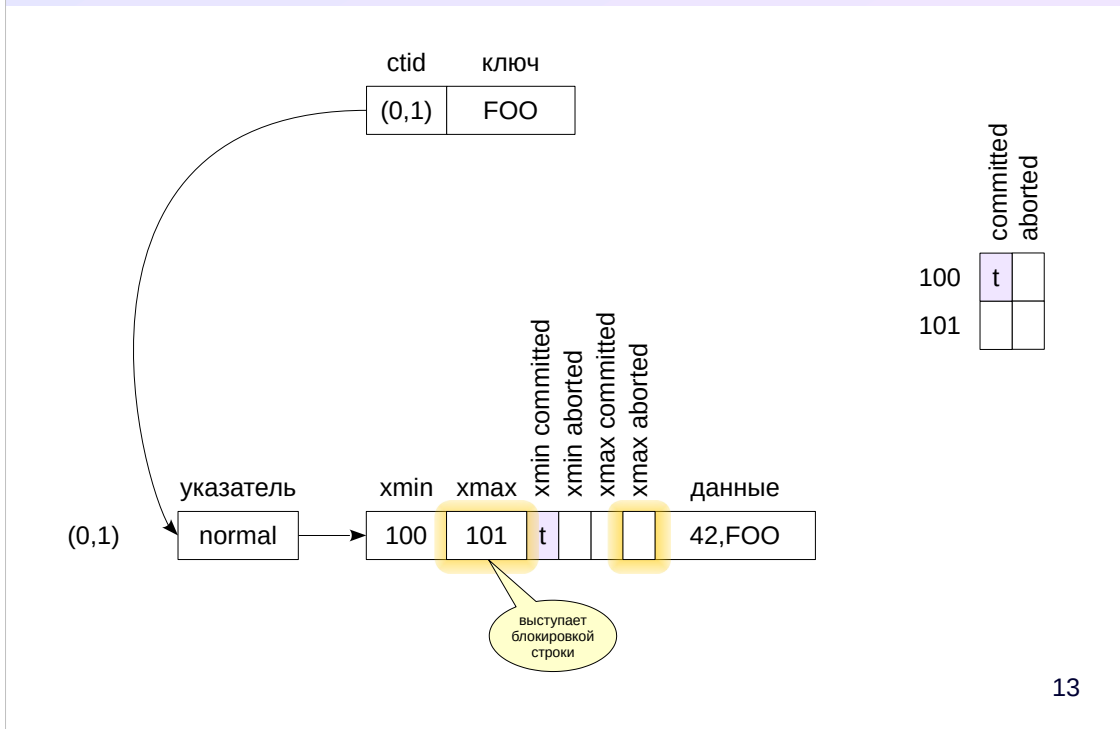
n	s
42	F00

(1 row)

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

ctid	state	xmin	xmax	xmin_c	xmin_a	xmax_c	xmax_a
(0,1)	normal	749	0	t			t

(1 row)



При удалении строки в поле xmax текущей версии записывается номер текущей удаляющей транзакции, а бит xmax aborted сбрасывается. Больше ничего не происходит.

Заметим, что установленное значение xmax, соответствующее активной транзакции (что определяется другими транзакциями по списку активных), выступает в качестве признака блокировки. Если другая транзакция намерена обновить или удалить эту строку, она будет вынуждена дожидаться завершения транзакции xmax.

Подробнее блокировки рассматриваются в одноименном модуле. Пока отметим только, что число блокировок строк ничем не ограничено. Они не занимают место в оперативной памяти и производительность системы не страдает от их количества (разумеется, за исключением того, что первый процесс, обратившийся к странице, должен будет проставить биты-подсказки).

## Удаление

Теперь удалим строку.

=> **BEGIN;**

BEGIN

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

DELETE 1

Номер транзакции записался в поле xтах:

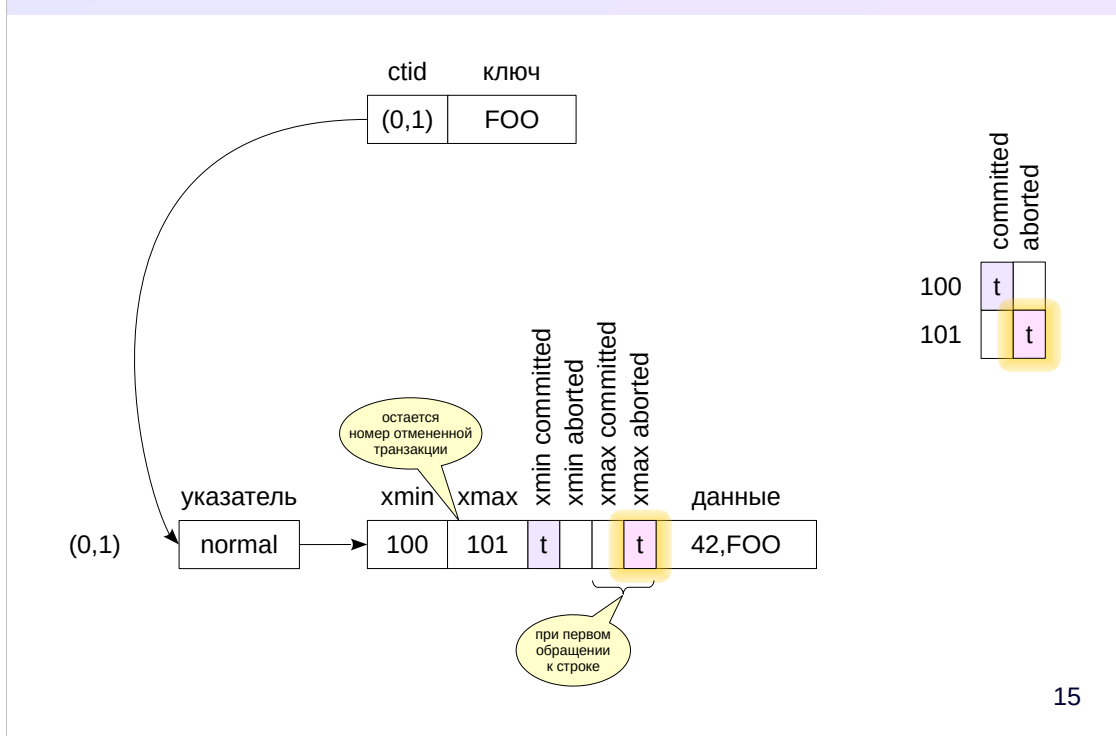
```
=> SELECT * FROM t_v;
```

```

ctid | state | xmin | xmax | xmin_c | xmin_a | xmax_c | xmax_a
-----+-----+-----+-----+-----+-----+-----+-----
(0,1) | normal | 749 | 750 | t       |         |         |
(1 row)

```

# Отмена изменений



15

Отмена изменений работает аналогично фиксации, только в CLOG для транзакции выставляется бит aborted. Отмена выполняется так же быстро, как и фиксация — не требуется выполнять откат выполненных действий.

Номер прерванной транзакции остается в поле xmax — его можно было бы стереть, но в этом нет смысла. При обращении к странице будет проверен статус и в версию строки будет установлен бит подсказки xmax aborted. Это будет означать, что на поле xmax смотреть не нужно.



## Отмена изменений

При обрыве транзакции номер xmax остается в заголовке.

```
=> ROLLBACK;
```

```
ROLLBACK
```

```
=> SELECT pg_xact_status('750');
```

```
pg_xact_status
-----
aborted
(1 row)
```

```
=> SELECT * from t_v;
```

ctid	state	xmin	xmax	xmin_c	xmin_a	xmax_c	xmax_a
(0,1)	normal	749	750	t			

```
(1 row)
```

А при первом обращении к странице выставляется соответствующий бит:

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

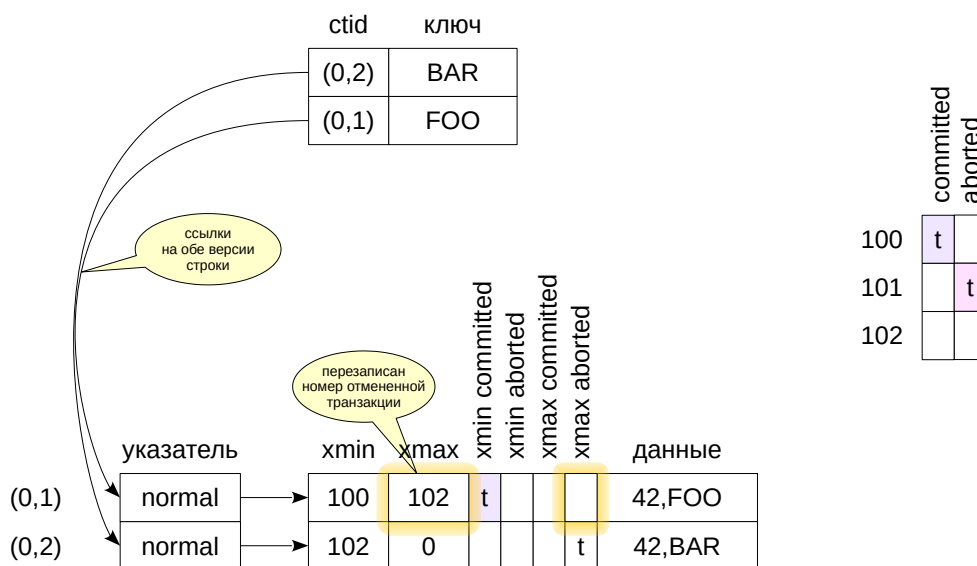
n	s
42	F00

```
(1 row)
```

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

ctid	state	xmin	xmax	xmin_c	xmin_a	xmax_c	xmax_a
(0,1)	normal	749	750	t			t

```
(1 row)
```



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 отменена. Кроме того, биты xmax committed и xmax aborted старой версии строки сброшены, так как статус текущей транзакции еще не известен.

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

Так же, как и при удалении, значение xmax в первой версии строки служит признаком того, что строка заблокирована.

## Обновление

Теперь проверим обновление.

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

```
UPDATE 1
```

Запрос выдает одну строку (новую версию):

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

n	s
42	BAR

(1 row)

Но в странице мы видим обе версии. Причем новый номер транзакции записался на место старого (поскольку старая транзакция была отменена).

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

ctid	state	xmin	xmax	xmin_c	xmin_a	xmax_c	xmax_a
(0,1)	normal	749	751	t		t	
(0,2)	normal	751	0	t			t

(2 rows)

При этом в индексной странице обнаруживаем указатели на обе версии:

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

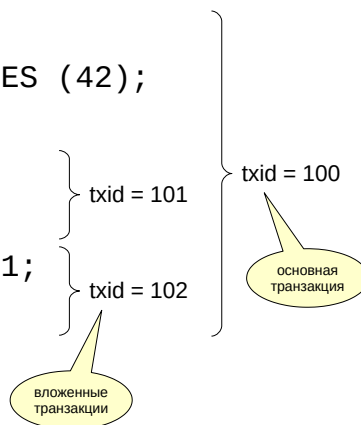
itemoffset	ctid
1	(0,2)
2	(0,1)

(2 rows)

Индексные записи внутри страницы упорядочены по значению ключа. Поэтому первой идет запись с ключом BAR (ссылается на версию (0,2)), а второй — запись с ключом FOO (ссылается на версию (0,1)).

## Возможность откатить часть транзакции

```
BEGIN;  
INSERT INTO t(n) VALUES (42);  
SAVEPOINT SP;  
DELETE FROM t;  
ROLLBACK TO SP;  
UPDATE t SET n = n + 1;  
COMMIT;
```



	committed	aborted
100	t	
101		t
102	t	

Тонкий момент представляет функционал точек сохранения, позволяющий отменить часть операций текущей транзакции. Это не укладывается в приведенную выше схему, поскольку физически никакие данные не откатываются, а лишь изменяется статус всей транзакции целиком.

Поэтому транзакция с точкой сохранения состоит из отдельных вложенных (не путать с автономными!) транзакций (subtransactions), статусом которых можно управлять отдельно.

## Собственный номер и статус в CLOG

конечный статус зависит от статуса основной транзакции

## Информация о вложенности сохраняется на диске

каталог PGDATA/pg\_subtrans

данные кешируются в буферах общей памяти (аналогично CLOG)

## Примеры использования

точка сохранения SAVEPOINT

обработка исключений в PL/pgSQL (EXCEPTION)

режим psql ON\_ERROR\_ROLLBACK = on/interactive

Вложенные транзакции имеют свой номер (бóльший, чем номер основной транзакции). Статус вложенных транзакций записывается обычным образом в CLOG, однако финальный статус зависит от статуса основной транзакции: если она отменена, то отменяются также и все вложенные транзакции.

Информация о вложенности транзакций хранится в каталоге PGDATA/pg\_subtrans. Обращение к файлам происходит через буферы в общей памяти сервера, организованные так же, как и буферы CLOG.

Вложенные транзакции нельзя использовать явно, то есть нельзя начать новую транзакцию, не завершив текущую. Этот механизм задействуется неявно при использовании точек сохранения, при обработке исключений PL/pgSQL и т. п.

Особенный интерес представляет режим ON\_ERROR\_ROLLBACK в psql, при включении которого транзакция, выполнившая ошибочную операцию, не прерывается, а продолжает работать. Почему этот режим не включен по умолчанию? Дело в том, что ошибка может произойти где-то в середине выполнения оператора, и таким образом нарушится атомарность выполнения оператора. Единственный способ отменить изменения, уже сделанные этим оператором, не трогая остальные изменения — использовать вложенные транзакции. Поэтому режим ON\_ERROR\_ROLLBACK фактически ставит перед каждой командой неявную точку сохранения. А это чревато существенными накладными расходами.

## Точки сохранения и вложенные транзакции

Опустошим таблицу (при этом опустошаются файлы таблицы и индекса):

```
=> TRUNCATE t;
```

TRUNCATE TABLE

Начинаем транзакцию и вставляем строку.

```
=> BEGIN;
```

BEGIN

```
=> INSERT INTO t(n) VALUES (42)
RETURNING *, ctid, xmin, xmax;
```

n	s	ctid	xmin	xmax
42		(0,1)	753	0

(1 row)

INSERT 0 1

Ставим точку сохранения и удаляем строку.

```
=> SAVEPOINT sp;
```

SAVEPOINT

```
=> DELETE FROM t RETURNING *, ctid, xmin, xmax;
```

n	s	ctid	xmin	xmax
42		(0,1)	753	754

(1 row)

DELETE 1

Обратите внимание: функция `pg_current_xact_id()` выдает номер основной, а не вложенной, транзакции:

```
=> SELECT pg_current_xact_id();
```

pg_current_xact_id
753

(1 row)

Откатимся к точке сохранения. Версии строк в таблице остаются на месте, но изменится статус вложенной транзакции:

```
=> ROLLBACK TO sp;
```

ROLLBACK

```
=> SELECT pg_xact_status('753') xid,
pg_xact_status('754') subxid;
```

xid	subxid
in progress	aborted

(1 row)

Запрос к таблице снова покажет строку — ее удаление было отменено:

```
=> SELECT *, ctid, xmin, xmax FROM t;
```

n	s	ctid	xmin	xmax
42		(0,1)	753	754

(1 row)

Для дальнейших изменений создается новая вложенная транзакция. Ее номер будет записан в поле `xmax` первой версии строки вместо номера отмененной вложенной транзакции 754 и в поле `xmin` новой версии:

```
=> UPDATE t SET n = n + 1
RETURNING *, ctid, xmin, xmax;
```

n	s	ctid	xmin	xmax
43		(0,2)	755	0

(1 row)

UPDATE 1

```
=> SELECT pg_xact_status('753') xid,
         pg_xact_status('754') subxid1,
         pg_xact_status('755') subxid2;
```

xid	subxid1	subxid2
in progress	aborted	in progress

(1 row)

Фиксируем изменения. При этом в таблице, как и прежде, ничего не меняется:

```
=> COMMIT;
```

COMMIT

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

ctid	state	xmin	xmax	xmin_c	xmin_a	xmax_c	xmax_a
(0,1)	normal	753	755				
(0,2)	normal	755	0				t

(2 rows)

А в CLOG основная транзакция и все вложенные, которые еще не завершены, получают статус committed:

```
=> SELECT pg_xact_status('753') xid,
         pg_xact_status('754') subxid1,
         pg_xact_status('755') subxid2;
```

xid	subxid1	subxid2
committed	aborted	committed

(1 row)

Информация о вложенности транзакций хранится в подкаталоге pg\_subtrans каталога PGDATA. Она кешируется в общей памяти, как и информация о статусах транзакций:

```
=> SELECT name, blks_hit, blks_read, blks_written
FROM pg_stat_slru WHERE name = 'Subtrans';
```

name	blks_hit	blks_read	blks_written
Subtrans	2	0	5

(1 row)

В табличных страницах может храниться несколько версий одной и той же строки, ограниченных номерами транзакций  $x_{min}$  и  $x_{max}$

В индексных записях нет информации о версии

Фиксация и откат выполняются одинаково быстро

Для точек сохранения используются вложенные транзакции



1. Создайте таблицу и вставьте в нее одну строку. Затем дважды обновите эту строку и удалите ее. Сколько версий строк находится сейчас в таблице?  
Проверьте, используя расширение `pageinspect`.
2. Определите, в какой странице находится строка таблицы `pg_class`, относящаяся к самой таблице `pg_class`. Сколько актуальных версий строк находится в той же странице?
3. Включите в `psql` параметр `ON_ERROR_ROLLBACK` и убедитесь, что этот режим использует вложенные транзакции.

## 1. Версии строк

Создаем расширение и таблицу.

```
=> CREATE DATABASE mvcc_tuples;
```

```
CREATE DATABASE
```

```
=> \c mvcc_tuples
```

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

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

```
CREATE EXTENSION
```

```
=> CREATE TABLE t(s text);
```

```
CREATE TABLE
```

Вставляем строку, обновляем ее и затем удаляем:

```
=> INSERT INTO t VALUES ('FOO');
```

```
INSERT 0 1
```

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

```
UPDATE 1
```

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

```
UPDATE 1
```

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

```
DELETE 1
```

В таблице ничего нет:

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

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

Проверяем версии в странице:

```
=> SELECT '(0, ' || lp || ')' AS ctid,
        t_xmin as xmin,
        t_xmax as xmax,
        CASE WHEN (t_infomask & 256) > 0 THEN 't' END AS xmin_c,
        CASE WHEN (t_infomask & 512) > 0 THEN 't' END AS xmin_a,
        CASE WHEN (t_infomask & 1024) > 0 THEN 't' END AS xmax_c,
        CASE WHEN (t_infomask & 2048) > 0 THEN 't' END AS xmax_a
FROM heap_page_items(get_raw_page('t', 0))
ORDER BY lp;
```

ctid	xmin	xmax	xmin_c	xmin_a	xmax_c	xmax_a
(0,1)	746	747	t		t	
(0,2)	747	748	t		t	
(0,3)	748	749	t		t	

(3 rows)

## 2. Версии строк на определенной странице

Номер страницы содержится в первой компоненте значения ctid:

```
=> SELECT ctid FROM pg_class WHERE relname = 'pg_class';
```

```
ctid
-----
(7,65)
(1 row)
```

К сожалению, тип данных tid не позволяет непосредственно получить номер страницы, но можно, например,

воспользоваться приведением к типу point:

```
=> SELECT (ctid::text::point)[0]::integer FROM pg_class WHERE relname = 'pg_class';

 ctid
-----
      7
(1 row)
```

Количество строк на той же странице:

```
=> SELECT count(*)
FROM pg_class
WHERE (ctid::text::point)[0]::integer = (
  SELECT (ctid::text::point)[0]::integer FROM pg_class WHERE relname = 'pg_class'
);

 count
-----
      12
(1 row)
```

### 3. Режим ON\_ERROR\_ROLLBACK

Включим режим:

```
=> \set ON_ERROR_ROLLBACK on
```

Начнем транзакцию и вставим строку.

```
=> BEGIN;

BEGIN

=> INSERT INTO t VALUES ('FOO')
RETURNING s, xmin, pg_current_xact_id();

 s | xmin | pg_current_xact_id
---+-----+-----
FOO | 751 | 750
(1 row)
```

```
INSERT 0 1
```

Вставим еще одну строку.

```
=> INSERT INTO t VALUES ('BAR')
RETURNING s, xmin, pg_current_xact_id();

 s | xmin | pg_current_xact_id
---+-----+-----
BAR | 752 | 750
(1 row)
```

```
INSERT 0 1
```

Каждая команда происходит в отдельной вложенной транзакции, что и требовалось установить.

```
=> COMMIT;
```

```
COMMIT
```

Установленный режим будет действовать вплоть до завершения сеанса psql. А чтобы эта настройка не помешала дальнейшей работе в этом же сеансе, вернем прежний режим:

```
=> \set ON_ERROR_ROLLBACK off
```