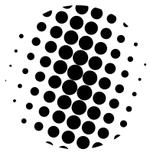
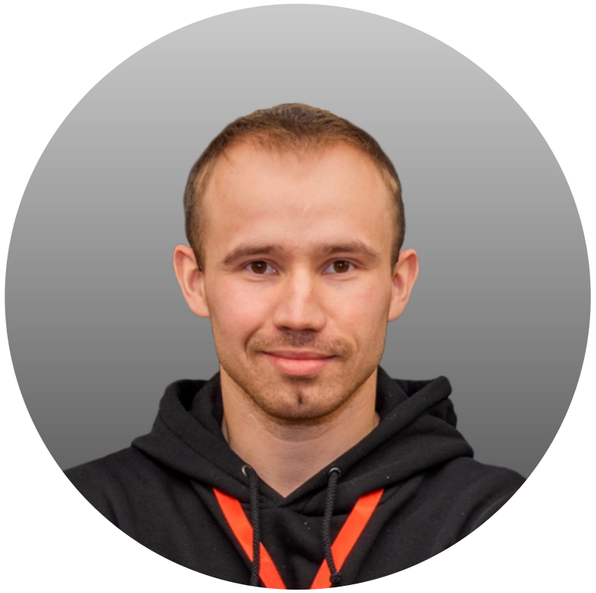


SmartData

SQL миграции в Postgres под нагрузкой

Николай Аверин
Miro

О докладчике



Николай Аверин

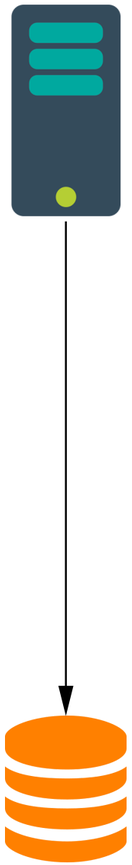
backend engineer

- ~ 3 года в Miro
- мигрируем Redis → PostgreSQL
- реализуем multi-tenant архитектуру хранения данных
- реализуем шардирование данных на уровне приложения

План

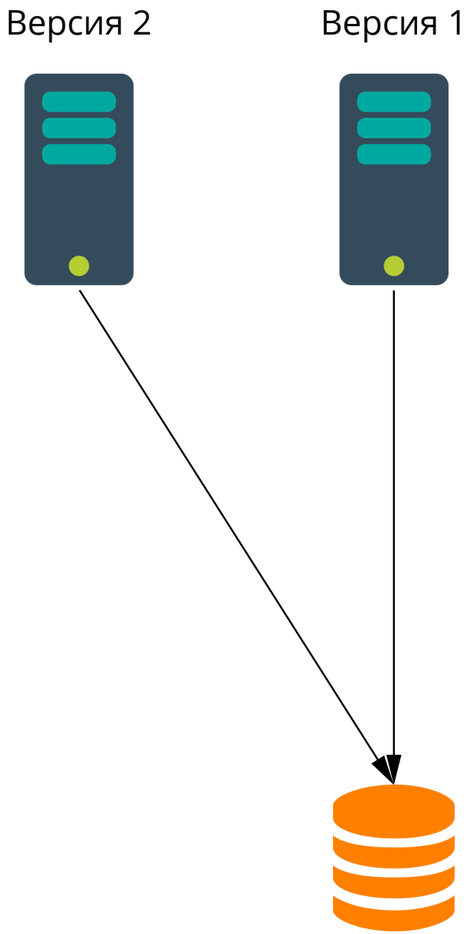
- В чем проблема
- Базовые операции:
    - add column
    - create index
    - и пр.
- Более сложные миграции:
    - обновление большой таблицы
    - разбиение таблиц

Суть проблемы

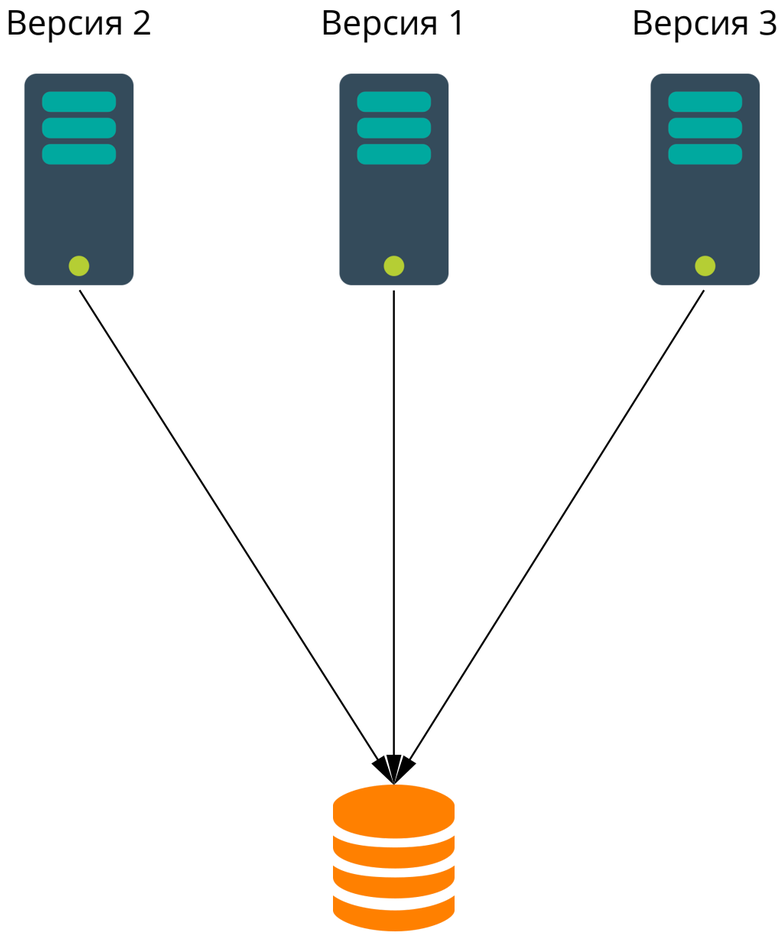


- downtime при релизе
- сервер приложений и БД синхронны

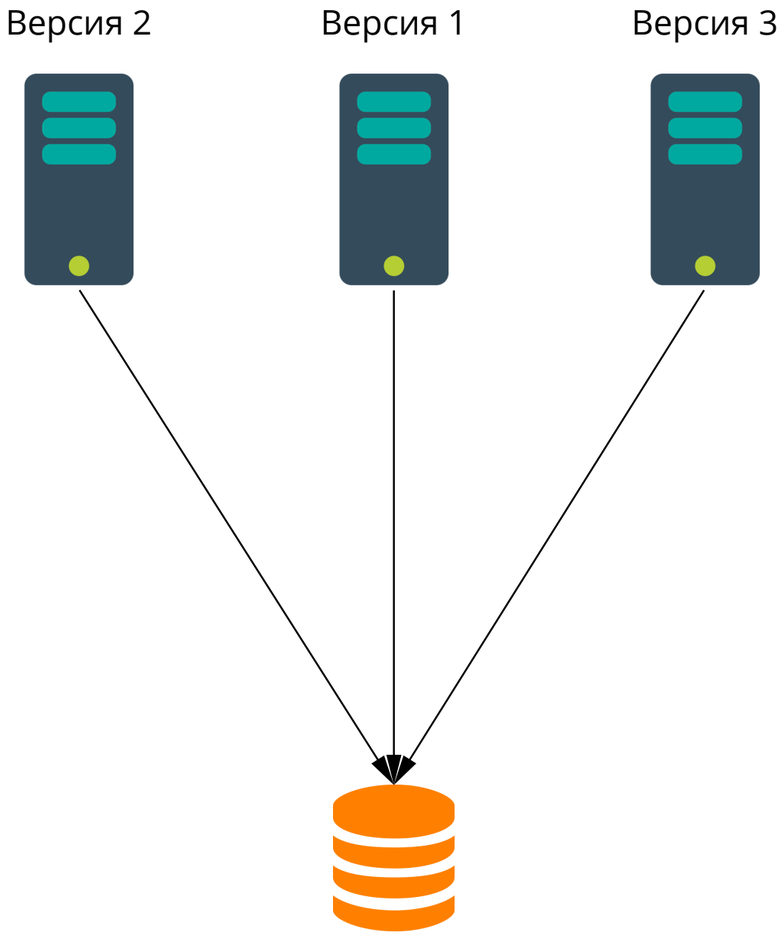
Суть проблемы



Суть проблемы

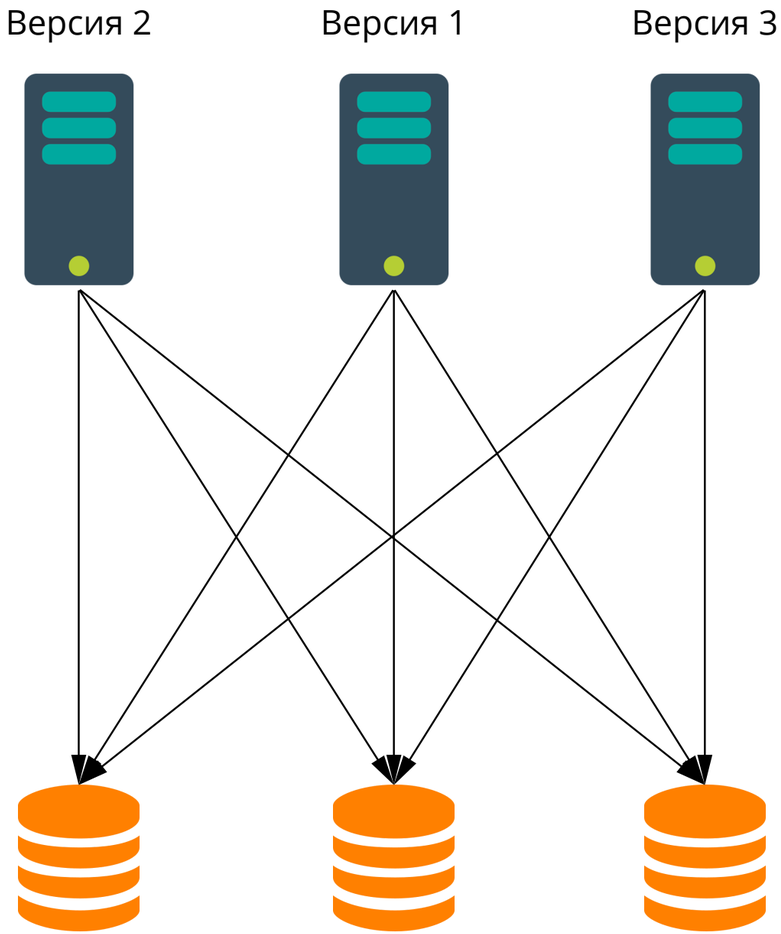


Суть проблемы



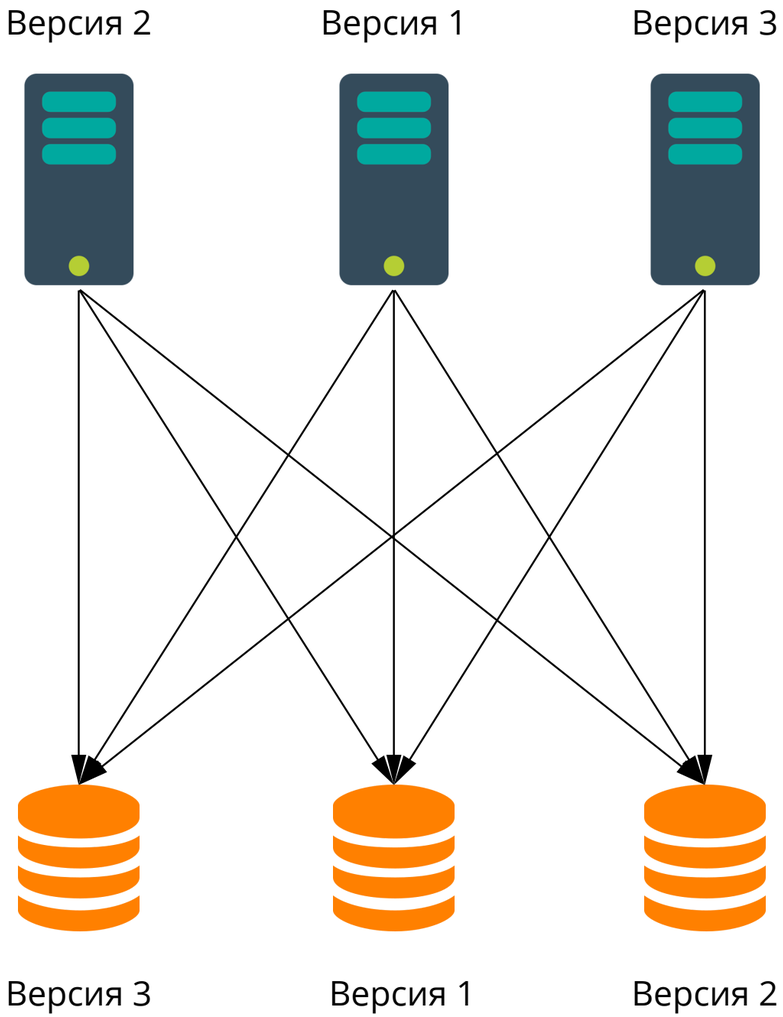
- downtime → 0
- разный код на серверах приложений

Суть проблемы



- downtime → 0
- разный код на серверах приложений

Суть проблемы



- downtime → 0
- разный код на серверах приложений
- разные схемы в БД

ADD COLUMN

```
ALTER TABLE my_table ADD COLUMN new_column INTEGER
```

ADD COLUMN

✔️
```
ALTER TABLE my_table ADD COLUMN new_column INTEGER -- быстро и дешево*
```

ADD COLUMN

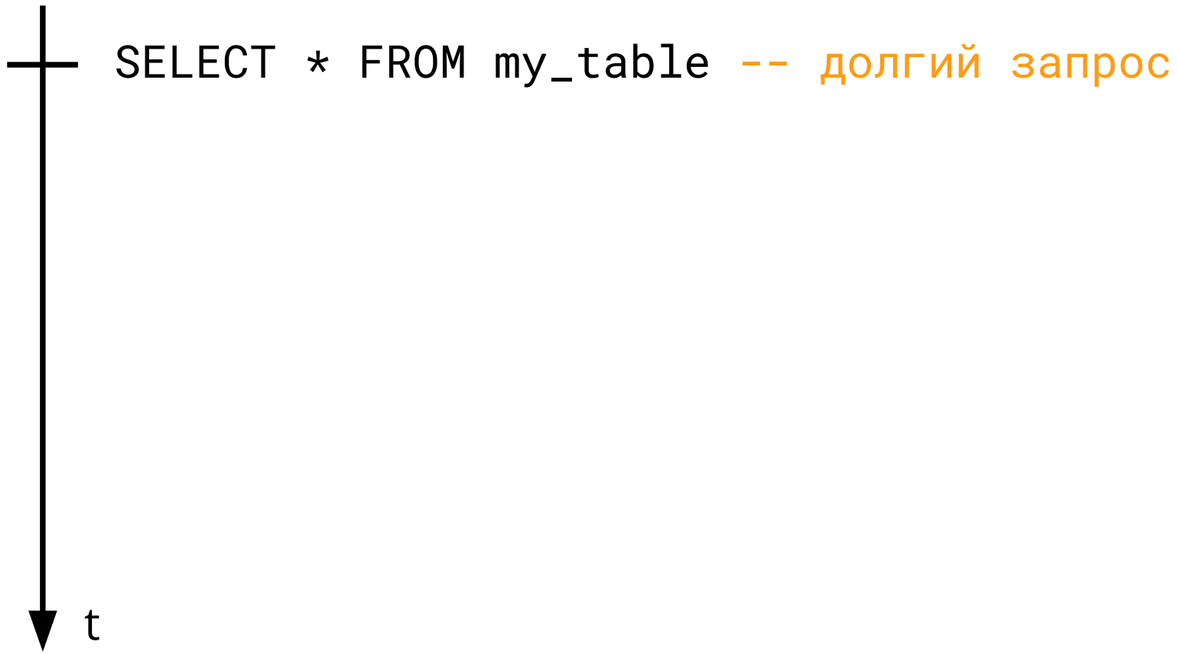
✔ `ALTER TABLE my_table ADD COLUMN new_column INTEGER -- быстро и дешево*`

*если нет долгих блокировок

ADD COLUMN

✔ `ALTER TABLE my_table ADD COLUMN new_column INTEGER -- быстро и дешево*`

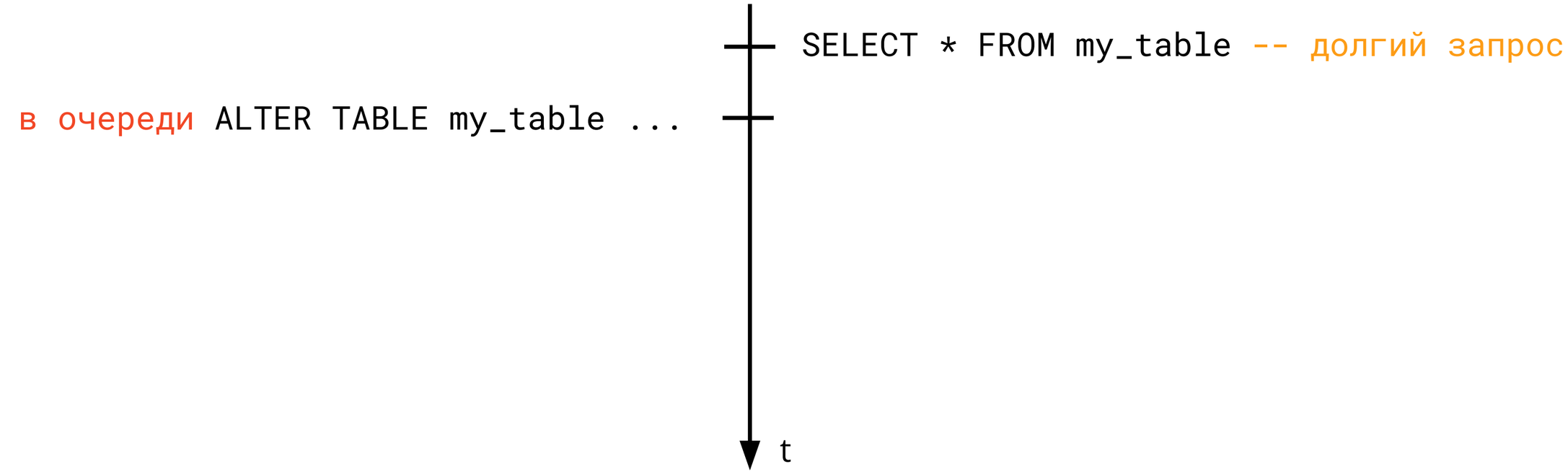
*если нет долгих блокировок



ADD COLUMN

✔ `ALTER TABLE my_table ADD COLUMN new_column INTEGER -- быстро и дешево*`

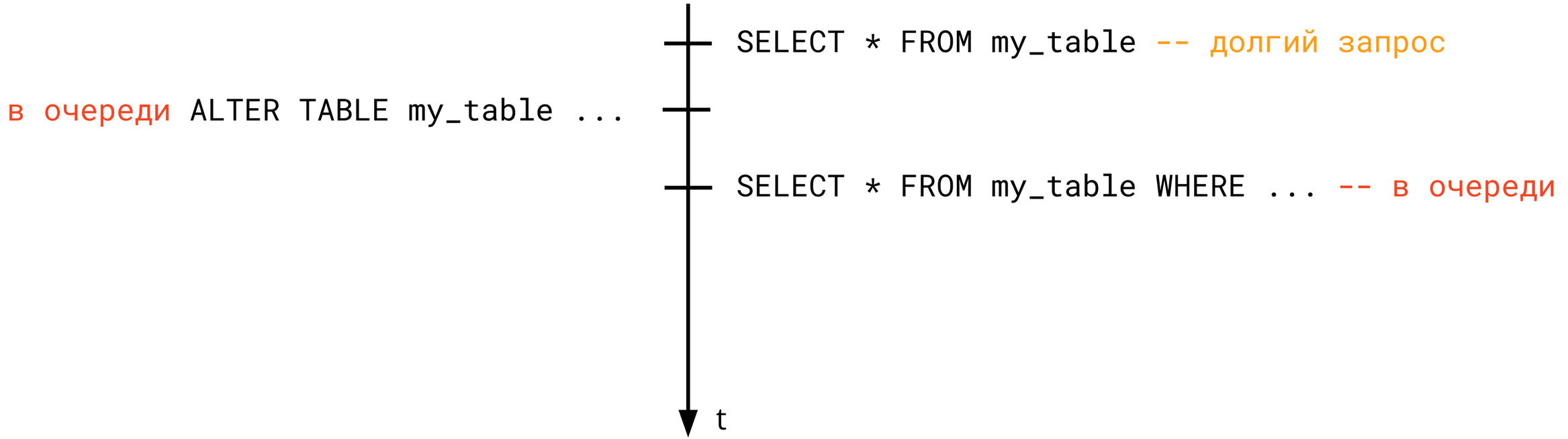
*если нет долгих блокировок



ADD COLUMN

✔ `ALTER TABLE my_table ADD COLUMN new_column INTEGER -- быстро и дешево*`

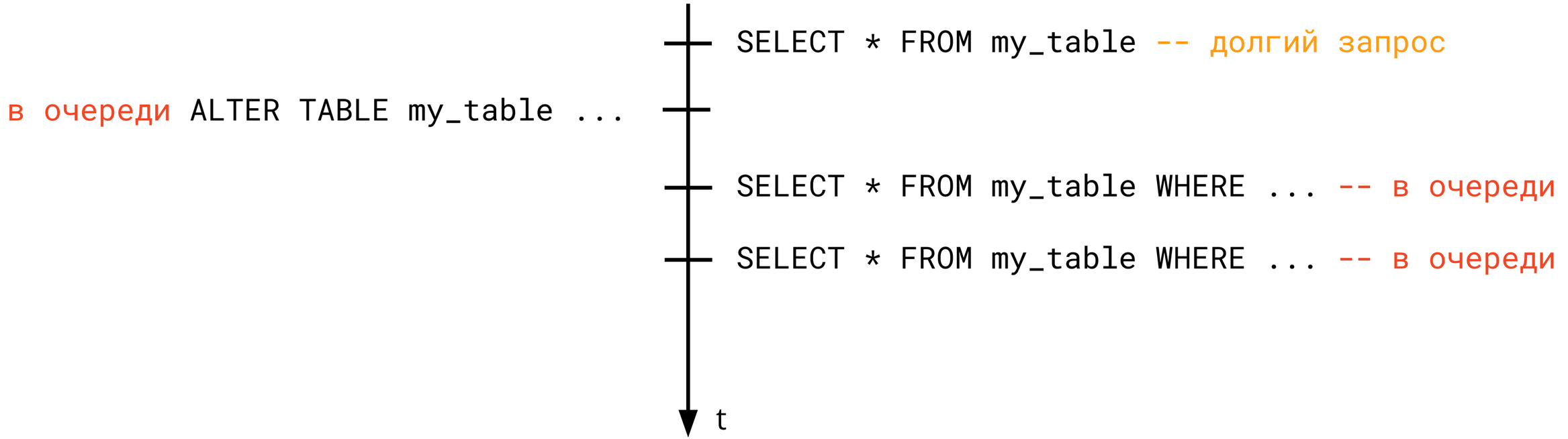
*если нет долгих блокировок



ADD COLUMN

✔ `ALTER TABLE my_table ADD COLUMN new_column INTEGER -- быстро и дешево*`

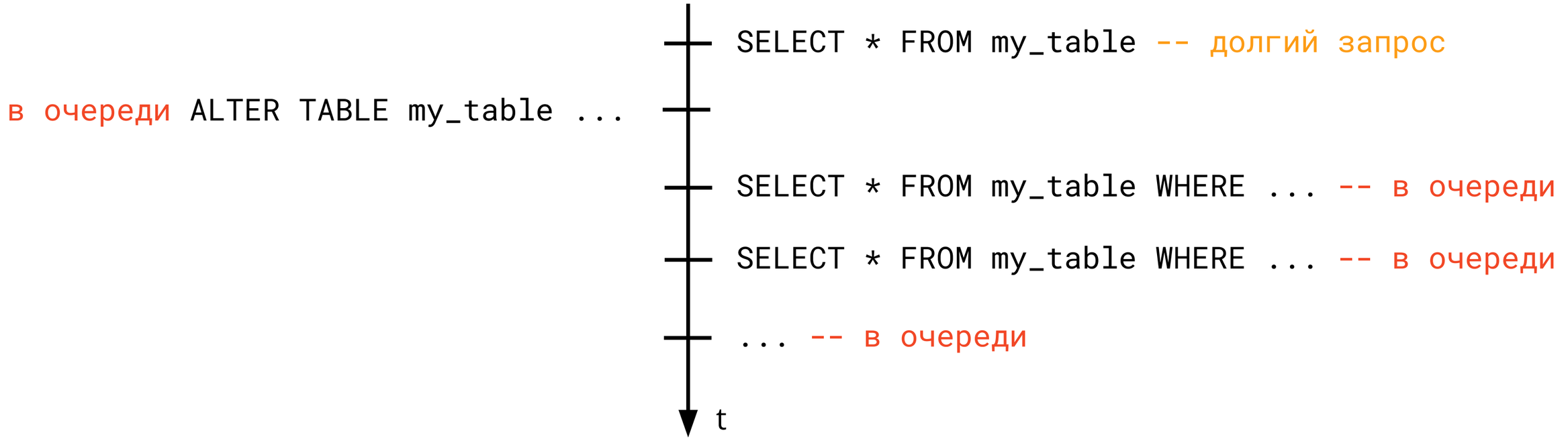
*если нет долгих блокировок



ADD COLUMN

✔ `ALTER TABLE my_table ADD COLUMN new_column INTEGER -- быстро и дешево*`

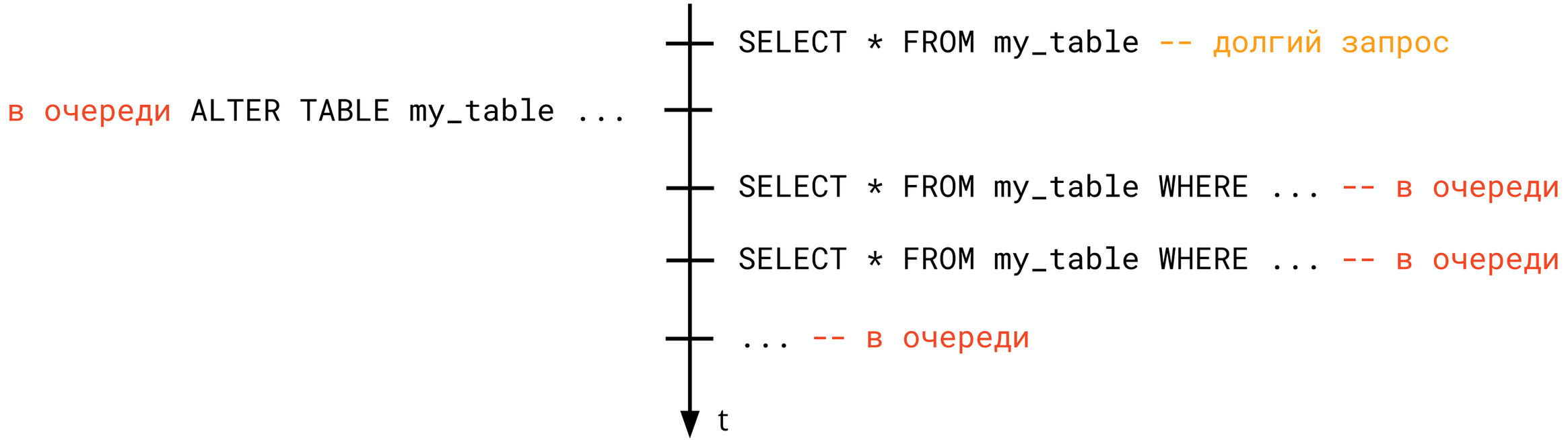
*если нет долгих блокировок



ADD COLUMN

✔ `ALTER TABLE my_table ADD COLUMN new_column INTEGER -- быстро и дешево*`

*если нет долгих блокировок



✔ `SET LOCAL lock_timeout TO '100ms'`

ADD COLUMN with DEFAULT

❌ `ALTER TABLE my_table ADD COLUMN new_column INTEGER DEFAULT 42*`

*в версии до **PG 11**

ADD COLUMN with DEFAULT

❌ `ALTER TABLE my_table ADD COLUMN new_column INTEGER DEFAULT 42*`

*в версии до **PG 11**

✔ *в версии от **PG 11** -- дешево и быстро (pg_attribute) + можно NOT NULL

ADD COLUMN with DEFAULT

❌ `ALTER TABLE my_table ADD COLUMN new_column INTEGER DEFAULT 42*`

*в версии до **PG 11**

✔ *в версии от **PG 11** -- дешево и быстро (pg_attribute) + можно NOT NULL

до **PG 11**:

✔ `1) ALTER TABLE my_table ADD COLUMN new_column INTEGER (Tx 1)`

ADD COLUMN with DEFAULT

❌ `ALTER TABLE my_table ADD COLUMN new_column INTEGER DEFAULT 42*`

*в версии до **PG 11**

✔ *в версии от **PG 11** -- дешево и быстро (pg_attribute) + можно NOT NULL

до **PG 11**:

✔
```
1) ALTER TABLE my_table ADD COLUMN new_column INTEGER          (Tx 1)
   ALTER TABLE my_table ALTER COLUMN new_column SET DEFAULT 42
```

ADD COLUMN with DEFAULT

❌ `ALTER TABLE my_table ADD COLUMN new_column INTEGER DEFAULT 42*`

*в версии до **PG 11**

✔ *в версии от **PG 11** -- дешево и быстро (pg_attribute) + можно NOT NULL

до **PG 11**:

✔
```
1) ALTER TABLE my_table ADD COLUMN new_column INTEGER          (Tx 1)
   ALTER TABLE my_table ALTER COLUMN new_column SET DEFAULT 42
```

❓
```
2) UPDATE my_table set new_column = 42 (Tx 2)
```

DROP COLUMN

✔️
```
ALTER TABLE my_table DROP COLUMN new_column -- быстро и дешево
```

DROP COLUMN

✔ `ALTER TABLE my_table DROP COLUMN new_column -- быстро и дешево`

1) Снять ограничения со столбца (NOT NULL, CHECK, ...)

```
ALTER TABLE my_table ALTER COLUMN new_column DROP NOT NULL
```

DROP COLUMN

✔ `ALTER TABLE my_table DROP COLUMN new_column -- быстро и дешево`

1) Снять ограничения со столбца (NOT NULL, CHECK, ...)

```
ALTER TABLE my_table ALTER COLUMN new_column DROP NOT NULL
```

2) Не использовать столбец в коде приложения

- `SELECT * FROM my_table ...`
- `Hibernate - @Transient`
- `JOOQ - <excludes>my_table.new_column</excludes>`

DROP COLUMN

✔ `ALTER TABLE my_table DROP COLUMN new_column -- быстро и дешево`

1) Снять ограничения со столбца (NOT NULL, CHECK, ...)

```
ALTER TABLE my_table ALTER COLUMN new_column DROP NOT NULL
```

2) Не использовать столбец в коде приложения

- `SELECT * FROM my_table ...`
- `Hibernate - @Transient`
- `JOOQ - <excludes>my_table.new_column</excludes>`

3) ALTER TABLE my_table DROP COLUMN new_column

CREATE INDEX

❌
```
CREATE INDEX my_table_index ON my_table (name) -- блокировка таблицы
```

CREATE INDEX

❌ `CREATE INDEX my_table_index ON my_table (name) -- блокировка таблицы`

✔️ `CREATE CONCURRENTLY INDEX my_table_index ON my_table (name)` **PG 8.2**

CREATE INDEX

❌
```
CREATE INDEX my_table_index ON my_table (name) -- блокировка таблицы
```

✔️
```
CREATE CONCURRENTLY INDEX my_table_index ON my_table (name)
```
PG 8.2

```
SELECT pg_index.indisvalid
FROM pg_class, pg_index
WHERE pg_index.indexrelid = pg_class.oid
  AND pg_class.relname = 'my_table_index'
```

CREATE INDEX

❌
```
CREATE INDEX my_table_index ON my_table (name) -- блокировка таблицы
```

✔️
```
CREATE CONCURRENTLY INDEX my_table_index ON my_table (name)
```
PG 8.2

```
SELECT pg_index.indisvalid
FROM pg_class, pg_index
WHERE pg_index.indexrelid = pg_class.oid
  AND pg_class.relname = 'my_table_index'
```

✔️
```
DROP INDEX CONCURRENTLY my_table_index
```
PG 9.2

CREATE INDEX

❌ `CREATE INDEX my_table_index ON my_table (name) -- блокировка таблицы`

✔️ `CREATE CONCURRENTLY INDEX my_table_index ON my_table (name)` **PG 8.2**

```
SELECT pg_index.indisvalid
FROM pg_class, pg_index
WHERE pg_index.indexrelid = pg_class.oid
  AND pg_class.relname = 'my_table_index'
```

✔️ `DROP INDEX CONCURRENTLY my_table_index` **PG 9.2**

❌ `REINDEX my_table_index -- блокировка таблицы`

CREATE INDEX

❌
```
CREATE INDEX my_table_index ON my_table (name) -- блокировка таблицы
```

✔️
```
CREATE CONCURRENTLY INDEX my_table_index ON my_table (name)
```
PG 8.2

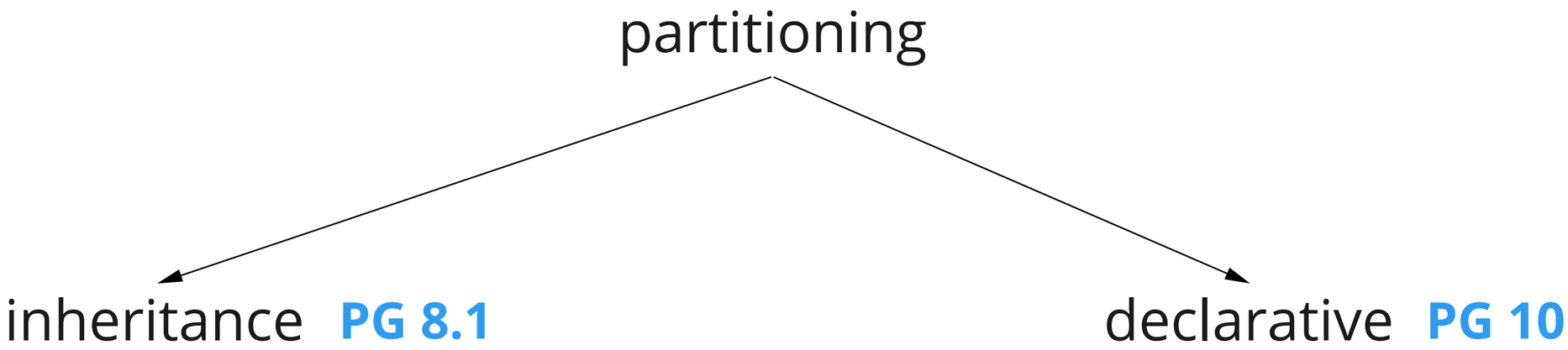
```
SELECT pg_index.indisvalid
FROM pg_class, pg_index
WHERE pg_index.indexrelid = pg_class.oid
  AND pg_class.relname = 'my_table_index'
```

✔️
```
DROP INDEX CONCURRENTLY my_table_index
```
PG 9.2

❌
```
REINDEX my_table_index -- блокировка таблицы
```

✔️
```
REINDEX CONCURRENTLY my_table_index
```
PG 12

CREATE INDEX on a partitioned table



CREATE INDEX on a partitioned table

partitioning using inheritance

```
CREATE TABLE my_table (
    ...
    reg_date   date not null
)

CREATE TABLE my_table_y2020 (
CHECK ( reg_date >= DATE '2020-01-01' AND reg_date < DATE '2021-01-01' ))
INHERITS (my_table);

CREATE TABLE my_table_y2021 (
CHECK ( reg_date >= DATE '2021-01-01' AND reg_date < DATE '2022-01-01' ))
INHERITS (my_table);

-- indexes for each partition
-- an insert trigger / rule
```

CREATE INDEX on a partitioned table

partitioning using inheritance

```
-- indexes for each partition:
CREATE INDEX ON my_table_y2020 (reg_date);
CREATE INDEX ON my_table_y2021 (reg_date);
```

CREATE INDEX on a partitioned table

partitioning using inheritance

```
-- indexes for each partition:
CREATE INDEX ON my_table_y2020 (reg_date);
CREATE INDEX ON my_table_y2021 (reg_date);
```

Новый индекс:

✔
```
CREATE CONCURRENTLY INDEX my_table_y2020_index
ON my_table_y2020 (name)
```

✔
```
CREATE CONCURRENTLY INDEX my_table_y2021_index
ON my_table_y2021 (name)
```

CREATE INDEX on a partitioned table

declarative partitioning

```
CREATE TABLE my_table (...) PARTITION BY RANGE (reg_date);

CREATE TABLE my_table_y2020 PARTITION OF my_table
FOR VALUES FROM ('2020-01-01') TO ('2020-12-31');

CREATE TABLE my_table_y2021 PARTITION OF my_table
FOR VALUES FROM ('2021-01-01') TO ('2021-12-31');

-- indexes for each partition
-- OR
-- an index for the parent table
CREATE INDEX ON my_table (reg_date);
```

PG 11

CREATE INDEX on a partitioned table

declarative partitioning

❌
```
CREATE INDEX ON my_table (name); -- блокировка таблицы
```

CREATE INDEX on a partitioned table

declarative partitioning

❌
```
CREATE CONCURRENTLY INDEX ON my_table (name); -- не реализовано
```

CREATE INDEX on a partitioned table

declarative partitioning

❌
```
CREATE CONCURRENTLY INDEX ON my_table (name); -- не реализовано
```

✔️
```
CREATE INDEX my_table_index ON ONLY my_table (name);
```
PG 11

CREATE INDEX on a partitioned table

declarative partitioning

❌
```
CREATE CONCURRENTLY INDEX ON my_table (name); -- не реализовано
```

✔
```
CREATE INDEX my_table_index ON ONLY my_table (name);
```
PG 11

✔
```
CREATE CONCURRENTLY INDEX my_table_y2020_index
ON my_table_y2020 (name);
```

CREATE INDEX on a partitioned table

declarative partitioning

❌
```
CREATE CONCURRENTLY INDEX ON my_table (name); -- не реализовано
```

✔
```
CREATE INDEX my_table_index ON ONLY my_table (name);
```
PG 11

✔
```
CREATE CONCURRENTLY INDEX my_table_y2020_index
ON my_table_y2020 (name);
```

✔
```
ALTER INDEX my_table_index ATTACH PARTITION my_table_y2020_index;
...
```

ADD NOT NULL CONSTRAINT

❌
```
ALTER TABLE my_table ALTER COLUMN name SET NOT NULL  -- блокировка таблицы
```

ADD NOT NULL CONSTRAINT

❌
```
ALTER TABLE my_table ALTER COLUMN name SET NOT NULL  -- блокировка таблицы
```

✔️
```
1) ALTER TABLE my_table ADD CONSTRAINT chk_name_not_null
   CHECK (name IS NOT NULL) NOT VALID (Tx 1)
```

PG 9.2

ADD NOT NULL CONSTRAINT

❌
```
ALTER TABLE my_table ALTER COLUMN name SET NOT NULL  -- блокировка таблицы
```

✔️
```
1) ALTER TABLE my_table ADD CONSTRAINT chk_name_not_null
   CHECK (name IS NOT NULL) NOT VALID (Tx 1)
```

PG 9.2

✔️
```
2) обновление данных, если нужно (Tx 2)
```

ADD NOT NULL CONSTRAINT

❌
```
ALTER TABLE my_table ALTER COLUMN name SET NOT NULL  -- блокировка таблицы
```

✔️
```
1) ALTER TABLE my_table ADD CONSTRAINT chk_name_not_null
   CHECK (name IS NOT NULL) NOT VALID (Tx 1)
```

PG 9.2

✔️
```
2) обновление данных, если нужно (Tx 2)
```

✔️
```
3) ALTER TABLE my_table VALIDATE CONSTRAINT chk_name_not_null (Tx 3)
```

ADD FOREIGN KEY

❌
```
ALTER TABLE my_table ADD CONSTRAINT fk_group FOREIGN KEY (group_id)
REFERENCES groups(id) -- блокировка обеих таблиц
```

ADD FOREIGN KEY

❌
```
ALTER TABLE my_table ADD CONSTRAINT fk_group FOREIGN KEY (group_id)
REFERENCES groups(id) -- блокировка обеих таблиц
```

✔️
```
1) ALTER TABLE my_table ADD CONSTRAINT fk_group FOREIGN KEY (group_id)
   REFERENCES groups(id) NOT VALID (Tx 1)
```

PG 9.1

ADD FOREIGN KEY

❌
```
ALTER TABLE my_table ADD CONSTRAINT fk_group FOREIGN KEY (group_id)
REFERENCES groups(id) -- блокировка обеих таблиц
```

✔️
```
1) ALTER TABLE my_table ADD CONSTRAINT fk_group FOREIGN KEY (group_id)
   REFERENCES groups(id) NOT VALID (Tx 1)
```

PG 9.1

✔️
```
2) обновление данных, если нужно (Tx 2)
```

ADD FOREIGN KEY

❌
```
ALTER TABLE my_table ADD CONSTRAINT fk_group FOREIGN KEY (group_id)
REFERENCES groups(id) -- блокировка обеих таблиц
```

✔️
```
1) ALTER TABLE my_table ADD CONSTRAINT fk_group FOREIGN KEY (group_id)
   REFERENCES groups(id) NOT VALID (Tx 1)
```

PG 9.1

✔️
```
2) обновление данных, если нужно (Tx 2)
```

✔️
```
3) ALTER TABLE my_table VALIDATE CONSTRAINT fk_group_id (Tx 3)
```

ADD UNIQUE/PRIMARY KEY

❌
```
ALTER TABLE my_table ADD CONSTRAINT uk_my_table_id UNIQUE (id)
```
-- блокировка таблицы

ADD UNIQUE/PRIMARY KEY

❌
```
ALTER TABLE my_table ADD CONSTRAINT uk_my_table_id UNIQUE (id)
```
-- блокировка таблицы

✔️
```
1) CREATE UNIQUE INDEX CONCURRENTLY uk_my_table_id ON my_table(id)
```

ADD UNIQUE/PRIMARY KEY

❌
```
ALTER TABLE my_table ADD CONSTRAINT uk_my_table_id UNIQUE (id)
```
-- блокировка таблицы

✔️
```
1) CREATE UNIQUE INDEX CONCURRENTLY uk_my_table_id ON my_table(id)
```

✔️
```
2) ALTER TABLE my_table ADD CONSTRAINT uk_my_table_id UNIQUE
   USING INDEX uk_my_table_id
```
PG 9.1

ADD UNIQUE/PRIMARY KEY

❌
```
ALTER TABLE my_table ADD CONSTRAINT uk_my_table_id UNIQUE (id)
```
-- блокировка таблицы

✔️
```
1) CREATE UNIQUE INDEX CONCURRENTLY uk_my_table_id ON my_table(id)
```

✔️
```
2) ALTER TABLE my_table ADD CONSTRAINT uk_my_table_id UNIQUE
   USING INDEX uk_my_table_id
```
PG 9.1

✔️
```
ALTER TABLE my_table ADD CONSTRAINT uk_my_table_id PRIMARY KEY
USING INDEX uk_my_table_id*
```
-- если id is **NOT NULL**

ADD UNIQUE/PRIMARY KEY

❌
```
ALTER TABLE my_table ADD CONSTRAINT uk_my_table_id UNIQUE (id)
```
-- блокировка таблицы

✔️
```
1) CREATE UNIQUE INDEX CONCURRENTLY uk_my_table_id ON my_table(id)
```

✔️
```
2) ALTER TABLE my_table ADD CONSTRAINT uk_my_table_id UNIQUE
   USING INDEX uk_my_table_id
```
PG 9.1

✔️
```
ALTER TABLE my_table ADD CONSTRAINT uk_my_table_id PRIMARY KEY
USING INDEX uk_my_table_id*
```
-- если id is **NOT NULL**, CHECK(id is not null) **не поможет**

ADD PRIMARY KEY in PG 11

✔1) `ALTER TABLE my_table ADD COLUMN` **`new_id`** `INTEGER` **`NOT NULL DEFAULT -1`**

ADD PRIMARY KEY in PG 11

✔ 1) `ALTER TABLE my_table ADD COLUMN **new_id** INTEGER **NOT NULL DEFAULT -1**`

✔ 2) создаем **триггер** для репликации **id** в **new_id**:

```
CREATE FUNCTION on_insert_or_update() RETURNS TRIGGER AS
$$
BEGIN
    NEW.new_id = NEW.id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

ADD PRIMARY KEY in PG 11

✔ 1) `ALTER TABLE my_table ADD COLUMN **new_id** INTEGER **NOT NULL DEFAULT -1**`

✔ 2) создаем **триггер** для репликации **id** в **new_id**:

```
CREATE FUNCTION on_insert_or_update() RETURNS TRIGGER AS
$$
BEGIN
    NEW.new_id = NEW.id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg BEFORE INSERT OR UPDATE ON my_table
FOR EACH ROW EXECUTE PROCEDURE on_insert_or_update();
```

ADD PRIMARY KEY in PG 11

✔️3) `UPDATE my_table SET new_id = id WHERE` **`new_id = -1`**

ADD PRIMARY KEY in PG 11

✔️3) `UPDATE my_table SET new_id = id WHERE` **`new_id = -1`**

✔️4)
```
ALTER TABLE my_table RENAME COLUMN id TO old_id;
ALTER TABLE my_table RENAME COLUMN new_id TO id;
ALTER TABLE my_table RENAME COLUMN old_id TO new_id;
```

ADD PRIMARY KEY in PG 11

✔️3) `UPDATE my_table SET new_id = id WHERE` **`new_id = -1`**

✔️4)
```
ALTER TABLE my_table RENAME COLUMN id TO old_id;
ALTER TABLE my_table RENAME COLUMN new_id TO id;
ALTER TABLE my_table RENAME COLUMN old_id TO new_id;
```

✔️5)
```
DROP TRIGGER trg ON my_table;
DROP FUNCTION on_insert_or_update();
ALTER TABLE my_table DROP COLUMN new_id;
```

ADD PRIMARY KEY in PG 11

✔️3) `UPDATE my_table SET new_id = id WHERE` **`new_id = -1`**

✔️4)
```
ALTER TABLE my_table RENAME COLUMN id TO old_id;
ALTER TABLE my_table RENAME COLUMN new_id TO id;
ALTER TABLE my_table RENAME COLUMN old_id TO new_id;
```

✔️5)
```
DROP TRIGGER trg ON my_table;
DROP FUNCTION on_insert_or_update();
ALTER TABLE my_table DROP COLUMN new_id;
```

✔️6) создаем PRIMARY KEY

Вопросы?

Обновление большой таблицы

❌
```
UPDATE my_table set new_column = 42
```

Обновление большой таблицы

❌
```
UPDATE my_table set new_column = 42
```

Проблемы:

- блокировка параллельных запросов на запись
- раздувание таблицы (bloat)

Обновление большой таблицы: блокировки

Multi
Version
Concurrency
Control

ctid	xmin	xmax	...	data

Обновление большой таблицы: блокировки

tx 435: insert

ctid	xmin	xmax	...	data
(0,1)	435	0	...	...

(0,1)

Обновление большой таблицы: блокировки

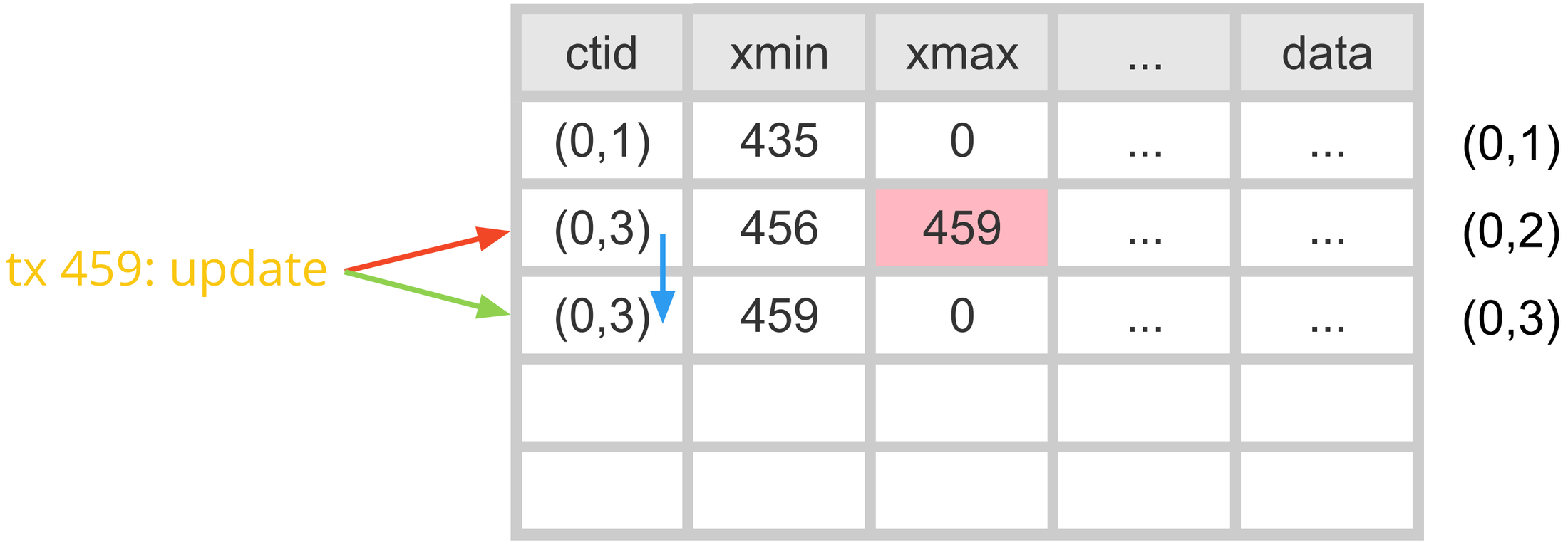
tx 456: insert

ctid	xmin	xmax	...	data
(0,1)	435	0	...	...
(0,2)	456	0	...	...

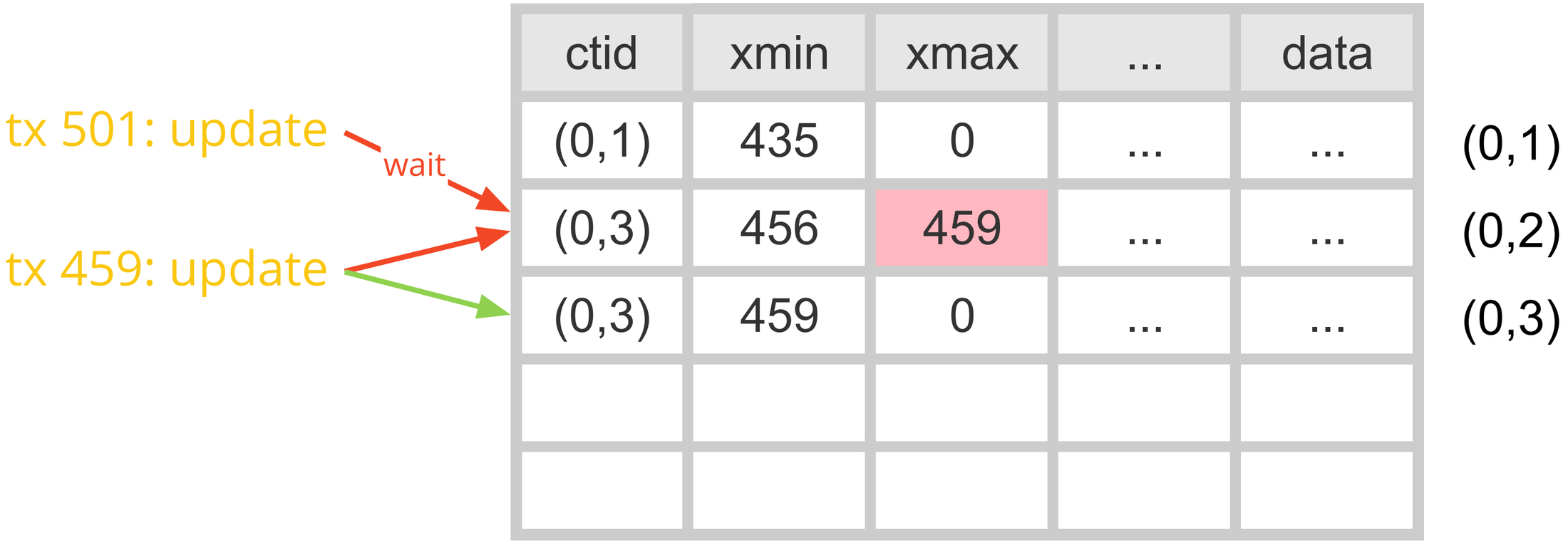
(0,1)

(0,2)

Обновление большой таблицы: блокировки



Обновление большой таблицы: блокировки



Обновление большой таблицы: блокировки

ctid	xmin	xmax	...	data

Обновление большой таблицы: блокировки

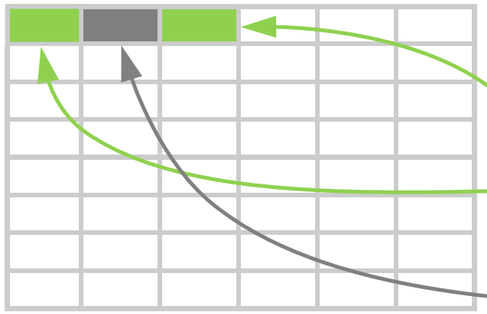
ctid	xmin	xmax	...	data

Обновление большой таблицы: bloat

ctid	xmin	xmax	...	data

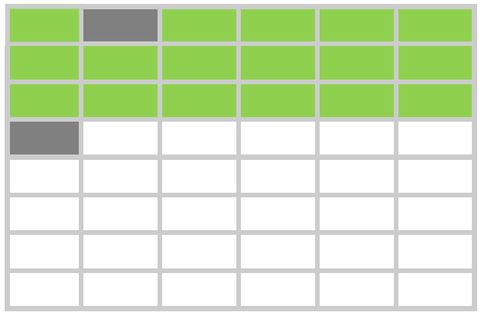
Обновление большой таблицы: bloat

page 8 kb

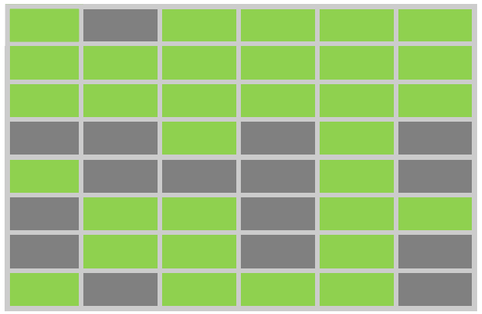


ctid	xmin	xmax	...	data
(0,1)	435	0	...	...
(0,3)	456	459	...	...
(0,3)	459	0	...	...

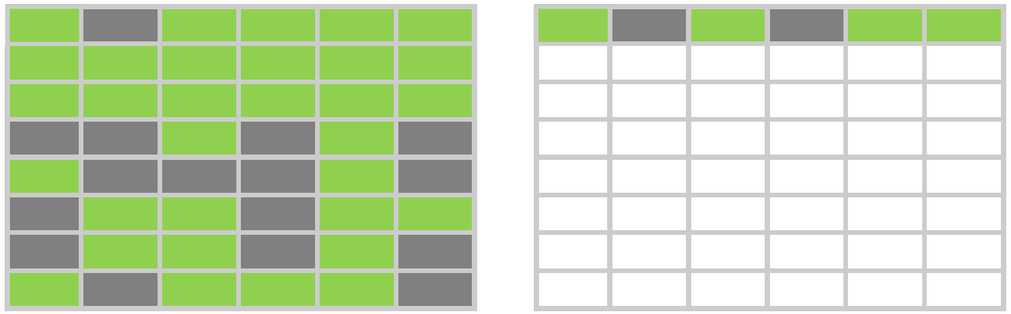
Обновление большой таблицы: bloat



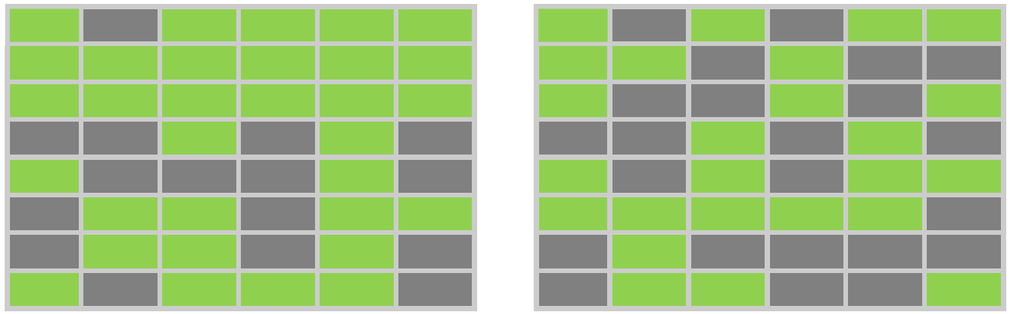
Обновление большой таблицы: bloat



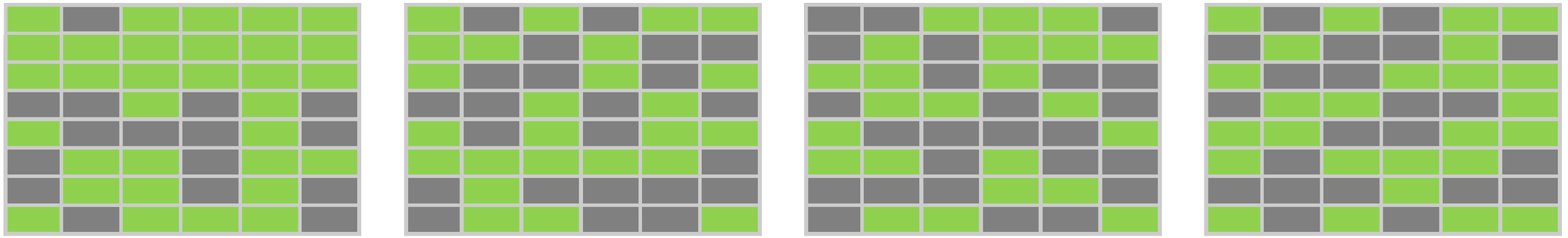
Обновление большой таблицы: bloat



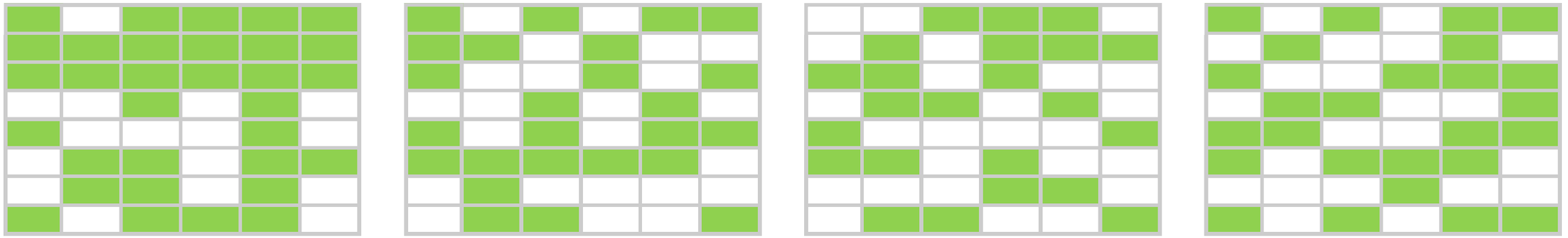
Обновление большой таблицы: bloat



Обновление большой таблицы: bloat



Обновление большой таблицы: bloat



Обновление большой таблицы

Сценарий 1 - замена таблицы

- SELECT >> INSERT + UPDATE + DELETE

Обновление большой таблицы

Сценарий 1 - замена таблицы

- SELECT >> INSERT + UPDATE + DELETE
- Часть операций на запись можно потерять или выполнить с большой задержкой

Обновление большой таблицы

Сценарий 1 - замена таблицы

- SELECT >> INSERT + UPDATE + DELETE
- Часть операций на запись можно потерять или выполнить с большой задержкой
- Есть свободное место (~100% - bloat + новые данные)

Обновление большой таблицы

```
BEGIN;
LOCK TABLE my_table IN SHARE MODE;
```

Обновление большой таблицы

```
BEGIN;
LOCK TABLE my_table IN SHARE MODE;

CREATE TABLE new_my_table (id int, ...);
```

Обновление большой таблицы

```
BEGIN;
LOCK TABLE my_table IN SHARE MODE;

CREATE TABLE new_my_table (id int, ...);

INSERT INTO new_my_table SELECT ... FROM my_table; -- update logic
```

Обновление большой таблицы

```
BEGIN;
LOCK TABLE my_table IN SHARE MODE;

CREATE TABLE new_my_table (id int, ...);

INSERT INTO new_my_table SELECT ... FROM my_table;
-- more constraints, indices, triggers?
```

Обновление большой таблицы

```
BEGIN;
LOCK TABLE my_table IN SHARE MODE;

CREATE TABLE new_my_table (id int, ...);

INSERT INTO new_my_table SELECT ... FROM my_table;
-- more constraints, indices, triggers?

DROP TABLE my_table;
ALTER TABLE new_my_table RENAME TO my_table;
```

Обновление большой таблицы

```
BEGIN;
LOCK TABLE my_table IN SHARE MODE;

CREATE TABLE new_my_table (id int, ...);

INSERT INTO new_my_table SELECT ... FROM my_table;
-- more constraints, indices, triggers?

DROP TABLE my_table;
ALTER TABLE new_my_table RENAME TO my_table;

COMMIT;
END;
```

Обновление большой таблицы

```
BEGIN;
LOCK TABLE my_table IN SHARE MODE;

CREATE TABLE new_my_table (id int, ...);

INSERT INTO new_my_table SELECT ... FROM my_table;
-- more constraints, indices, triggers?

DROP TABLE my_table;
ALTER TABLE new_my_table RENAME TO my_table;

COMMIT;
END;
```

PG 9.2

Обновление большой таблицы

```
BEGIN;
LOCK TABLE my_table IN SHARE MODE;

CREATE TABLE new_my_table (id int, ...);

INSERT INTO new_my_table SELECT ... FROM my_table;
-- more constraints, indices, triggers?

DROP TABLE my_table;
ALTER TABLE new_my_table RENAME TO my_table;

COMMIT;
END;
```

PG 9.2

Обновление большой таблицы

Плюсы:

- самый быстрый способ

Обновление большой таблицы

Плюсы:

- самый быстрый способ
- нет раздувания таблицы

Обновление большой таблицы

Плюсы:

- самый быстрый способ
- нет раздувания таблицы

Минусы:

- необходимо наличие свободного места

Обновление большой таблицы

Плюсы:

- самый быстрый способ
- нет раздувания таблицы

Минусы:

- необходимо наличие свободного места
- блокировка операций на запись

Обновление большой таблицы

Плюсы:

- самый быстрый способ
- нет раздувания таблицы

Минусы:

- необходимо наличие свободного места
- блокировка операций на запись
- смена OID таблицы (иногда критично)

Обновление большой таблицы

Сценарий 2 - использование временной таблицы

- SELECT >> INSERT + UPDATE + DELETE
- Часть операций на запись можно потерять или выполнить с большой задержкой
- Есть свободное место
- Важно сохранить исходную таблицу

Обновление большой таблицы

```
BEGIN;
LOCK TABLE my_table IN SHARE MODE;

CREATE TABLE new_my_table (id int, ...);

INSERT INTO new_my_table SELECT ... FROM my_table;
-- more constraints, indices, triggers?

DROP TABLE my_table;
ALTER TABLE new_my_table RENAME TO my_table;

COMMIT;
END;
```

Обновление большой таблицы

```
BEGIN;
LOCK TABLE my_table IN SHARE MODE;

SET LOCAL temp_buffers = '????MB';

CREATE TEMP TABLE new_my_table AS SELECT ... FROM my_table;

DROP TABLE my_table;
ALTER TABLE new_my_table RENAME TO my_table;

COMMIT;
END;
```

Обновление большой таблицы

```
BEGIN;
LOCK TABLE my_table IN SHARE MODE;

SET LOCAL temp_buffers = '????MB';

CREATE TEMP TABLE new_my_table AS SELECT ... FROM my_table;

DROP TABLE my_table;
ALTER TABLE new_my_table RENAME TO my_table;

COMMIT;
END;
```

Обновление большой таблицы

```
BEGIN;
LOCK TABLE my_table IN SHARE MODE;

SET LOCAL temp_buffers = '????MB';

CREATE TEMP TABLE new_my_table AS SELECT ... FROM my_table;

--drop constraints, indexes, triggers
TRUNCATE my_table;
INSERT INTO my_table SELECT * FROM new_my_table;
--create constraints, indexes, triggers

COMMIT;
END;
```

Обновление большой таблицы

Плюсы:

- быстрый способ

Обновление большой таблицы

Плюсы:

- быстрый способ
- остается исходная таблица

Обновление большой таблицы

Плюсы:

- быстрый способ
- остается исходная таблица
- нет раздувания таблицы

Обновление большой таблицы

Плюсы:

- быстрый способ
- остается исходная таблица
- нет раздувания таблицы

Минусы:

- необходимо наличие свободного места

Обновление большой таблицы

Плюсы:

- быстрый способ
- остается исходная таблица
- нет раздувания таблицы

Минусы:

- необходимо наличие свободного места
- блокировка операций на запись (на все время выполнения)

Обновление большой таблицы

Плюсы:

- быстрый способ
- остается исходная таблица
- нет раздувания таблицы

Минусы:

- необходимо наличие свободного места
- блокировка операций на запись (на все время выполнения)
- блокировка операций на чтение (~ 1/2 времени выполнения)

Обновление большой таблицы

Сценарий 3 - обновление порциями

- SELECT > INSERT + UPDATE + DELETE
- Необходимо поддержать параллельные операции на запись

Обновление большой таблицы

❌
```
UPDATE my_table set new_column = 42
```

Проблемы:

- блокировка параллельных запросов на запись
- раздувание таблицы (bloat)

Обновление большой таблицы

```
UPDATE my_table SET new_column = 42
  WHERE id BETWEEN 1 AND 10000 AND new_column IS NULL;
COMMIT;
```

Обновление большой таблицы

```
UPDATE my_table SET new_column = 42
  WHERE id BETWEEN 1 AND 10000 AND new_column IS NULL;
COMMIT;


UPDATE my_table SET new_column = 42
  WHERE id BETWEEN 10001 AND 20000 AND new_column IS NULL;
COMMIT;

...
```

Обновление большой таблицы

```
UPDATE my_table SET new_column = 42
  WHERE id BETWEEN 1 AND 10000 AND new_column IS NULL;
COMMIT;
VACUUM my_table;

UPDATE my_table SET new_column = 42
  WHERE id BETWEEN 10001 AND 20000 AND new_column IS NULL;
COMMIT;
VACUUM my_table;
...
```

Обновление большой таблицы

```
FOR i IN 1..batch_count LOOP
    id_from := ...
    id_to := ...
    UPDATE my_table SET new_column = 42
      WHERE id BETWEEN id_from AND id_to AND new_column IS NULL;
    COMMIT;
    VACUUM my_table;
END LOOP;
```

Обновление большой таблицы

```
FOR i IN 1..batch_count LOOP
    id_from := ...
    id_to := ...
    UPDATE my_table SET new_column = 42
      WHERE id BETWEEN id_from AND id_to AND new_column IS NULL;
    COMMIT;
    pid := pg_background_launch('vacuum my_table');
    PERFORM * FROM pg_background_result(pid) as p (r text);
END LOOP;
```

Обновление большой таблицы

Плюсы:

- минимальное время блокировок

Обновление большой таблицы

Плюсы:

- минимальное время блокировок
- минимальное раздувание таблицы

Обновление большой таблицы

Плюсы:

- минимальное время блокировок
- минимальное раздувание таблицы

Минусы:

- медленный способ

Обновление большой таблицы

Плюсы:

- минимальное время блокировок
- минимальное раздувание таблицы

Минусы:

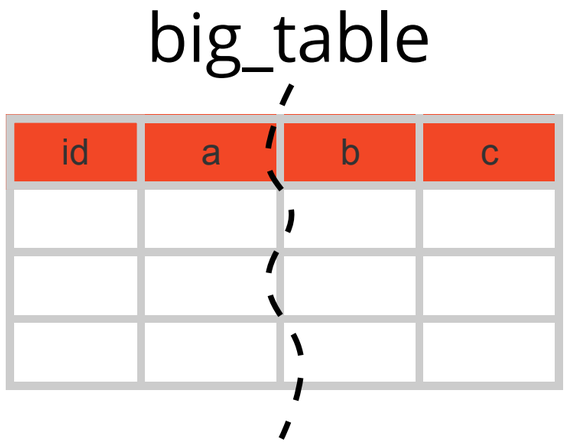
- медленный способ
- есть риск получить взаимоблокировку

Разбиение таблицы на две

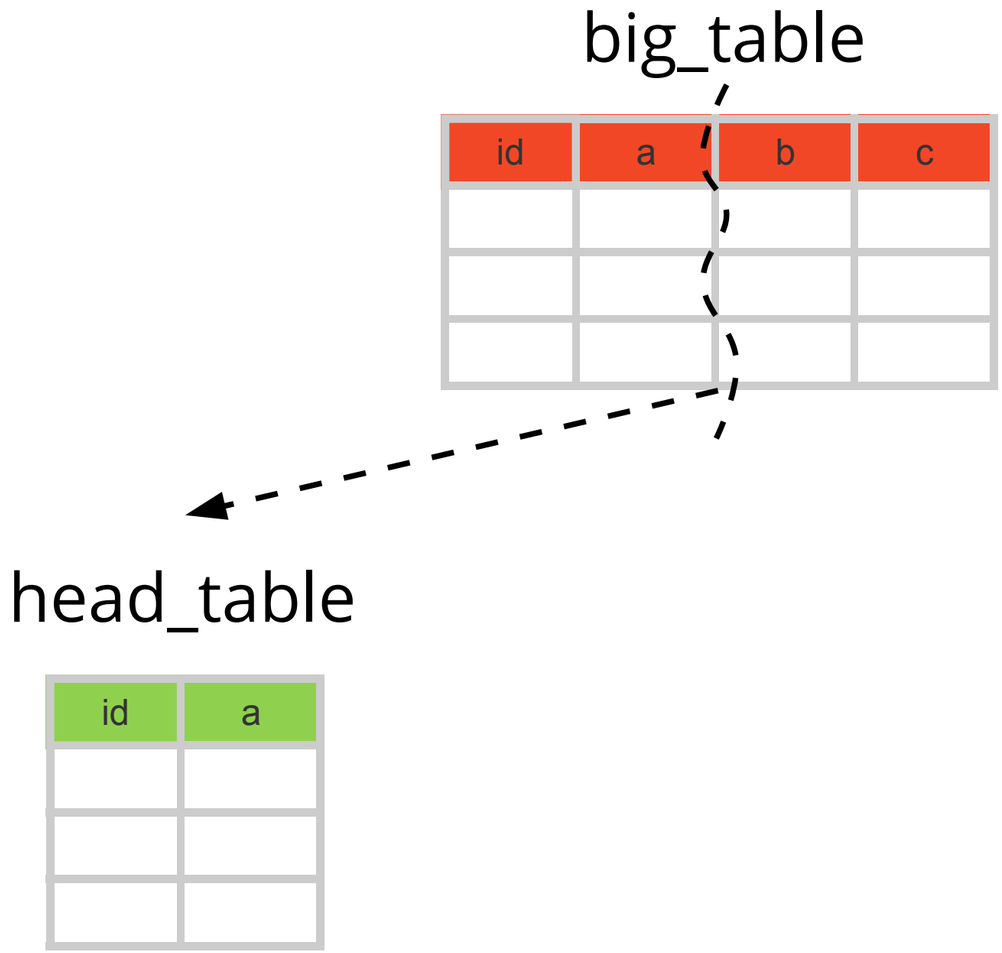
big_table

id	a	b	c

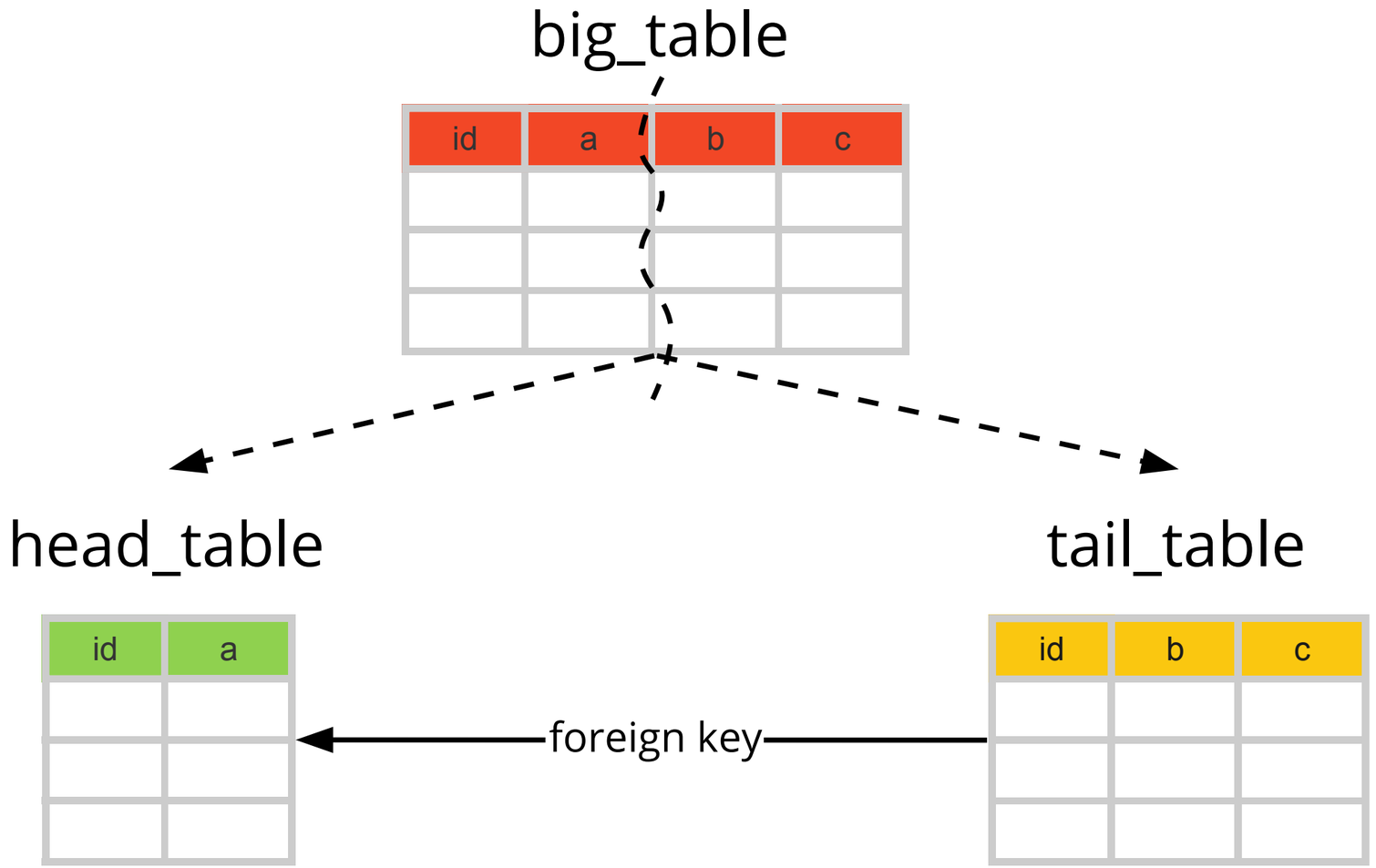
Разбиение таблицы на две



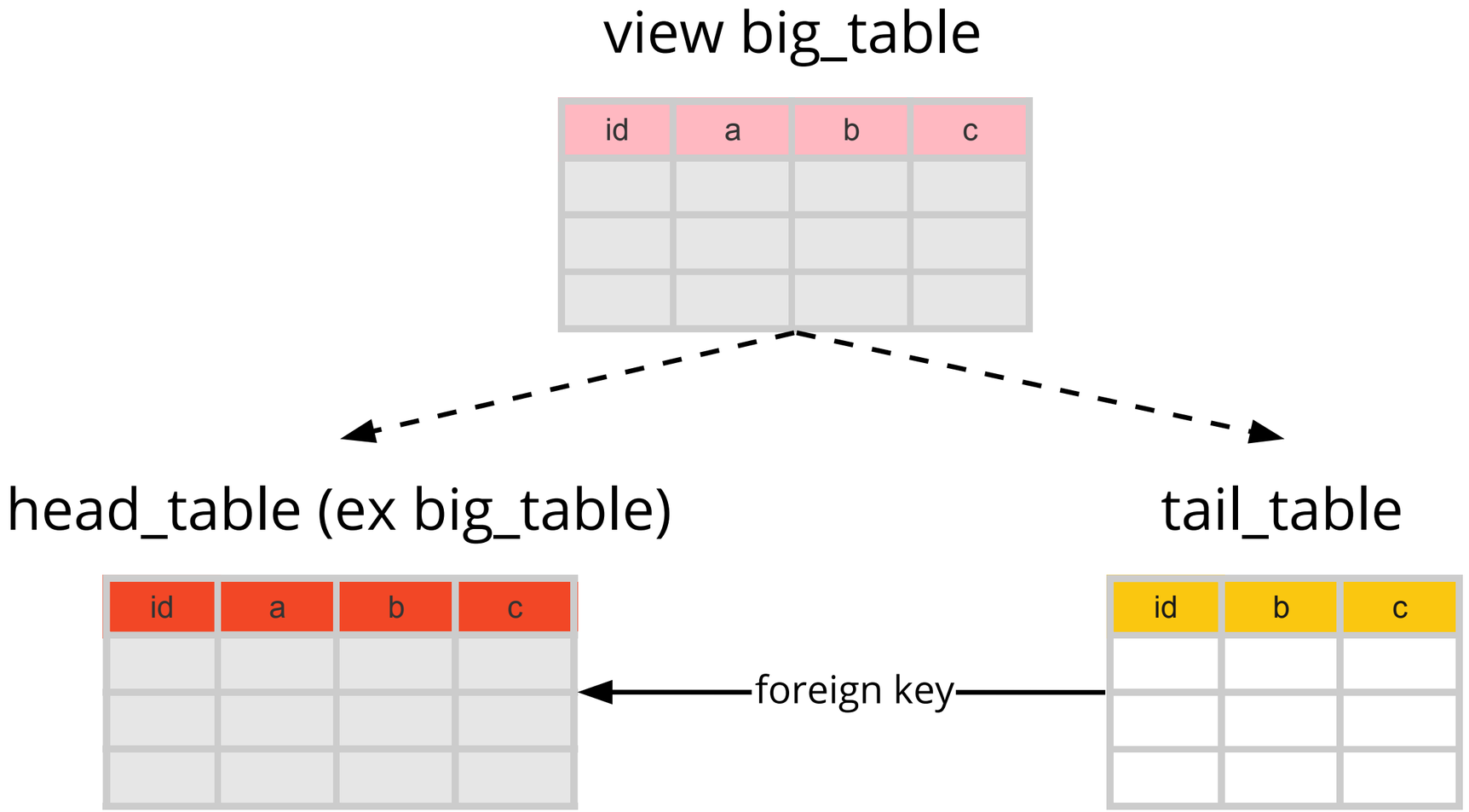
Разбиение таблицы на две



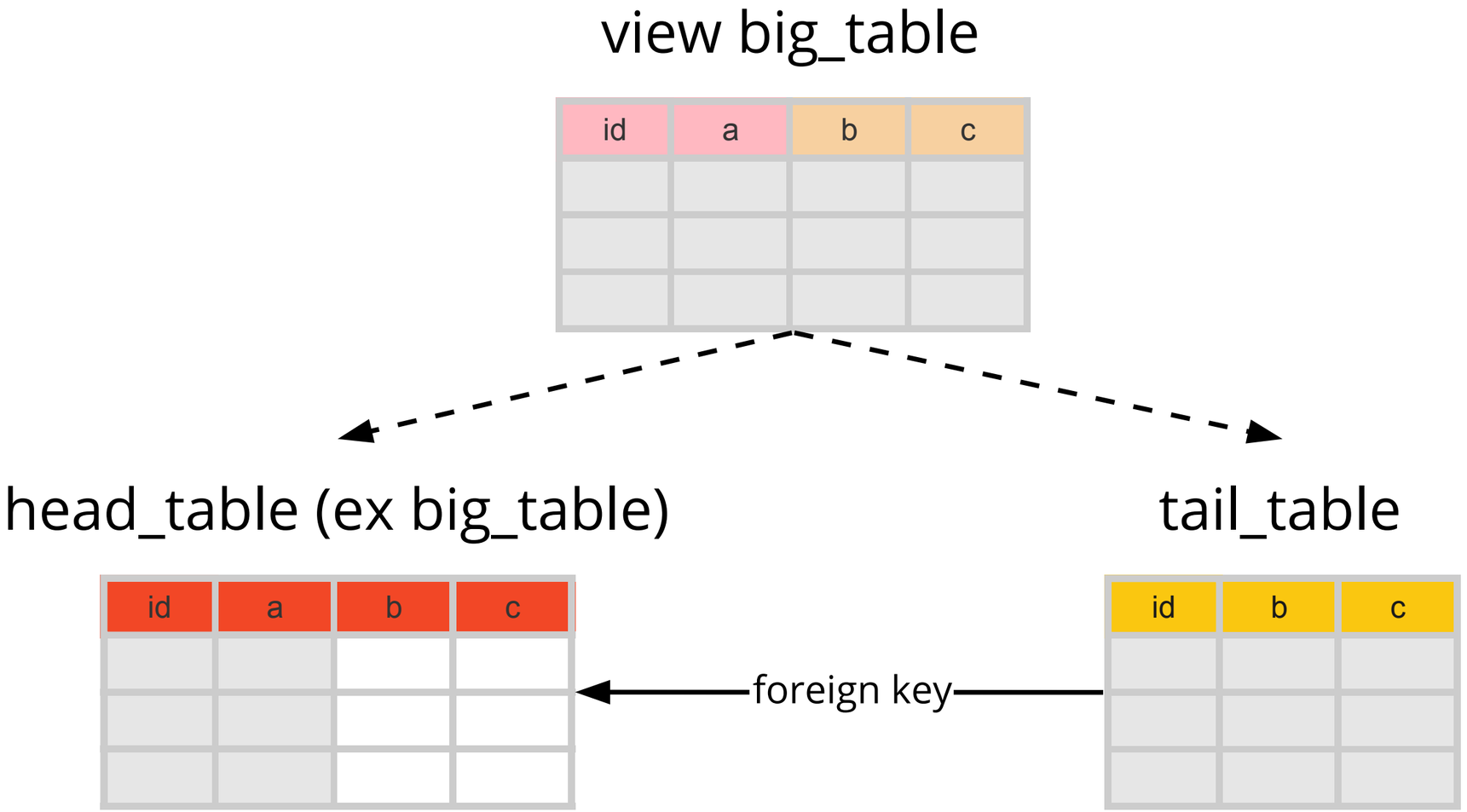
Разбиение таблицы на две



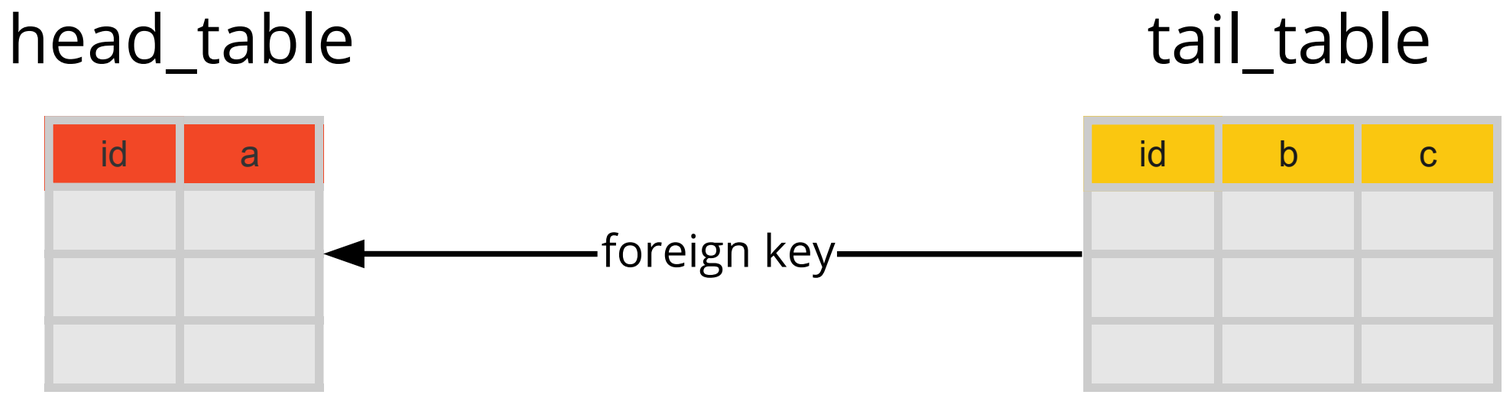
Разбиение таблицы на две



Разбиение таблицы на две



Разбиение таблицы на две



Разбиение таблицы на две

```
CREATE TABLE tail_table
(
    id int primary key,
    b  int,
    c  int,
    CONSTRAINT fk_tail_table FOREIGN KEY (id) REFERENCES big_table(id)
    ON DELETE CASCADE
)
```

Разбиение таблицы на две

```
CREATE TABLE tail_table
(
    id int primary key,
    b  int,
    c  int,
    CONSTRAINT fk_tail_table FOREIGN KEY (id) REFERENCES big_table(id)
    ON DELETE CASCADE
)
```

1) Tx 1:

```
ALTER TABLE big_table RENAME TO head_table;
CREATE VIEW big_table as SELECT * FROM head_table;
```

Разбиение таблицы на две

```
CREATE OR REPLACE FUNCTION big_table_trigger_func() RETURNS TRIGGER AS
$$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO head_table VALUES (NEW.*);
        INSERT INTO tail_table values (NEW.id, NEW.b, NEW.c);
        RETURN NEW;
    ELSEIF TG_OP = 'UPDATE' THEN
        ...
    ELSEIF TG_OP = 'DELETE' THEN
        ...
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Разбиение таблицы на две

```
CREATE OR REPLACE FUNCTION big_table_trigger_func() RETURNS TRIGGER AS $
BEGIN
    IF TG_OP = 'INSERT' THEN
        ...
    ELSEIF TG_OP = 'UPDATE' THEN
        UPDATE head_table set a=new.a, b=new.b, c=new.c
          WHERE id = new.id;
        INSERT INTO tail_table values (NEW.id, NEW.b, NEW.c)
          ON CONFLICT (id) DO UPDATE set b=new.b, c=new.c;
        RETURN NEW;
    ELSEIF TG_OP = 'DELETE' THEN
        ...
    END IF;
END; $$ LANGUAGE plpgsql;
```

Разбиение таблицы на две

```
CREATE OR REPLACE FUNCTION big_table_trigger_func() RETURNS TRIGGER AS
$$
BEGIN
    IF TG_OP = 'INSERT' THEN
        ...
    ELSEIF TG_OP = 'UPDATE' THEN
        ...
    ELSEIF TG_OP = 'DELETE' THEN
        DELETE FROM head_table WHERE id = OLD.id;
        RETURN OLD;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Разбиение таблицы на две

```
CREATE TRIGGER big_table_trigger
INSTEAD OF INSERT OR UPDATE OR DELETE ON big_table
FOR EACH ROW EXECUTE PROCEDURE big_table_trigger_func();

Конец Tx 1
```

PG 9.1

Разбиение таблицы на две

2) Заполняем данные tail_table:

```
INSERT INTO tail_table(id, b, c) SELECT id, b, c FROM head_table
  WHERE id BETWEEN 1 AND 10000 ON CONFLICT DO NOTHING;
COMMIT;

INSERT INTO tail_table(id, b, c) SELECT id, b, c FROM head_table
  WHERE id BETWEEN 10001 AND 20000 ON CONFLICT DO NOTHING;
COMMIT;
...
```

Разбиение таблицы на две

```
BEGIN
FOR i IN 1..batch_count LOOP
    id_from := ...
    id_to := ...
    INSERT INTO tail_table(id, b, c) SELECT id, b, c FROM head_table
      WHERE id BETWEEN id_from AND id_to ON CONFLICT DO NOTHING;
    COMMIT;
END LOOP;
END;
```

PG 11

Разбиение таблицы на две

Не делайте так!

```
BEGIN
FOR r in SELECT * FROM head_table LOOP --holdable cursor
    INSERT INTO tail_table(id, b, c) VALUES (r.id, r.b, r.c)
      ON CONFLICT DO NOTHING;
    i := i + 1;
    IF i % batch_size = 0 THEN
        COMMIT;
    END IF;
END LOOP;
END;
```

Разбиение таблицы на две

```
3) CREATE OR REPLACE VIEW big_table AS
   SELECT h.id, h.a, t.b, t.c
     FROM head_table h
   LEFT JOIN tail_table t ON h.id = t.id;
```

Разбиение таблицы на две

3)
```
CREATE OR REPLACE VIEW big_table AS
  SELECT h.id, h.a, t.b, t.c
    FROM head_table h
  LEFT JOIN tail_table t ON h.id = t.id;
```

4) Переводим приложение на работу с **head_table** и **tail_table**

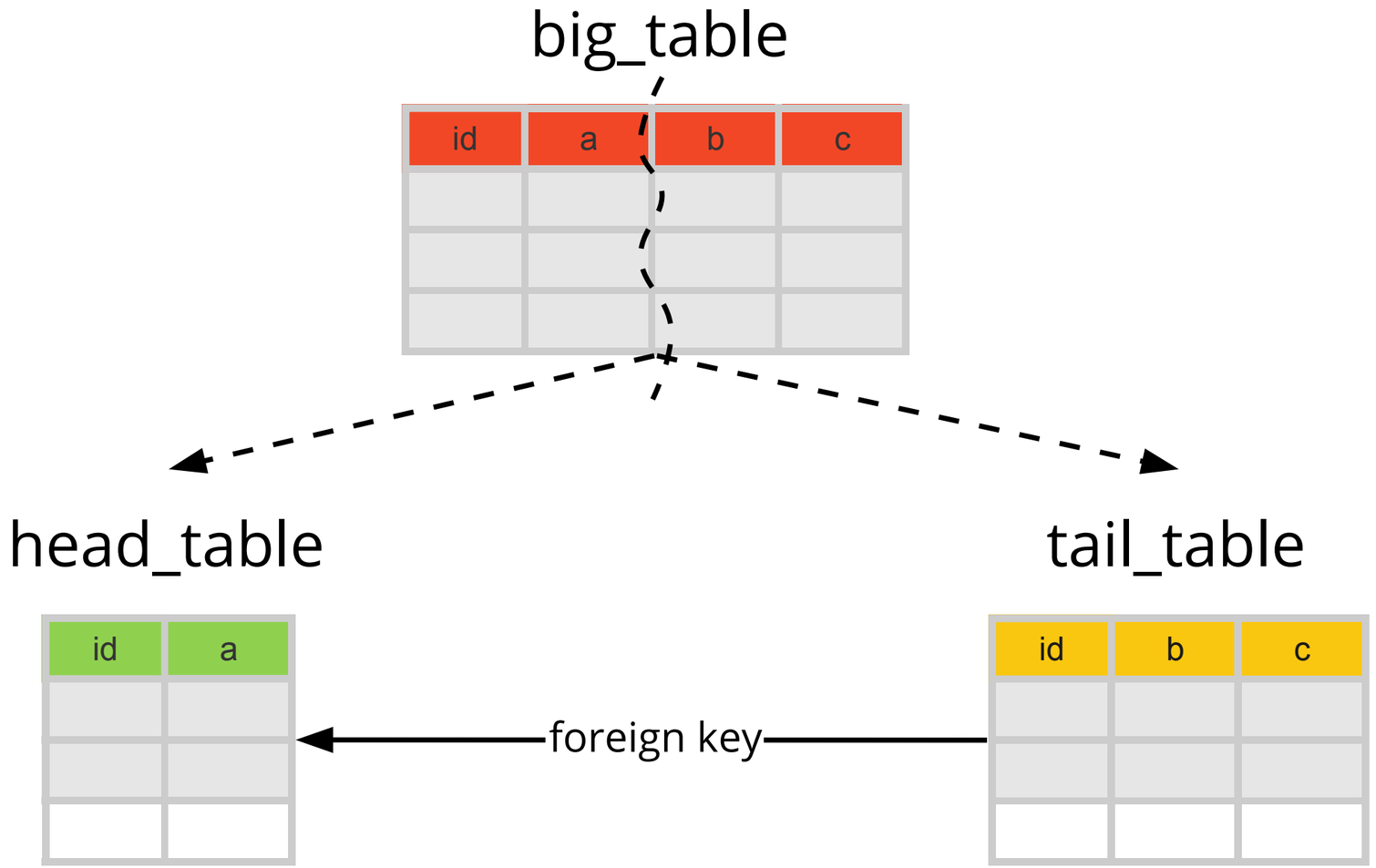
Разбиение таблицы на две

3)
```
CREATE OR REPLACE VIEW big_table AS
SELECT h.id, h.a, t.b, t.c
  FROM head_table h
LEFT JOIN tail_table t ON h.id = t.id;
```

4) Переводим приложение на работу с **head_table** и **tail_table**

5)
```
DROP VIEW big_table;
DROP FUNCTION big_table_trigger_func();
ALTER TABLE head_table DROP COLUMN b, DROP COLUMN c;
```

Разбиение таблицы на две



Спасибо за внимание!

Николай Аверин

 averin@miro.com

 @nikolai_averin

 @nikolai-averin