Postgres PROFESSIONAL

Снимки и блокировки

Темы

Как устроен снимок

Определение принадлежности версии строки снимку

Экспорт снимка

Виртуальные транзакции

Блокировки на уровне строк

Блокировки на уровне объектов БД

Взаимоблокировки

Видимость версий строк

Видимость версии строки ограничена xmin и xmax

Версия попадает в снимок, когда

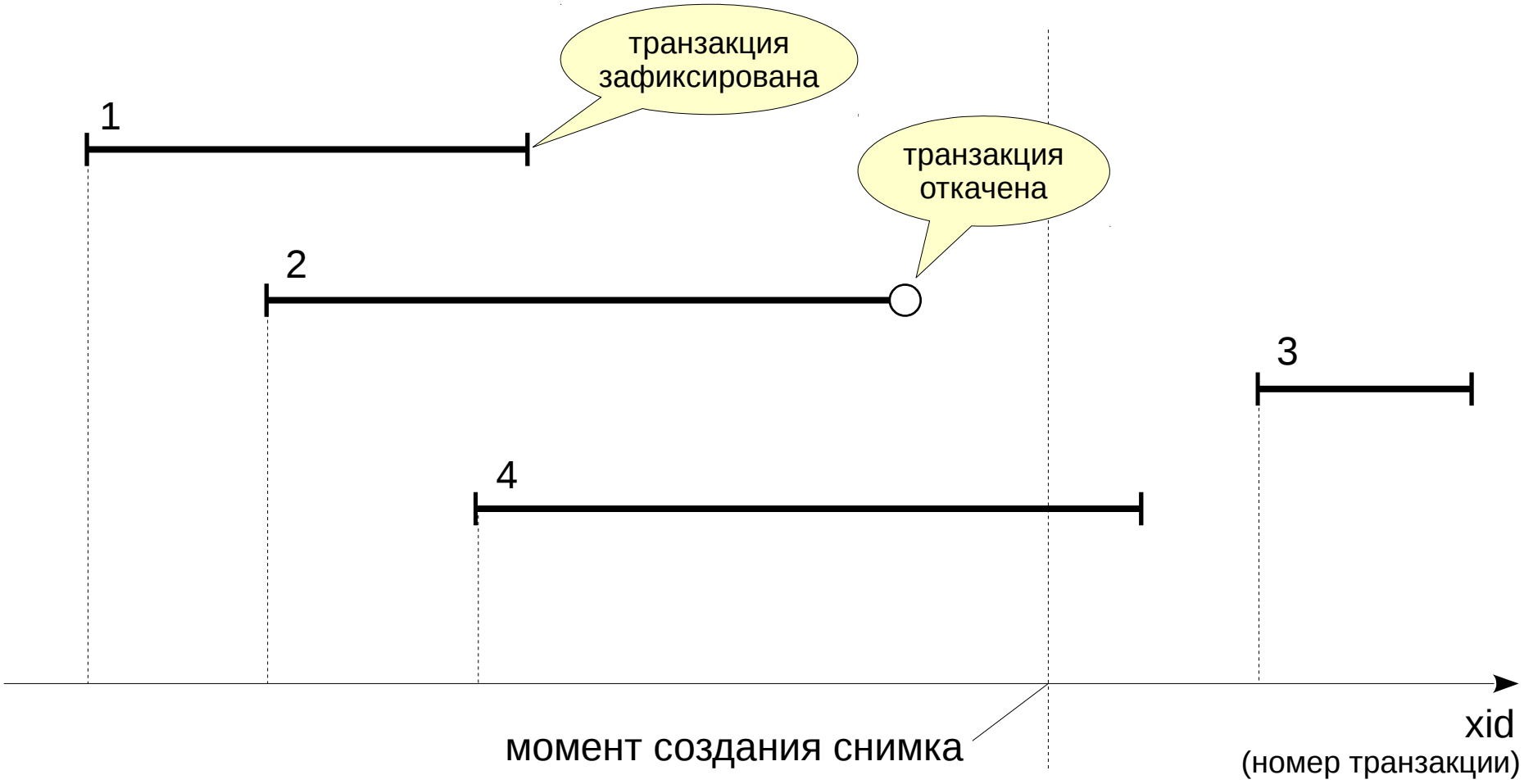
- изменения транзакции xmin видны для снимка,
- изменения транзакции xmax не видны для снимка

Изменения транзакции видны, когда

- либо это та же самая транзакция, что создала снимок,
- либо она завершилась фиксацией до момента создания снимка

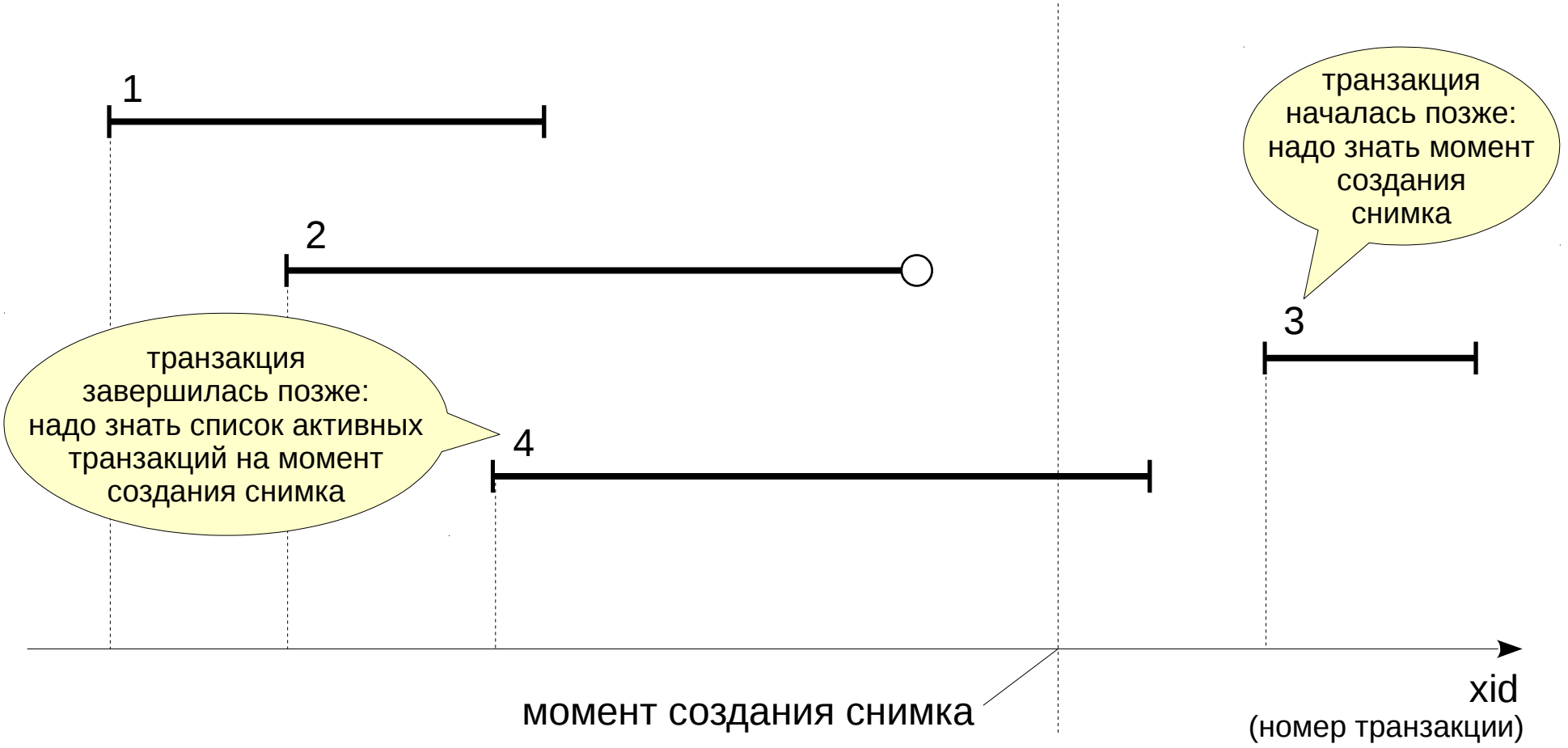
Видимость версий строк

Изменения каких транзакций видны для снимка?



Видимость версий строк

Видны изменения только первой транзакции.



Снимок

Содержит

номер следующей (еще не существующей) транзакции

список активных транзакций

Создается

READ COMMITTED — при выполнении каждого запроса

REPEATABLE READ, SERIALIZABLE — при выполнении первого запроса в транзакции

снимок можно создать только на текущий момент времени

Информация о снимке

`txid_current_snapshot()`

Экспорт снимка

Задача

работа распределена между несколькими одновременно работающими транзакциями, но они должны видеть одни и те же данные

пример: `pg_dump -j N`

Механизм

одна транзакция экспортирует снимок:
`pg_export_snapshot()`

другие транзакции используют этот снимок:
`set transaction snapshot` *`идентификатор_снимка`*`;`

Ограничения и особенности

снимок существует, пока активна создавшая его транзакция

Блокировки версий строк

Возникают

- при удалении или обновлении строк — `delete`, `update`
- при явном блокировании — `select for share/update`

Защищают

- от одновременного изменения другой транзакцией

Устройство

- поле xmax, содержащее номер активной блокирующей транзакции
- количество блокировок ничем не ограничено,
  большое число не приводит к потере производительности

Блокировки объектов БД

Возникают

при любых операциях с таблицами,
а также с индексами, представлениями, последовательностями...

при явном блокировании

Защищают (в зависимости от режима)

от изменения определения или данных объекта при его использовании

от использования изменяющегося объекта

Устройство

информация в общей памяти сервера (представление `pg_locks`)

ждущие процессы не потребляют ресурсы

количество блокировок ограничено:
max_locks_per_transaction × *max_connections*

Блокировки объектов БД

Для уровня изоляции serializable

- отдельные предикатные блокировки
  на уровне таблицы или страницы индекса
- изоляция достигается не блокированием, а отслеживанием зависимостей, которые могут привести к нарушению
- количество предикатных блокировок ограничено:
  max_pred_locks_per_transaction × *max_connections*

Блокировки объектов БД

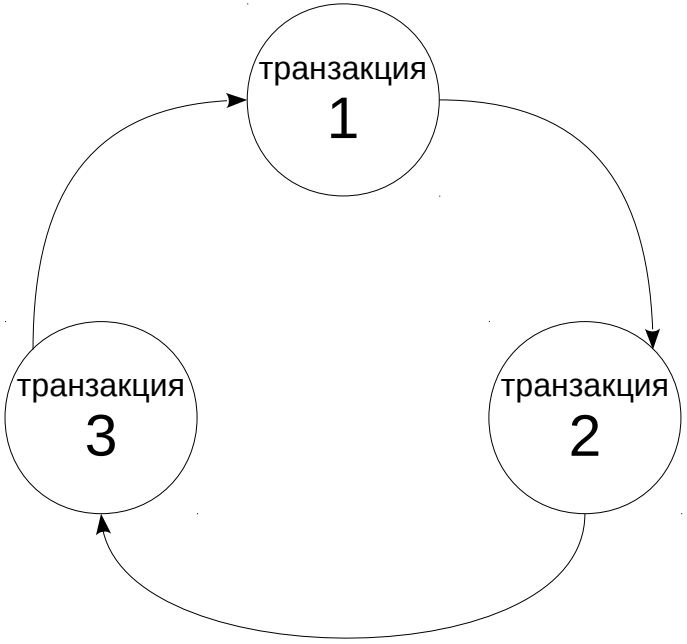
Взаимоблокировки

обнаруживаются сервером путем поиска петлей в графе ожиданий

одна из транзакций откатывается, чтобы остальные могли продолжать

проверка выполняется после ожидания *deadlock_timeout*

взаимоблокировки обычно означают, что приложение спроектировано неправильно



Мониторинг блокировок

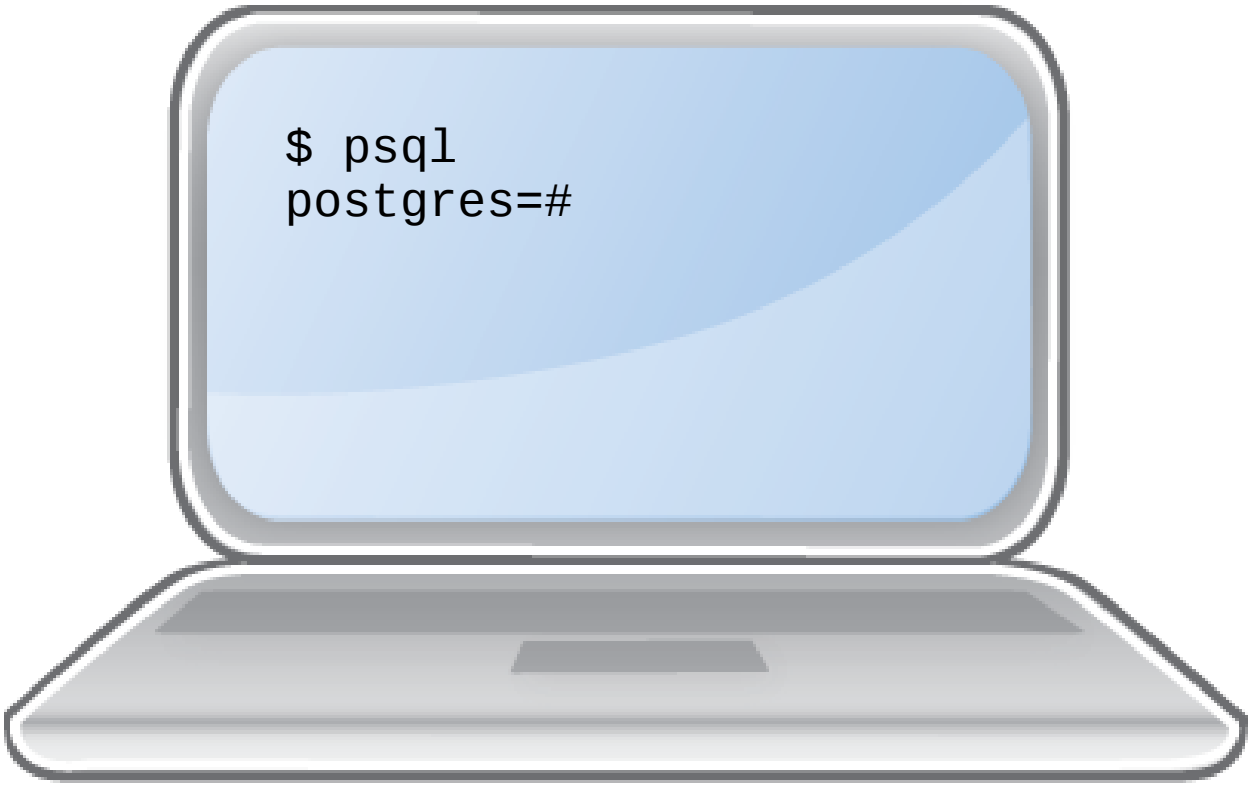
Вывод сообщений в журнал сервера

параметр *log_lock_waits:*
выводит сообщение об ожидании дольше *deadlock_timeout*

Запросы для изучения текущих блокировок

на основе представлений `pg_locks` и `pg_stat_activity`

Демонстрация



Итоги

Снимок содержит информацию, необходимую для определения принадлежности версии строки

Изоляция обеспечивается снимками и блокировками

Блокировки уровня строки находятся внутри страницы, их количество не ограничено

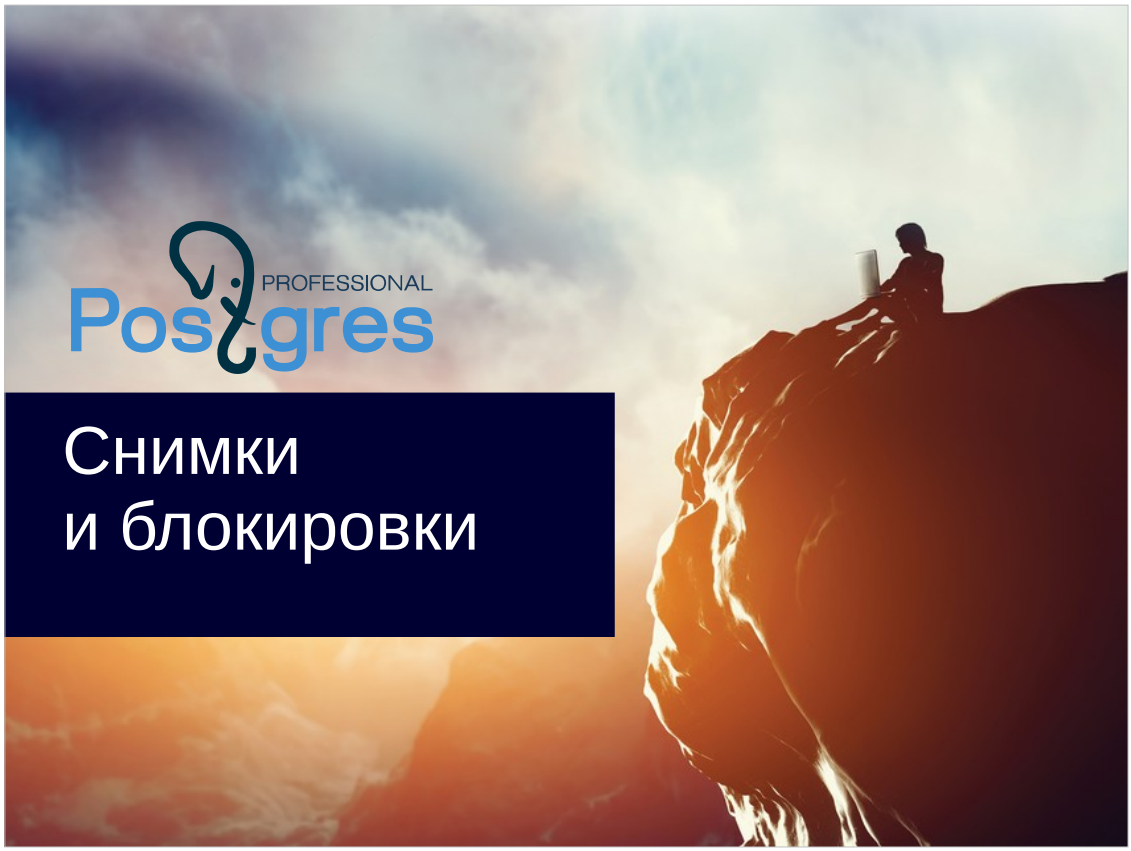
Блокировки уровня объектов БД находятся в памяти сервера и организуют очереди ожидания с обнаружением взаимоблокировок

Практика

Выполняется в базе данных DB4.

1. Воспроизвести ситуацию взаимоблокировки трех транзакций, обращающихся к таблице.
   Достаточно ли данных о возникшей ситуации содержит журнал?
2. Настроить сервер так, чтобы в журнал сбрасывалась информация о блокировках, удерживаемых более ста миллисекунд.
   Воспроизвести ситуацию, при которой в журнале появятся такие сообщения.
3. Воспроизвести ситуацию, при которой одна транзакция еще видит удаленную строку, а другая — уже нет.
   Посмотреть снимки этих транзакций и поля xmin и xmax удаленной строки; объяснить видимость на основе этих данных.

4*. Если в запросе вызывается функция на PL/pgSQL, содержащая другой запрос, то какой снимок в ней будет использоваться при уровне изоляции read committed? Проверьте.



Авторы: Егор Рогов, Павел Лузанов

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

Темы

Как устроен снимок

Определение принадлежности версии строки снимку

Экспорт снимка

Виртуальные транзакции

Блокировки на уровне строк

Блокировки на уровне объектов БД

Взаимоблокировки

Видимость версий строк

Видимость версии строки ограничена xmin и xmax

Версия попадает в снимок, когда

- изменения транзакции xmin видны для снимка,
- изменения транзакции xmax не видны для снимка

Изменения транзакции видны, когда

- либо это та же самая транзакция, что создала снимок,
- либо она завершилась фиксацией до момента создания снимка

Будет или нет данная версия строки видна в снимке, определяется двумя полями ее заголовка — xmin и xmax, — то есть номерами создавшей и удалившей транзакций. Такие интервалы не пересекаются, поэтому одна строка представлена в любом снимке максимум одной своей версией.

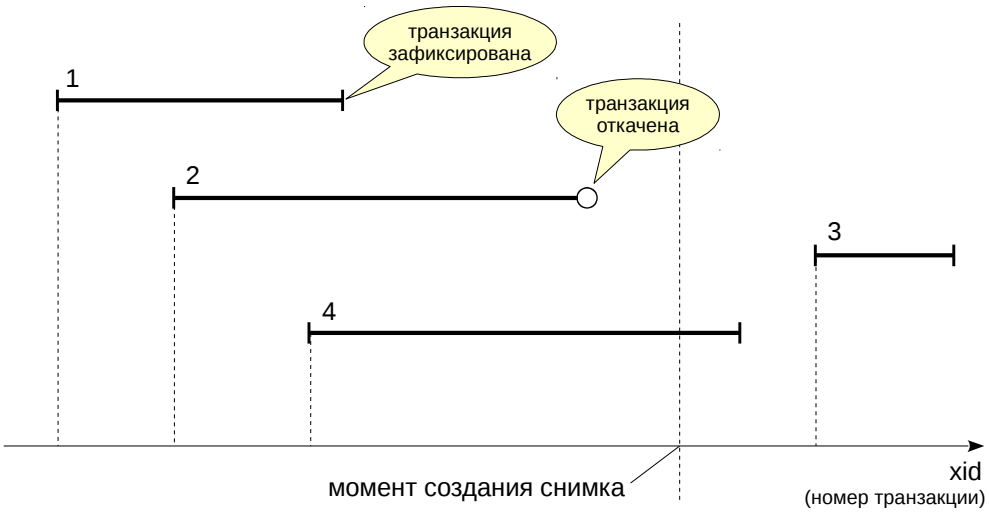
Точные правила видимости довольно сложны, поскольку учитывают всевозможные «крайние» случаи. Но в общем можно сказать, что версия строки видна, когда в снимке видны изменения, сделанные транзакцией xmin, и не видны изменения, сделанные транзакцией xmax.

В свою очередь, изменения транзакции видны в снимке, если либо это та же самая транзакция, что создала снимок (она сама видит свои же изменения), либо транзакция была зафиксирована до момента создания снимка.

Несколько усложняет картину случай определения видимости собственных изменений транзакции. Здесь может потребоваться видеть только часть таких изменений. Например, курсор, открытый в определенный момент, ни при каком уровне изоляции не должен увидеть изменений, сделанных после этого момента. Для этого в заголовке версии строки есть специальное поле (псевдостолбцы cmin и cmax), показывающее номер операции внутри транзакции, и он тоже принимается во внимание.

Видимость версий строк

Изменения каких транзакций видны для снимка?

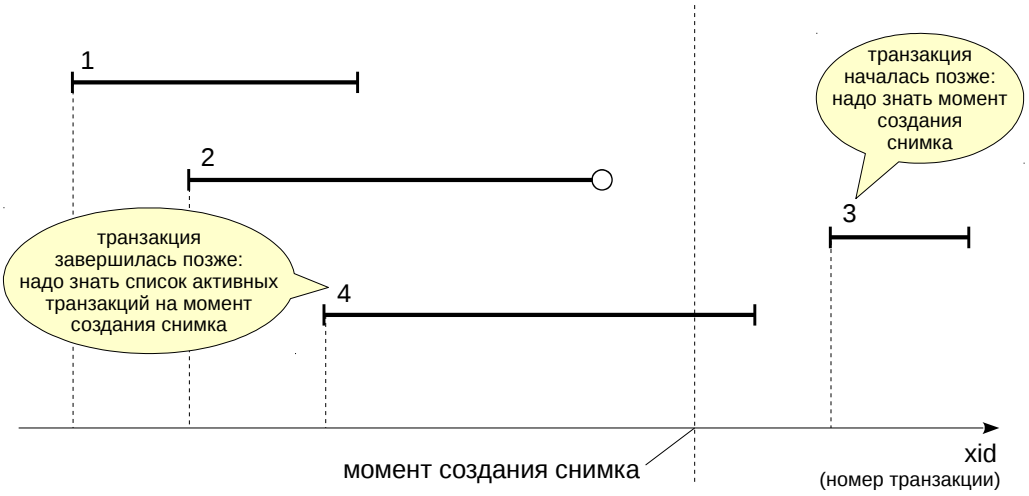


4

Рассмотрим пример. На рисунке приведены четыре транзакции, одна из которых завершилась откатом, а остальные зафиксированы.
Изменения каких транзакций будут видны в снимке, момент создания которого обозначен?

Видимость версий строк

Видны изменения только первой транзакции.



Будут видны изменения только первой транзакции, так как она завершилась фиксацией до момента создания снимка.

Вторая транзакция не видна, поскольку откачена.

Третья транзакция не видна, поскольку началась после создания снимка. Если мы знаем момент создания снимка (то есть соответствующий номер транзакции), то это нетрудно проверить.

Четверная транзакция не видна, поскольку хоть и завершилась фиксацией, но на момент создания снимка была еще не завершена. Это более сложный случай, потому что момент окончания транзакции неизвестен: ведь у нас есть только номер транзакции, а это по сути время ее начала.

Чтобы учесть четвертый случай, в снимок должен входить список активных транзакций на момент создания снимка.

Снимок

Содержит

номер следующей (еще не существующей) транзакции

список активных транзакций

Создается

READ COMMITTED — при выполнении каждого запроса

REPEATABLE READ, SERIALIZABLE — при выполнении первого запроса в транзакции

снимок можно создать только на текущий момент времени

Информация о снимке

`txid_current_snapshot()`

Итак, снимок состоит из двух частей:

1. Номер следующей (еще не существующей) транзакции — он определяет момент создания снимка.

2. Список активных транзакций на момент создания снимка.

(Конечно, в снимке также содержится и номер команды в текущей транзакции, и ряд другой вспомогательной информации.)

Из этого следует, что снимок можно создать только в текущий момент, поскольку не хранится историческая информация об активности транзакций. Нельзя, например, сейчас создать снимок, соответствующий пяти минутам назад.

В зависимости от выбранного уровня изоляции, снимок создается либо в начале каждой операции (read committed), либо в начале первой операции транзакции (repeatable read, serializable).

Стоит подчеркнуть, что снимок создается именно при выполнении запроса, а не при команде begin, как можно было бы подумать.

Информацию о снимке можно получить с помощью функции txid_current_snapshot() и нескольких других, см. http://www.postgresql.org/docs/current/static/functions-info.html

Подробнее это рассматривается в демонстрации.

Экспорт снимка

Задача

работа распределена между несколькими одновременно работающими транзакциями, но они должны видеть одни и те же данные

пример: `pg_dump -j N`

Механизм

одна транзакция экспортирует снимок:
`pg_export_snapshot()`

другие транзакции используют этот снимок:
`set transaction snapshot` *`идентификатор_снимка`*`;`

Ограничения и особенности

снимок существует, пока активна создавшая его транзакция

Бывают ситуации, когда несколько параллельных транзакций должны гарантированно видеть одну и ту же картину данных. Разумеется, нельзя полагаться на то, что картины совпадут просто из-за того, что транзакции запущены «одновременно».

Для этого есть механизм экспорта и импорта снимка Функция pg_export_snapshot() возвращает идентификатор снимка, который может быть передан (внешними средствами) в другую транзакцию. Другая транзакция может импортировать снимок с помощью команды set transaction snapshot сразу после начала транзакции (до выполнения первого запроса).

Время жизни экспортированного снимка совпадает со временим жизни экспортирующей транзакции.

http://www.postgresql.org/docs/current/static/functions-admin.html

http://www.postgresql.org/docs/current/static/sql-set-transaction.html

Блокировки версий строк

Возникают

при удалении	`delete`
или обновлении строк	`update`
при явном блокировании	`select for share/update`

Защищают

от одновременного изменения другой транзакцией

Устройство

поле xmax, содержащее номер активной блокирующей транзакции

количество блокировок ничем не ограничено,
большое число не приводит к потере производительности

Хотя при многоверсионности читатели никого не блокируют, а писатели не блокируют читателей, однако одновременное изменение одной и той же строки блокировать необходимо. Блокировки нужны только при удалении версий строк (как следствие, и при обновлениях, которые устроены как удаление и вставка). При создании новых версий блокировка не требуется (такая версия будет невидима для других транзакций).

При удалении версии строки в поле xmax записывается номер удаляющей транзакции. Это поле и является признаком блокировки.

Число таких блокировок 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, что обсуждалось в прошлой теме).

Также блокировка потребуется и при ручном блокировании строки при помощи select for update или select for no key update (эксклюзивная блокировка). Она также реализуется установкой поля xmax, но с признаком, что это только блокировка, а не удаление.

Существует также разделяемая блокировка с помощью select for share или select for key share. Ее особенность в том, что блокировку могут захватывать несколько транзакций. Подробнее этот случай рассматривается в демонстрации.

http://www.postgresql.org/docs/current/static/explicit-locking.html

Блокировки объектов БД

Postgres PROFESSIONAL

Возникают

- при любых операциях с таблицами,
  а также с индексами, представлениями, последовательностями...
- при явном блокировании

Защищают (в зависимости от режима)

- от изменения определения или данных объекта при его использовании
- от использования изменяющегося объекта

Устройство

- информация в общей памяти сервера (представление `pg_locks`)
- ждущие процессы не потребляют ресурсы
- количество блокировок ограничено:
  max_locks_per_transaction × *max_connections*

Помимо блокировок строк, есть необходимость и в блокировках на более высоких уровнях, в частности, на уровне таблиц, индексов и других объектов (relations). Такие блокировки защищают объекты от одновременного изменения или от использования в то время, когда объект изменяется. PostgreSQL использует восемь различных уровней блокировок, чтобы порождать как можно меньше ожиданий. Некоторые примеры будут приведены в демонстрации, однако подробно рассматривать различные уровни мы не будем.

Блокировки объектов располагаются в общей памяти сервера. Поэтому их количество ограничено двумя параметрами: *max_locks_per_transaction* и *max_connections*. Пул блокировок общий для всех транзакций, то есть одна транзакция может захватить больше блокировок, чем *max_locks_per_transaction* — важно лишь, чтобы общее число блокировок в системе не превысило установленный предел.

Важный момент: ожидающие блокировок транзакции не потребляют ресурсы процессора. Они встают в очередь и «засыпают», и пробуждаются только когда ресурс освобождается.

Блокировки можно посмотреть с помощью представления pg_locks.

http://www.postgresql.org/docs/current/static/runtime-config-locks.html

Блокировки объектов БД

Для уровня изоляции serializable

- отдельные предикатные блокировки
  на уровне таблицы или страницы индекса
- изоляция достигается не блокированием, а отслеживанием
  зависимостей, которые могут привести к нарушению
- количество предикатных блокировок ограничено:
  max_pred_locks_per_transaction × *max_connections*

Уровень изоляции serializable обеспечивается специальным механизмом, который (в дополнение к обычному многоверсионному снимку) отслеживает зависимости между транзакциями, которые могут привести к нарушению изоляции.

Зависимости проверяются на основе специальных предикатных блокировок (SIReadLocks), устанавливаемых на уровне страниц или объектов. Их число также ограничено, но они используют другую область памяти, ограниченную параметрами *max_pred_locks_per_transaction* и *max_connections*.

http://www.postgresql.org/docs/9.4/static/transaction-iso.html

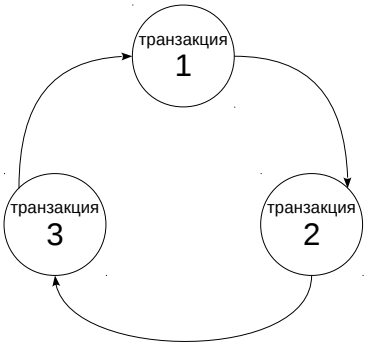
https://wiki.postgresql.org/wiki/SSI

Блокировки объектов БД

Взаимоблокировки

обнаруживаются сервером путем поиска петлей в графе ожиданий
одна из транзакций откатывается, чтобы остальные могли продолжать
проверка выполняется после ожидания *deadlock_timeout*

взаимоблокировки обычно означают, что приложение спроектировано неправильно



Возможна ситуация взаимоблокировки, когда одна транзакция пытается захватить ресурс, уже захваченные другой транзакцией, в то время, как другая транзакция пытается захватить ресурс, захваченный первой. Взаимоблокировка возможна и при нескольких транзакциях. Визуально ее удобно представлять как петлю в графе ожиданий.

PostgreSQL автоматически отслеживает взаимоблокировки, если транзакция ожидает освобождение ресурса дольше, чем *deadlock_timeout*. Если такая ситуация обнаруживается, одна из транзакций принудительно откатывается, чтобы остальные могли продолжить работу.

Взаимоблокировки обычно означают, что приложение спроектировано неправильно. Если в журнале сервера появляются сообщения о взаимоблокировках, это повод задуматься о причинах.

http://www.postgresql.org/docs/current/static/explicit-locking.html

Мониторинг блокировок

Вывод сообщений в журнал сервера

параметр *log_lock_waits:*
выводит сообщение об ожидании дольше *deadlock_timeout*

Запросы для изучения текущих блокировок

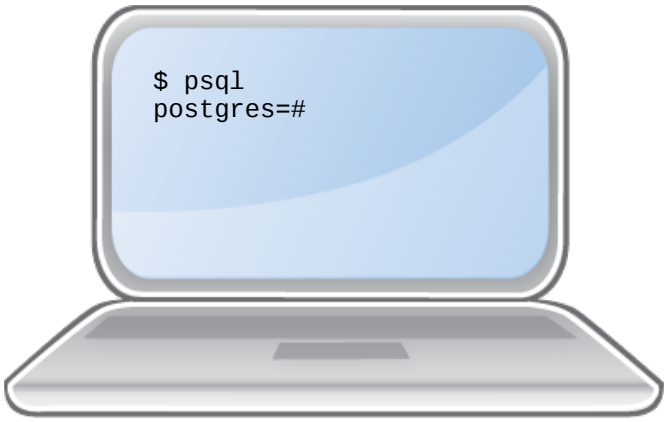
на основе представлений `pg_locks` и `pg_stat_activity`

Возникающие в системе блокировки необходимы для обеспечения целостности и изоляции, однако могут приводить к нежелательным ожиданиям. Такие ожидания можно отслеживать, чтобы разобраться в их причине и по возможности устранить (например, изменив алгоритм работы приложения).

Один способ состоит в том, чтобы включить параметр *log_lock_waits*. В этом случае в журнал сообщений сервера будет попадать информация, если транзакция ждала дольше, чем *deadlock_timeout*.

Второй способ состоит в том, чтобы в момент возникновения долгой блокировки (или на периодической основе) выполнять запрос к представлению pg_locks, расшифровывая блокирующие и блокируемые транзакции при помощи pg_stat_activity. Примеры таких запросов можно найти здесь: https://wiki.postgresql.org/wiki/Lock Monitoring

Демонстрация



Итоги

Снимок содержит информацию, необходимую для определения принадлежности версии строки

Изоляция обеспечивается снимками и блокировками

Блокировки уровня строки находятся внутри страницы, их количество не ограничено

Блокировки уровня объектов БД находятся в памяти сервера и организуют очереди ожидания с обнаружением взаимоблокировок

Практика

Выполняется в базе данных DB4.

1. Воспроизвести ситуацию взаимоблокировки трех транзакций, обращающихся к таблице.
   Достаточно ли данных о возникшей ситуации содержит журнал?
2. Настроить сервер так, чтобы в журнал сбрасывалась информация о блокировках, удерживаемых более ста миллисекунд.
   Воспроизвести ситуацию, при которой в журнале появятся такие сообщения.
3. Воспроизвести ситуацию, при которой одна транзакция еще видит удаленную строку, а другая — уже нет.
   Посмотреть снимки этих транзакций и поля xmin и xmax удаленной строки; объяснить видимость на основе этих данных.

4*. Если в запросе вызывается функция на PL/pgSQL, содержащая другой запрос, то какой снимок в ней будет использоваться при уровне изоляции read committed? Проверьте.

4. Для написания функции можно воспользоваться следующим шаблоном:

```
create function test() returns integer as $$
declare
  cnt integer;
begin
  select count(*) into cnt from t;
  return cnt;
end;
$$ language plpgsql;
```

Также может пригодиться функция pg_sleep(n), вызывающая задержку на n секунд.