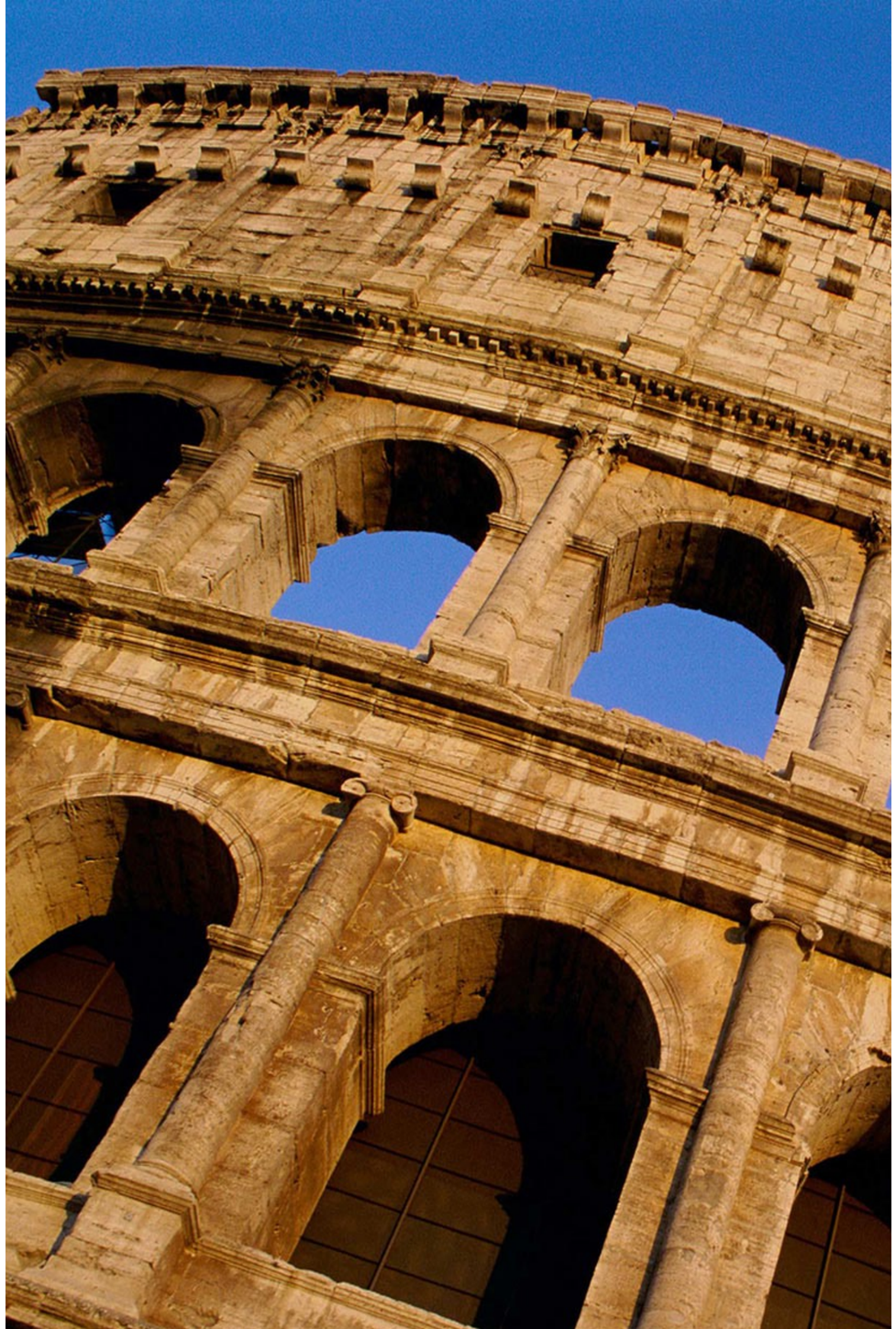
Лок, лок — дедлок!

~~Дедпул~~
Тюрин Михаил
mtyurin@bk.ru



почему нам это может быть интересно?

- «локи на базе, ахтунг, всё пропало! всё упало!» — приходилось такое слышать — очень эмоционально
- «*ERROR: deadlock detected DETAIL: Process 8835 waits for ShareLock on transaction 193588236; blocked by process 8834. Process 8834 waits for ShareLock on transaction 193588228; blocked by process 8835*» — выглядит страшно и хочется такого не видеть
- локи и, да, транзакции — это всё как-то сложно и пора еще раз попытаться внести ясность
- академическая пятиминутка

документация содержит все ответы // 1

- как всегда! но хорошо бы и примеров из практики
- https://www.postgresql.org/docs/9.6/static/explicit-locking.html — 13.3.4. Deadlocks :
  - automatically detects deadlocks and resolves them
  - all applications acquire locks on multiple objects in a consistent order
  - the first lock acquired on an object is the most restrictive mode that will be needed

документация содержит все ответы // 2

- https://www.postgresql.org/docs/9.6/static/sql-lock.html
  - deadlock is possible :-)
  - transactions acquire locks on the same objects in the same order
  - if multiple lock modes are involved for a single object, then transactions should always acquire the most restrictive mode first
- https://www.postgresql.org/docs/9.6/static/explicit-locking.html — кто и кого может ждать:
  - 13-2. Conflicting Lock Modes
  - **13-3. Conflicting Row-level Locks**
  - 3.3.5. Advisory Locks

документация содержит все ответы // 3

- https://www.postgresql.org/docs/9.6/static/runtime-config-locks.html —19.12. Lock Management:
  - deadlock_timeout (integer) — to wait on a lock before checking to see if there is a deadlock; the check is **expensive**
- https://www.postgresql.org/docs/9.6/static/runtime-config-logging.html
  - log_lock_waits (boolean) — log when a session waits longer than **deadlock_timeout** to acquire a lock; useful in determining if lock waits are causing poor performance; default is off.

транзакции и их уровни изоляции

дефолтный Read Committed

- https://www.postgresql.org/docs/current/static/transaction-iso.html
  - **Read Committed**
    - «здоровый компромисс» для OLTP: баланс оверхеда и пропускной способности
      - «много» тысяч наперед известных tps
      - программист (часто уже не db-девелопер) **сам** решает, как управлять блокировками!

хитрый пример про Read Committed

https://www.postgresql.org/docs/9.6/static/transaction-iso.html

13.2.1. Read Committed Isolation Level

BEGIN;

UPDATE website SET hits = hits + 1;

-- run from another session: DELETE FROM website WHERE hits = 10;

COMMIT;

домашнее задание!

обратно к блокировкам

классический пример дедлока

T1: UPDATE accounts SET balance = balance + 100.00 WHERE acctnum = 11111;

T2: UPDATE accounts SET balance = balance + 100.00 WHERE acctnum = 22222;

T2: UPDATE accounts SET balance = balance - 100.00 WHERE acctnum = 11111;

T1: UPDATE accounts SET balance = balance - 100.00 WHERE acctnum = 22222;

T1: (acctnum = 11111, acctnum = 22222), T2: (22222, 11111) — deadlock

менее классический пример

T1: UPDATE user;

T2: UPDATE user_items;

T2: UPDATE user;

T1: UPDATE user_items;

T1: (user, user_items), T2: (user_items, user) — deadlock

сложный пример // масштаб катастрофы

T1: UPDATE users WHERE user_id __condition1__ —> users_update_plan1

T2: UPDATE users WHERE user_id __condition2__ —> users_update_plan2

T1: (update_plan1), T2: (update_plan2) — possible deadlock

еще пример

T1: share user1; share user2;

T2: share user1; share user2;

T2: update user1;

T1: update user2;

T1: (share, update), T2: (share, update) — deadlock

еще кейз из реального бизнеса

фоновая обработка задач и бекофис конкурируют с пользовательской нагрузкой

- одним надо одно
- другим другое

одно из решений было: внутрибатчевое переупорядочивание событий при фоновой обработке очереди

хаки

set statement_timeout = 20;

set deadlock_timeout = 10; // отстрел вакуума

alter mytable add mycolumn integer;

итого!

- порядок важен:
  - на одном уровне единый порядок
  - блокировки от общего к частному
  - более сильная блокировка вперед
- но и это может не помочь: обрабатывайте ошибки сериализация на клиенте

обобщим

- если есть конкуренция в системе за ресурсы, то может быть дедлок
- но дедлокдетектора может не быть
- избегаем «увязывания» сетевых ресурсов: пока находимся в одном ресурсе, не делаем вызов в другой
- выход на Dependency Hell in Microservices … нда

вопросы и ответы

спасибо!

mtyurin@bk.ru