

Блокировки Блокировки объектов



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.

Общая информация о блокировках

Блокировки отношений и других объектов

Предикатные блокировки

Задача и механизм использования блокировок

Блокируемые ресурсы

Факторы, влияющие на эффективность

Время жизни блокировок

Задача: упорядочение конкурентного доступа к разделяемым ресурсам

Механизм

перед обращением к данным процесс захватывает блокировку, после обращения — освобождает
блокировки приводят к очередям

Альтернативы

многоверсионность — несколько версий данных
оптимистичные блокировки — процессы не блокируются, но при неудачном стечении обстоятельств возникает ошибка

Блокировки используются, чтобы упорядочить конкурентный доступ к разделяемым ресурсам.

Под конкурентным доступом понимается одновременный доступ нескольких процессов. Сами процессы могут выполняться как параллельно (если позволяет аппаратура), так и последовательно в режиме деления времени.

Блокировки не нужны, если нет конкуренции (одновременно к данным обращается только один процесс) или если нет разделяемого ресурса (например, общий буферный кэш нуждается в блокировках, а локальный — нет).

Перед тем, как обратиться к ресурсу, защищенному блокировкой, процесс должен захватить эту блокировку. После того, как ресурс больше не нужен процессу, он освобождает блокировку, чтобы ресурсом могли воспользоваться другие процессы.

Захват блокировки возможен не всегда: ресурс может оказаться уже занятым кем-то другим. Тогда процесс либо встает в очередь ожидания, либо повторяет попытку захвата блокировки через определенное время. Так или иначе это приводит к тому, что процесс вынужден простаивать в ожидании освобождения блокировки.

Иногда удастся применить другие, неблокирующие, стратегии. Например, в одноименном модуле мы обсуждали механизм многоверсионности. Еще один пример — оптимистичные блокировки, которые не блокируют процесс, но в случае неудачи приводят к ошибке.

Ресурс

все, что можно идентифицировать

Примеры ресурсов

реальные хранимые объекты: страницы, таблицы, строки и т. п.
структуры данных в общей памяти (хеш-таблицы, буферы...)
абстрактные ресурсы (число)

Ресурсом, защищаемым блокировкой, в принципе может быть все, что угодно, лишь бы ресурс можно было однозначно идентифицировать.

Например, ресурсом может быть объект, с которым работает СУБД, такой как страница данных (идентифицируется именем файла и позицией внутри файла), таблица (oid в системной каталоге), табличная строка (страница и смещение внутри страницы).

Ресурсом может быть структура в памяти, такая как хеш-таблица, буфер и т. п. (идентифицируется заранее присвоенным номером).

Иногда бывает удобно использовать даже абстрактные ресурсы, не имеющие никакого физического смысла (идентифицируются числом).

Гранулярность блокировки

степень детализации, уровень в иерархии ресурсов
например: таблица → страница → строки, хеш-таблица → корзины
выше гранулярность — больше возможностей для параллелизма

Режимы блокировок

совместимость режимов определяется матрицей
больше совместимых режимов — больше возможностей для параллелизма

На эффективность блокировок оказывают влияние много факторов, из которых мы выделим всего несколько.

Гранулярность (степень детализации) важна, если ресурсы образуют иерархию. Например, таблица состоит из страниц, которые содержат табличные строки. Все эти объекты могут выступать в качестве ресурсов. Если процесс заинтересован всего в нескольких строках, а блокировка устанавливается на уровне таблицы, то другие процессы не смогут работать с остальными строками.

Поэтому чем выше гранулярность – тем лучше для возможности распараллеливания, но это приводит к увеличению числа блокировок (информацию о которых надо где-то хранить).

Блокировки могут захватываться в разных **режимах**. Имена режимов могут быть абсолютно произвольными, важна лишь матрица их совместимости друг с другом.

Режим, несовместимый ни с каким режимом, принято называть *исключительным* (exclusive). Если режимы совместимы, то блокировка может захватываться несколькими процессами одновременно; такие режимы называют *разделяемыми* (shared).

В целом, чем больше можно найти режимов, совместимых друг с другом, тем больше возможностей для параллелизма.

Долговременные блокировки

обычно захватываются до конца транзакции
и относятся к хранимым данным
большое число режимов
развитая «тяжеловесная» инфраструктура, мониторинг

Краткосрочные блокировки

обычно захватываются на доли секунды
и относятся к структурам в оперативной памяти
минимум режимов
«легковесная» инфраструктура, мониторинг может отсутствовать

По времени использования блокировки можно разделить на длительные и короткие.

Долговременные блокировки захватываются на потенциально большое время (обычно до конца транзакции) и чаще всего относятся к таким ресурсам, как таблицы (отношения) и строки. Как правило, PostgreSQL управляет такими блокировками автоматически, но пользователь, тем не менее, имеет определенный контроль над этим процессом.

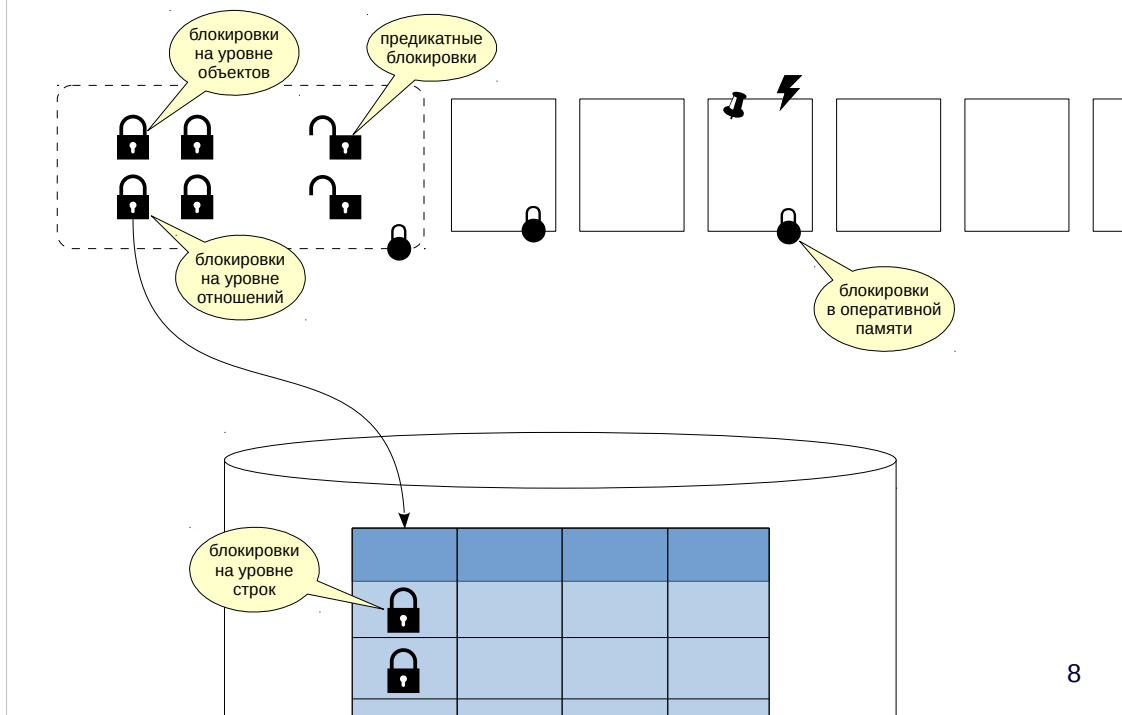
Для длительных блокировок характерно большое число режимов, чтобы разрешать как можно больше одновременных действий над данными.

Обычно для долговременных блокировок имеется развитая инфраструктура (например, поддержка очередей и обнаружение взаимоблокировок) и средства мониторинга.

Краткосрочные блокировки захватываются на небольшое время (от нескольких тактов процессора до долей секунд) и обычно относятся к структурам данных в общей памяти. Такими блокировками PostgreSQL управляет полностью автоматически — о их существовании надо просто знать.

Для коротких блокировок характерны простые режимы (исключительный и разделяемый), простая инфраструктура. В ряде случаев средства мониторинга могут отсутствовать.

Виды блокировок



В PostgreSQL используются разные виды блокировок.

Блокировки на уровне объектов относятся к длительным, «тяжеловесным». В качестве ресурсов здесь выступают **отношения** и другие объекты.

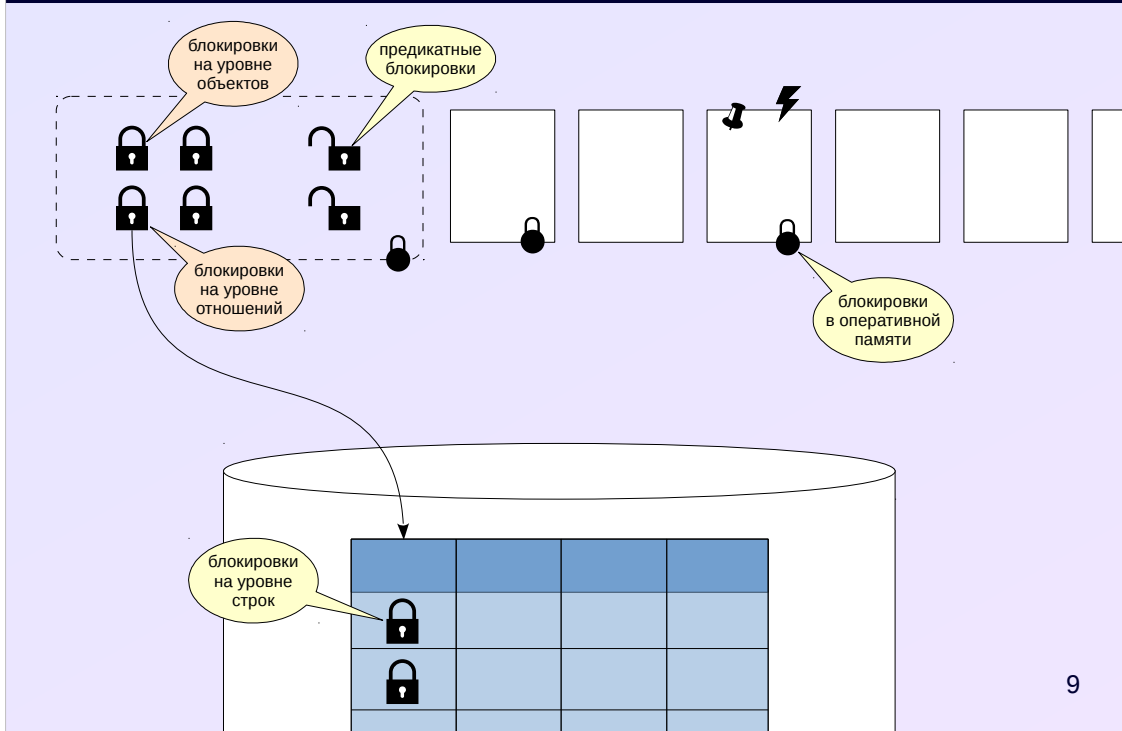
Еще один класс блокировок (оптимистических) — **предикатные**.

Информация о всех этих блокировках хранится однотипным образом в оперативной памяти. Эти виды блокировок подробно рассматриваются дальше в данной теме.

Среди длительных блокировок отдельно выделяются **блокировки на уровне строк**. Их реализация отличается от остальных длительных блокировок из-за потенциально огромного их количества (представьте обновление миллиона строк). Такие блокировки будут рассмотрены в теме «Блокировки строк».

К коротким блокировкам относятся различные блокировки структур оперативной памяти. Они рассматриваются в теме «**Блокировки в оперативной памяти**».

Блокировки объектов



Информация в общей памяти сервера

представление `pg_locks`:
`locktype` — тип блокируемого ресурса,
`mode` — режим блокировки

Ограниченное количество

$max_locks_per_transaction \times max_connections$

Инфраструктура

очередь ожидания: ждущие процессы не потребляют ресурсы
обнаружение взаимоблокировок

Начнем рассмотрение блокировок с блокировок уровня объектов, таких как таблицы, индексы и подобные им (*relations*). Такие блокировки защищают объекты от одновременного изменения или использования в то время, когда объект изменяется, а также для других нужд.

Блокировки объектов располагаются в общей памяти сервера. Их количество ограничено произведением значений двух параметров: *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.

Все блокировки можно посмотреть в представлении `pg_locks`.

Если ресурс уже заблокирован в неразделяемом режиме, транзакция, пытающаяся захватить этот ресурс, ставится в очередь и ожидает освобождения блокировки. Ожидающие транзакции не потребляют ресурсы процессора, они «засыпают» и пробуждаются только при освобождении ресурса. Ряд команд SQL позволяют указать ключевое слово `NOWAIT`: в этом случае попытка захватить занятый ресурс приводит не к ожиданию, а к ошибке.

Возможна ситуация взаимоблокировки (тупика), при которой две или более транзакций ждут друг друга. PostgreSQL автоматически определяет такие ситуации и аварийно прерывает бесконечное ожидание.

Тип ресурса: Relation

Режимы

Access Share	SELECT	} допускают параллельное изменение данных в таблице
Row Share	SELECT FOR UPDATE/SHARE	
Row Exclusive	UPDATE, DELETE, INSERT	
Share Update Exclusive	VACUUM, ALTER TABLE, CREATE INDEX CONCURRENTLY	
Share	CREATE INDEX	
Share Row Exclusive	CREATE TRIGGER, ALTER TABLE	
Exclusive	REFRESH MAT. VIEW CONCURRENTLY	
Access Exclusive	DROP, TRUNCATE, VACUUM FULL, LOCK TABLE, ALTER TABLE, REFRESH MAT. VIEW	

11

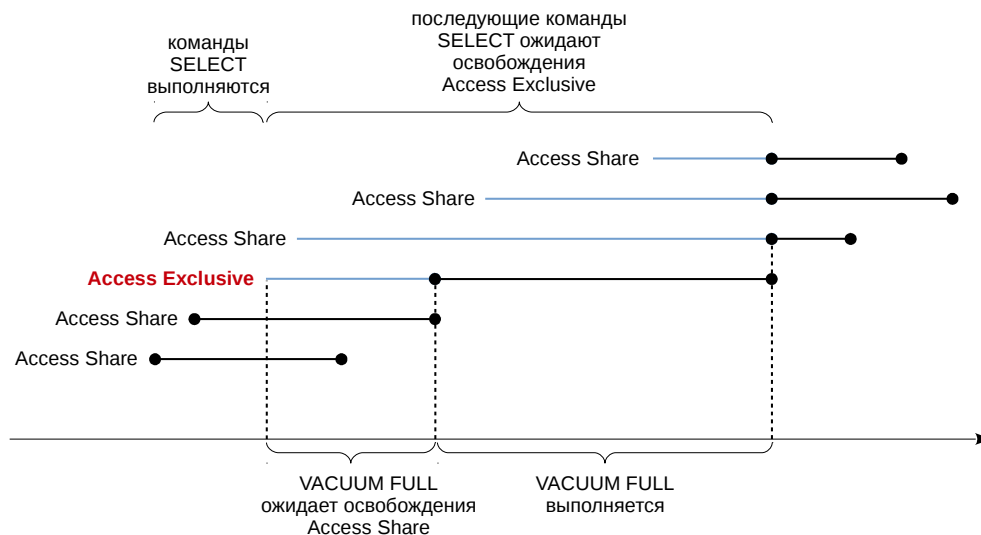
Важный частный случай блокировок — блокировки отношений (таблиц, индексов, последовательностей и т. п.). Такие блокировки имеют тип Relation в pg_locks.

Для них определено целых 8 различных режимов, которые показаны на слайде. Матрица совместимости, которая показывает, какие блокировки можно захватывать совместно, приведена в документации:

<https://postgrespro.ru/docs/postgresql/10/explicit-locking#LOCKING-TABLES>

Такое количество режимов существуют для того, чтобы как можно большее количество команд, относящихся к одной таблице, могло выполняться одновременно.

Как понять, к чему может привести выполнение какой-либо «административной» команды, например CREATE INDEX? Находим в документации, что эта команда устанавливает блокировку в режиме Share. По матрице определяем, что команда совместима сама с собой (можно одновременно создавать несколько индексов) и с читающими командами. Таким образом, команды SELECT продолжают работу, а команды UPDATE, DELETE, INSERT будут заблокированы. (Поэтому существует вариант команды — CREATE INDEX CONCURRENTLY — работающий дольше, но допускающий одновременное изменение данных.)



Чтобы лучше представить, к чему приводит появление несовместимой блокировки, можно посмотреть на приведенный пример.

Вначале на одной и той же таблице выполняются две команды SELECT, которые запрашивают и получают блокировку уровня Access Share (левые нижние отрезки).

Затем администратор выполняет команду VACUUM FULL, которой требуется блокировка уровня Access Exclusive, несовместимая с Access Share. Транзакция встает в очередь (третий снизу синий отрезок).

Затем в системе появляются еще несколько команд SELECT. И, хотя какие-то из них теоретически могли бы «проскочить», пока VACUUM FULL ждет своей очереди, все они честно занимают место за VACUUM FULL.

После того, как первые две транзакции с командами SELECT завершаются и освобождают блокировки, VACUUM FULL начинает выполняться.

И только после того, как VACUUM FULL завершит свою работу и снимет блокировку, все накопившиеся в очереди команды SELECT смогут захватить блокировку Access Share и начать выполняться.

Таким образом, неаккуратно выполненная команда может парализовать работу системы на время, значительно превышающее время выполнения самой команды.

Типы ресурсов

- Extend — добавление страниц к файлу отношения
- Object — не-отношение: база данных, схема и т. п.
- Page — страница (используется некоторыми типами индексов)
- Tuple — версия строки
- Advisory — рекомендательная блокировка
- Transactionid — транзакция
- Virtualxid — виртуальная транзакция

Режимы

- исключительный
- разделяемый

Кроме блокировок отношений есть еще несколько типов блокировок. Все они захватываются либо только в исключительном режиме, либо в исключительном и разделяемом. К ним относятся:

- **Extend** при добавлении страниц к файлу какого-либо отношения;
- **Object** для блокирования объекта, который не является отношением (примеры таких объектов: база данных, схема, подписка и т. п.);
- **Page** для блокирования страницы (редкая блокировка, используется некоторыми типами индексов);
- **Tuple** используется в некоторых случаях для установки приоритета среди нескольких транзакций, ожидающих блокировку одной строки (подробнее см. тему «Блокировка строк»);
- **Advisory** для рекомендательных блокировок:
<https://postgrespro.ru/docs/postgresql/10/explicit-locking#ADVISORY-LOCK>

Особый интерес представляют блокировки типа **Transactionid** и **Virtualxid**. Каждая транзакция сама удерживает исключительную блокировку своего собственного номера. Такие блокировки удобно использовать, когда какой-либо транзакции надо дождаться окончания другой транзакции. Подробнее это рассматривается в теме «Блокировки на уровне строк» этого модуля.

Вывод сообщений в журнал сервера

параметр *log_lock_waits*:

выводит сообщение об ожидании дольше *deadlock_timeout*

Текущие блокировки

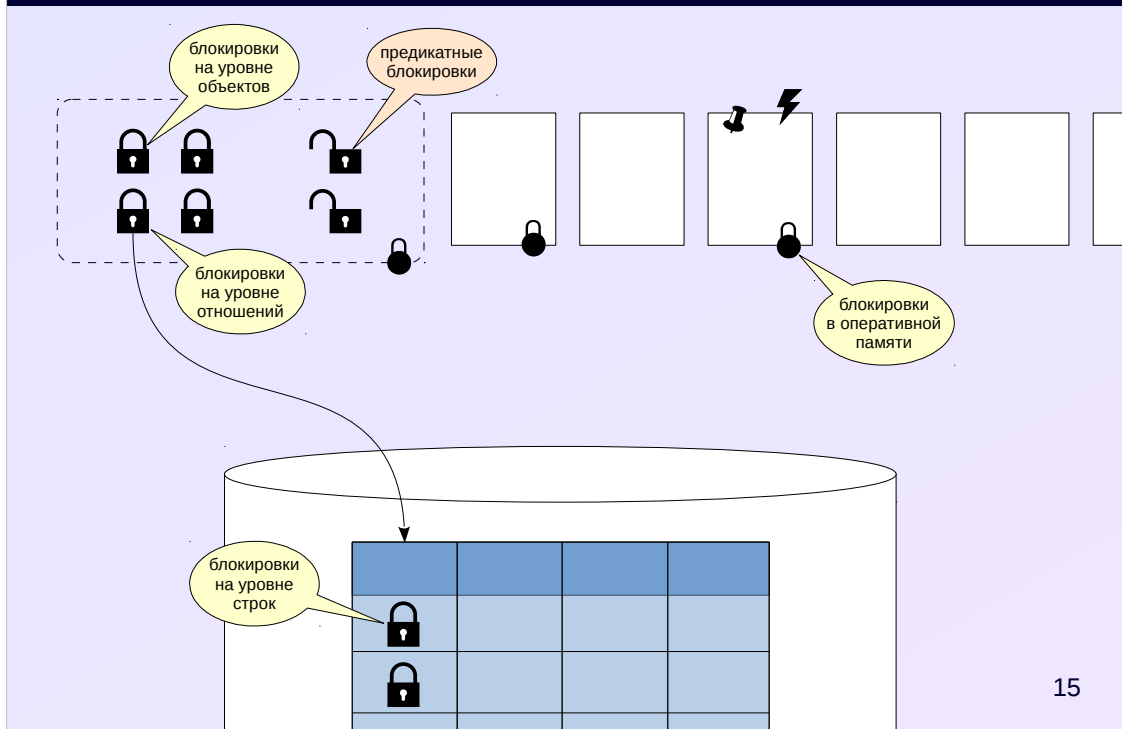
представление *pg_locks*

функция *pg_blocking_pids*

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* (несмотря на то, что используется параметр для взаимоблокировок, речь идет об обычных ожиданиях).

Второй способ состоит в том, чтобы в момент возникновения долгой блокировки (или на периодической основе) выполнять запрос к представлению *pg_locks*, смотреть на блокируемые и блокирующие транзакции (функция *pg_blocking_pids*) и расшифровывать их при помощи *pg_stat_activity*.



Задача: реализация уровня изоляции Serializable

используются в дополнение к обычной изоляции на снимках данных
оптимистичные блокировки, название сложилось исторически

Информация в общей памяти сервера

представление pg_locks

Ограниченное количество

$max_pred_locks_per_transaction \times max_connections$

Повышение уровня блокировок

$max_pred_locks_per_relation$

$max_pred_locks_per_page$

Термин *предикатная блокировка* появился давно, еще при первых попытках реализовать полную изоляцию (Serializable) на основе блокировок в ранних СУБД. Идея состояла в том, что блокировать надо не только определенные строки, но и предикаты. Например, при выполнении запроса с условием $a > 10$ надо заблокировать диапазон $a > 10$, чтобы избежать появления фантомных строк и других аномалий.

В PostgreSQL уровень Serializable реализован поверх существующего механизма снимков данных, но термин остался. Фактически такие «блокировки» ничего не блокируют, а используются для отслеживания зависимостей транзакций по данным.

Как и для обычных блокировок, информация о предикатных блокировках отображается в представлении pg_locks со специальным режимом SIReadLock.

Число предикатных блокировок ограничено параметрами $max_pred_locks_per_transaction$ и $max_connections$. В отличие от обычных блокировок, при увеличении их количества происходит автоматическое повышение уровня (эскалация): более мелкие блокировки освобождаются, а вместо них захватывается одна блокировка более высокого уровня. Начиная с версии PostgreSQL 10, этим процессом можно управлять с помощью параметров.

<https://postgrespro.ru/docs/postgresql/10/transaction-iso#XACT-SERIALIZABLE>

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

https://github.com/postgres/postgres/blob/REL_10_STABLE/src/backend/storage/lmgr/README-SSI

Типы ресурсов

Relation — отношение
вся таблица при полном сканировании

Page — страница
индексные страницы при индексном доступе

Tuple — версия строки
табличные строки при индексном доступе

Режим

SIRead

Фактически предикатные блокировки захватываются на трех уровнях.

При полном сканировании таблицы блокировка устанавливается **на уровне всей таблицы**.

При индексном сканировании устанавливаются блокировки тех **страниц индекса**, которые соответствуют условию доступа (предикату). Кроме того, устанавливаются блокировки на уровне **отдельных табличных версий строк** (это не то же самое, что блокировки строк, которые рассматриваются в одноименной теме).

Все предикатные блокировки устанавливаются в одном специальном режиме: SIRead (Serializable Isolation Read).

Если число блокировок строк одной страницы превышает значение параметра *max_pred_locks_per_page*, вместо них захватывается одна блокировка уровня страницы. Если число блокировок строк одной таблицы (или индекса) превышает значение параметра *max_pred_locks_per_relation*, вместо них захватывается одна блокировка на все отношение. Повышение уровня блокировок приводит к тому, что большее число транзакций ложно завершается ошибкой сериализации и пропускная способность системы падает.

Именно с использованием предикатных блокировок связано ограничение, что для достижения полной изоляции *все* транзакции должны работать на уровне Serializable. Отслеживание зависимостей будет работать только для транзакций, которые устанавливают предикатные блокировки.



Блокировки отношений и других объектов БД используются для организации конкурентного доступа к общим ресурсам

- поддерживаются очереди и обнаружение взаимоблокировок
- рекомендательные блокировки для ресурсов, не связанных с хранимыми объектами

Предикатные блокировки используются для реализации уровня изоляции Serializable

- ничего не блокируют, отслеживают зависимости транзакций по данным

1. Какие блокировки на уровне изоляции Read Committed удерживает транзакция, прочитавшая одну строку таблицы по первичному ключу? Проверьте на практике.
2. Повторите предыдущий пункт для уровня изоляции Serializable.
3. Настройте сервер так, чтобы в журнал сообщений сбрасывалась информация о блокировках, удерживаемых более 100 миллисекунд. Воспроизведите ситуацию, при которой в журнале появятся такие сообщения.