

# Многоверсионность Заморозка



## **Авторские права**

© Postgres Professional, 2016–2022.

Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

## **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.

Проблема переполнения счетчика транзакций

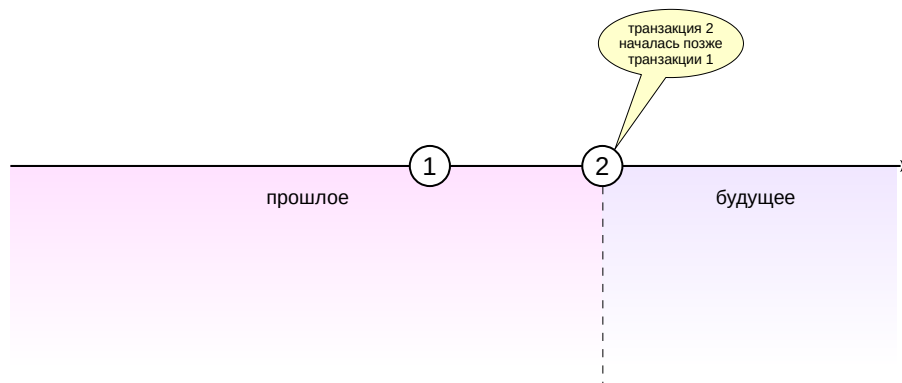
Заморозка версий строк и правила видимости

Настройка автоочистки для выполнения заморозки

Заморозка вручную

# Переполнение счетчика

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



3

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

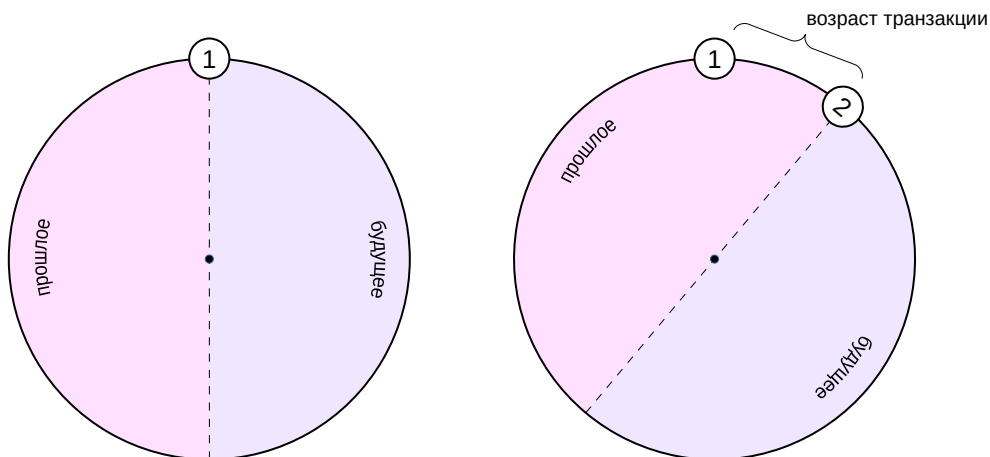
Под номер транзакции в PostgreSQL выделено 32 бита. Это довольно большое число (около 4 млрд), но при активной работе сервера оно вполне может быть исчерпано. Например при нагрузке 1000 транзакций в секунду это произойдет всего через полтора месяца непрерывной работы.

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

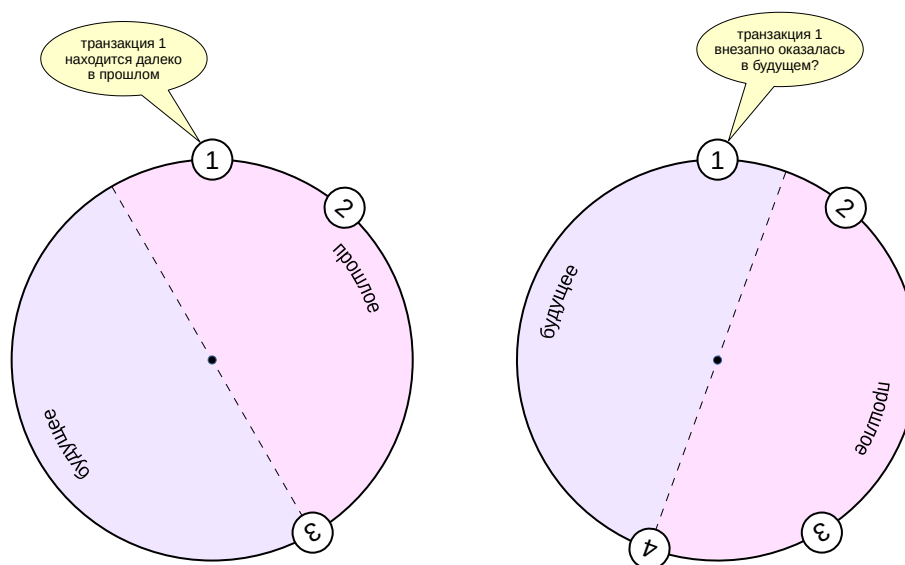


4

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

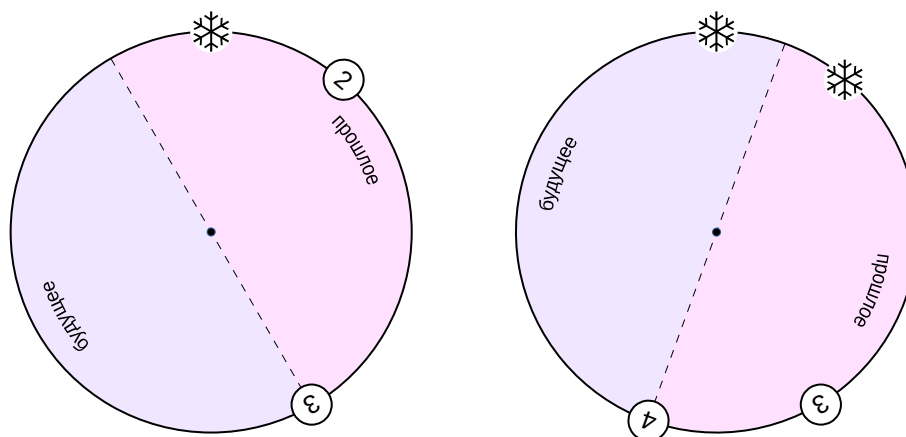


5

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



6

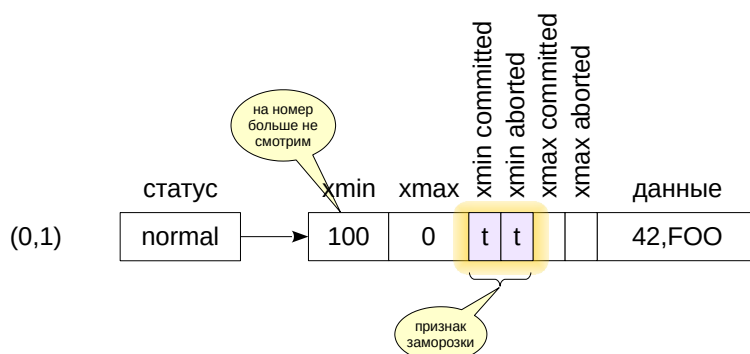
Чтобы не допустить путешествий из прошлого в будущее, процесс очистки выполняет еще одну задачу. Он находит достаточно старые и «холодные» версии строк (которые видны во всех снимках и изменение которых уже маловероятно) и специальным образом помечает — «замораживает» — их. Замороженная версия строки 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



7

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.

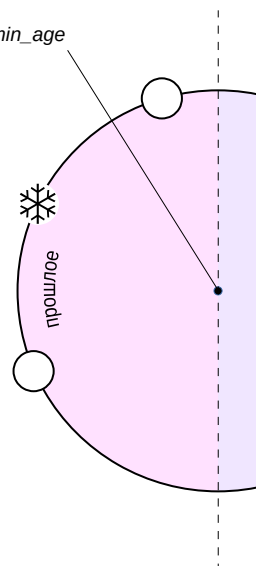
Важно, чтобы версии строк замораживались вовремя. Если возникнет ситуация, при которой еще не замороженный номер транзакции рискует попасть в будущее, PostgreSQL аварийно остановится. Это возможно в двух случаях: либо транзакция не завершена и, следовательно, не может быть заморожена, либо не сработала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

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

***vacuum\_freeze\_min\_age***

минимальный возраст,  
с которого начинается заморозка

*vacuum\_freeze\_min\_age*



8

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.

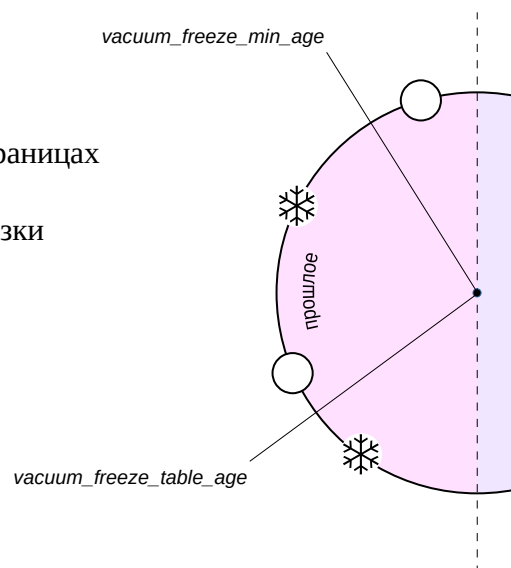
Поэтому более молодые версии строк замораживаются только в тех случаях, когда это точно не добавляет работы, например, при полной очистке таблицы.

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.



## `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.relrozenxid`), 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.

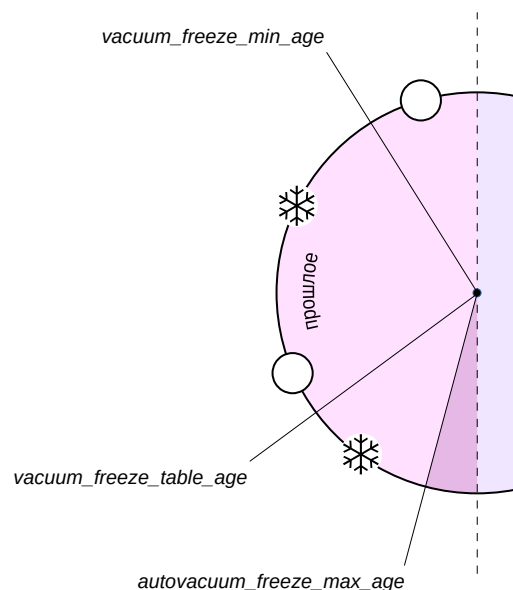
Даже в агрессивном режиме версии строк с транзакциями младше `vacuum_freeze_min_age` не замораживаются, поэтому после заморозки новый возраст транзакции `relrozenxid` будет равен не нулю, а `vacuum_freeze_min_age`. Таким образом, заморозка всех страниц выполняется раз в  $(vacuum\_freeze\_table\_age - vacuum\_freeze\_min\_age)$  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.

## *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.

## 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
```

11

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	588	1	t		0
(0,2)	normal	588	1	t		0
(1,1)	normal	588	1	t		0
(1,2)	normal	588	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	588	2	t		589
(0,2)	normal	588	2	t		0
(0,3)	normal	589	1			0
(1,1)	normal	588	2	t		0
(1,2)	normal	588	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	588	2	t	t	0
(0,3)	normal	589	1	t		0
(1,1)	normal	588	2	t		0
(1,2)	normal	588	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 наибольший номер транзакции, для которого все версии строк с меньшими номерами `xmin` гарантированно заморожены. Ее возраст и сравнивается со значением параметра.

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

relfrozenxid	age
588	2

(1 row)

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

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

txid_current
590

(1 row)

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

relfrozenxid	age
588	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
590	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	588	3	t	t	0
(0,3)	normal	589	2	t	t	0
(1,1)	normal	588	3	t	t	0
(1,2)	normal	588	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	478	113
student	478	113
template1	478	113
template0	478	113
mvcc_freeze	478	113

(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.

## 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>



Пространство номеров транзакций закольцовано  
Достаточно старые версии строк замораживаются  
процессом очистки  
Для оптимизации используется карта заморозки

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).

15

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	28867	3	t	t	0	(0,1)
(0,2)	normal	28867	3	t	t	0	(0,2)
(0,3)	normal	28867	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
-----
28870
(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, datfrozensxid, age(datfrozensxid) FROM pg_database;
```

```

datname | datfrozensxid | age
-----+-----+---
postgres | 28872 | 0
student | 28872 | 0
template1 | 28872 | 0
template0 | 478 | 28394
mvcc_freeze | 28872 | 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 0.16 s, remaining 0.00 s)
```

```
vacuuming...
```

```
creating primary keys...
```

```
done in 0.54 s (drop tables 0.00 s, create tables 0.03 s, client-side generate 0.23 s, vacuum 0.12 s, primary keys 0.17 s).
```

```
student$ pgbench -t 100000 -P 5 mvcc_freeze
```

```
starting vacuum...end.
```

```
progress: 5.0 s, 121.4 tps, lat 8.206 ms stddev 4.053
```

```
progress: 10.0 s, 146.8 tps, lat 6.809 ms stddev 1.478
```

```
progress: 15.0 s, 120.6 tps, lat 8.282 ms stddev 2.121
```

```
progress: 20.0 s, 118.4 tps, lat 8.450 ms stddev 2.342
```

```
progress: 25.0 s, 139.8 tps, lat 7.150 ms stddev 1.355
```

```
progress: 30.0 s, 144.4 tps, lat 6.931 ms stddev 1.380
```

```
progress: 35.0 s, 136.4 tps, lat 7.324 ms stddev 1.355
```

```
progress: 40.0 s, 141.2 tps, lat 7.082 ms stddev 1.432
```

```
progress: 45.0 s, 131.8 tps, lat 7.581 ms stddev 2.481
```

```
progress: 50.0 s, 126.4 tps, lat 7.916 ms stddev 1.307
```

```
progress: 55.0 s, 133.2 tps, lat 7.508 ms stddev 1.154
```

```
progress: 60.0 s, 135.6 tps, lat 7.372 ms stddev 1.415
```

```
progress: 65.0 s, 127.6 tps, lat 7.828 ms stddev 1.092
```

```
progress: 70.0 s, 124.8 tps, lat 8.022 ms stddev 1.334
```

```
progress: 75.0 s, 125.4 tps, lat 7.960 ms stddev 1.011
```

```
progress: 80.0 s, 121.2 tps, lat 8.250 ms stddev 2.624
```

```
progress: 85.0 s, 124.5 tps, lat 8.029 ms stddev 1.463
```

```
progress: 90.0 s, 138.9 tps, lat 7.201 ms stddev 1.371
```

```
progress: 95.0 s, 136.8 tps, lat 7.312 ms stddev 1.213
```

```
progress: 100.0 s, 117.2 tps, lat 8.533 ms stddev 2.053
```

```
progress: 105.0 s, 129.3 tps, lat 7.738 ms stddev 1.130
```

```
progress: 110.0 s, 130.0 tps, lat 7.689 ms stddev 1.096
```

```
progress: 115.0 s, 128.8 tps, lat 7.765 ms stddev 1.274
```

```
progress: 120.0 s, 135.6 tps, lat 7.371 ms stddev 1.289
```

```
progress: 125.0 s, 146.8 tps, lat 6.815 ms stddev 1.387
```

```
progress: 130.0 s, 136.0 tps, lat 7.346 ms stddev 1.503
```

```
progress: 135.0 s, 134.6 tps, lat 7.427 ms stddev 2.097
```

```
progress: 140.0 s, 134.0 tps, lat 7.463 ms stddev 1.110
```

```
progress: 145.0 s, 130.0 tps, lat 7.687 ms stddev 1.221
```

```
progress: 150.0 s, 128.6 tps, lat 7.774 ms stddev 1.126
```

```
progress: 155.0 s, 140.6 tps, lat 7.115 ms stddev 1.467
```

```
progress: 160.0 s, 137.6 tps, lat 7.260 ms stddev 1.380
```

```
progress: 165.0 s, 127.8 tps, lat 7.824 ms stddev 1.544
```

```
progress: 170.0 s, 134.2 tps, lat 7.454 ms stddev 1.354
```

```
progress: 175.0 s, 143.0 tps, lat 6.992 ms stddev 1.426
```

```
progress: 180.0 s, 134.2 tps, lat 7.447 ms stddev 1.471
```

```
progress: 185.0 s, 134.2 tps, lat 7.456 ms stddev 1.387
```

```
progress: 190.0 s, 127.6 tps, lat 7.831 ms stddev 1.362
```

```
progress: 195.0 s, 123.8 tps, lat 8.073 ms stddev 1.034
```

```
progress: 200.0 s, 124.2 tps, lat 8.052 ms stddev 1.117
```

```
progress: 205.0 s, 123.0 tps, lat 8.132 ms stddev 0.910
```

```
progress: 210.0 s, 132.8 tps, lat 7.525 ms stddev 1.257
```

```
progress: 215.0 s, 132.8 tps, lat 7.535 ms stddev 1.240
```

```
progress: 220.0 s, 127.4 tps, lat 7.845 ms stddev 1.096
```

```
progress: 225.0 s, 130.6 tps, lat 7.654 ms stddev 1.323
```

```
progress: 230.0 s, 148.2 tps, lat 6.748 ms stddev 1.501
```

```
progress: 235.0 s, 129.4 tps, lat 7.724 ms stddev 1.471
```

```
progress: 240.0 s, 130.6 tps, lat 7.661 ms stddev 1.187
```

```
progress: 245.0 s, 134.4 tps, lat 7.440 ms stddev 1.378
```

```
progress: 250.0 s, 137.2 tps, lat 7.285 ms stddev 1.365
```

```
progress: 255.0 s, 130.2 tps, lat 7.679 ms stddev 1.269
```

```
progress: 260.0 s, 120.4 tps, lat 8.297 ms stddev 0.928
```

```
progress: 265.0 s, 128.0 tps, lat 7.814 ms stddev 1.093
```

```
progress: 270.0 s, 123.0 tps, lat 8.132 ms stddev 1.227
```

```
progress: 275.0 s, 125.6 tps, lat 7.952 ms stddev 1.007
```

```
progress: 280.0 s, 122.8 tps, lat 8.140 ms stddev 1.027
```

```
progress: 285.0 s, 124.0 tps, lat 8.071 ms stddev 1.078
```

```
progress: 290.0 s, 133.6 tps, lat 7.485 ms stddev 1.469
```

```
progress: 295.0 s, 139.4 tps, lat 7.169 ms stddev 1.545
```

progress: 300.0 s, 139.4 tps, lat 7.173 ms stddev 1.561  
progress: 305.0 s, 122.6 tps, lat 8.152 ms stddev 2.363  
progress: 310.0 s, 133.4 tps, lat 7.499 ms stddev 1.487  
progress: 315.0 s, 122.0 tps, lat 8.188 ms stddev 1.244  
progress: 320.0 s, 133.8 tps, lat 7.481 ms stddev 1.567  
progress: 325.0 s, 134.4 tps, lat 7.433 ms stddev 1.486  
progress: 330.0 s, 135.4 tps, lat 7.385 ms stddev 1.322  
progress: 335.0 s, 133.8 tps, lat 7.478 ms stddev 1.413  
progress: 340.0 s, 125.2 tps, lat 7.984 ms stddev 1.338  
progress: 345.0 s, 127.6 tps, lat 7.838 ms stddev 1.092  
progress: 350.0 s, 122.2 tps, lat 8.183 ms stddev 1.000  
progress: 355.0 s, 130.4 tps, lat 7.658 ms stddev 1.329  
progress: 360.0 s, 134.5 tps, lat 7.441 ms stddev 1.331  
progress: 365.0 s, 141.3 tps, lat 7.074 ms stddev 1.300  
progress: 370.0 s, 134.2 tps, lat 7.444 ms stddev 1.438  
progress: 375.0 s, 126.2 tps, lat 7.922 ms stddev 2.246  
progress: 380.0 s, 135.0 tps, lat 7.404 ms stddev 1.182  
progress: 385.0 s, 126.4 tps, lat 7.919 ms stddev 1.061  
progress: 390.0 s, 125.7 tps, lat 7.940 ms stddev 1.053  
progress: 395.0 s, 128.1 tps, lat 7.818 ms stddev 1.353  
progress: 400.0 s, 122.2 tps, lat 8.178 ms stddev 0.827  
progress: 405.0 s, 128.4 tps, lat 7.792 ms stddev 1.468  
progress: 410.0 s, 125.8 tps, lat 7.945 ms stddev 1.248  
progress: 415.0 s, 128.6 tps, lat 7.772 ms stddev 1.162  
progress: 420.0 s, 128.2 tps, lat 7.795 ms stddev 1.109  
progress: 425.0 s, 126.6 tps, lat 7.894 ms stddev 1.046  
progress: 430.0 s, 125.2 tps, lat 7.998 ms stddev 1.208  
progress: 435.0 s, 126.4 tps, lat 7.907 ms stddev 1.125  
progress: 440.0 s, 128.0 tps, lat 7.804 ms stddev 1.186  
progress: 445.0 s, 129.8 tps, lat 7.709 ms stddev 2.178  
progress: 450.0 s, 116.4 tps, lat 8.584 ms stddev 2.764  
progress: 455.0 s, 124.0 tps, lat 8.060 ms stddev 1.016  
progress: 460.0 s, 134.2 tps, lat 7.459 ms stddev 1.211  
progress: 465.0 s, 134.2 tps, lat 7.450 ms stddev 1.284  
progress: 470.0 s, 126.2 tps, lat 7.919 ms stddev 1.082  
progress: 475.0 s, 139.2 tps, lat 7.183 ms stddev 1.582  
progress: 480.0 s, 135.8 tps, lat 7.363 ms stddev 1.422  
progress: 485.0 s, 131.8 tps, lat 7.587 ms stddev 1.211  
progress: 490.0 s, 140.0 tps, lat 7.140 ms stddev 1.433  
progress: 495.0 s, 130.2 tps, lat 7.680 ms stddev 1.135  
progress: 500.0 s, 128.4 tps, lat 7.775 ms stddev 1.070  
progress: 505.0 s, 127.6 tps, lat 7.842 ms stddev 1.308  
progress: 510.0 s, 124.0 tps, lat 8.069 ms stddev 0.990  
progress: 515.0 s, 126.6 tps, lat 7.888 ms stddev 1.294  
progress: 520.0 s, 124.2 tps, lat 8.051 ms stddev 1.266  
progress: 525.0 s, 132.6 tps, lat 7.548 ms stddev 1.166  
progress: 530.0 s, 129.0 tps, lat 7.748 ms stddev 1.181  
progress: 535.0 s, 140.2 tps, lat 7.125 ms stddev 1.280  
progress: 540.0 s, 136.8 tps, lat 7.309 ms stddev 1.181  
progress: 545.0 s, 124.4 tps, lat 8.040 ms stddev 1.218  
progress: 550.0 s, 131.4 tps, lat 7.607 ms stddev 2.020  
progress: 555.0 s, 136.6 tps, lat 7.325 ms stddev 1.403  
progress: 560.0 s, 137.8 tps, lat 7.249 ms stddev 1.316  
progress: 565.0 s, 137.6 tps, lat 7.264 ms stddev 1.255  
progress: 570.0 s, 127.8 tps, lat 7.831 ms stddev 1.058  
progress: 575.0 s, 177.0 tps, lat 5.644 ms stddev 1.304  
progress: 580.0 s, 129.2 tps, lat 7.744 ms stddev 1.475  
progress: 585.0 s, 126.6 tps, lat 7.901 ms stddev 1.206  
progress: 590.0 s, 122.2 tps, lat 8.170 ms stddev 1.495  
progress: 595.0 s, 133.0 tps, lat 7.518 ms stddev 3.875  
progress: 600.0 s, 102.4 tps, lat 9.748 ms stddev 17.096  
progress: 605.0 s, 113.4 tps, lat 8.838 ms stddev 5.928  
progress: 610.0 s, 123.2 tps, lat 8.112 ms stddev 1.213  
progress: 615.0 s, 121.4 tps, lat 8.243 ms stddev 1.107  
progress: 620.0 s, 123.8 tps, lat 8.075 ms stddev 1.442  
progress: 625.0 s, 122.6 tps, lat 8.146 ms stddev 1.169  
progress: 630.0 s, 112.0 tps, lat 8.933 ms stddev 2.054  
progress: 635.0 s, 122.8 tps, lat 8.143 ms stddev 1.013  
progress: 640.0 s, 120.4 tps, lat 8.305 ms stddev 1.084  
progress: 645.0 s, 121.8 tps, lat 8.204 ms stddev 1.048  
progress: 650.0 s, 121.0 tps, lat 8.270 ms stddev 1.184  
progress: 655.0 s, 124.0 tps, lat 8.058 ms stddev 1.031  
progress: 660.0 s, 113.4 tps, lat 8.820 ms stddev 2.394  
progress: 665.0 s, 123.2 tps, lat 8.112 ms stddev 1.150  
progress: 670.0 s, 124.1 tps, lat 8.058 ms stddev 1.320  
progress: 675.0 s, 120.8 tps, lat 8.275 ms stddev 0.937  
progress: 680.0 s, 120.4 tps, lat 8.311 ms stddev 1.005  
progress: 685.0 s, 123.2 tps, lat 8.118 ms stddev 1.019  
progress: 690.0 s, 119.8 tps, lat 8.346 ms stddev 0.930  
progress: 695.0 s, 120.0 tps, lat 8.326 ms stddev 1.206  
progress: 700.0 s, 113.8 tps, lat 8.793 ms stddev 1.613  
progress: 705.0 s, 119.4 tps, lat 8.363 ms stddev 0.997  
progress: 710.0 s, 124.0 tps, lat 8.065 ms stddev 1.037  
progress: 715.0 s, 121.6 tps, lat 8.224 ms stddev 1.034  
progress: 720.0 s, 120.6 tps, lat 8.286 ms stddev 0.937  
progress: 725.0 s, 122.2 tps, lat 8.186 ms stddev 1.043  
progress: 730.0 s, 122.6 tps, lat 8.156 ms stddev 1.167

```

progress: 735.0 s, 114.6 tps, lat 8.727 ms stddev 2.341
progress: 740.0 s, 123.2 tps, lat 8.109 ms stddev 0.918
progress: 745.0 s, 120.4 tps, lat 8.311 ms stddev 1.005
progress: 750.0 s, 118.6 tps, lat 8.419 ms stddev 1.536
progress: 755.0 s, 122.0 tps, lat 8.196 ms stddev 1.135
progress: 760.0 s, 122.1 tps, lat 8.181 ms stddev 1.012
progress: 765.0 s, 122.1 tps, lat 8.193 ms stddev 1.081
progress: 770.0 s, 121.8 tps, lat 8.216 ms stddev 0.999
progress: 775.0 s, 118.0 tps, lat 8.476 ms stddev 1.075
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 = 7.783 ms
latency stddev = 1.997 ms
tps = 128.449752 (including connections establishing)
tps = 128.452480 (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	28872	100012
student	28872	100012
template1	28872	100012
template0	478	128406
mvcc_freeze	28872	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
40073 postgres: 13/main: checkpointer
40074 postgres: 13/main: background writer
40075 postgres: 13/main: walwriter
40076 postgres: 13/main: stats collector
40077 postgres: 13/main: logical replication launcher
40115 postgres: 13/main: student mvcc_freeze [local] idle
40767 postgres: 13/main: autovacuum worker template0

```

И через некоторое время транзакции окажутся замороженными:

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

datname	datfrozenxid	age
postgres	127884	1000
student	127884	1000
template1	128884	0
template0	128884	0
mvcc_freeze	127884	1000

(5 rows)