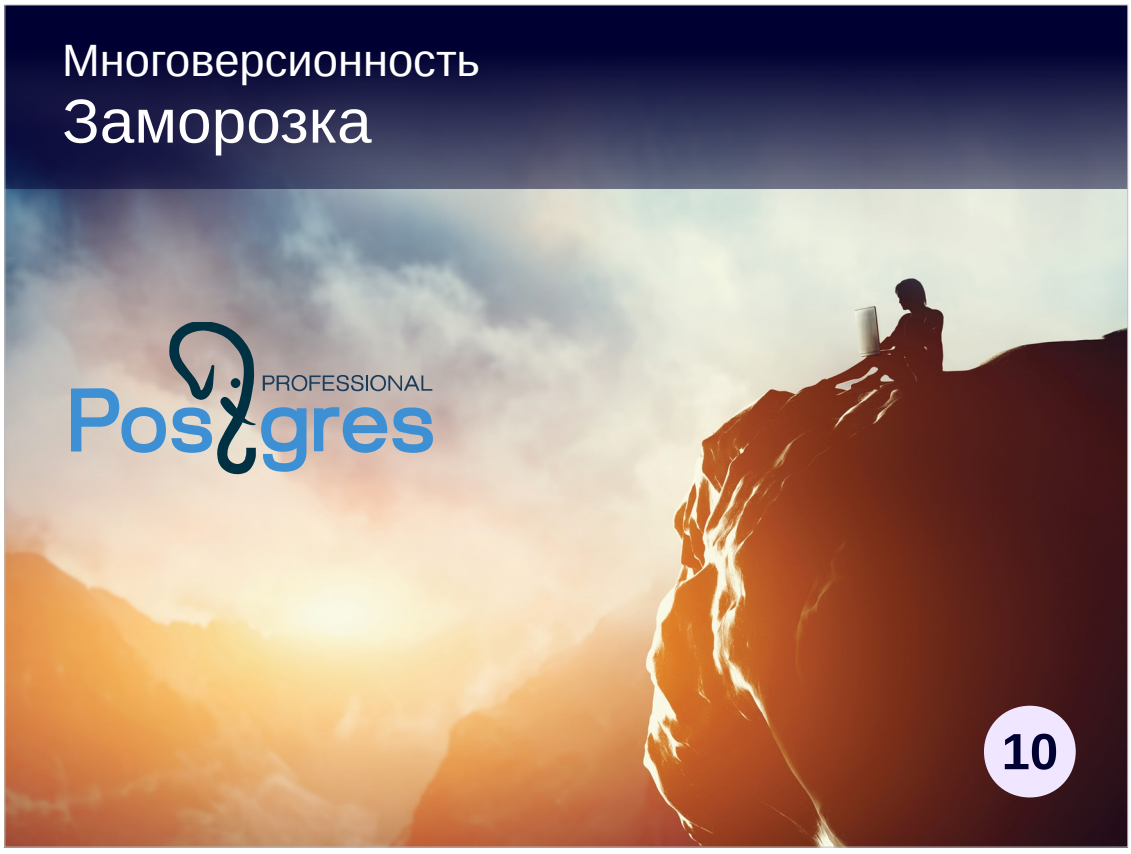


Авторы: Егор Рогов, Павел Лузанов

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

Темы

Проблема переполнения счетчика транзакций

Заморозка версий строк и правила видимости

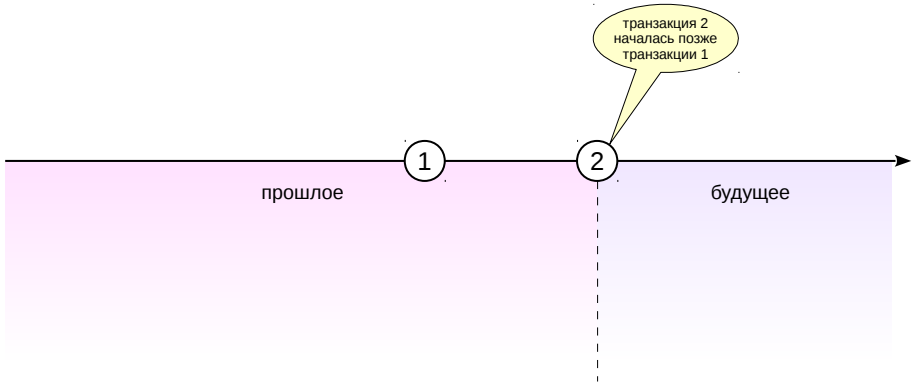
Настройка автоочистки для выполнения заморозки

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.

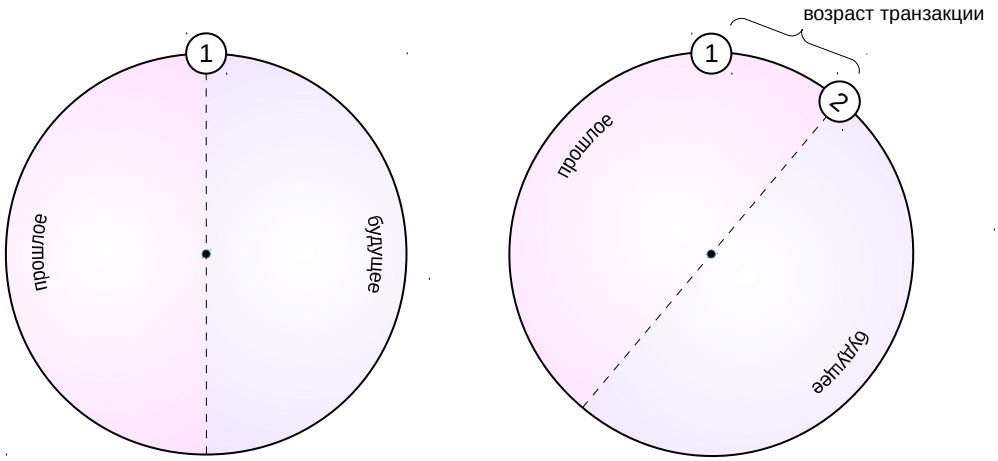
Под номер транзакции в PostgreSQL выделено 32 бита. Это довольно большое число (около 4 млрд), но при активной работе сервера оно вполне может быть исчерпано. Например при нагрузке 1000 транзакций в секунду это произойдет всего через полтора месяца непрерывной работы.

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

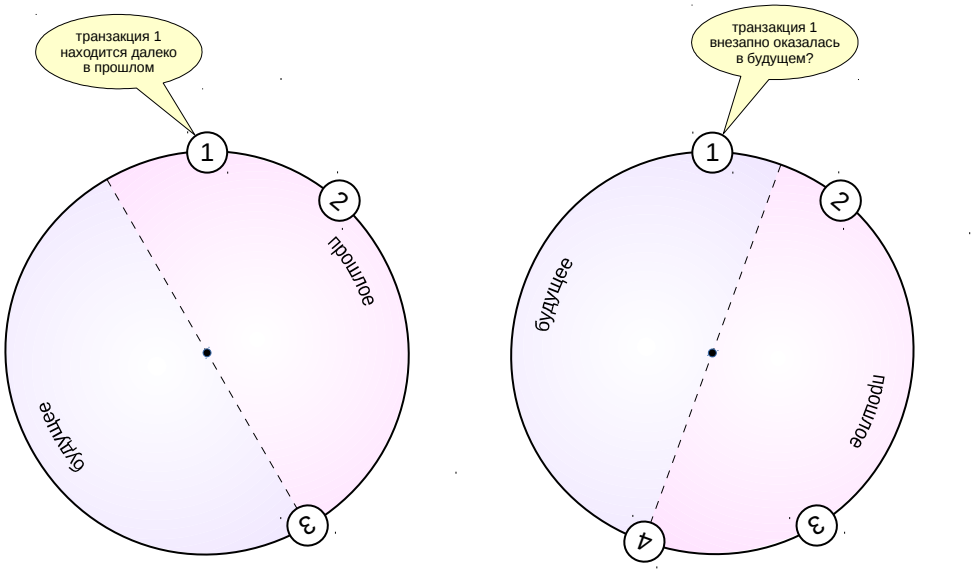
пространство номеров транзакций закольцовано
половина номеров — прошлое, половина — будущее



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).

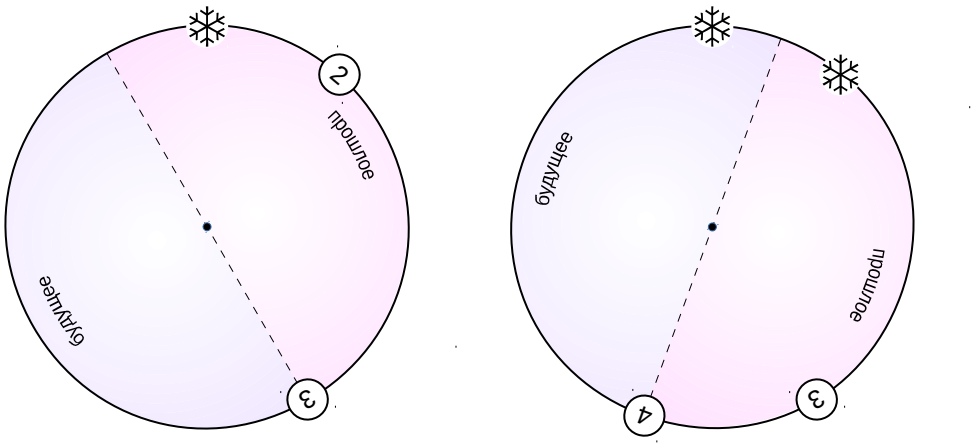
Проблема видимости



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, нарушает правила видимости и привело бы к проблемам.

Заморозка версий строк

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



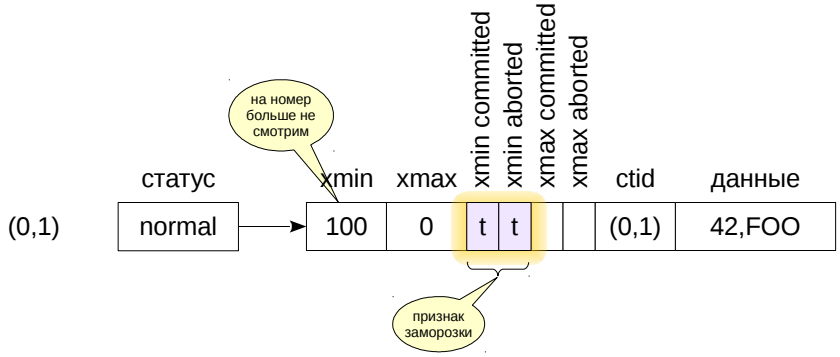
Чтобы не допустить путешествий из прошлого в будущее, процесс очистки выполняет еще одну задачу. Он находит достаточно старые и «холодные» версии строк (которые видны во всех снимках и изменение которых уже маловероятно) и специальным образом помечает — «замораживает» — их. Замороженная версия строки 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/10/routine-vacuuming#VACUUM-FOR-WRAPAROUND

Заморозка версии 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 номер транзакции xmin как замороженный, 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, которым помечаются замороженные транзакции. 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 замороженная транзакция 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 работать дальше.

Настройка

vacuum_freeze_min_age

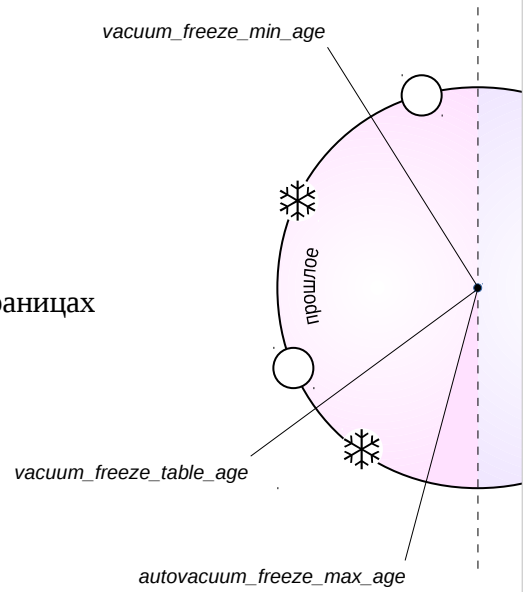
минимальный возраст,
с которого начинается заморозка

vacuum_freeze_table_age

при достижении такого возраста
замораживаются версии строк на всех страницах
используется карта заморозки

autovacuum_freeze_max_age

при достижении такого возраста
заморозка запускается принудительно
определяет размер XACT



Заморозкой управляют три основных параметра.

vacuum_freeze_min_age определяет минимальный возраст транзакции xmin, при котором версию строки можно замораживать. Чем меньше это значение, тем больше накладных расходов: версии строк могут быть «горячими» и активно меняться — их заморозка пропадет без пользы, а вместо этого придется снова замораживать уже новые версии.

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 определяет возраст транзакции, при котором пора выполнять заморозку на всех страницах. Для этого каждая таблица хранит номер транзакции, который был заморожен в прошлый раз (pg_class.relfrozenxid).

До версии 9.6 для этого выполнялось полное сканирование таблицы. Начиная с 9.6, в карту видимости была встроена *карта заморозки,* в которой отмечены страницы с замороженными версиями строк.

Так или иначе, заморозка всех страниц выполняется раз в (*vacuum_freeze_table_age* - *vacuum_freeze_min_age*) транзакций. Таким образом, большое значение *vacuum_freeze_min_age* увеличивает накладные расходы.

autovacuum_freeze_max_age определяет возраст транзакции, при котором заморозка будет выполняться принудительно — автоочистка запустится, даже если она отключена. Этот параметр также определяет размер XACT.

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. Это сделано для ограничения размера XACT (два бита на транзакцию, т. е. примерно 50 МБ при значении по умолчанию). Скорее всего, значение можно увеличивать в разы для уменьшения избыточных накладных расходов.

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 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 нарушаются правила изоляции транзакции

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

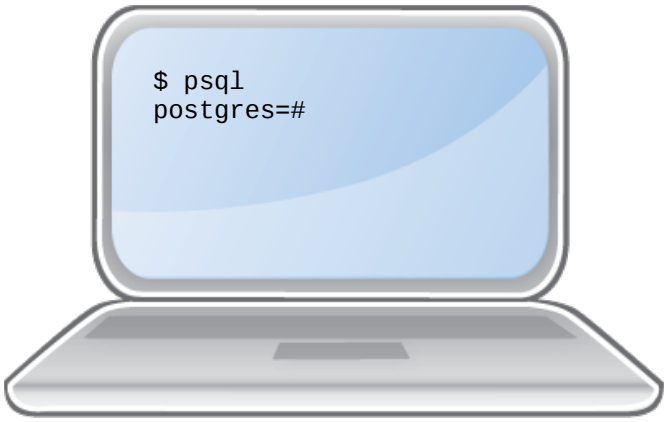
Заморозку можно вызвать вручную командой VACUUM FREEZE — при этом будут заморожены все версии строк, без оглядки на возраст транзакций (как будто параметр *autovacuum_freeze_min_age* = 0). При перестройке таблицы командами VACUUM FULL или CLUSTER все строки также замораживаются.

https://postgrespro.ru/docs/postgresql/10/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), хотя обычно не не представляет проблемы. Подробнее такой случай рассматривается в практике.

https://postgrespro.ru/docs/postgresql/10/sql-copy

Демонстрация



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/10/pgbench