Авторские права
© Postgres Professional, 2019 год.
Авторы: Егор Рогов, Павел Лузанов

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

Отказ от ответственности
Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или непрямым, 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.

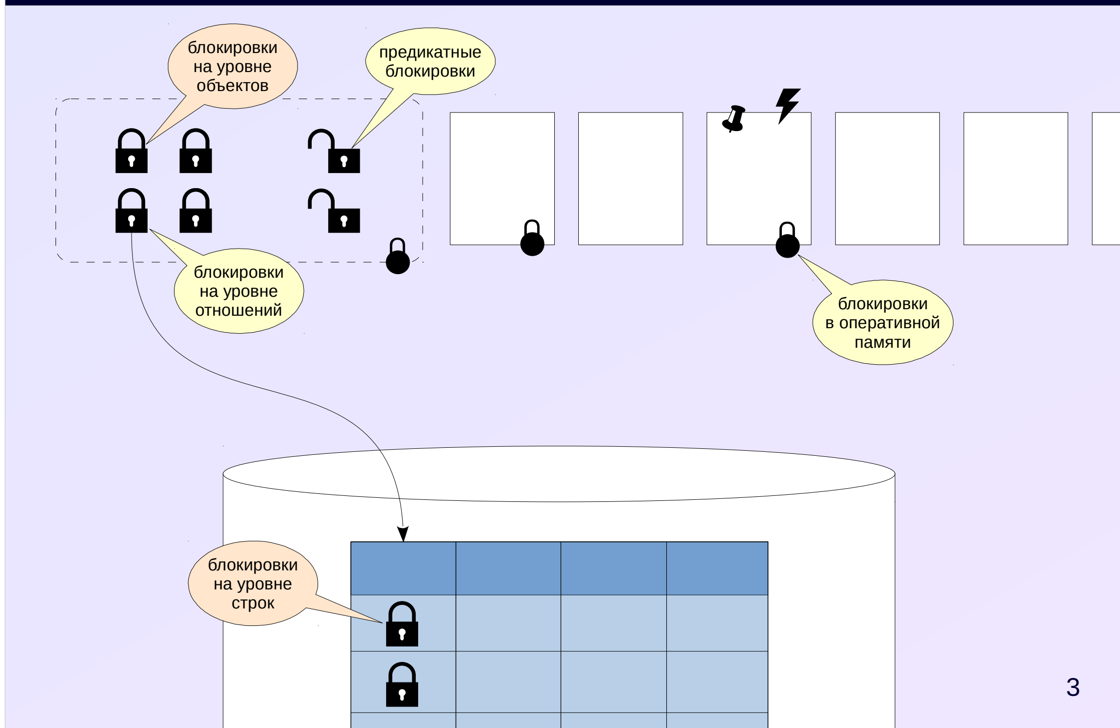
Темы

Исключительные и разделяемые блокировки строк

Мультитранзакции и заморозка

Реализация очереди ожидания

Взаимоблокировки


Блокировки строк
PROFESSIONAL
Postgres
блокировки
на уровне
объектов
предикатные
блокировки
блокировки
на уровне
отношений
блокировки
в оперативной
памяти
блокировки
на уровне
строк
3

Устройство

Postgres PROFESSIONAL

Информация *только* в страницах данных

поле `xmax` заголовка версии строки + информационные биты

в оперативной памяти ничего не хранится

Неограниченное количество

большое число не влияет на производительность

Инфраструктура

очередь ожидания организована с помощью блокировок объектов

4

В случае блокировок объектов, рассмотренных в предыдущей теме этого модуля, каждый ресурс представлен собственной блокировкой в оперативной памяти. Со строками так не получается: заведение отдельной блокировки для каждой табличной строки (которых могут быть миллионы и миллиарды) потребует непомерных накладных расходов и огромного объема оперативной памяти. А если повышать уровень блокировок, то будет страдать пропускная способность.

Поэтому в PostgreSQL информация о том, что строка заблокирована, хранится только и исключительно в версии строки внутри страницы данных. Там она представлена номером блокирующей транзакции (xmax) и дополнительными информационными битами.

За счет этого блокировок уровня строки может быть неограниченное количество. Это не приводит к потреблению каких-либо ресурсов и не снижает производительность системы.

Обратная сторона такого подхода — сложность организации очереди ожидания. Для этого все-таки приходится использовать блокировки более высокого уровня, но удается обойтись очень небольшим их количеством (пропорциональным числу процессов, а не числу заблокированных строк). И это так же не снижает производительность системы.

Исключительный режим

UPDATE

удаление строки
или изменение всех полей

`SELECT FOR UPDATE`

`UPDATE`
(с изменением ключевых полей)

`DELETE`

NO KEY UPDATE

изменение любых полей,
кроме ключевых

`SELECT FOR NO KEY UPDATE`

`UPDATE`
(без изменения ключевых полей)

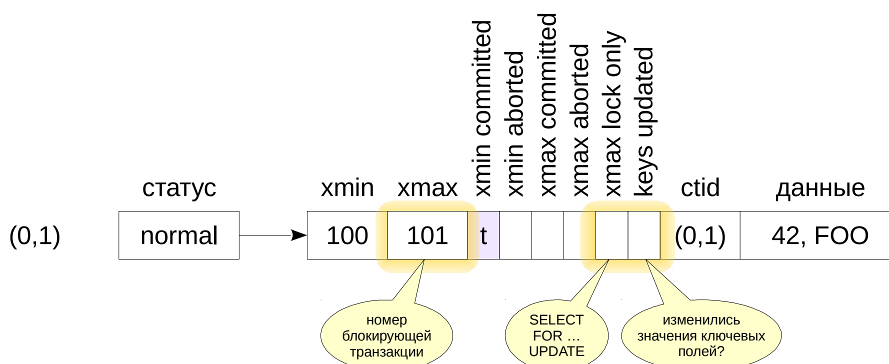
Всего существует 4 режима, в которых можно заблокировать строку (в версии строки режим проставляется с помощью дополнительных информационных битов).

Два режима представляют исключительные (exclusive) блокировки, которые одновременно может удерживать только одна транзакция. Режим UPDATE предполагает полное изменение (или удаление) строки, а режим NO KEY UPDATE — изменение только тех полей, которые не входят в уникальные индексы (иными словами, при таком изменении все внешние ключи остаются без изменений).

Команда UPDATE сама выбирает минимальный подходящий режим блокировки; обычно строки блокируются в режиме NO KEY UPDATE.

https://postgrespro.ru/docs/postgresql/10/explicit-locking#LOCKING-ROWS

Исключительный режим



При изменении или удалении строки в поле xmax актуальной версии записывается номер текущей транзакции. Установленное значение xmax, соответствующее активной транзакции, выступает в качестве блокировки.

То же самое происходит и при явном блокировании строки командой SELECT FOR UPDATE, но проставляется дополнительный информационный бит (xmax lock only), который говорит о том, что версия строки по-прежнему актуальна, хоть и заблокирована.

Режим блокировки определяется еще одним информационным битом (keys updated).

(На самом деле используется большее количество битов с более сложными условиями и проверками, что связано с поддержкой совместимости с предыдущими версиями. Но это не принципиально.)

Если другая транзакция намерена обновить или удалить заблокированную строку в несовместимом режиме, она будет вынуждена дождаться завершения транзакции с номером xmax.

Разделяемый режим

PostgreSQL Professional

SHARE

запрет изменения
любых полей строки

`SELECT FOR SHARE`

KEY SHARE

запрет изменения
ключевых полей строки

`SELECT FOR KEY SHARE`
и проверка внешних ключей

Матрица совместимости режимов

	KEY SHARE	SHARE	NO KEY UPDATE	UPDATE
KEY SHARE				×
SHARE			×	×
NO KEY UPDATE		×	×	×
UPDATE	×	×	×	×

Еще два режима представляют разделяемые (shared) блокировки, которые могут удерживаться несколькими транзакциями.

Режим SHARE применяется, когда нужно прочитать строку, но при этом нельзя допустить, чтобы она как-либо изменилась другой транзакцией.

Режим KEY SHARE допускает изменение строки, но только неключевых полей. Этот режим, в частности, автоматически используется PostgreSQL при проверке внешних ключей.

Общая матрица совместимости режимов приведена внизу слайда.
Из нее видно, что:

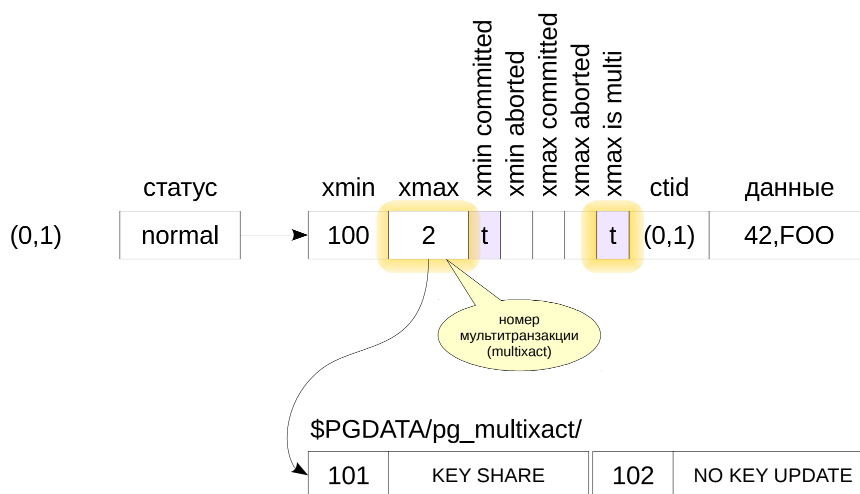
- исключительные режимы конфликтуют между собой;

- разделяемые режимы совместимы между собой;

- разделяемый режим KEY SHARE совместим с исключительным режимом NO KEY UPDATE (то есть можно обновлять неключевые поля и быть уверенным в том, что ключ не изменится).

https://postgrespro.ru/docs/postgresql/10/explicit-locking#LOCKING-ROWS

Разделяемый режим



Мы говорили о том, что блокировка представляется номером блокирующей транзакции в поле xmax. Разделяемые блокировки могут удерживаться несколькими транзакциями, но в одно поле xmax нельзя записать несколько номеров.

Поэтому для разделяемых блокировок применяются так называемые *мультитранзакции* (MultiXact). Им выделяются отдельные номера, которые соответствуют не одной транзакции, а целой группе. Чтобы отличить мультитранзакцию от обычной, используется еще один информационный бит (xmax is multi), а детальная информация об участниках такой группы и режимах блокировки находятся в каталоге $PGDATA/pg_multixact/. Естественно, последние использованные данные хранятся в буферах в общей памяти сервера для ускорения доступа.

Настройка заморозки

PostgreSQL PROFESSIONAL

Параметры для мультитранзакций

vacuum_multixact_freeze_min_age = 5 000 000
vacuum_multixact_freeze_table_age = 150 000 000

autovacuum_multixact_freeze_max_age = 400 000 000

Параметры хранения таблиц

autovacuum_multixact_freeze_min_age
toast.autovacuum_multixact_freeze_min_age

vacuum_multixact_freeze_table_age
toast.vacuum_multixact_freeze_table_age

autovacuum_multixact_freeze_max_age
toast.autovacuum_multixact_freeze_max_age

Поскольку для мультитранзакций выделяются отдельные номера, которые записываются в поле xmax версий строк, то из-за ограничения разрядности счетчика с ними возникают такие же сложности, как и с обычным номером транзакции. Речь идет о проблеме переполнения (xid wraparound), которая рассматривалась в теме «Заморозка» модуля «Многоверсионность».

Поэтому для номеров мультитранзакций тоже необходимо выполнять аналог заморозки — старые номера multixactid заменяются на новые (или на обычный номер, если в текущий момент блокировка уже удерживается только одной транзакцией).

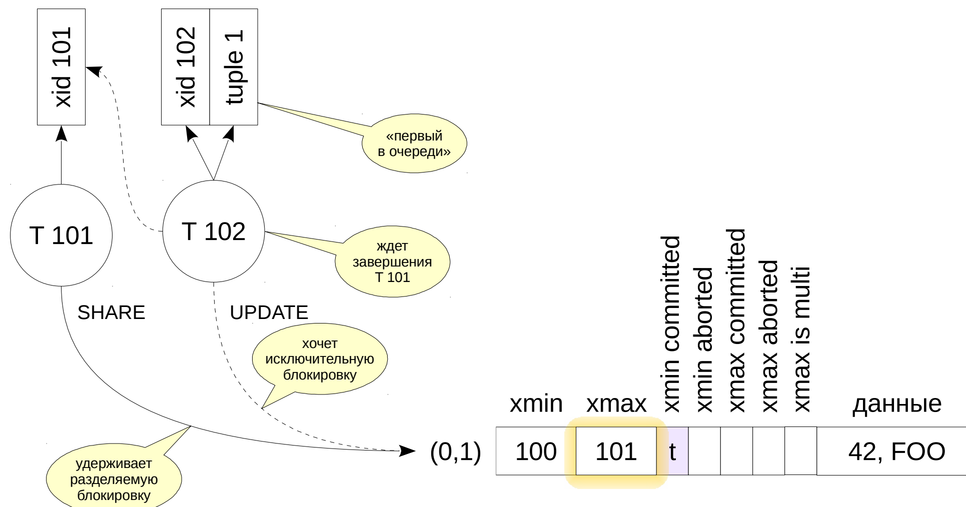
Заметим, что заморозка «обычных» номеров транзакций выполняется для поля xmin (если у версии строки непустое поле xmax, то либо это уже неактуальная версия и она будет очищена, либо транзакция xmax отменена и ее номер нас не интересует). А для мультитранзакций речь идет о поле xmax актуальной версии строки, которая может оставаться актуальной, но при этом постоянно блокироваться разными транзакциями в разделяемом режиме.

За заморозку мультитранзакций отвечают параметры, аналогичные параметрам обычной заморозки.

«Очередь» ожидания

Postgres PROFESSIONAL

Блокировки строк не представлены в памяти,
взамен используется блокировка номера транзакции



Как мы рассмотрели, транзакция блокирует строку, проставляя в поле xmax свой номер (101 в примере на слайде). Другая транзакция (102), желая получить блокировку, обращается к строке и видит, что строка заблокирована. Если режимы блокировок конфликтуют, она должна каким-то образом «встать в очередь», чтобы система «разбудила» ее, когда блокировка освободится. Но блокировки на уровне строк не предоставляют такой возможности — они никак не представлены в оперативной памяти, это просто байты внутри страницы данных.

Поэтому используется своего рода трюк. Как мы видели в теме «Блокировки объектов», каждая транзакция удерживает исключительную блокировку своего номера. Поскольку транзакция 102 фактически должна дождаться завершения транзакции 101 (ведь блокировка строки освобождается только при завершении транзакции), она запрашивает блокировку номера 101.

(Здесь и далее на рисунках сплошной линией показаны захваченные блокировки, а пунктирной — блокировки, которые еще не удалось захватить.)

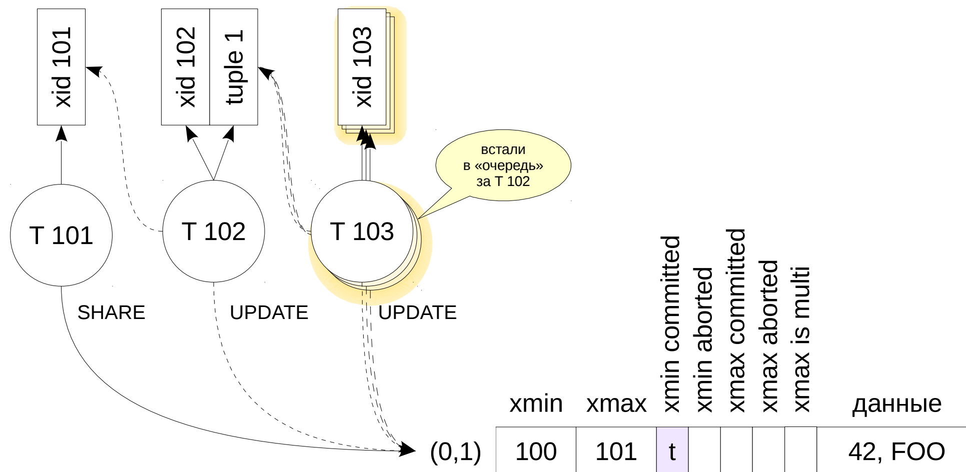
Когда транзакция 101 завершится, заблокированный ресурс освободится (фактически — просто исчезнет), транзакция 102 будет разбужена и сможет заблокировать строку (установить xmax = 102 в соответствующей версии строки).

Кроме того, ожидающая транзакция удерживает блокировку типа tuple (версия строки), показывая, что она первая в «очереди».

«Очередь» ожидания

Postgres PROFESSIONAL

Транзакции, конфликтующие с текущей блокировкой, встают за транзакцией, которая удерживает блокировку типа tuple



11

Если появляются другие транзакции, конфликтующие с текущей блокировкой строки, первым делом они пытаются захватить блокировку типа tuple для этой строки.

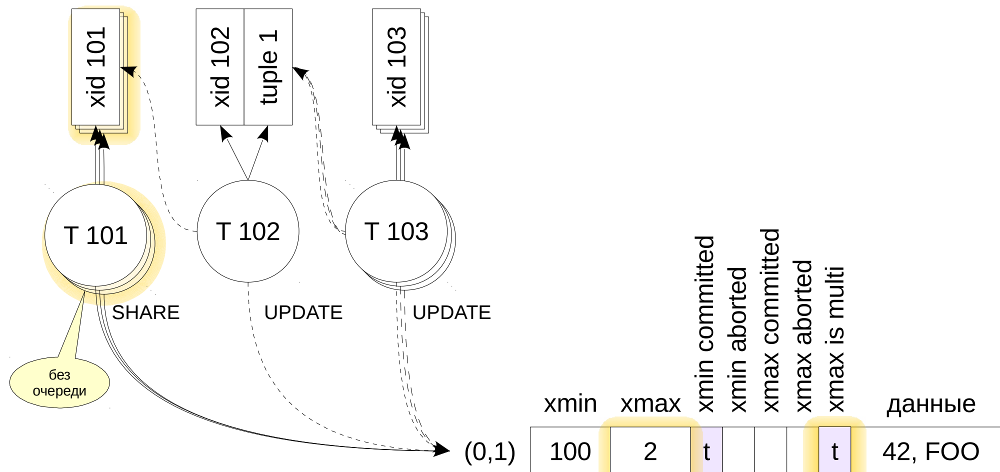
В нашем примере, поскольку блокировка уже удерживается транзакцией 102, другие транзакции ждут освобождения этой блокировки. Получается своеобразная «очередь», в которой есть *первый* и *все остальные*.

Когда транзакция 101 завершится, транзакция 102 получит возможность первой записать свой номер в поле xmax, после чего она освободит блокировку tuple. Тогда одна случайная транзакция из *всех остальных* успеет захватить блокировку tuple и станет *первой*.

«Очередь» ожидания

Postgres PROFESSIONAL

Транзакции, не конфликтующие с текущей блокировкой,
проходят без очереди



12

Изначальная идея такой двухуровневой схемы блокирования состояла в том, чтобы избежать ситуации вечного ожидания «невезучей» транзакции. Тем не менее, такая ситуация вполне возможна.

В нашем примере транзакция 101 заблокировала строку в разделяемом (SHARE) режиме. Пока транзакция 102 ожидает завершения транзакции 101, может появиться еще одна транзакция, желающая получить блокировку строки в разделяемом режиме, совместимом с блокировкой транзакции 101. Ничто не мешает ей работать со строкой. Поэтому она формирует мультитранзакцию, состоящую из нее самой и транзакции 101, и записывает номер мультитранзакции в xmax.

Когда транзакция 101 завершится, транзакция 102 будет разбужена, но не сможет заблокировать строку, поскольку теперь строка заблокирована другой транзакцией. Транзакция 102 будет вынуждена снова «заснуть». При постоянном потоке разделяемых блокировок пишущая транзакция может ждать своей очереди бесконечно. Это ситуация называется по-английски locker starvation.

Заметим, что такой проблемы в принципе не возникает при блокировках объектов (таких, как отношения). В этом случае каждый ресурс представлен собственной блокировкой, и все ждущие процессы выстраиваются в «честную» очередь.

https://git.postgresql.org/gitweb/?p=postgresql.git;a=blob;f=src/backend/access/heap/README.tuplock;hb=HEAD

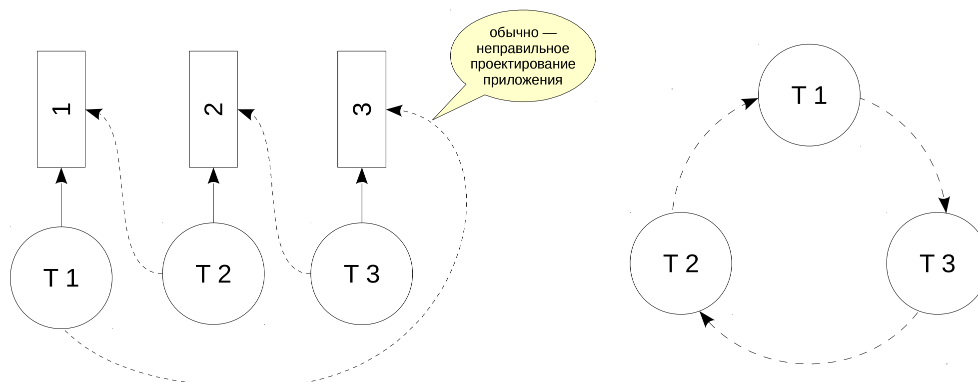
Взаимоблокировки

PostgresPROFESSIONAL

Обнаруживаются поиском контуров в графе ожиданий

проверка выполняется после ожидания *deadlock_timeout*

Одна из транзакций обрывается, остальные продолжают



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: на слайде показан пример такой ситуации для трех транзакций.

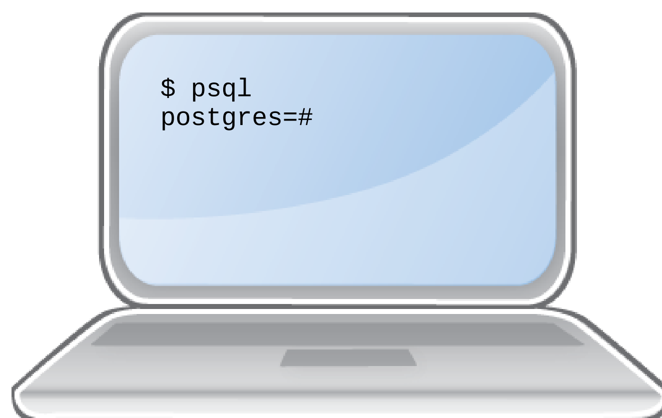
Визуально взаимоблокировку удобно представлять, построив граф ожиданий. Для этого мы убираем конкретные ресурсы и оставляем только транзакции, отмечая, какая транзакция какую ожидает. Если в графе есть контур (из некоторой вершины можно по стрелкам добраться до нее же самой) — это взаимоблокировка.

Если взаимоблокировка возникла, участвующие транзакции уже не могут ничего с этим сделать — они будут ждать бесконечно. Поэтому PostgreSQL автоматически отслеживает взаимоблокировки. Проверка выполняется, если какая-либо транзакция ожидает освобождения ресурса дольше, чем указано в параметре *deadlock_timeout*. Если выявлена взаимоблокировка, одна из транзакций принудительно прерывается, чтобы остальные могли продолжить работу.

Взаимоблокировки обычно означают, что приложение спроектировано неправильно. Сообщения в журнале сервера или увеличивающееся значение pg_stat_database.deadlocks — повод задуматься о причинах.

https://postgrespro.ru/docs/postgresql/10/explicit-locking#LOCKING-DEADLOCKS

Демонстрация



Итоги

Postgres PROFESSIONAL

Блокировки строк хранятся в страницах данных

- из-за потенциально большого количества

Очереди и обнаружение взаимоблокировок обеспечиваются блокировками объектов

- приходится прибегать к сложным схемам блокирования

Практика

PostgresPROFESSIONAL

1. Смоделируйте ситуацию обновления одной и той же строки тремя командами UPDATE в разных сеансах.
   Изучите возникшие блокировки в представлении pg_locks и убедитесь, что все они понятны.
2. Воспроизведите взаимоблокировку трех транзакций.
   Можно ли разобраться в ситуации постфактум, изучая журнал сообщений?
3. Могут ли две транзакции, выполняющие единственную команду UPDATE одной и той же таблицы, заблокировать друг друга? Попробуйте воспроизвести такую ситуацию.