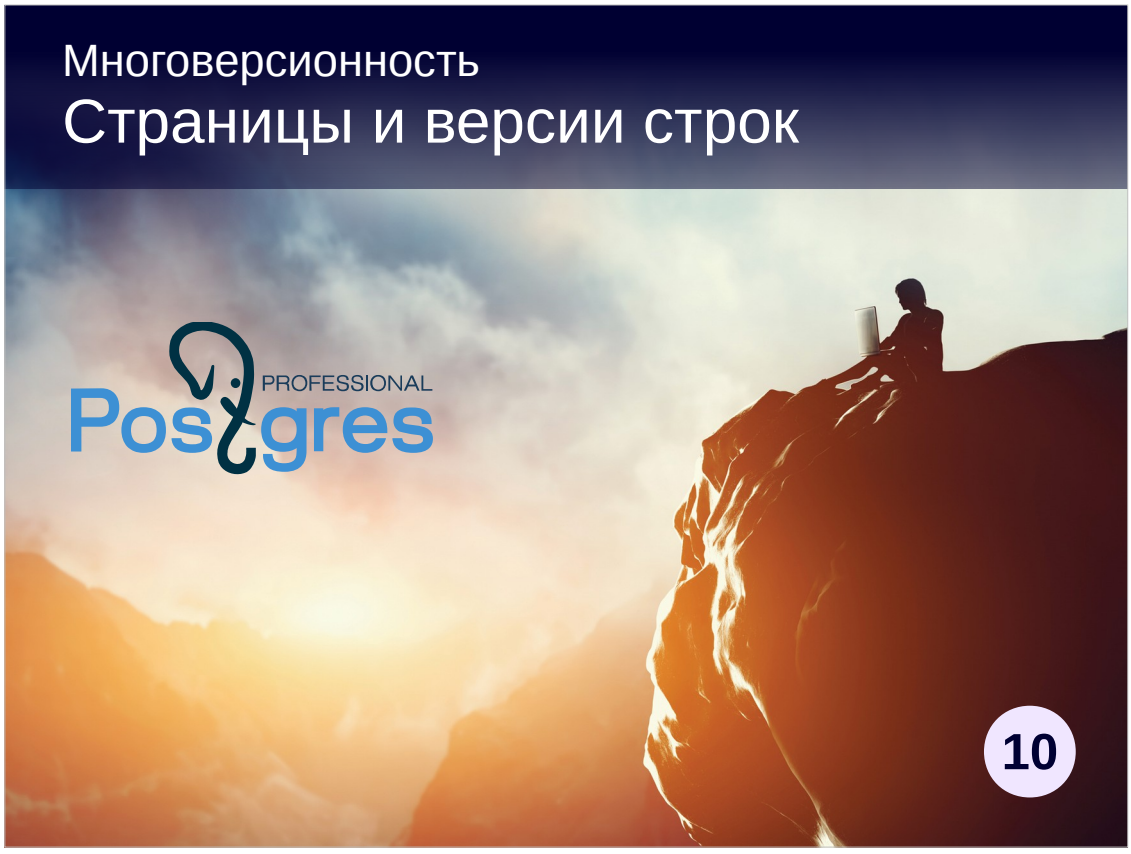


Авторы: Егор Рогов, Павел Лузанов

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

Темы

Структура страниц и версий строк

Как работают операции над данными

Вложенные транзакции

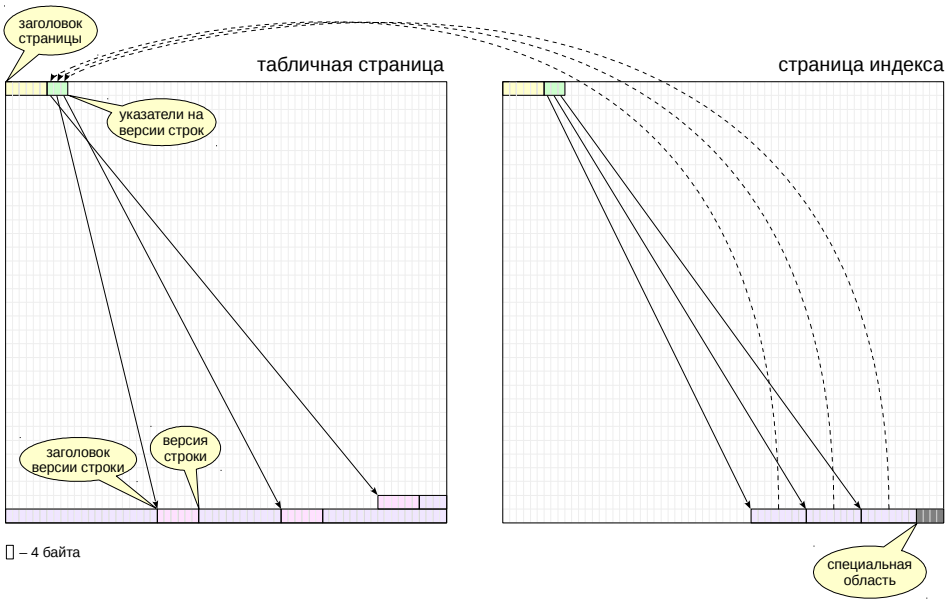
Структура данных

Структура страницы

Структура версии строки

Формат данных

Страницы



4

Размер страницы составляет 8 килобайт. Это значение можно увеличить (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 «строка».

Указатели имеют фиксированный размер (4 байта) и составляют массив, позиция в котором определяет идентификатор строки (tuple id, tid). Указатели ссылаются на собственно версии строк (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, позволяет перемещать строку внутри блока, не ломая ссылки из индексов.

Версия строки, в свою очередь, имеет заголовок и данные.

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

https://postgrespro.ru/docs/postgresql/10/storage-page-layout

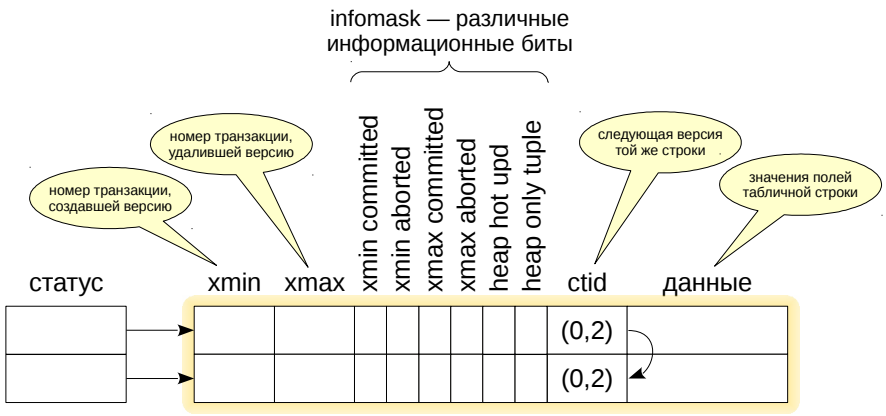
Указатели на версии строк

Страница содержит массив указателей на версии строк.

Каждый указатель (занимающий 4 байта) содержит:

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

Версии строк в таблице



Версии строк (tuples) в табличных страницах (heap pages) 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.

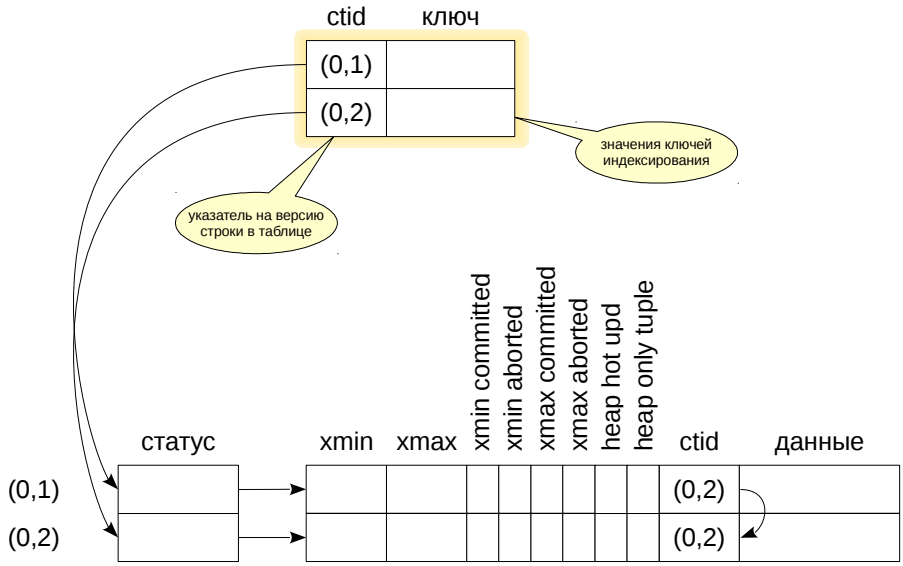
- **infomask** содержит ряд битов, определяющих свойства данной версии. На рисунке показаны основные из них, но далеко не все. Часть показанных битов будет рассмотрена в этой теме, часть — в других темах этого модуля.

- **ctid** является ссылкой на следующую, более новую, версию той же строки. У самой новой, актуальной, версии строки ctid ссылается на саму эту версию.

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 внутрь одной страницы. Если версия строки имеет большой размер, PostgresSQL попробует сжать часть полей или вынести часть полей во внешнее TOAST-хранилище (это рассматривается в модуле «Физическая организация данных» курса DBA1).

Строки в индексе



Информация в индексной странице сильно зависит от типа индекса. И даже у одного типа индекса бывают разные виды страниц. Например, у B-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). Кроме того, в конце страницы отводится место под специальные данные.

Строки в индексах тоже могут иметь очень разную структуру в зависимости от типа индекса. Например, для B-дерева строки, относящиеся к листовым страницам, содержат значение ключа индексирования и ссылку (ctid) на соответствующую строку таблицы (подробно структура B-дерева разбирается в учебном курсе «Оптимизация запросов»).

В общем случае индекс может быть устроен совсем другим образом, но как правило он все равно будет содержать ссылки на версии строк.

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

Важный момент состоит в том, что никакой индекс не содержит информацию о версионности (нет полей xmin и xmax). Прочитав строку индекса, невозможно определить видимость этой строки, не заглянув в табличную страницу (для оптимизации служит карта видимости).

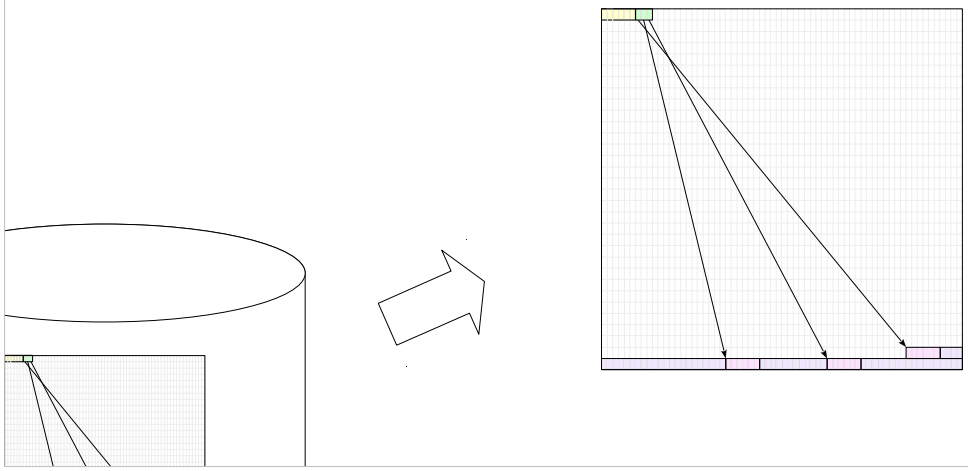
На рисунке показаны строки обычного индекса-B-дерева. Для простоты указатели на эти строки опущены.

Формат данных

Страницы читаются в оперативную память «как есть»

данные не переносимы между разными платформами

между полями данных возможны пропуски из-за выравнивания



8

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

Поэтому файлы данных на одной платформе (разрядность, порядок байтов и т. п.) не совместимы 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 будут выровнены по границе 8-байтных слов.

Из-за этого размер табличной строки зависит от порядка расположения полей. Обычно этот эффект не сильно заметен, но в некоторых случаях он может привести к существенному увеличению размера. Например, если располагать поля типов char(1) и integer вперемешку, между ними, как правило, будет пропадать 3 байта.

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

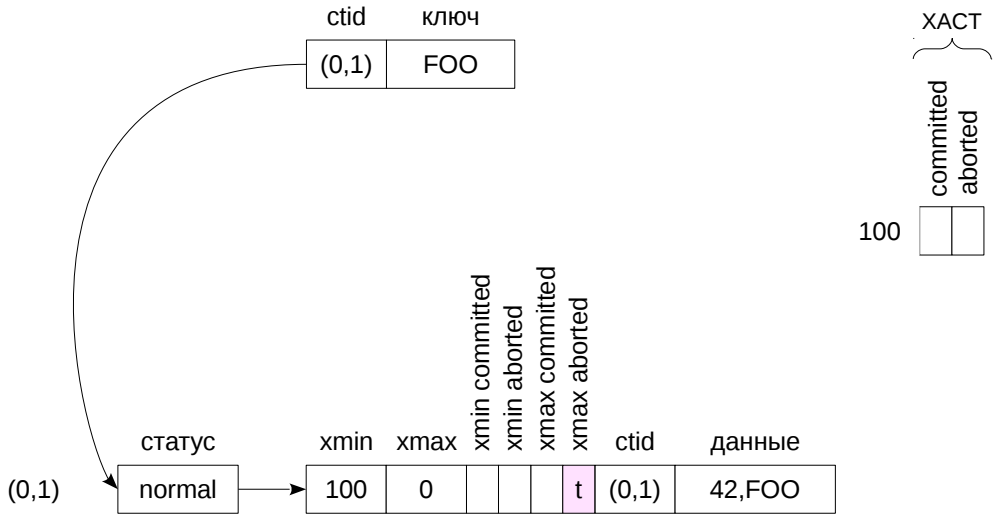
Выполнение операций

Вставка, обновление, удаление

Блокировка

Фиксация и отмена транзакций

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-дерево.

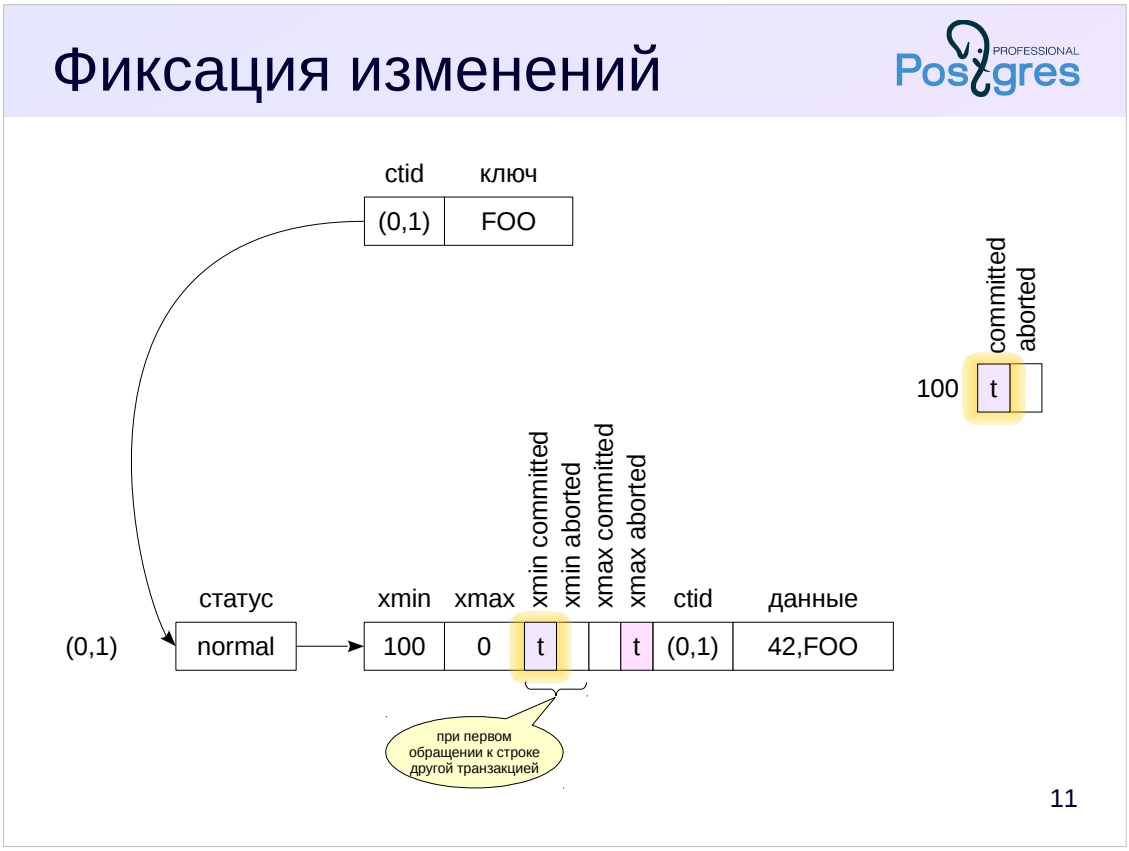
При вставке строки в табличной странице появится указатель с номером 1, ссылающийся на первую и единственную версию строки.

В версии строки поле xmin заполнено номером текущей транзакции (100 в нашем примере). Поскольку изменения еще не фиксировались и транзакция активна, то в журнале статуса транзакций (XACT) соответствующая запись заполнена нулями. XACT можно представить себе как массив, в котором для каждой транзакции (начиная с некоторой) отводится ровно два бита.

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.

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.



При фиксации изменений в XACT 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 процессов и их транзакций; такая структура в общей памяти имеет название ProcArray),

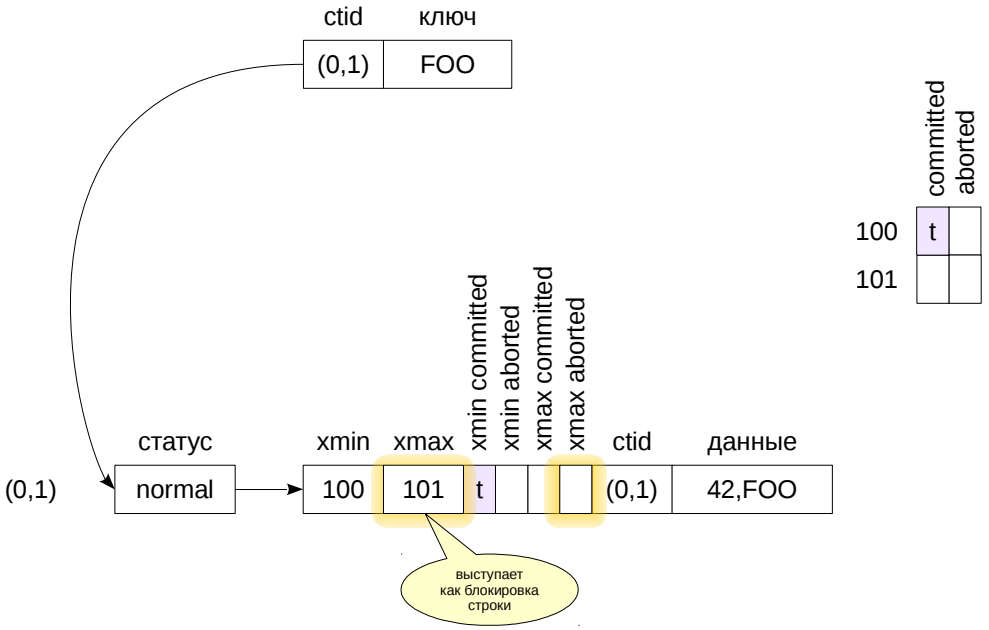
2. а если завершилась, то фиксацией или отменой (свериться с XACT).

Поскольку выполнять проверку по XACT 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 XACT.

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 кэша 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 кэше и породить новые журнальные записи.

Удаление

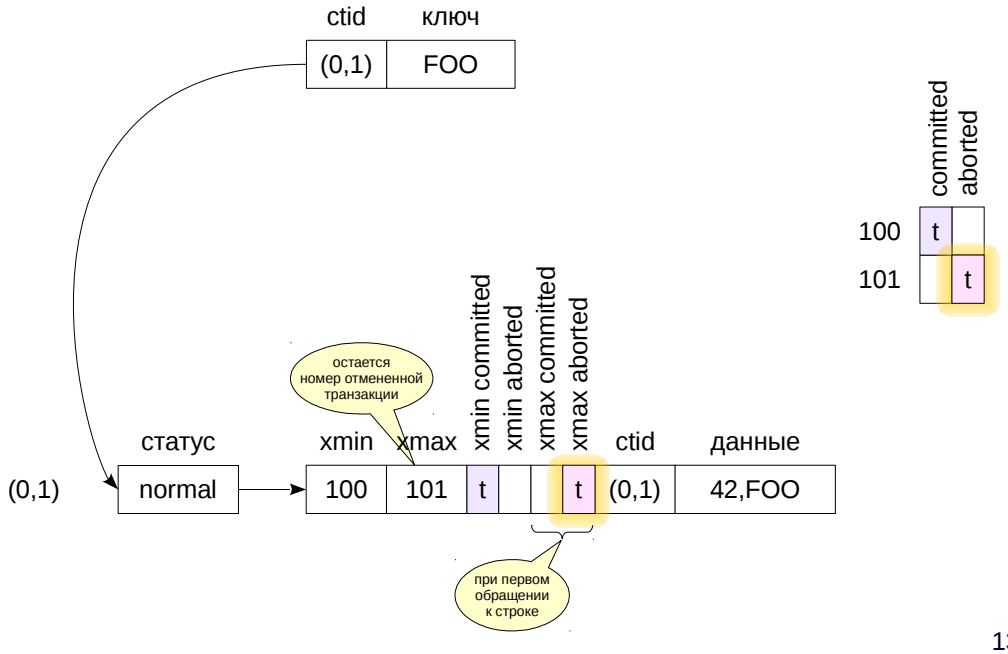


При удалении строки в поле xmax текущей версии записывается номер текущей удаляющей транзакции, а бит xmax aborted сбрасывается. Больше ничего не происходит.

Заметим, что установленное значение xmax, соответствующее активной транзакции (что определяется другими транзакциями по ProcArray), выступает в качестве блокировки. Если другая транзакция намерена обновить или удалить эту строку, она будет вынуждена дождаться 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, производительность системы не страдает от их количества (разумеется, за исключением того, что первый процесс, обратившийся к странице, должен будет проставить биты-подсказки).

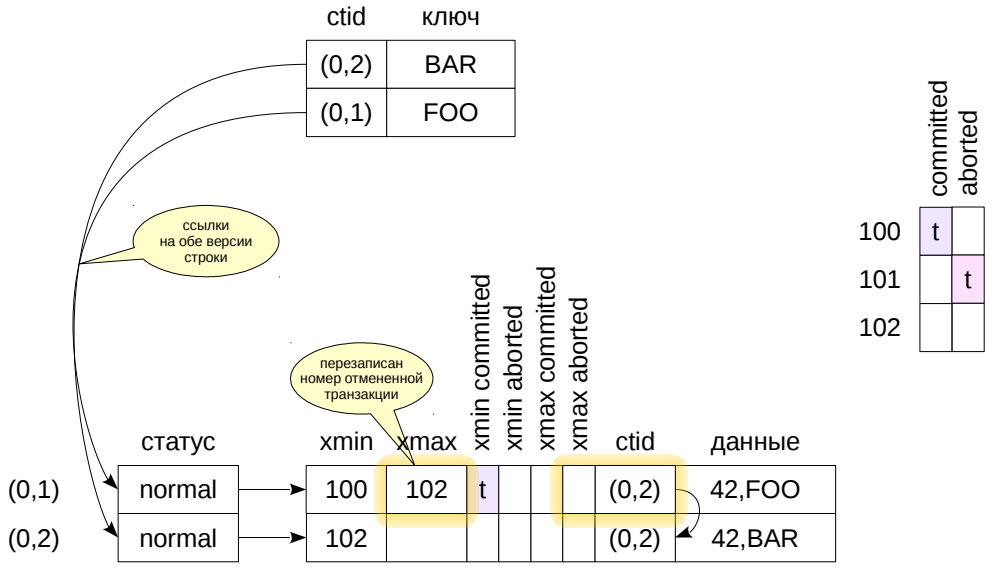
Отмена изменений



Отмена изменений работает аналогично фиксации, только в XACT 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.

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 отменена. 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.

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

Вложенные транзакции

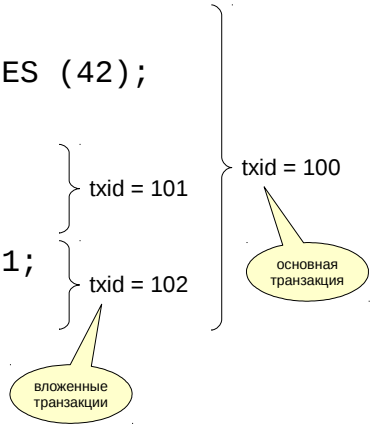
Точка сохранения и откат к ней

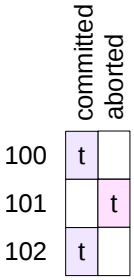
Вложенные транзакции как механизм реализации

Точка сохранения

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;
```





Тонкий момент представляет функционал точек сохранения, позволяющий отменить часть операцией 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.

Вложенные транзакции

Собственный номер и статус в XACT

- конечный статус зависит от статуса основной транзакции

Информация о вложенности сохраняется на диске

- каталог $PGDATA/pg_subtrans/
- данные кэшируются в буферах общей памяти (аналогично XACT)

Примеры использования

- точка сохранения SAVEPOINT
- обработка исключений в PL/pgSQL (EXCEPTION)
- режим psql ON_ERROR_ROLLBACK = on/interactive

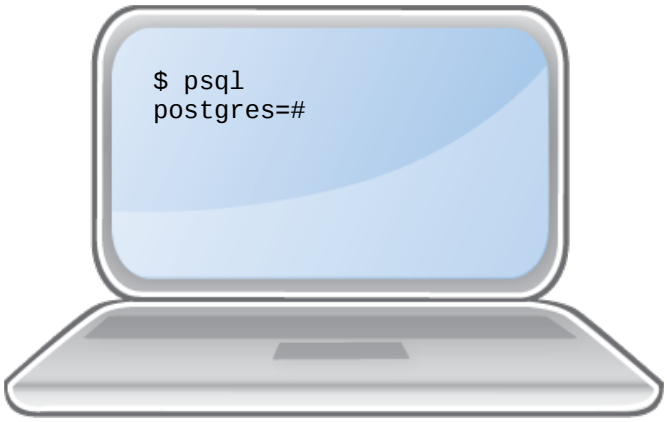
Вложенные транзакции имеют свой номер (бóльший, чем номер основной транзакции). Статус вложенных транзакций записывается обычным образом в XACT, однако финальный статус зависит от статуса основной транзакции: если она отменена, то отменяются также и все вложенные транзакции.

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 XACT.

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.

Демонстрация



Итоги

В табличных страницах может храниться несколько версий одной и той же строки, ограниченных номерами транзакций xmin и xmax

В индексных строках нет информации о версионности

Фиксация и откат выполняются одинаково быстро

Для точек сохранения используются вложенные транзакции

Практика

1. Внутри одной транзакции:

   – вставьте строку в таблицу;
   – создайте точку сохранения;
   – вставьте в таблицу еще одну строку;
   – откатите изменения до точки сохранения;
   – вставьте еще одну строку;
   – зафиксируйте изменения.

   При этом после каждой операции:

   – выведите номер текущей транзакции;
   – проверьте значения псевдостолбцов xmin и xmax;
   – проверьте содержимое страницы с помощью pageinspect.

Для выполнения задания потребуется установить расширение pageinspect.

Для удобного просмотра содержимого табличной страницы создайте представление аналогичное t_v из демонстрации.