Практические работы по дисциплине «Технологии обработки транзакций клиент-серверных приложений» направления подготовки бакалавриата 09.03.04 «Программная инженерия»

Практическая работа №7

Транзакции. Блокировки. Блокировки объектов

Теория для понимания практики:

Темы

- Общая информация о блокировках
- Блокировки отношений и других объектов
- Предикатные блокировки

Общая информация

- Задача и механизм использования блокировок
- Блокируемые ресурсы
- Факторы, влияющие на эффективность
- Время жизни блокировок

Блокировки

Задача: упорядочение конкурентного доступак 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 разделения 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 удается применить другие, неблокирующие, стратегии. Например, в одноименном модуле мы обсуждали механизм многоверсионности. Еще один пример — оптимистические блокировки, которые не блокируют процесс, но в случае неудачи приводят к ошибке.

Ресурсы

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 системном 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).

Факторы эффективности

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.

Время жизни

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 надлительные и короткие.

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

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

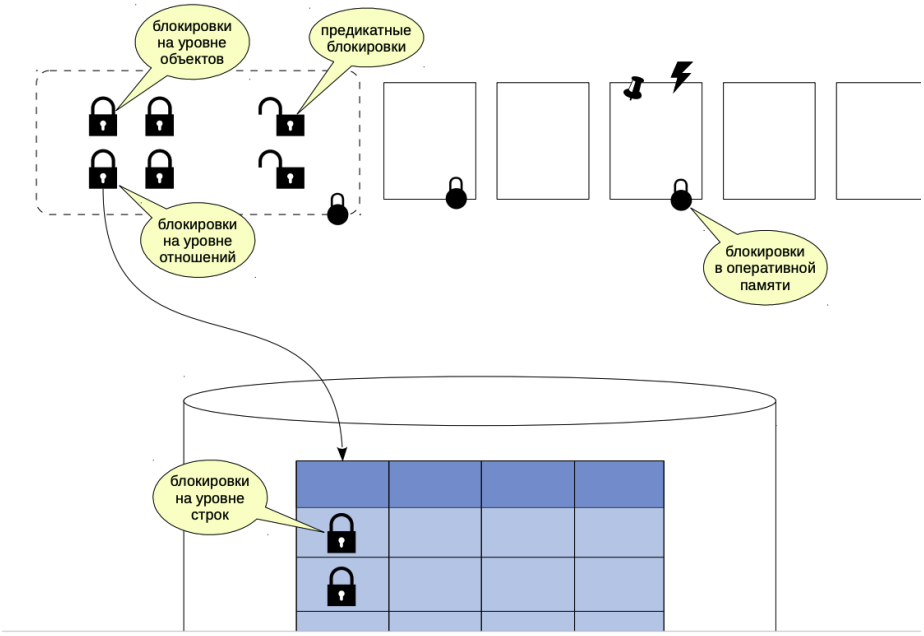

Рисунок 1 – Виды блокировок

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

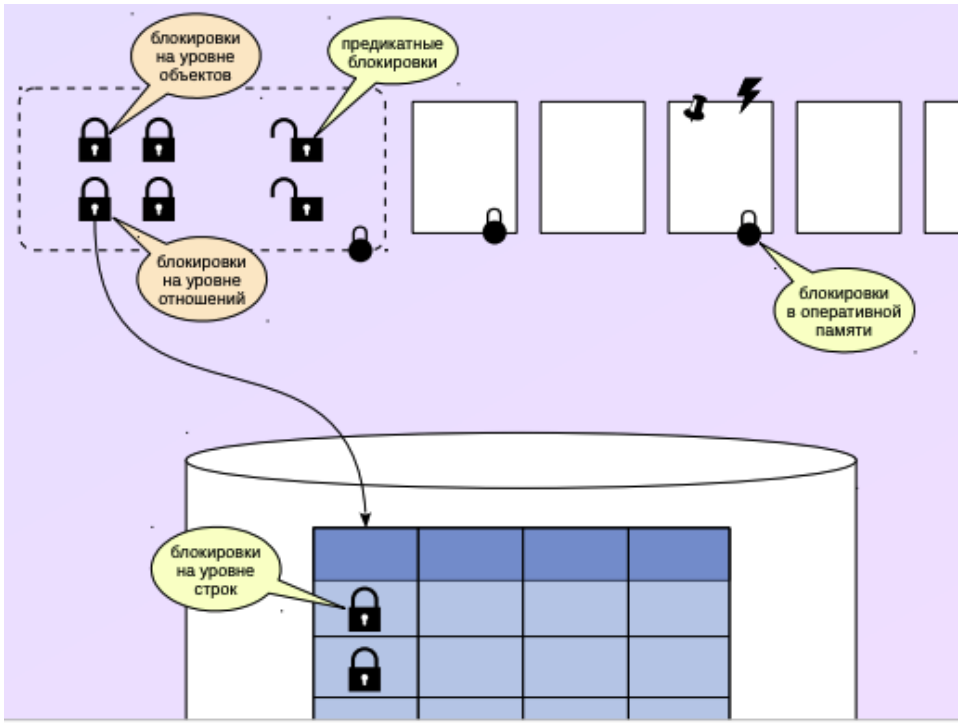

Рисунок 2 – Блокировки объектов

Устройство

Информация в общей памяти сервера

- представление pg_locks:
- locktype – тип блокируемого ресурса,
- mode – режим блокировки

Ограниченное количество

- *max_locks_per_transaction* × *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.

Блокировки отношений

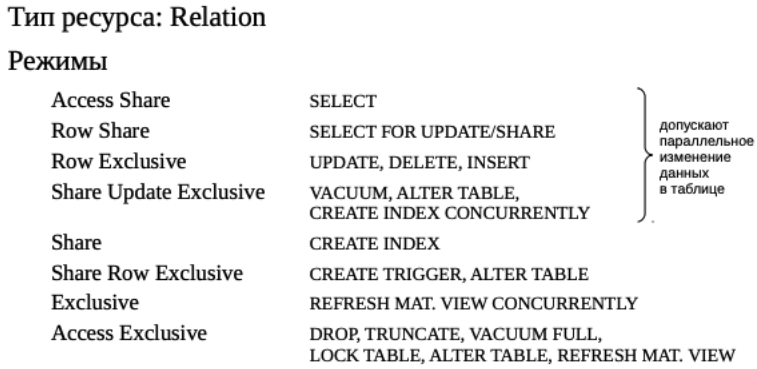

Рисунок 3 – Блокировки отношений

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 рисунке 3. Единственное, что действительно отличает один режим блокировки от другого, это набор режимов, с которыми конфликтует каждый из них:

Запрашиваемый режим блокировки	Существующий режим блокировки							
	ACCESS SHARE	ROW SHARE	ROW EXCL.	SHARE UPDATE EXCL.	SHARE	SHARE ROW EXCL.	EXCL.	ACCESS EXCL.
ACCESS SHARE								X
ROW SHARE							X	X
ROW EXCL.					X	X	X	X
SHARE UPDATE EXCL.				X	X	X	X	X
SHARE			X	X		X	X	X
SHARE ROW EXCL.			X	X	X	X	X	X
EXCL.		X	X	X	X	X	X	X
ACCESS EXCL.	X	X	X	X	X	X	X	X

Рисунок 4 – Конфликтующие режимы блокировки

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 продолжат работу, а команды UPDATE, DELETE, INSERT будут заблокированы. (Поэтому существует вариант команды – CREATE INDEX CONCURRENTLY – работающий дольше, но допускающий одновременное изменение данных.)

Очередь ожидающих

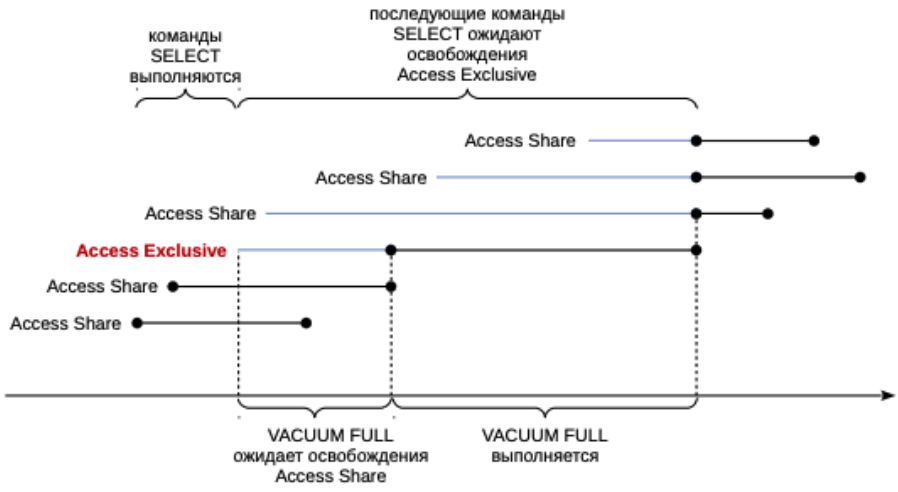

Рисунок 5 – Очередь ожидающих

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 выполнение самой команды.

Блокировки других типов

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. Рекомендательные блокировки бывают полезны для реализаций стратегий блокирования, плохо вписывающихся в модель MVCC. Например, рекомендательные блокировки часто применяются для исполнения стратегии пессимистичной блокировки, типичной для систем управления данными «плоский файл». Хотя для этого можно использовать и дополнительные флаги в таблицах, рекомендательные блокировки работают быстрее, не нагружают таблицы и автоматически ликвидируется сервером в конце сеанса.

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.

Средства мониторинга

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.

Предикатные блокировки

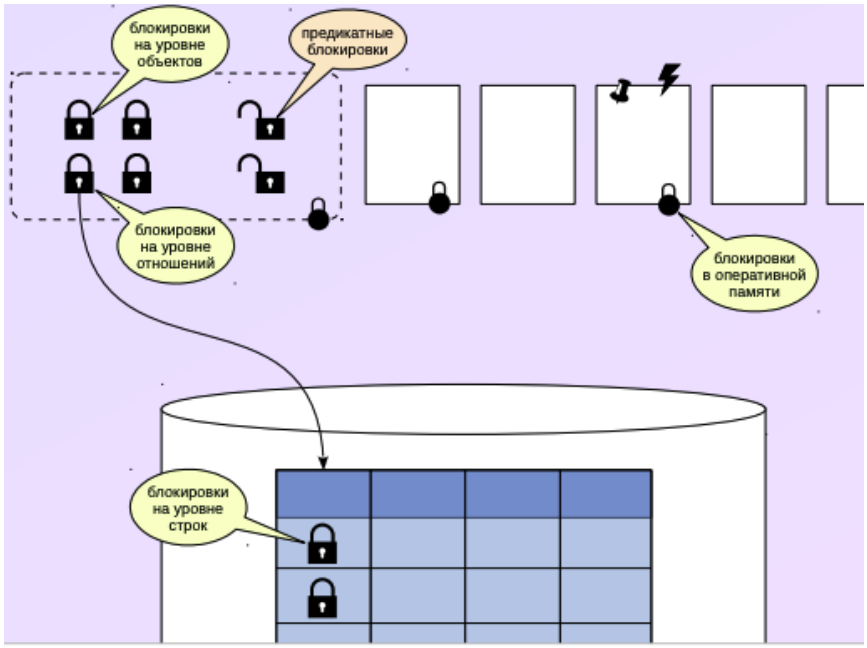

Рисунок 6 – Предикатные блокировки

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* × *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 специальнымрежимом 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.

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 приводитк 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.

Практика 1

Блокировки отношений и других объектов

Создадим таблицу "банковских" счетов. В ней храним номер счета и сумму.

```
=> CREATE DATABASE locks_objects;
CREATE DATABASE
=> \c locks_objects
```

```
You are now connected to database "locks_objects" as user
"postgres".
=> CREATE TABLE accounts(acc_no integer, amount numeric);
CREATE TABLE
=> INSERT INTO accounts VALUES (1,1000.00), (2,2000.00),
(3,3000.00);
INSERT 0 3
```

Во втором сеансе начнем транзакцию. Нам понадобится номер обслуживающего процесса.

```
=> \c locks_objects
You are now connected to database "locks_objects" as user
"postgres".
=> SELECT pg_backend_pid();
 pg_backend_pid
----------------
          11123
(1 row)

=> BEGIN;
BEGIN
```

Какие блокировки удерживает только что начавшаяся транзакция?

```
=> SELECT locktype, relation::REGCLASS, virtualxid AS virtxid,
transactionid AS xid, mode, granted
FROM pg_locks WHERE pid = 11123;
  locktype  | relation | virtxid | xid |     mode      | granted
------------+----------+---------+-----+---------------+--------
 virtualxid |          | 3/16    |     | ExclusiveLock | t
(1 row)
```

Только блокировку собственного виртуального номера.

Теперь обновим строку таблицы. Как изменится ситуация?

```
=> UPDATE accounts SET amount = amount + 100 WHERE acc_no = 1;
UPDATE 1
```

```
=> SELECT locktype, relation::REGCLASS, virtualxid AS virtxid,
transactionid AS xid, mode, granted
FROM pg_locks WHERE pid = 11123;
   locktype    | relation | virtxid |  xid  |      mode       | granted
---------------+----------+---------+-------+-----------------+---------
 relation      | accounts |         |       | RowExclusiveLock | t
 virtualxid    |          | 3/16    |       | ExclusiveLock    | t
 transactionid |          |         | 26986 | ExclusiveLock    | t
(3 rows)
```

Добавилась блокировка отношения в режиме RowExclusiveLock (что соответствует команде UPDATE), и исключительная блокировка собственного номера (который появился, как только транзакция начала изменять данные).

Теперь попробуем в еще одном сеансе создать индекс по таблице.

```
=> \c locks_objects
You are now connected to database "locks_objects" as user
"postgres".
=> SELECT pg_backend_pid();
 pg_backend_pid
----------------
          11279
(1 row)

=> CREATE INDEX ON accounts(acc_no);
```

Команда не выполняется - ждет освобождения блокировки. Какой?

Проверим:

```
=> SELECT locktype, relation::REGCLASS, virtualxid AS virtxid,
transactionid AS xid, mode, granted
FROM pg_locks WHERE pid = 11279;
  locktype  | relation | virtxid | xid |     mode      | granted
------------+----------+---------+-----+---------------+---------
 virtualxid |          | 4/54    |     | ExclusiveLock | t
 relation   | accounts |         |     | ShareLock     | f
(2 rows)
```

Видим, что транзакция пыталась получить блокировку таблицы в режиме ShareLock, но не смогла (granted = f).

Мы можем найти номер блокирующего процесса (в общем виде - несколько номеров)...

```
=> SELECT pg_blocking_pids(11279);
 pg_blocking_pids
------------------
 {11123}
(1 row)
```

...и посмотреть информацию о сеансах, к которым они относятся:

```
=> SELECT * FROM pg_stat_activity
WHERE pid = ANY(pg_blocking_pids(11279)) \gx
-[ RECORD 1 ]----+------------------------------------------------------------
datid            | 24842
datname          | locks_objects
pid              | 11123
usesysid         | 10
usename          | postgres
application_name | psql
client_addr      |
client_hostname  |
client_port      | -1
backend_start    | 2019-08-12 17:16:12.66099+03
xact_start       | 2019-08-12 17:16:12.74424+03
query_start      | 2019-08-12 17:16:12.826119+03
state_change     | 2019-08-12 17:16:12.827085+03
wait_event_type  | Client
wait_event       | ClientRead
state            | idle in transaction
backend_xid      | 26986
backend_xmin     |
query            | UPDATE accounts SET amount = amount + 100 WHERE acc_no = 1;
backend_type     | client backend
```

После завершения транзакции блокировки снимаются и индекс создается.

```
=> COMMIT;
COMMIT

CREATE INDEX
```

Рекомендательные блокировки

Рассмотрим еще один тип блокировок: рекомендательные. В отличие от других блокировок (таких, как блокировки отношений), рекомендательные блокировки устанавливаются только пользователем, но не автоматически. Их используют, если приложению требуется логика блокирования, которую неудобно реализовывать с помощью других, "обычных" блокировок.

Получим блокировку некого условного ресурса. Начнем транзакцию.

```
=> BEGIN;
BEGIN
```

В качестве идентификатора используется число; если ресурс имеет имя, удобно получить число с помощью функции хеширования:

```
=> SELECT hashtext('ресурс1');
 hashtext
-----------
 243773337
(1 row)

=> SELECT pg_advisory_lock(hashtext('ресурс1'));
 pg_advisory_lock
------------------


(1 row)
```

Информация о рекомендательных блокировках доступна в pg_locks:

```
=> SELECT locktype, objid, virtualxid AS virtxid, mode, granted
FROM pg_locks WHERE pid = 11123;
  locktype  |   objid   | virtxid |     mode      | granted
```

```
------------+----------+---------+---------------+---------
 virtualxid |          | 3/17    | ExclusiveLock | t
 advisory   | 243773337 |         | ExclusiveLock | t
(2 rows)
```

Если другой сеанс попробует захватить ту же блокировку, он будет ждать ее освобождения:

```
=> SELECT pg_advisory_lock(hashtext('pecypc1'));
```

В приведенном примере блокировка действует до конца сеанса, а не транзакции, как обычно.

```
=> COMMIT;
COMMIT

=> SELECT locktype, objid, virtualxid AS virtxid, mode, granted
FROM pg_locks WHERE pid = 11123;
 locktype |   objid   | virtxid |     mode      | granted
----------+-----------+---------+---------------+---------
 advisory | 243773337 |         | ExclusiveLock | t
(1 row)
```

Ее можно явно освободить:

```
=> SELECT pg_advisory_unlock(hashtext('pecypc1'));
 pg_advisory_unlock
--------------------
 t
(1 row)


 pg_advisory_lock
------------------

(1 row)
```

Существуют другие варианты функций для получения рекомендательных блокировок до конца транзакции, для получения разделяемых блокировок и т. п.

Практика 2

Блокировки читающей транзакции, Read Committed

Создадим таблицу как в демонстрации; номер счета будет первичным ключом.

```
=> CREATE DATABASE locks_objects;
CREATE DATABASE
=> \c locks_objects
You are now connected to database "locks_objects" as user
"postgres".
=> CREATE TABLE accounts(acc_no integer PRIMARY KEY, amount
numeric);
CREATE TABLE
=> INSERT INTO accounts VALUES
(1,1000.00),(2,2000.00),(3,3000.00);
INSERT 0 3
```

Начинаем транзакцию и читаем одну строку.

```
=> \c locks_objects
You are now connected to database "locks_objects" as user
"postgres".
=> SELECT pg_backend_pid();
 pg_backend_pid
----------------
          28663
(1 row)

=> BEGIN;
BEGIN
=> SELECT * FROM accounts WHERE acc_no = 1;
 acc_no | amount
--------+---------
```

```
      1 | 1000.00
(1 row)
```

Блокировки:

```
=> SELECT locktype, relation::REGCLASS, virtualxid AS virtxid,
transactionid AS xid, mode, granted
FROM pg_locks WHERE pid = 28663;
  locktype  |   relation    | virtxid | xid |      mode       | granted
------------+---------------+---------+-----+-----------------+---------
 relation   | accounts_pkey |         |     | AccessShareLock | t
 relation   | accounts      |         |     | AccessShareLock | t
 virtualxid |               | 2/75    |     | ExclusiveLock   | t
(3 rows)
```

Здесь мы видим:

- Блокировка таблицы accounts в режиме AccessShareLock;
- Блокировка индекса accounts_pkey, созданного для первичного ключа, в том же режиме;
- Исключительная блокировка собственного номера виртуальной транзакции.

Если смотреть блокировки в самой транзакции, к ним добавится блокировка на таблицу pg_locks:

```
=> SELECT locktype, relation::REGCLASS, virtualxid AS virtxid,
transactionid AS xid, mode, granted
FROM pg_locks WHERE pid = 28663;
  locktype  |   relation    | virtxid | xid |      mode       | granted
------------+---------------+---------+-----+-----------------+---------
 relation   | pg_locks      |         |     | AccessShareLock | t
 relation   | accounts_pkey |         |     | AccessShareLock | t
 relation   | accounts      |         |     | AccessShareLock | t
 virtualxid |               | 2/75    |     | ExclusiveLock   | t
(4 rows)

=> COMMIT;
COMMIT
```

Блокировки читающей транзакции, Serializable

Начинаем транзакцию и читаем одну строку.

```
=> BEGIN ISOLATION LEVEL SERIALIZABLE;
BEGIN
=> SELECT * FROM accounts WHERE acc_no = 1;
 acc_no | amount
--------+---------
      1 | 1000.00
(1 row)
```

Блокировки:

```
=> SELECT locktype, relation::REGCLASS, page, tuple, virtualxid
AS virtxid, transactionid AS xid, mode, granted
FROM pg_locks WHERE pid = 28663;
  locktype  |   relation    | page | tuple | virtxid | xid |      mode       | granted
------------+---------------+------+-------+---------+-----+-----------------+--------
 relation   | accounts_pkey |      |       |         |     | AccessShareLock | t
 relation   | accounts      |      |       |         |     | AccessShareLock | t
 virtualxid |               |      |       | 2/76    |     | ExclusiveLock   | t
 page       | accounts_pkey |    1 |       |         |     | SIReadLock      | t
 tuple      | accounts      |    0 |     1 |         |     | SIReadLock      | t
(5 rows)
```

К предыдущим блокировкам добавились:

- Предикатная блокировка страницы индекса;
- Предикатная блокировка прочитанной версии строки.

```
=> COMMIT;
COMMIT
```

Вывести в журнал информацию о блокировках

Требуется изменить параметры:

```
=> ALTER SYSTEM SET log_lock_waits = on;
ALTER SYSTEM
=> ALTER SYSTEM SET deadlock_timeout = '100ms';
ALTER SYSTEM
=> SELECT pg_reload_conf();
```

```
 pg_reload_conf
----------------
 t
(1 row)
```

Воспроизведем блокировку.

```
=> BEGIN;
BEGIN
=> UPDATE accounts SET amount = 10.00 WHERE acc_no = 1;
UPDATE 1
=> BEGIN;
BEGIN
=> UPDATE accounts SET amount = 100.00 WHERE acc_no = 1;
```

В первом сеансе выполним задержку и после этого завершим транзакцию.

```
=> SELECT pg_sleep(1);
 pg_sleep
----------


(1 row)

=> COMMIT;
COMMIT

UPDATE 1
=> COMMIT;
COMMIT
```

Вот что попало в журнал:

```
postgres$ tail -n 7 /home/postgres/logfile
2019-08-12 17:25:51.399 MSK [28663] LOG:  process 28663 still
waiting for ShareLock on transaction 215332 after 100.319 ms
2019-08-12 17:25:51.399 MSK [28663] DETAIL:  Process holding the
lock: 28610. Wait queue: 28663.
2019-08-12 17:25:51.399 MSK [28663] CONTEXT:  while updating
tuple (0,1) in relation "accounts"
```

```
2019-08-12 17:25:51.399 MSK [28663] STATEMENT:  UPDATE accounts
SET amount = 100.00 WHERE acc_no = 1;
2019-08-12 17:25:52.369 MSK [28663] LOG:  process 28663 acquired
ShareLock on transaction 215332 after 1070.271 ms
2019-08-12 17:25:52.369 MSK [28663] CONTEXT:  while updating
tuple (0,1) in relation "accounts"
2019-08-12 17:25:52.369 MSK [28663] STATEMENT:  UPDATE accounts
SET amount = 100.00 WHERE acc_no = 1;
```

Итоги

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

Задание на практическую работу:

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.