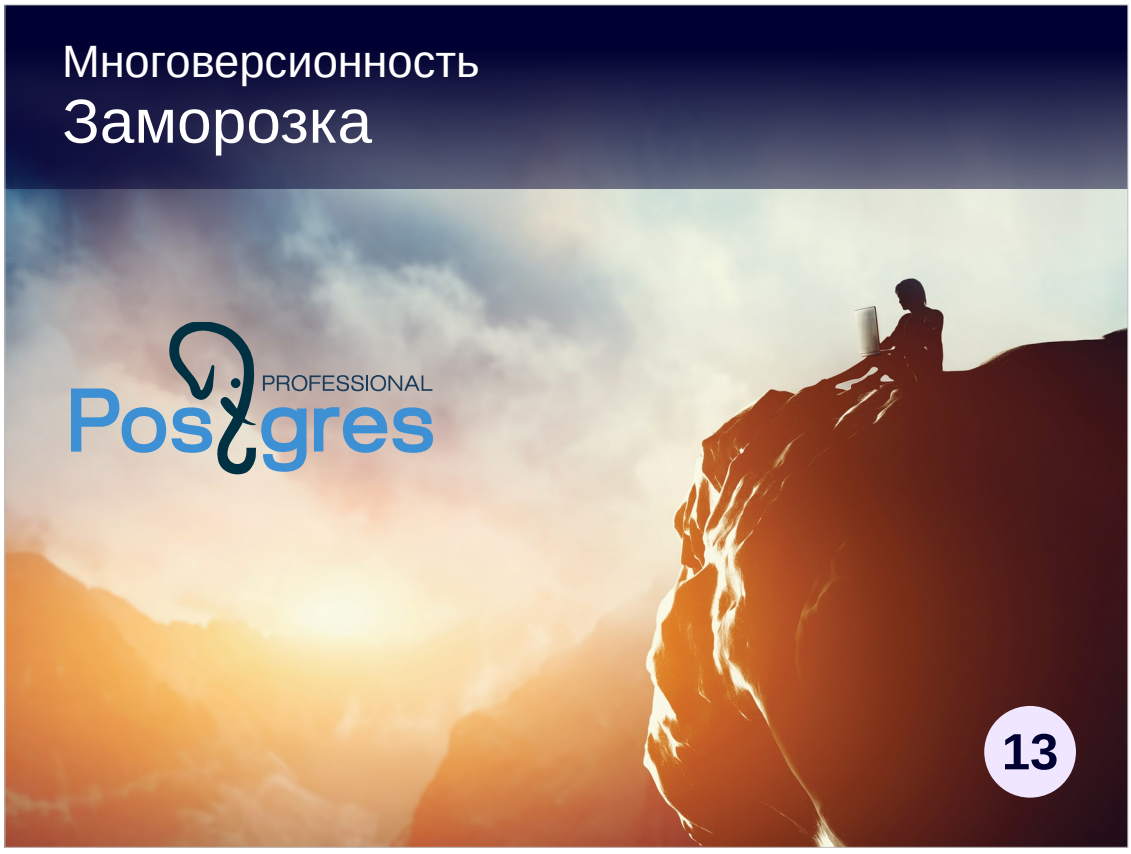


Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

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

Темы

Проблема переполнения счетчика транзакций

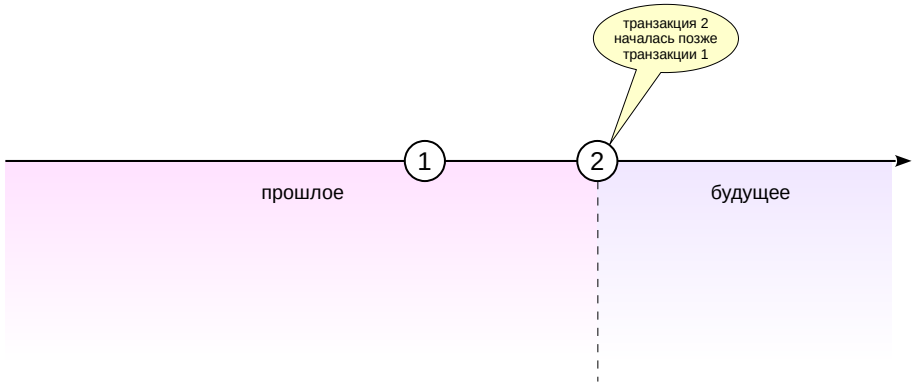
Заморозка версий строк и правила видимости

Настройка автоочистки для выполнения заморозки

Заморозка вручную

Переполнение счетчика

меньшие номера — прошлое, бóльшие — будущее
разрядность счетчика — 32 бита, что делать при переполнении?



Кроме освобождения места в страницах, очистка выполняет также задачу по предотвращению проблем, связанных с переполнением счетчика транзакций.

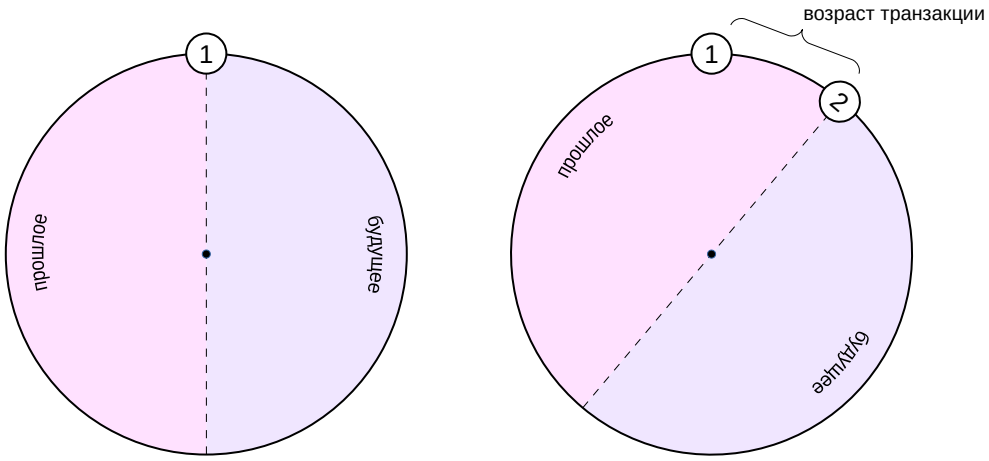
Под номер транзакции в PostgreSQL выделено 32 бита. Это довольно большое число (около 4 млрд), но при активной работе сервера оно вполне может быть исчерпано. Например при нагрузке 1000 транзакций в секунду это произойдет всего через полтора месяца непрерывной работы.

Но мы говорили о том, что механизм многоверсионности полагается на последовательную нумерацию транзакций — из двух транзакций транзакция с меньшим номером считается начавшейся раньше. Понятно, что нельзя просто обнулить счетчик и продолжить нумерацию заново.

Почему под номер транзакции не выделено 64 бита — ведь это полностью исключило бы проблему? Дело в том, что (как рассматривалось в теме «Страницы и версии строк») в заголовке каждой версии строки хранятся два номера транзакций — xmin и xmax. Заголовок и так достаточно большой, а увеличение разрядности привело бы к его увеличению еще на 8 байт.

Нумерация по кругу

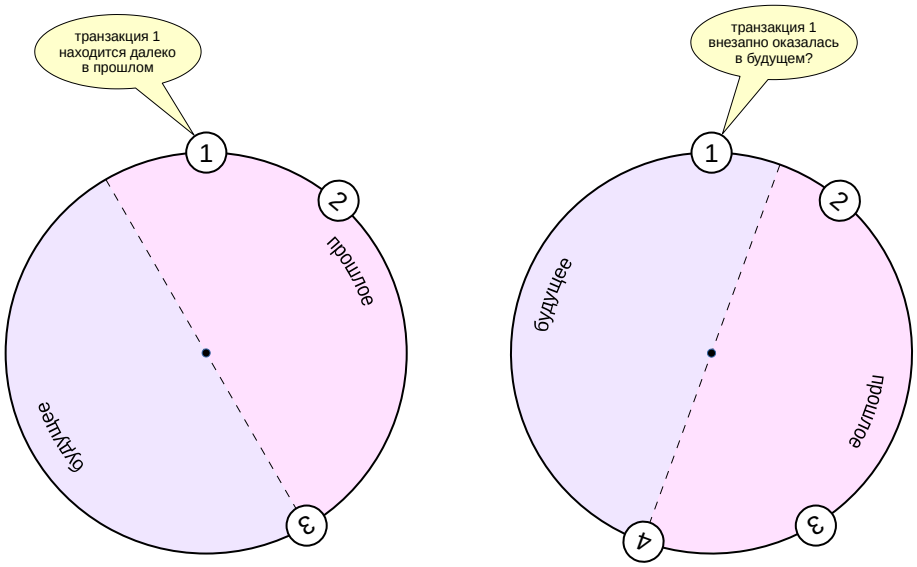
пространство номеров транзакций закольцовано
половина номеров — прошлое, половина — будущее



Поэтому вместо линейной схемы все номера транзакций закольцованы. Для любой транзакции половина номеров «против часовой стрелки» считается принадлежащей прошлому, а половина «по часовой стрелке» — будущему.

Возрастом транзакции называется число транзакций, прошедших с момента ее появления в системе (независимо от того, переходил ли счетчик через ноль или нет).

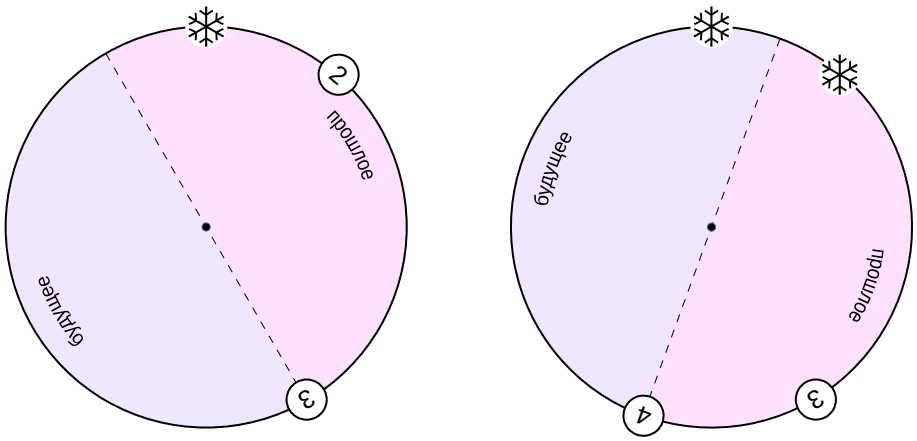
Проблема видимости



В такой закольцованной схеме возникает неприятная ситуация. Транзакция, находившаяся в далеком прошлом (транзакция 1 на слайде), через некоторое время окажется в той половине круга, которая относится к будущему. Это, конечно, нарушит правила видимости и приведет к проблемам.

Заморозка версий строк

замороженные версии строк считаются «бесконечно старыми»
номер транзакции xmin может быть использован заново



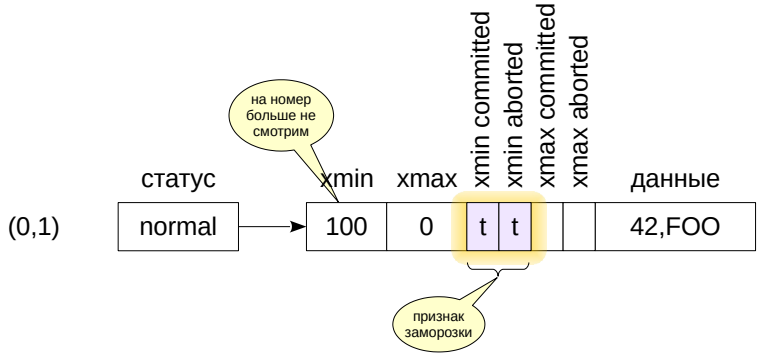
Чтобы не допустить путешествий из прошлого в будущее, процесс очистки выполняет еще одну задачу. Он находит достаточно старые и «холодные» версии строк (которые видны во всех снимках и изменение которых уже маловероятно) и специальным образом помечает — «замораживает» — их. Замороженная версия строки считается старше любых обычных данных и всегда видна во всех снимках данных. При этом уже не требуется смотреть на номер транзакции xmin, и этот номер может быть безопасно использован заново. Таким образом, замороженные версии строк всегда остаются в прошлом.

https://postgrespro.ru/docs/postgresql/13/routine-vacuuming#VACUUM-FOR-WRAPAROUND

Заморозка версий строк

Еще одна задача процесса очистки

если вовремя не заморозить версии строк, они окажутся в будущем
и сервер остановится для предотвращения ошибки



Для того чтобы пометить версию строки как замороженную, для транзакции xmin выставляются одновременно оба бита-подсказки — бит фиксации и бит отмены.

Заметим, что транзакцию xmax замораживать не нужно. Ее наличие означает, что данная версия строки больше не актуальна. После того, как она перестанет быть видимой в снимках данных, такая версия строки будет очищена.

Многие источники (включая документацию) упоминают специальный номер FrozenTransactionId = 2, который записывается на место xmin в замороженных версиях. Такая система действовала до версии 9.4, но сейчас заменена на биты-подсказки — это позволяет сохранить в версии строки исходный номер транзакции, что удобно для целей поддержки и отладки. Однако транзакции с номером 2 еще могут встретиться в старых системах, даже обновленных до последних версий.

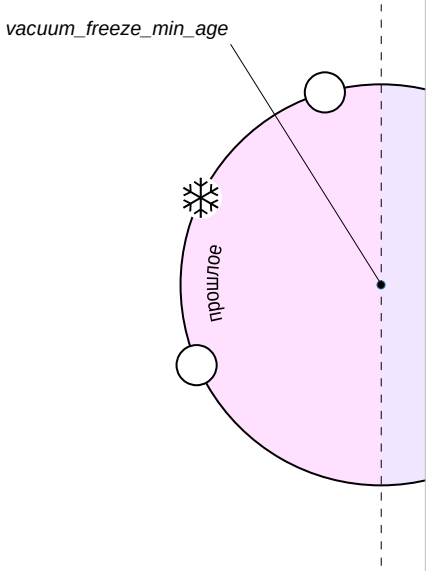
Важно, чтобы версии строк замораживались вовремя. Если возникнет ситуация, при которой еще не замороженный номер транзакции рискует попасть в будущее, PostgreSQL аварийно остановится. Это возможно в двух случаях: либо транзакция не завершена и, следовательно, не может быть заморожена, либо не сработала очистка.

При запуске сервера транзакция будет автоматически отменена; дальше администратор должен вручную выполнить очистку, и после этого система сможет продолжить работу.

Настройка

vacuum_freeze_min_age

минимальный возраст,
с которого начинается заморозка



Заморозкой управляют три основных параметра.

Параметр ***vacuum_freeze_min_age*** определяет минимальный возраст транзакции xmin, с которого начинается заморозка.

Чем меньше это значение, тем больше может быть накладных расходов. Если строка «горячая» и активно меняется, заморозка ее версий будет пропадать без пользы: уже замороженные версии будут вычищаться, а новые версии придется снова замораживать.

Поэтому более молодые версии строк замораживаются только в тех случаях, когда это точно не добавляет работы, например, при полной очистке таблицы.

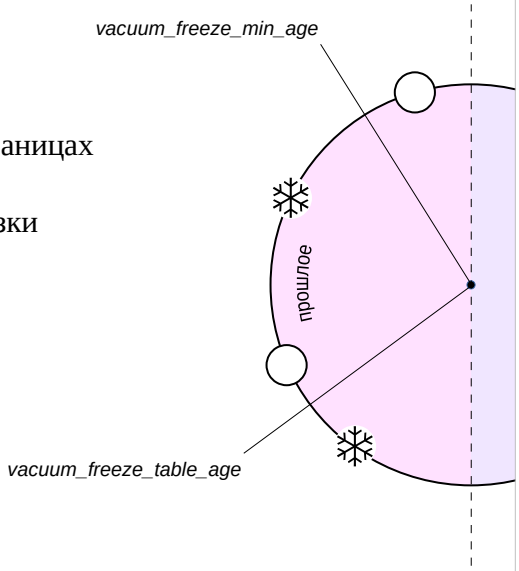
Заметим, что очистка просматривает только страницы, не отмеченные в карте видимости. Если на странице остались только актуальные версии, то очистка не придет в такую страницу и не заморозит их.

Настройка

vacuum_freeze_table_age

при достижении такого возраста
замораживаются версии строк на всех страницах
(«агрессивная» заморозка)

для ускорения используется карта заморозки



Параметр ***vacuum_freeze_table_age*** определяет возраст транзакции, при котором пора выполнять заморозку версий строк на всех страницах таблицы. Такая заморозка называется «агрессивной».

Каждая таблица хранит номер транзакции (pg_class.relfrozenxid), для которого известно, что в версиях строк не осталось более старых незамороженных номеров транзакций. Возраст этой транзакции и сравнивается со значением параметра.

Чтобы не просматривать всю таблицу целиком, вместе с картой видимости ведется *карта заморозки.* В ней отмечены страницы, в которых заморожены все версии строк. Такие страницы можно пропускать при заморозке.

Даже в агрессивном режиме версии строк с транзакциями младше *vacuum_freeze_min_age* не замораживаются, поэтому после заморозки новый возраст транзакции relfrozenxid будет равен не нулю, а *vacuum_freeze_min_age.* Таким образом, заморозка всех страниц выполняется раз в (*vacuum_freeze_table_age* − *vacuum_freeze_min_age*) транзакций.

Мы уже говорили, что слишком маленькое значение параметра *vacuum_freeze_min_age* увеличивает накладные расходы на очистку. Но при больших значениях агрессивная заморозка будет выполняться слишком часто, что тоже плохо. Установка этого параметра требует компромисса.

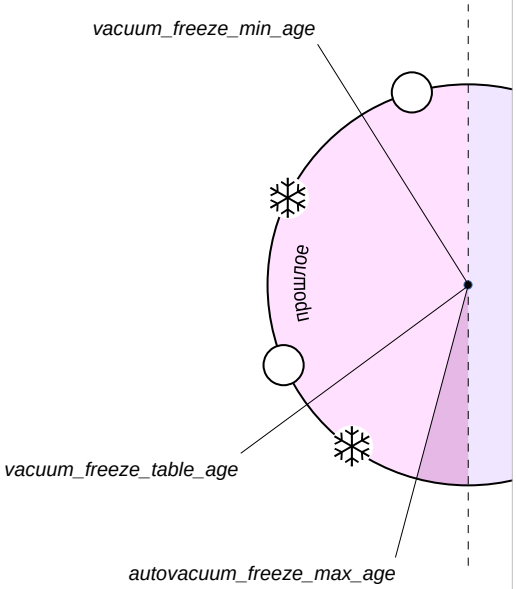
Настройка

autovacuum_freeze_max_age

при достижении такого возраста заморозка запускается принудительно

определяет размер CLOG

VACUUM (index_cleanup off)



Параметр ***autovacuum_freeze_max_age*** определяет возраст транзакции, при котором заморозка будет выполняться принудительно. Автоочистка для предотвращения последствий переполнения счетчика транзакций запустится, даже если она отключена параметрами.

Этот параметр также определяет размер структуры CLOG: данные о статусе более старых транзакций точно никогда не понадобятся, поэтому часть файлов из PGDATA/pg_xact может быть удалена.

Если администратор понимает, что автоочистка не успеет заморозить версии строк до переполнения счетчика транзакций, можно воспользоваться ручной очисткой с параметром index_cleanup off. В этом случае индексы не будут очищаться, но за счет этого версии строк в таблицах будут заморожены быстрее.

Настройка

Конфигурационные параметры

vacuum_freeze_min_age = 50 000 000
vacuum_freeze_table_age = 150 000 000
autovacuum_freeze_max_age = 200 000 000

↑?

Параметры хранения таблиц

autovacuum_freeze_min_age
toast.autovacuum_freeze_min_age

autovacuum_freeze_table_age
toast.autovacuum_freeze_table_age

autovacuum_freeze_max_age
toast.autovacuum_freeze_max_age

Значения по умолчанию довольно консервативны. Предел для *autovacuum_freeze_max_age* составляет порядка 2 млрд транзакций, а используется значение, в 10 раз меньшее. Можно увеличить значения *vacuum_freeze_table_age* и *autovacuum_freeze_max_age* для уменьшения накладных расходов, но важно понимать, что если по каким-то причинам (например, из-за незавершенной транзакции) автоочистка не справится вовремя с заморозкой, у администратора останется мало времени для принятия мер.

Обратите внимание, что изменение параметра *autovacuum_freeze_max_age* требует перезапуска сервера.

Параметры также можно устанавливать на уровне отдельных таблиц с помощью параметров хранения. Это имеет смысл делать только в особенных случаях, когда таблица действительно требует особого обхождения. Обратите внимание, что имена параметров на уровне таблиц немного отличаются от имен конфигурационных параметров.

В модуле «Блокировки» рассматриваются т. н. мультитранзакции и дополнительные параметры настройки заморозки для них.

Заморозка

Установим для демонстрации параметры заморозки.

Небольшой возраст транзакции:

```
=> ALTER SYSTEM SET vacuum_freeze_min_age = 1;
```

```
ALTER SYSTEM
```

Возраст, после которого будет выполняться заморозка всех страниц:

```
=> ALTER SYSTEM SET vacuum_freeze_table_age = 3;
```

```
ALTER SYSTEM
```

И отключим автоматическую очистку, чтобы запускать ее вручную в нужный момент.

```
=> ALTER SYSTEM SET autovacuum = off;
```

```
ALTER SYSTEM
```

```
=> SELECT pg_reload_conf();
```

```
 pg_reload_conf
----------------
 t
(1 row)
```

Создадим таблицу с данными. Установим минимальный fillfactor: на каждой странице будет всего две строки.

```
=> CREATE DATABASE mvcc_freeze;
```

```
CREATE DATABASE
```

```
=> \c mvcc_freeze
```

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

```
=> CREATE TABLE t(id integer, s char(300)) WITH (fillfactor = 10);
```

```
CREATE TABLE
```

Создадим представление для наблюдения за битами-подсказками на первых двух страницах таблицы.

Сейчас нас интересует только xmin и биты, которые относятся к нему, поскольку версии строк с ненулевым xmax будут очищены. Кроме того, выведем и возраст транзакции xmin.

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

```
CREATE EXTENSION
```

```
=> CREATE VIEW t_v AS
SELECT '('||blkno||','||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,
       age(t_xmin) AS xmin_age,
       CASE WHEN (t_infomask & 256) > 0 THEN 't' END AS xmin_c,
       CASE WHEN (t_infomask & 512) > 0 THEN 't' END AS xmin_a,
       t_xmax AS xmax
FROM (
  SELECT 0 blkno, * FROM heap_page_items(get_raw_page('t',0))
  UNION ALL
  SELECT 1 blkno, * FROM heap_page_items(get_raw_page('t',1))
) q
ORDER BY blkno, lp;
```

```
CREATE VIEW
```

Для того чтобы заглянуть в карту видимости и заморозки, воспользуемся еще одним расширением:

```
=> CREATE EXTENSION pg_visibility;
```

```
CREATE EXTENSION
```

Вставляем данные. Сразу выполним очистку, чтобы заполнить карту видимости.

```
=> INSERT INTO t(id, s) SELECT g.id, 'FOO' FROM generate_series(1,100) g(id);

INSERT 0 100

=> VACUUM t;

VACUUM
```

Сразу после очистки обе страницы отмечены в карте видимости (all_visible):

```
=> SELECT * FROM generate_series(0,1) g(blkno), pg_visibility_map('t',g.blkno)
ORDER BY g.blkno;

 blkno | all_visible | all_frozen
-------+-------------+------------
     0 | t           | f
     1 | t           | f
(2 rows)
```

Каков возраст транзакции, создавшей строки?

```
=> SELECT * FROM t_v;

 ctid  | state  |  xmin  | xmin_age | xmin_c | xmin_a | xmax
-------+--------+--------+----------+--------+--------+------
 (0,1) | normal | 696056 |        1 | t      |        |    0
 (0,2) | normal | 696056 |        1 | t      |        |    0
 (1,1) | normal | 696056 |        1 | t      |        |    0
 (1,2) | normal | 696056 |        1 | t      |        |    0
(4 rows)
```

Возраст равен 1; версии строк с такой транзакцией еще не будут заморожены.

Обновим строку на нулевой странице. Новая версия попадет на ту же страницу благодаря небольшому значению fillfactor.

```
=> UPDATE t SET s = 'BAR' WHERE id = 1;

UPDATE 1

=> SELECT * FROM t_v;

 ctid  | state  |  xmin  | xmin_age | xmin_c | xmin_a |  xmax
-------+--------+--------+----------+--------+--------+--------
 (0,1) | normal | 696056 |        2 | t      |        | 696057
 (0,2) | normal | 696056 |        2 | t      |        |      0
 (0,3) | normal | 696057 |        1 |        |        |      0
 (1,1) | normal | 696056 |        2 | t      |        |      0
 (1,2) | normal | 696056 |        2 | t      |        |      0
(5 rows)
```

Сейчас нулевая страница уже будет обработана заморозкой:

- возраст транзакции превышает значение, установленное в vacuum_freeze_min_age;
- страница изменена и исключена из карты видимости.

```
=> SELECT * FROM generate_series(0,1) g(blkno), pg_visibility_map('t',g.blkno)
ORDER BY g.blkno;

 blkno | all_visible | all_frozen
-------+-------------+------------
     0 | f           | f
     1 | t           | f
(2 rows)
```

Выполняем очистку.

```
=> VACUUM t;

VACUUM
```

Очистка обработала измененную страницу. У одной версии строки установлены оба бита - это признак заморозки. Другая версия строки слишком молода и поэтому не была заморожена:

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

```
 ctid  |     state     |  xmin  | xmin_age | xmin_c | xmin_a | xmax
-------+---------------+--------+----------+--------+--------+------
 (0,1) | redirect to 3 |        |          |        |        |
 (0,2) | normal        | 696056 |        2 | t      | t      |    0
 (0,3) | normal        | 696057 |        1 | t      |        |    0
 (1,1) | normal        | 696056 |        2 | t      |        |    0
 (1,2) | normal        | 696056 |        2 | t      |        |    0
(5 rows)
```

Теперь обе страницы отмечены в карте видимости (все версии строк на них актуальны). Очистка теперь не будет обрабатывать ни одну из этих страниц, и незамороженные версии строк так и останутся незамороженными.

```
=> SELECT * FROM generate_series(0,1) g(blkno), pg_visibility_map('t',g.blkno)
ORDER BY g.blkno;
```

```
 blkno | all_visible | all_frozen
-------+-------------+------------
     0 | t           | f
     1 | t           | f
(2 rows)
```

Именно для такого случая и требуется параметр vacuum_freeze_table_age, определяющий, в какой момент нужно просмотреть страницы, отмеченные в карте видимости, если они не отмечены в карте заморозки.

Для каждой таблицы сохраняется номер самой последней (наименее старой) замороженной транзакции. Ее возраст и сравнивается со значением параметра.

```
=> SELECT relfrozenxid, age(relfrozenxid) FROM pg_class WHERE relname = 't';
```

```
 relfrozenxid | age
--------------+-----
       696056 |   2
(1 row)
```

Сымитируем выполнение еще одной транзакции, чтобы возраст relfrozenxid таблицы достиг значения параметра vacuum_freeze_table_age.

```
=> SELECT txid_current();
```

```
 txid_current
--------------
       696058
(1 row)
```

```
=> SELECT relfrozenxid, age(relfrozenxid) FROM pg_class WHERE relname = 't';
```

```
 relfrozenxid | age
--------------+-----
       696056 |   3
(1 row)
```

```
=> VACUUM t;
```

```
VACUUM
```

Теперь, поскольку гарантированно была проверена вся таблица, номер замороженной транзакции можно увеличить - мы уверены, что в страницах не осталось более старой незамороженной транзакции.

```
=> SELECT relfrozenxid, age(relfrozenxid) FROM pg_class WHERE relname = 't';
```

```
 relfrozenxid | age
--------------+-----
       696058 |   1
(1 row)
```

Вот что получилось в страницах:

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

```
 ctid  |     state     |  xmin  | xmin_age | xmin_c | xmin_a | xmax
-------+---------------+--------+----------+--------+--------+------
 (0,1) | redirect to 3 |        |          |        |        |
 (0,2) | normal        | 696056 |        3 | t      | t      |    0
 (0,3) | normal        | 696057 |        2 | t      | t      |    0
 (1,1) | normal        | 696056 |        3 | t      | t      |    0
 (1,2) | normal        | 696056 |        3 | t      | t      |    0
(5 rows)
```

Обе страницы теперь отмечены в карте заморозки.

```
=> SELECT * FROM generate_series(0,1) g(blkno), pg_visibility_map('t',g.blkno)
ORDER BY g.blkno;
```

```
 blkno | all_visible | all_frozen
-------+-------------+------------
     0 | t           | t
     1 | t           | t
(2 rows)
```

Номер последней замороженной транзакции есть и на уровне всей БД:

```
=> SELECT datname, datfrozenxid, age(datfrozenxid)
FROM pg_database;
```

```
   datname   | datfrozenxid |  age
-------------+--------------+-------
 postgres    |       639307 | 56752
 student     |       639307 | 56752
 template1   |       640307 | 55752
 template0   |       640307 | 55752
 mvcc_freeze |       640307 | 55752
(5 rows)
```

Он устанавливается в минимальное значение из relfrozenxid всех таблиц этой БД. Если возраст datfrozenxid превысит значение параметра autovacuum_freeze_max_age, автоочистка будет запущена принудительно.

Заморозка вручную

VACUUM FREEZE

- принудительная заморозка версий строк с xmin любого возраста
- тот же эффект и при VACUUM FULL, CLUSTER

COPY … WITH FREEZE

- принудительная заморозка сразу после загрузки
- таблица должна быть создана или опустошена в той же транзакции
- могут нарушиться правила изоляции транзакции

Иногда бывает удобно управлять заморозкой вручную, а не дожидаться автоочистки.

Заморозку можно вызвать вручную командой VACUUM FREEZE — при этом будут заморожены все версии строк, без оглядки на возраст транзакций (как будто параметр *autovacuum_freeze_min_age* = 0). При перестройке таблицы командами VACUUM FULL или CLUSTER все строки также замораживаются.

https://postgrespro.ru/docs/postgresql/13/sql-vacuum

Данные можно заморозить и при начальной загрузке с помощью команды COPY, указав параметр FREEZE. Для этого таблица должна быть создана (или опустошена командой TRUNCATE) в той же транзакции, что и COPY. Поскольку для замороженных строк действуют отдельные правила видимости, такие строки будут видны в снимках данных других транзакций в нарушение обычных правил изоляции (это касается транзакций с уровнем Repeatable Read или Serializable), но обычно это не представляет проблемы. Подробнее такой случай рассматривается в практике.

https://postgrespro.ru/docs/postgresql/13/sql-copy

Итоги

Пространство номеров транзакций закольцовано

Достаточно старые версии строк замораживаются процессом очистки

Для оптимизации используется карта заморозки

Практика

1. Проверьте с помощью расширения pageinspect, что при использовании команды COPY … WITH FREEZE версии строк действительно замораживаются.
2. Убедитесь, что даже на уровне изоляции Repeatable Read строки, загруженные командой COPY … WITH FREEZE, оказываются видны в снимке данных.
3. Уменьшив значение параметра *autovacuum_freeze_max_age* и отключив автоочистку, воспроизведите ситуацию принудительного срабатывания автоочистки, выполнив соответствующее количество транзакций.
   Учтите, что срабатывание произойдет не сразу, а при выполнении ручной очистки какой-нибудь таблицы (или при перезапуске сервера).

3. Чтобы транзакциям выделялись настоящие (не виртуальные) номера, в транзакции нужно менять данные.

Можно организовать цикл в bash, в котором вызывать psql с командой обновления:

```
psql -c 'UPDATE …'
```

Другой вариант – использовать для организации цикла PL/pgSQL. Поскольку внутри серверного кода явно управлять транзакциями нельзя, придется использовать блок с обработкой исключений. Тогда при перехвате исключения транзакция будет откатываться к неявной точке сохранения: фактически начнется новая вложенная транзакция (см. тему «Страницы и версии строк»).

Третий вариант – использовать утилиту pgbench:

https://postgrespro.ru/docs/postgresql/13/pgbench

1. Заморозка при COPY WITH FREEZE

Создаем таблицу и загружаем несколько строк в одной и той же транзакции:

```
=> CREATE DATABASE mvcc_freeze;
```

```
CREATE DATABASE
```

```
=> \c mvcc_freeze
```

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

```
=> BEGIN;
```

```
BEGIN
```

```
=> CREATE TABLE t(n integer);
```

```
CREATE TABLE
```

```
=> COPY t FROM stdin WITH FREEZE;
```

```
=> 1
```

```
=> 2
```

```
=> 3
```

```
=> \.
```

```
COPY 3
```

```
=> COMMIT;
```

```
COMMIT
```

Проверяем версии строк:

```
=> 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 AS xmin,
       age(t_xmin) AS xmin_age,
       CASE WHEN (t_infomask & 256) > 0 THEN 't' END AS xmin_c,
       CASE WHEN (t_infomask & 512) > 0 THEN 't' END AS xmin_a,
       t_xmax AS xmax,
       t_ctid
FROM heap_page_items(get_raw_page('t',0))
ORDER BY lp;
```

```
CREATE VIEW
```

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

```
 ctid  | state  |  xmin  | xmin_age | xmin_c | xmin_a | xmax | t_ctid
-------+--------+--------+----------+--------+--------+------+--------
 (0,1) | normal | 749764 |        3 | t      | t      |    0 | (0,1)
 (0,2) | normal | 749764 |        3 | t      | t      |    0 | (0,2)
 (0,3) | normal | 749764 |        3 | t      | t      |    0 | (0,3)
(3 rows)
```

2. COPY WITH FREEZE и изоляция

В другом сеансе начнем транзакцию с уровнем изоляции Repeatable Read.

```
=> \c mvcc_freeze
```

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

```
=> BEGIN ISOLATION LEVEL REPEATABLE READ;
```

```
BEGIN
```

```
=> SELECT txid_current();
```

```
 txid_current
--------------
       749767
(1 row)
```

Обратите внимание, что эта транзакция не должна обращаться к таблице t.

Теперь опустошим таблицу и загрузим в нее новые строки в одной транзакции. Если бы параллельная транзакция прочитала содержимое t, команда TRUNCATE ожидала бы ее завершения.

```
=> BEGIN;
```

```
BEGIN
```

```
=> TRUNCATE t;
```

```
TRUNCATE TABLE
```

```
=> COPY t FROM stdin WITH FREEZE;
```

```
=> 10
```

```
=> 20
```

```
=> 30
```

```
=> \.
```

```
COPY 3
```

```
=> COMMIT;
```

```
COMMIT
```

Теперь параллельная транзакция видит новые данные, хотя это и нарушает изоляцию:

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

```
 n
----
 10
 20
 30
(3 rows)
```

```
=> COMMIT;
```

```
COMMIT
```

```
=> \q
```

3. Аварийное срабатывание автоочистки

Предварительно заморозим все транзакции во всех базах. Для этого удобно воспользоваться командой vacuumdb:

```
student$ vacuumdb --all --freeze
```

```
vacuumdb: vacuuming database "mvcc_freeze"
vacuumdb: vacuuming database "postgres"
vacuumdb: vacuuming database "student"
vacuumdb: vacuuming database "template1"
```

Максимальный возраст незамороженных транзакций по всем БД:

```
=> SELECT datname, datfrozenxid, age(datfrozenxid) FROM pg_database;
```

```
   datname   | datfrozenxid |  age
-------------+--------------+--------
 postgres    |       749769 |      0
 student     |       749769 |      0
 template1   |       749769 |      0
 template0   |       640307 | 109462
 mvcc_freeze |       749769 |      0
(5 rows)
```

Отключаем автоочистку.

```
=> ALTER SYSTEM SET autovacuum = off;
```

```
ALTER SYSTEM
```

Уменьшаем значения параметров:

```
=> ALTER SYSTEM SET vacuum_freeze_min_age = 1000;
```

```
ALTER SYSTEM
```

```
=> ALTER SYSTEM SET vacuum_freeze_table_age = 10000;
```

```
ALTER SYSTEM
```

```
=> ALTER SYSTEM SET autovacuum_freeze_max_age = 100000;

ALTER SYSTEM
```

Требуется перезагрузка сервера.

```
student$ sudo pg_ctlcluster 13 main restart

student$ psql mvcc_freeze
```

Получить большое количество транзакций можно разными способами; например, можно воспользоваться утилитой pgbench. Попросим ее инициализировать свои таблицы и выполнить 100000 транзакций.

```
student$ pgbench -i mvcc_freeze

dropping old tables...
NOTICE:  table "pgbench_accounts" does not exist, skipping
NOTICE:  table "pgbench_branches" does not exist, skipping
NOTICE:  table "pgbench_history" does not exist, skipping
NOTICE:  table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
100000 of 100000 tuples (100%) done (elapsed 0.10 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 0.27 s (drop tables 0.00 s, create tables 0.01 s, client-side generate 0.13 s, vacuum 0.07 s, primary keys 0.06 s).

student$ pgbench -t 100000 -P 5 mvcc_freeze

starting vacuum...end.
progress: 5.0 s, 868.6 tps, lat 1.150 ms stddev 0.365
progress: 10.0 s, 808.6 tps, lat 1.236 ms stddev 0.763
progress: 15.0 s, 781.6 tps, lat 1.279 ms stddev 0.230
progress: 20.0 s, 786.4 tps, lat 1.271 ms stddev 0.240
progress: 25.0 s, 796.8 tps, lat 1.254 ms stddev 0.246
progress: 30.0 s, 837.2 tps, lat 1.194 ms stddev 0.234
progress: 35.0 s, 784.2 tps, lat 1.275 ms stddev 0.183
progress: 40.0 s, 811.2 tps, lat 1.232 ms stddev 0.394
progress: 45.0 s, 778.6 tps, lat 1.284 ms stddev 0.212
progress: 50.0 s, 788.2 tps, lat 1.268 ms stddev 0.201
progress: 55.0 s, 782.0 tps, lat 1.278 ms stddev 0.258
progress: 60.0 s, 785.0 tps, lat 1.274 ms stddev 0.253
progress: 65.0 s, 740.2 tps, lat 1.350 ms stddev 0.736
progress: 70.0 s, 863.6 tps, lat 1.158 ms stddev 0.648
progress: 75.0 s, 891.6 tps, lat 1.121 ms stddev 0.191
progress: 80.0 s, 774.4 tps, lat 1.291 ms stddev 0.250
progress: 85.0 s, 792.4 tps, lat 1.261 ms stddev 0.222
progress: 90.0 s, 772.4 tps, lat 1.295 ms stddev 0.219
progress: 95.0 s, 779.8 tps, lat 1.282 ms stddev 0.204
progress: 100.0 s, 783.4 tps, lat 1.276 ms stddev 0.324
progress: 105.0 s, 767.2 tps, lat 1.303 ms stddev 0.199
progress: 110.0 s, 763.8 tps, lat 1.309 ms stddev 0.197
progress: 115.0 s, 790.6 tps, lat 1.264 ms stddev 0.219
progress: 120.0 s, 826.4 tps, lat 1.210 ms stddev 0.223
progress: 125.0 s, 778.4 tps, lat 1.284 ms stddev 0.190
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
number of transactions per client: 100000
number of transactions actually processed: 100000/100000
latency average = 1.254 ms
latency stddev = 0.343 ms
tps = 797.369602 (including connections establishing)
tps = 797.386127 (excluding connections establishing)
```

Видно, что возраст незамороженных транзакций превышает установленное пороговое значение (100000):

```
=> SELECT datname, datfrozenxid, age(datfrozenxid) FROM pg_database;

   datname   | datfrozenxid |  age
-------------+--------------+--------
 postgres    |       749769 | 100012
 student     |       749769 | 100012
 template1   |       749769 | 100012
 template0   |       749769 | 100012
 mvcc_freeze |       749769 | 100012
(5 rows)
```

Теперь при выполнении команды VACUUM для любой таблицы будет запущен процесс автоочистки.

```
=> VACUUM t;

VACUUM
```

Среди процессов появился autovacuum worker:

```
postgres$ ps -o pid,command --ppid `head -n 1 /var/lib/postgresql/13/main/postmaster.pid`
```

```
    PID COMMAND
  40441 postgres: 13/main: checkpointer
  40442 postgres: 13/main: background writer
  40443 postgres: 13/main: walwriter
  40444 postgres: 13/main: stats collector
  40445 postgres: 13/main: logical replication launcher
  40482 postgres: 13/main: student mvcc_freeze [local] idle
  40748 postgres: 13/main: autovacuum worker postgres
```

И через некоторое время транзакции окажутся замороженными:

```
=> SELECT datname, datfrozenxid, age(datfrozenxid) FROM pg_database;
```

```
   datname   | datfrozenxid | age
-------------+--------------+------
 postgres    |       848781 | 1000
 student     |       848781 | 1000
 template1   |       849781 |    0
 template0   |       849781 |    0
 mvcc_freeze |       848781 | 1000
(5 rows)
```