

БАЗЫ ДАННЫХ

Лекция 6.

Блокировки в PostgreSQL.
Рекомендательные блокировки

Блокировки в PostgreSQL



Уровни изоляции в БД

Уровень изоляции	*«Грязное» чтение (dirty read)*	*Неповторяемое чтение (nonrepeatable read)*	*Фантомное чтение (phantom read)*	*Аномалия сериализации (serialization anomaly)*
READ UNCOMMITTED (Чтение Незафиксированных данных)	Допускается, но не в PG	Возможно	Возможно	Возможно
READ COMMITTED (Чтение зафиксированных данных)	Невозможно	Возможно	Возможно	Возможно
REPEATABLE READ (Повторяемое чтение)	Невозможно	Невозможно	Допускается, но не в PG	Возможно
SERIALIZABLE (Сериализуемость)	Невозможно	Невозможно	Невозможно	Невозможно

Феномен чтения – отличия при параллельном и изолированном выполнении транзакций

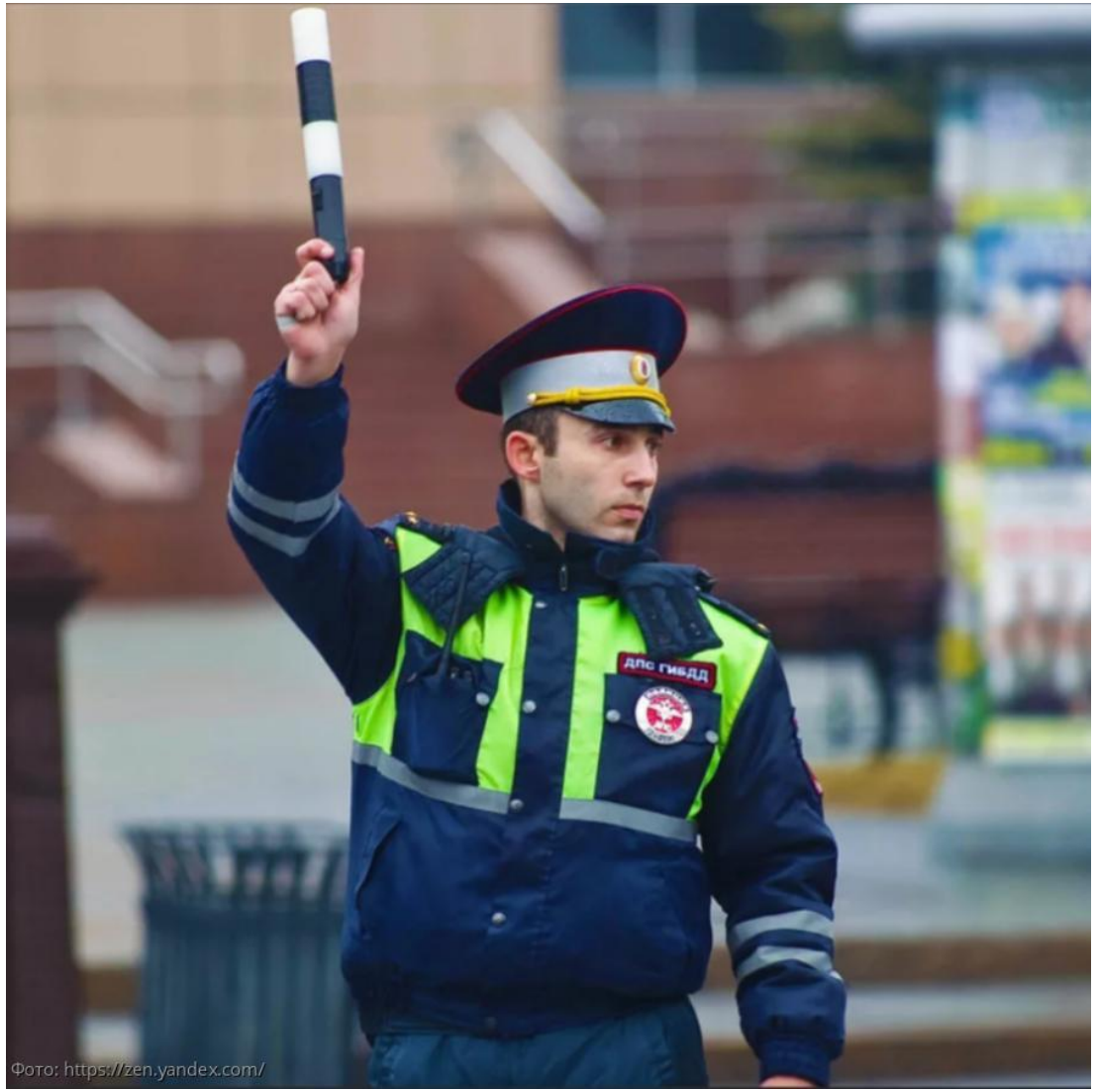
Определение блокировки

БЛОКИРОВКА

Используется для ограничений конкурентного доступа

Для определения строгого порядка доступа к ресурсу

- Инициируется пользовательским приложением или самой СУБД



Определение блокировки

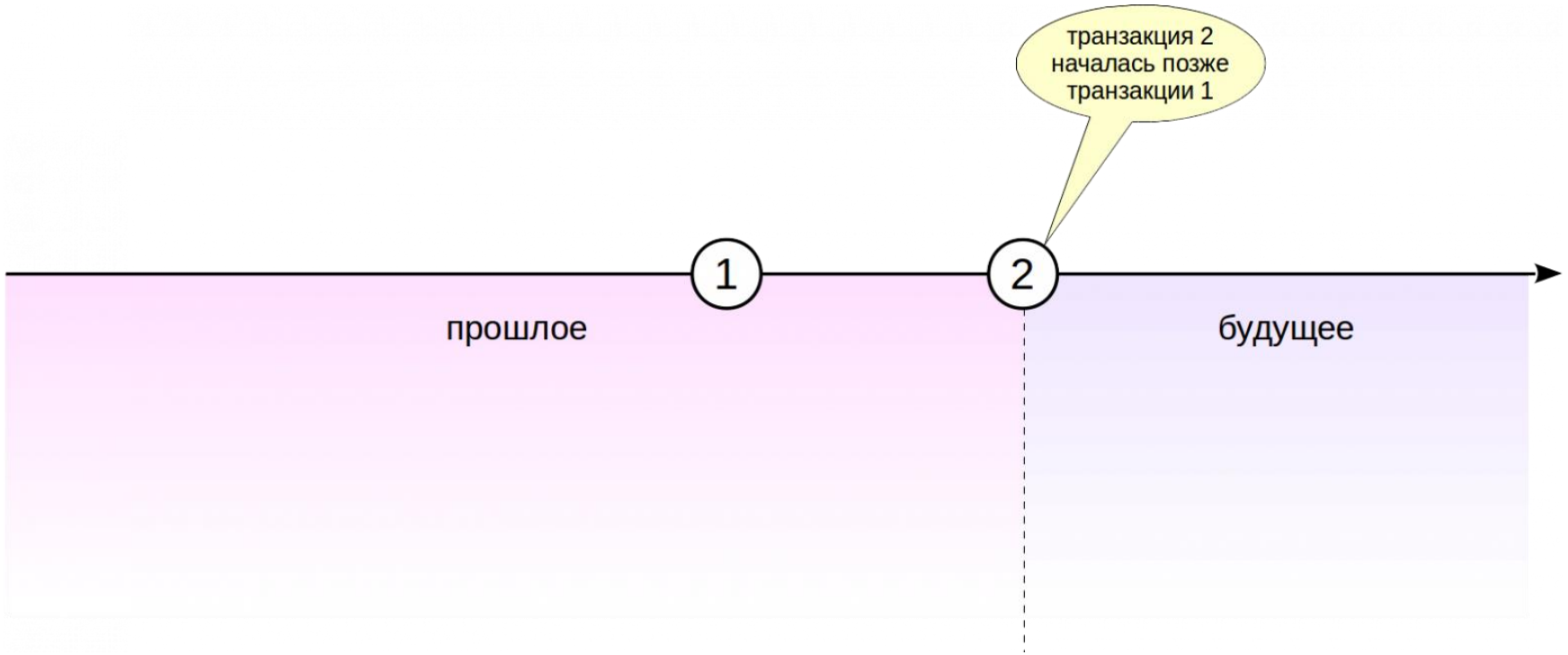
«Наиболее распространенным механизмом, используемым в СУБД для сериализации транзакций, является механизм блокировок».



Определение блокировки

«Наиболее распространенным механизмом, используемым в СУБД для сериализации транзакций, является механизм блокировок».

PostgreSQL: механизм MVCC – многоверсионность строк



Определение блокировки

«Наиболее распространенным механизмом, используемым в СУБД для сериализации транзакций, является механизм блокировок».

PostgreSQL: механизм MVCC – многоверсионность строк

Для управления параллельным доступом к данным в таблицах PostgreSQL предоставляет **несколько режимов явных блокировок**. Эти режимы могут применяться для блокировки данных со стороны приложения в ситуациях, *когда MVCC не даёт желаемый результат*.

Определение блокировки

«Наиболее распространенным механизмом, используемым в СУБД для сериализации транзакций, является механизм блокировок».

PostgreSQL: механизм MVCC – многоверсионность строк

Для управления параллельным доступом к данным в таблицах PostgreSQL предоставляет **несколько режимов явных блокировок**. Эти режимы могут применяться для блокировки данных со стороны приложения в ситуациях, *когда MVCC не даёт желаемый результат*.

Большинство команд PostgreSQL **автоматически** получают блокировки соответствующих режимов, защищающие от удаления или изменения задействованных таблиц, несовместимого с характером выполняемой команды.

Блокировка

- На низком уровне:
  участок разделяемой памяти, в котором некоторым образом отмечается, свободна ли блокировка или захвачена (и, возможно, записывается дополнительная информация: номер процесса, время захвата и т. п.).

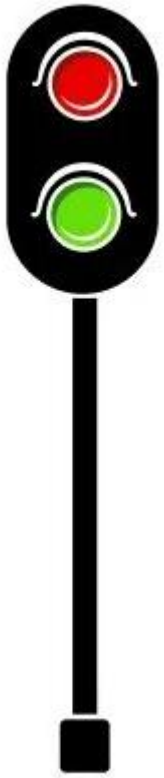
Блокировка

- На низком уровне:

  участок разделяемой памяти, в котором некоторым образом отмечается, свободна ли блокировка или захвачена (и, возможно, записывается дополнительная информация: номер процесса, время захвата и т. п.).

- На уровень ниже:

  специальные примитивы синхронизации (семафоры или мьютексы), предоставляемые ОС. Код, обращающийся к разделяемому ресурсу, одновременно выполняется только в одном процессе.



Блокировка

- На низком уровне:

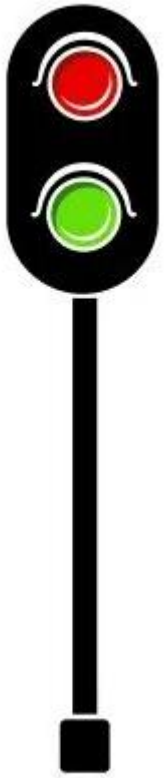
  участок разделяемой памяти, в котором некоторым образом отмечается, свободна ли блокировка или захвачена (и, возможно, записывается дополнительная информация: номер процесса, время захвата и т. п.).

- На уровень ниже:

  специальные примитивы синхронизации (семафоры или мьютексы), предоставляемые ОС. Код, обращающийся к разделяемому ресурсу, одновременно выполняется только в одном процессе.

- На самом низком уровне:

  реализация на основе атомарных инструкций процессора



Блокировка

Накладывается:

- на определенный ресурс
  реальный объект БД (таблица, индекс, запись, страница данных)

- **некоторый виртуальный ID**



Блокировка

Накладывается:

- Всегда **последовательно** –
  невозможно ничего заблокировать «одновременно»

Блокировка

Накладывается:

- Всегда **последовательно** –
  невозможно ничего заблокировать «одновременно»

- Если удерживать блокировку долго, то возникнут очереди



Блокировка

Накладывается:

- **Изолированно** для каждого соединения/участника
  невозможно управлять «чужими» блокировками

Блокировка

Накладывается:

- **Изолированно** для каждого соединения/участника
  невозможно управлять «чужими» блокировками

- Но можно «попросить» сервер «разобраться с кем надо»

  //общий предок –
  процесс-координатор –
  postmaster



Блокировка

Накладывается:

- **Изолированно** для каждого соединения/участника
  невозможно управлять «чужими» блокировками

- Но можно «попросить» сервер
  «разобраться с кем надо»
  //общий предок – процесс-координатор – **postmaster**

2 команды, которые реально что-то могут сделать

```
SELECT pg_cancel_backend(pid); --прерывание запроса в процессе
SELECT pg_terminate_backend(pid); --завершение запроса
```

Блокировка



Накладывается:

- **В момент возникновения потребности**

  выполнение определенного ЗАПРОСА в ходе транзакции

  изменение конкретной ЗАПИСИ в ходе запроса
  (для блокировки на уровне записи)

➢ Когда сервер реально добрался до конкретного исполнения

Блокировка



Снимается:

1. **ВНЕ ТРАНЗАКЦИИ:**

- при завершении процесса /обрыве соединения
- при поступлении явной команды на снятие блокировки

нетранзакционные блокировки pg_advisory_unlock()*

Блокировка



Снимается:

2. ВНУТРИ ТРАНЗАКЦИИ:

- при ее завершении (COMMIT/ROLLBACK)
- при откате к точке восстановления rollback to sp
  для блокировок наложенных после savepoint sp
- при завершении exception-блока в хранимой процедуре
  для блокировок наложенных в нем
  только при возникновении этого исключения

Варианты блокировок

Table-Level Locks
Уровня таблиц

Row-Table Locks
Уровня записей

Page-Level Locks
Уровня страниц

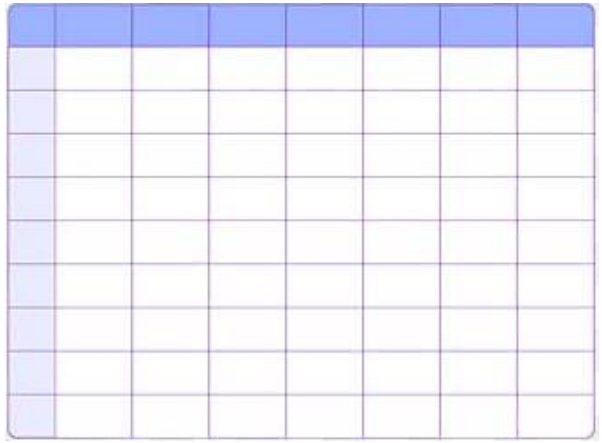
Advisory Locks
Рекомендательные блокировки

Deadlocks
Взаимоблокировки

Table-Level Locks

Блокируемый ресурс – «отношение»
(таблицы, последовательности, индексы и др.)

- может блокировать и чтение, и запись
- распространяется и на записи pg_catalog.*
  (записи таблиц с мета-описанием)



Возникновение

- автоматически при части операций
- при вызове команды LOCK

Table-Level Locks

Режимы блокировки отличаются только преопределенным набором конфликтов

Запрашиваемый режим		Текущий режим							
		AS	RS	RE	SUE	S	SRE	E	AE
Access Share	AS								x
Row Share	RS							x	x
Row Exclusive	RE					x	x	x	x
Share Update Exclusive	SUE				x	x	x	x	x
Share	S			x	x		x	x	x
Share Row Exclusive	SRE			x	x	x	x	x	x
Exclusive	E		x	x	x	x	x	x	x
Access Exclusive	AE	x	x	x	x	x	x	x	x

Table-Level Locks

Shared locks – разделяемые блокировки (S-блокировки) – блокировки с взаимным доступом, могут одновременно устанавливаться несколькими пользователями.

eXclusive locks – монопольные блокировки (X-блокировки) – жесткие блокировки без взаимного доступа, устанавливаются только одним пользователем, получающим эксклюзивный доступ к данным.

Table-Level Locks

Возникают, когда мы делаем какую-либо операцию

Запускаемая команда		Исполняющаяся команда							
		AS	RS	RE	SUE	S	SRE	E	AE
SELECT	AS								x
SELECT ... FOR ...	RS							x	x
INSERT / UPDATE / DELETE / MERGE	RE					x	x	x	x
VACUUM / ANALYZE CREATE INDEX CONCURRENTLY REINDEX CONCURRENTLY CREATE STATISTICS COMMENT ON ALTER INDEX / ALTER TABLE [*]	SUE				x	x	x	x	x
CREATE INDEX	S			x	x		x	x	x
CREATE TRIGGER ALTER TABLE [*]	SRE			x	x	x	x	x	x
REFRESH MATERIALIZED VIEW CONCURRENTLY	E		x	x	x	x	x	x	x
DROP TABLE VACUUM FULL TRUNCATE REINDEX CLUSTER REFRESH MATERIALIZED VIEW ALTER INDEX / ALTER TABLE [*] LOCK	AE	x	x	x	x	x	x	x	x

Table-Level Locks

На все таблицы запроса и все их индексы

Запускаемая команда		Исполняющаяся команда							
		AS	RS	RE	SUE	S	SRE	E	AE
SELECT	AS								x
SELECT ... FOR ...	RS							x	x
INSERT / UPDATE / DELETE / MERGE	RE					x	x	x	x
VACUUM / ANALYZE CREATE INDEX CONCURRENTLY REINDEX CONCURRENTLY CREATE STATISTICS COMMENT ON ALTER INDEX / ALTER TABLE [*]	SUE				x	x	x	x	x
CREATE INDEX	S			x	x		x	x	x
CREATE TRIGGER ALTER TABLE [*]	SRE			x	x	x	x	x	x
REFRESH MATERIALIZED VIEW CONCURRENTLY	E		x	x	x	x	x	x	x
DROP TABLE VACUUM FULL TRUNCATE REINDEX CLUSTER REFRESH MATERIALIZED VIEW ALTER INDEX / ALTER TABLE [*] LOCK	AE	x	x	x	x	x	x	x	x

Table-Level Locks

Как не ждать совсем

LOCK … NOWAIT

// Команда «обвязана» в транзакцию

```
ALTER TABLE x …;
-- ждем все предыдущие команды
-- все последующие команды
   тоже ждут
```

```
BEGIN;
LOCK x IN ACCESS EXCLUSIVE MODE NOWAIT,
-- либо успешно захватим блокировку
-- либо сразу выпадем в EXCEPTION
ALTER TABLE x …;
COMMIT;
```

Table-Level Locks

Как не ждать (хотя немножко можно)

lock_timeout

```
BEGIN;
LOCK x IN ACCESS EXCLUSIVE
        MODE NOWAIT,
-- сразу EXCEPTION при конфликте
ALTER TABLE x …;
COMMIT;
```

```
BEGIN;
SET LOCAL lock_timeout = 100; -- ms
LOCK x IN ACCESS EXCLUSIVE
        MODE NOWAIT,
-- EXCEPTION только после 100 мс ожидания
ALTER TABLE x …;
COMMIT;
```

Table-Level Locks

Как не ждать

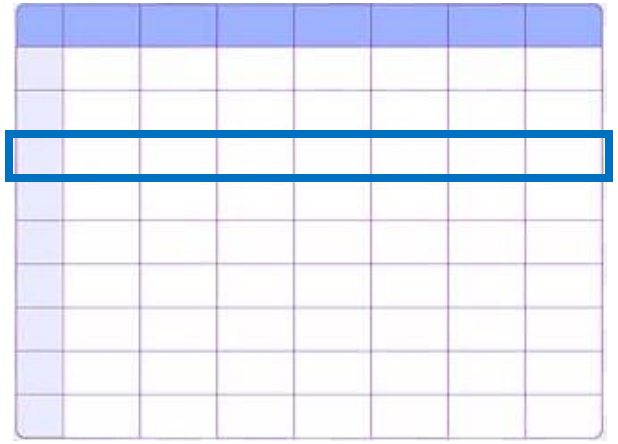
- кто-то открыл транзакцию и «ушел»

idle_in_transaction_session_timeout

Соединение (без активных запросов) закроется по истечении времени



Row-Level Locks



Блокируемый ресурс – строка таблицы

- может блокировать только запись (той же строки)
- не влияет на чтение

Возникновение

- автоматически при UPDATE/DELETE/MERGE
- при вызове команды SELECT … FOR …

SELECT … FOR UPDATE …

Row-Level Locks

Запускаемая команда		Исполняющаяся команда			
		RS	S	SRE	E
SELECT ... FOR KEY SHARE	RS				x
SELECT ... FOR SHARE	S			x	x
SELECT ... FOR NO KEY UPDATE UPDATE [*]	SRE		x	x	x
SELECT ... FOR UPDATE DELETE UPDATE (cols of UNIQUE INDEX)	E	x	x	x	x

Row-Level Locks

- Как не ждать

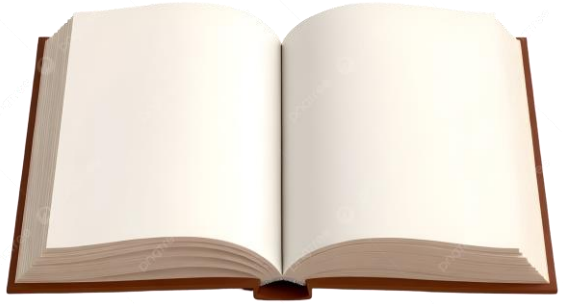
- SELECT … FOR … **NOWAIT**

- SELECT … FOR … **SKIP LOCKED** –
  «пропуск заблокированных строк»

Page-Level Locks

Блокируемый ресурс – запись страницы таблицы/индекса
(их физического представления)

- может блокировать и чтение, и запись



Возникновение

- кратковременные системные операции
  сброс страницы данных на диск, сплит дерева индекса …

Page-Level Locks

Блокировка:

- Кратковременная
- Физическая
- Никак не привязана к транзакциям
- Снимается сразу после завершения **операции**

Запрашиваемый режим		Текущий режим	
		S	E
Share	S		x
Exclusive	E	x	x

- Мы не может на них повлиять и скорее всего «не видим»
- Если видим, то либо тупик, либо проблемы с аппаратными ресурсами (долго пишется на диск)


PostgreSQL
advisory locks

Advisory Locks

Рекомендательные блокировки

«Обычные» блокировки в PG:

привязаны к конкретному объекту БД (таблице, записи, странице) и процессу, обслуживающему соединение.

Advisory locks:

тоже привязаны к процессу,
но вместо реального объекта — абстрактный идентификатор,
который можно задать как **(bigint)** или как **(integer, integer)** –
число, полученное как угодно (oid таблицы, pk и т.д.)

Advisory Locks

Рекомендательные блокировки

- Блокировки не транзакционные, не привязаны к транзакциям
- pg_advisory_lock()

  pg_advisory_unlock()
- pg_**try**_advisory_lock()

  pg_**try**_advisory_unlock()



Advisory Locks

Запрашиваемый режим		Текущий режим	
		S	E
Share	S		x
Exclusive	E	x	x

Advisory Locks

Преимущества

- можно блокировать «то, чего нет» (например, новую строку)
- можно не ждать занятую блокировку
- CAS-проверка возможности захвата блокировки

> CAS — это Compare-and-Set, то есть проверка возможности захвата и сам захват блокировки проходят как одна атомарная операция, и между ними заведомо никто не может «вклиниться».

- можно привязать к транзакции, а можно – к соединению

pg_try_advisory_xact_lock() Pg_try_advisory_lock()

Advisory Locks

Возможные проблемы:

- слишком много блокировок

«И рекомендательные, и обычные блокировки сохраняются в области общей памяти, размер которой определяется параметрами конфигурации **max_locks_per_transaction** и **max_connections**. Важно, чтобы этой памяти было достаточно, так как **в противном случае сервер не сможет выдать никакую блокировку**. Таким образом, число рекомендательных блокировок, которые может выдать сервер, ограничивается обычно десятками или сотнями тысяч в зависимости от конфигурации сервера.»

Advisory Locks

Возможные проблемы:

- мультиналожение одной advisory-блокировки
  (со стороны бизнес-логики: try-catch)

> «Когда поступает несколько запросов на блокировку сеансового уровня, **они накапливаются, так что если один идентификатор ресурса был заблокирован три раза, должны поступить три запроса на освобождение блокировки**, чтобы ресурс был разблокирован до завершения сеанса.»

Advisory Locks

Возможные проблемы:

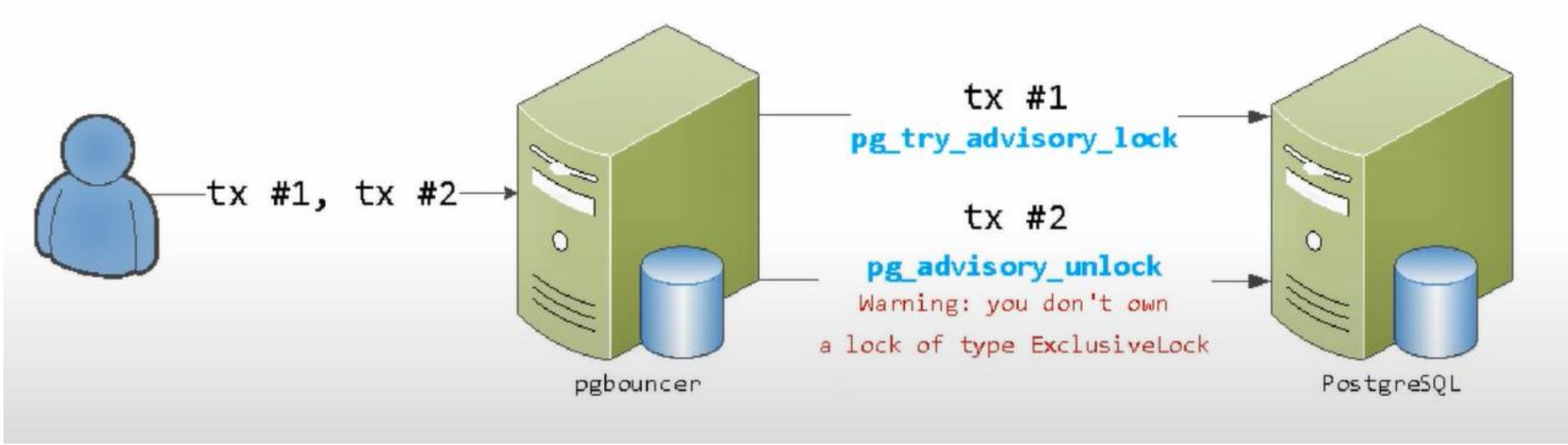
- утечки при фильтрации записей

```
SELECT pg_advisory_lock(id) FROM foo WHERE id = 12345;           -- ok
SELECT pg_advisory_lock(id) FROM foo WHERE id > 12345 LIMIT 100; -- опасно!
...
SELECT pg_advisory_unlock_all(); -- «Гильотина – лучшее средство от головной боли!»
```

Advisory Locks

Возможные проблемы:

- утечки через **pgbouncer в transaction mode**



Advisory Locks

Возможные проблемы:

- Забыть try:

  *pg_**try**_advisory_lock* – не ждет, сразу возвращает FALSE
  pg_advisory_lock **–** ждет освобождения блокирвоки

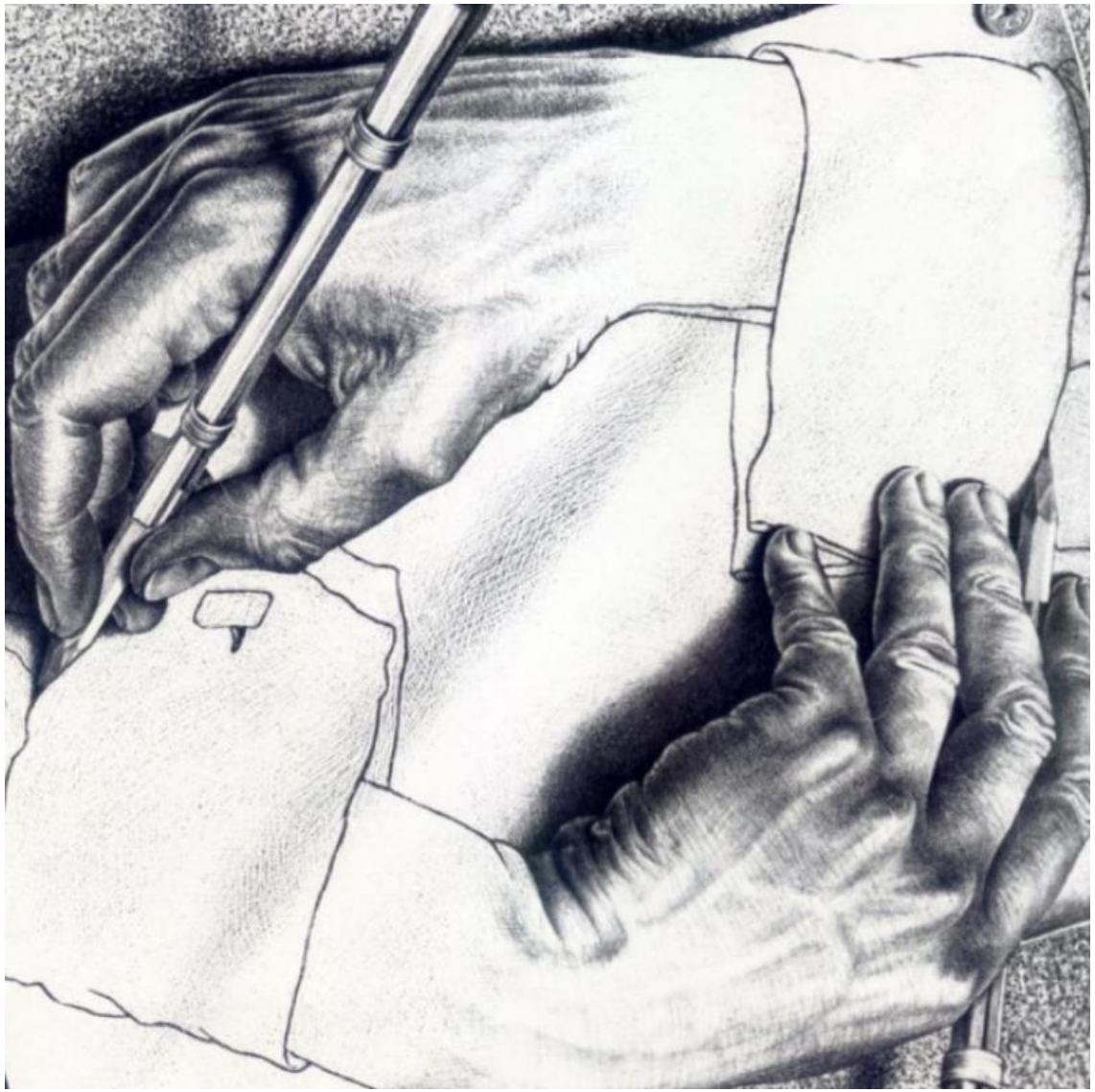
- Забыть xact:

  *pg_try_advisory_**xact**_lock* – снимается завершением транзакции
  pg_try_advisory_lock – остается висеть на соединении

Deadlock

- Взаимоблокировки
- Тупики

- **Тупиковые ситуации** возникают при взаимных блокировках транзакций, выполняющихся на пересекающихся множествах данных.



Deadlock

Несколько блокировок ждут друг друга

```
ALTER TABLE x …;
-- ok

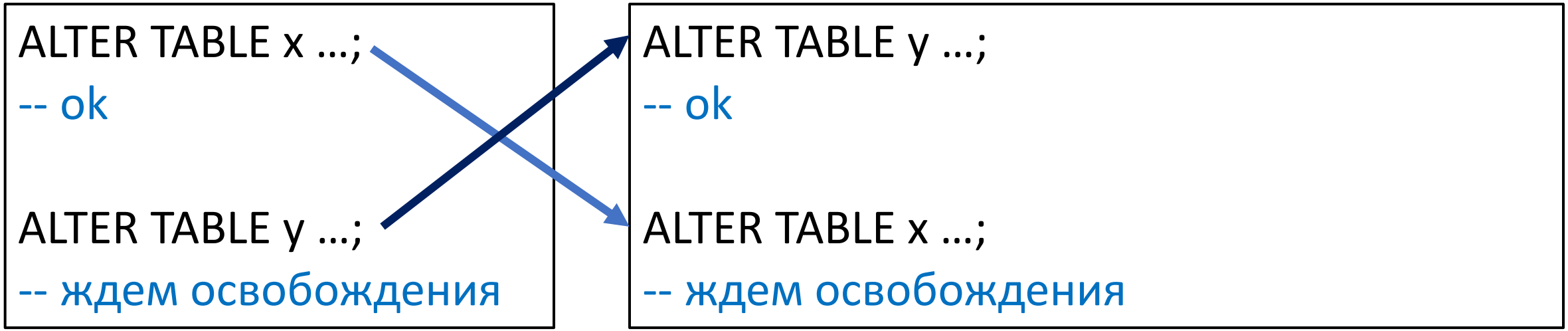
ALTER TABLE y …;
-- ждем освобождения
```

```
ALTER TABLE y …;
-- ok

ALTER TABLE x …;
-- ждем освобождения
```

Deadlock

Несколько блокировок ждут друг друга



Deadlock

Несколько блокировок ждут друг друга

```
ALTER TABLE x …;
-- ok

ALTER TABLE y …;
-- ждем освобождения
```

```
ALTER TABLE y …;
-- ok

ALTER TABLE x …;
-- ждем освобождения

-- ERROR: deadlock detected
-- откат транзакции и снятие блокировок
```

Deadlock

Как не ждать:

deadlock_timeout

Сервер сам распознает deadlock’и:

«Эта проверка довольно дорогостоящая, поэтому сервер не выполняет её при всяком ожидании блокировки. Мы оптимистично полагаем, что взаимоблокировки редки в производственных приложениях, и поэтому **просто ждём некоторое время, прежде чем пытаться выявить взаимоблокировку**. При увеличении значения этого параметра сокращается время, уходящее на ненужные проверки взаимоблокировки, но замедляется реакция на реальные взаимоблокировки.»

Deadlock

Не всегда может распознаться

```
DO $$
BEGIN
    SELECT dblink_exec(
      format('dbname=%s user=%s', current_database(), current_user),
      'CREATE INDEX CONCURRENTLY idx_cic ON x(pk);'
    );
END
$$

-- CONCURRENTLY ждет на своем пути окончания всех транзакций,
включая наш DO
```

Мониторинг блокировок

SELECT * FROM **pg_stat_activity** WHERE wait_event_type IS NOT NULL;

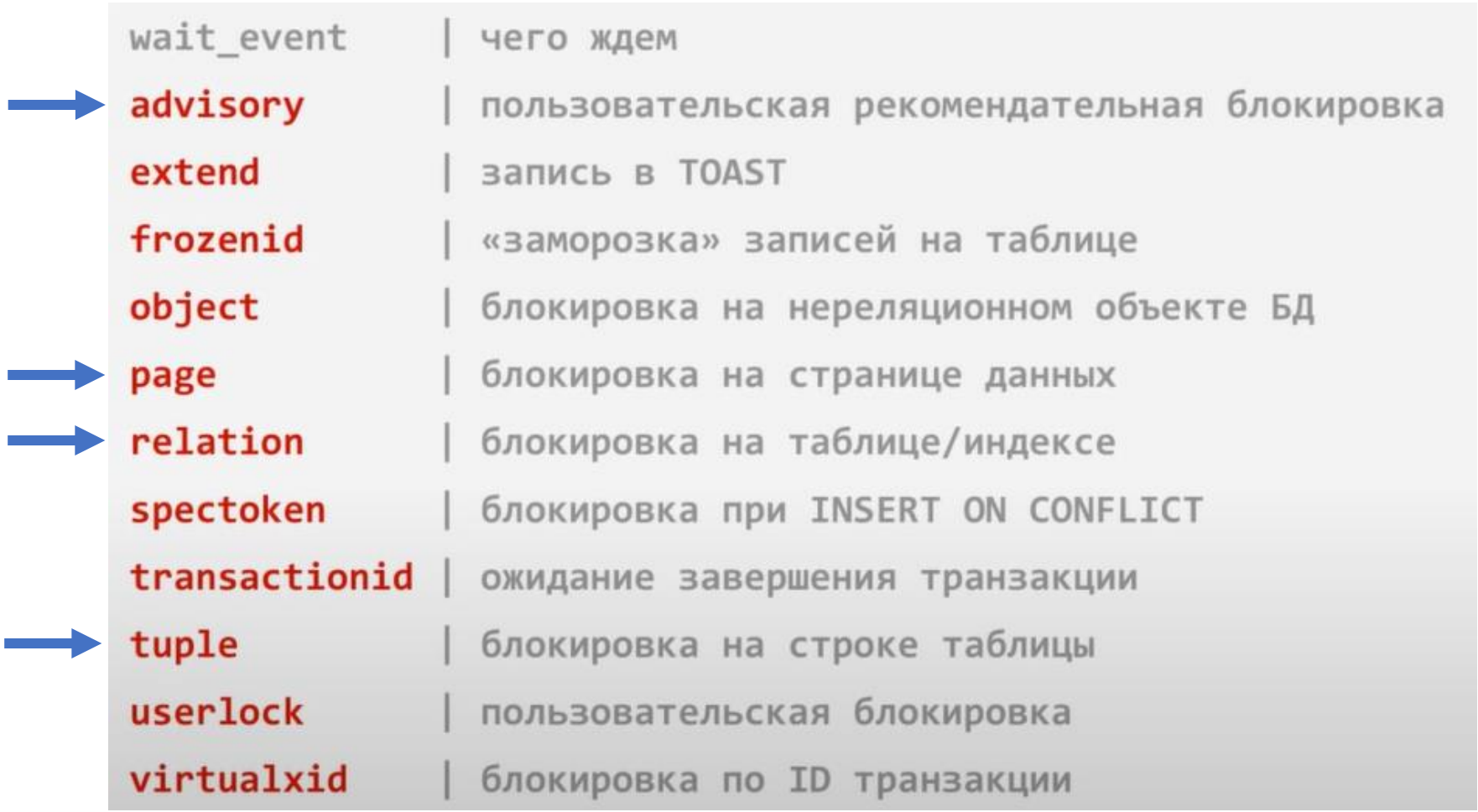
-- кто ждет

wait_event_type	чего ждем
Activity	пауза в системных фоновых процессах (autovacuum, archiver, checkpointer)
BufferPin	эксклюзивный доступ к странице данных
Client	ожидание активности со стороны клиента
Extension	разрешение условия внутри расширения
IO	ожидание ввода/вывода
IPC	ожидание межпроцессного обмена
Lock	ожидание классической «тяжелой» блокировки (объект БД/ID)
LWLock	ожидание «легкой» блокировки в памяти
Timeout	ждем таймаута (пауза в системных фоновых процессах или pg_sleep)

Мониторинг блокировок

SELECT * FROM **pg_stat_activity** WHERE wait_event_type = ‘Lock’;

-- чего ждет



Мониторинг блокировок

SELECT * FROM **pg_locks** WHERE NOT granted;

-- кто конкретно чего ждет

```
advisory      | (classid, objid, objsubid)
extend        | (relation)
frozenid      | (relation)
object        | (classid, objid, objsubid)
page          | (relation, page)
relation      | (relation)
spectoken     | (relation, page, tuple)
transactionid | (transactionid)
tuple         | (relation, page, tuple)
userlock      | (classid)
virtualxid    | (virtualxid)
```