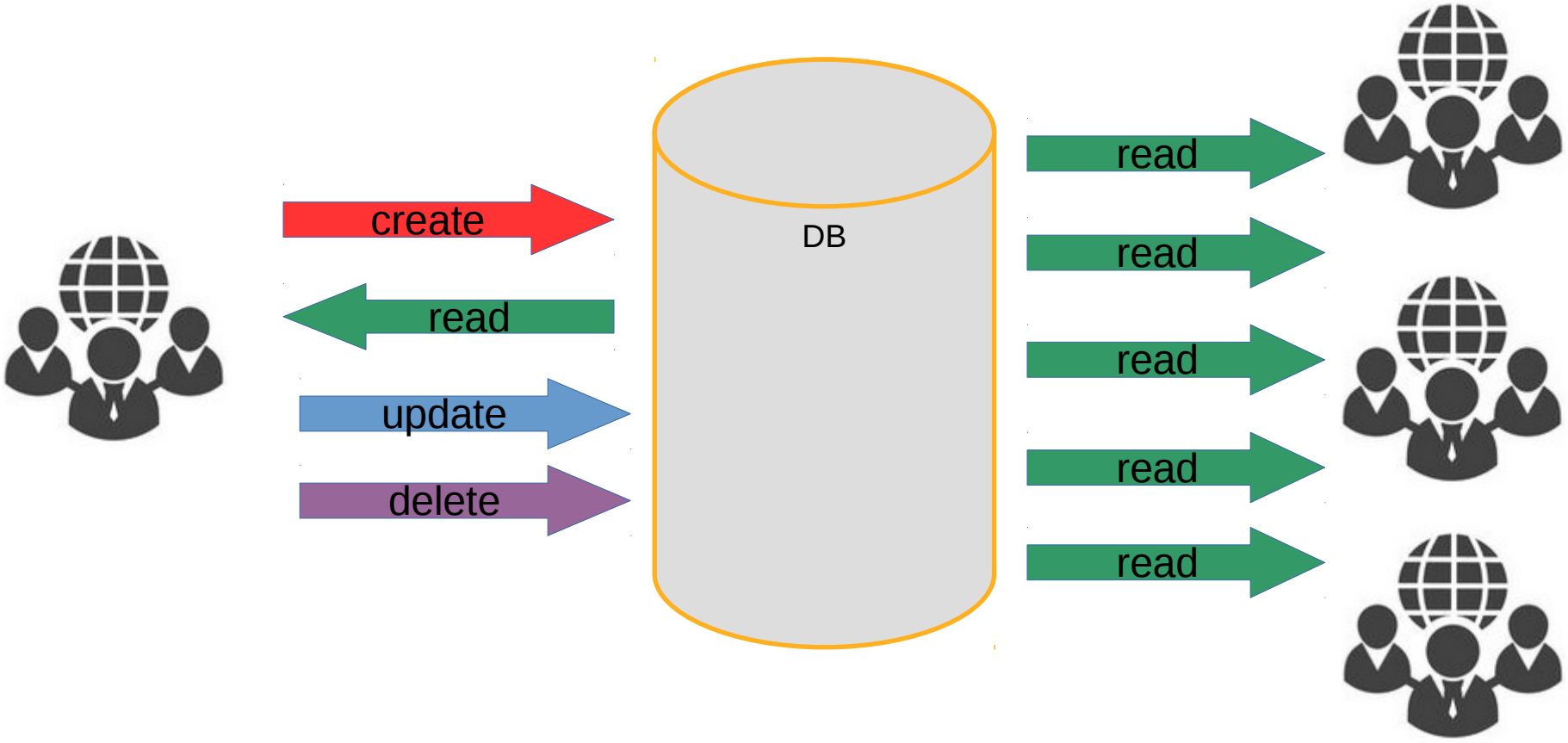
Building data streams

Константин Евтеев
kevteev@avito.ru

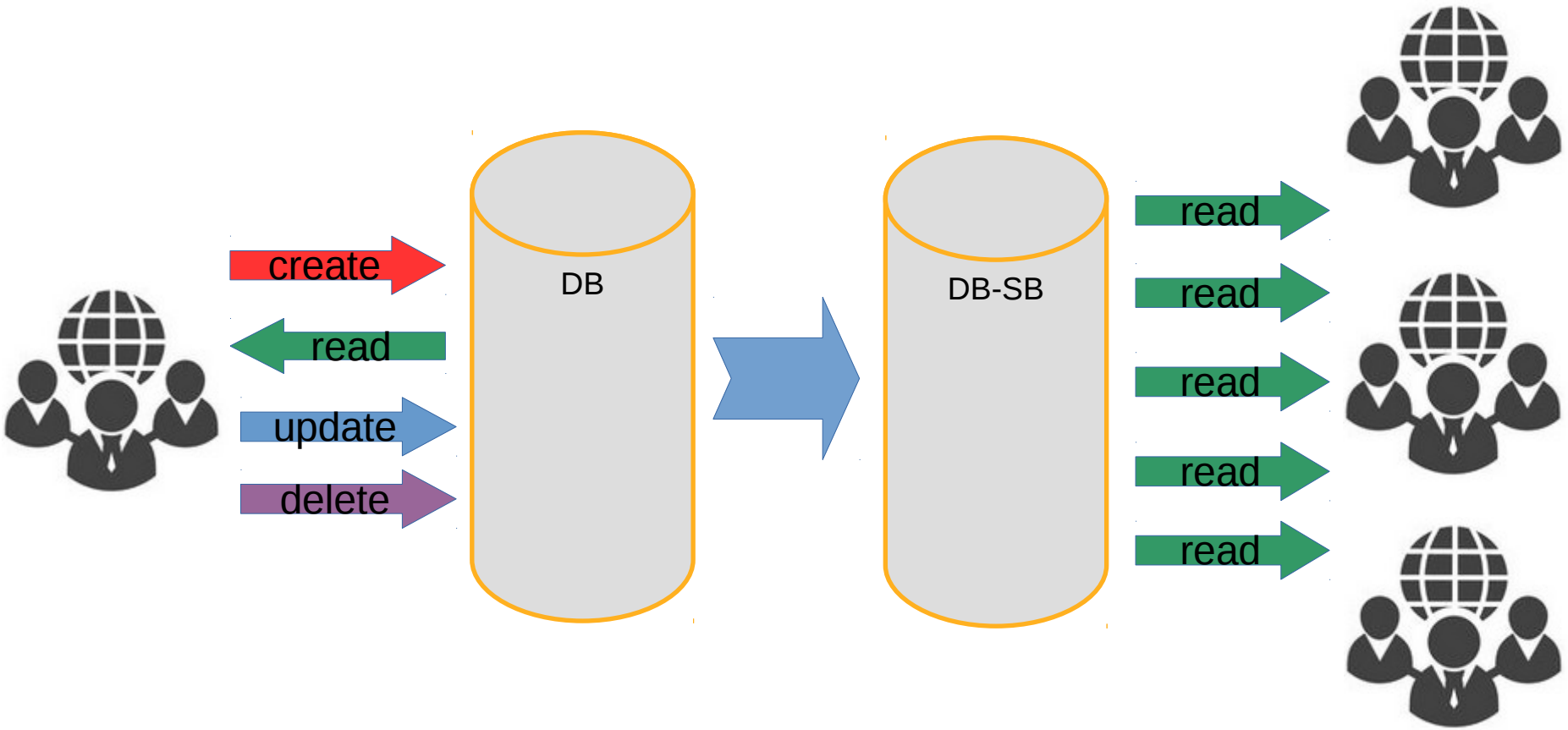
CRUD



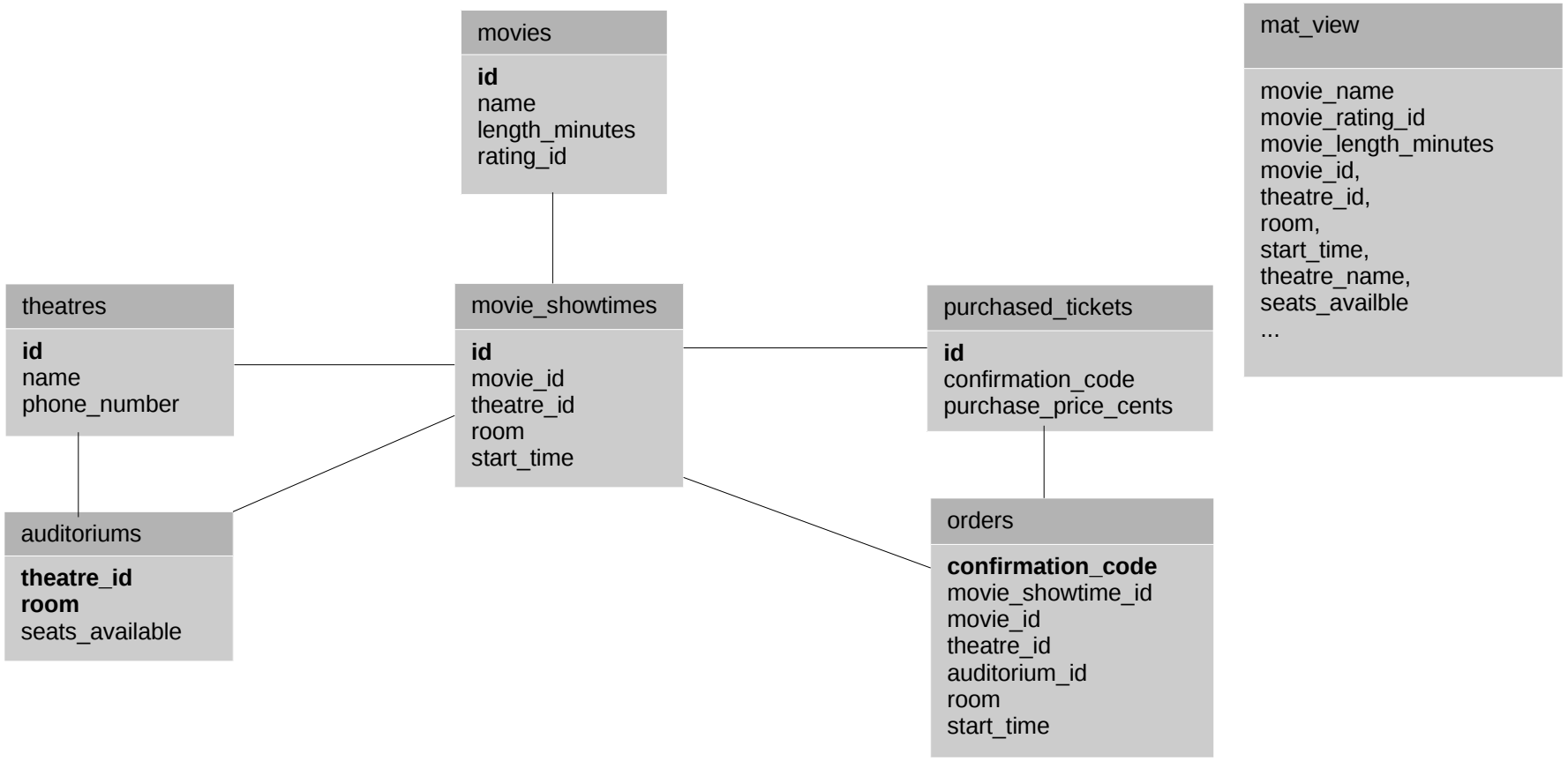
Optimization steps

1. Оптимизируем процедуры выборки
2. Вертикальное и функциональное масштабирование
3. Вводим в бой стендбаи бинарной репликации
4. Денормализуем данные для чтения
5. Шардирование
6. Денормализованные данные выносим на отдельные машины

C(R)UD & R



Денормализуем данные для чтения

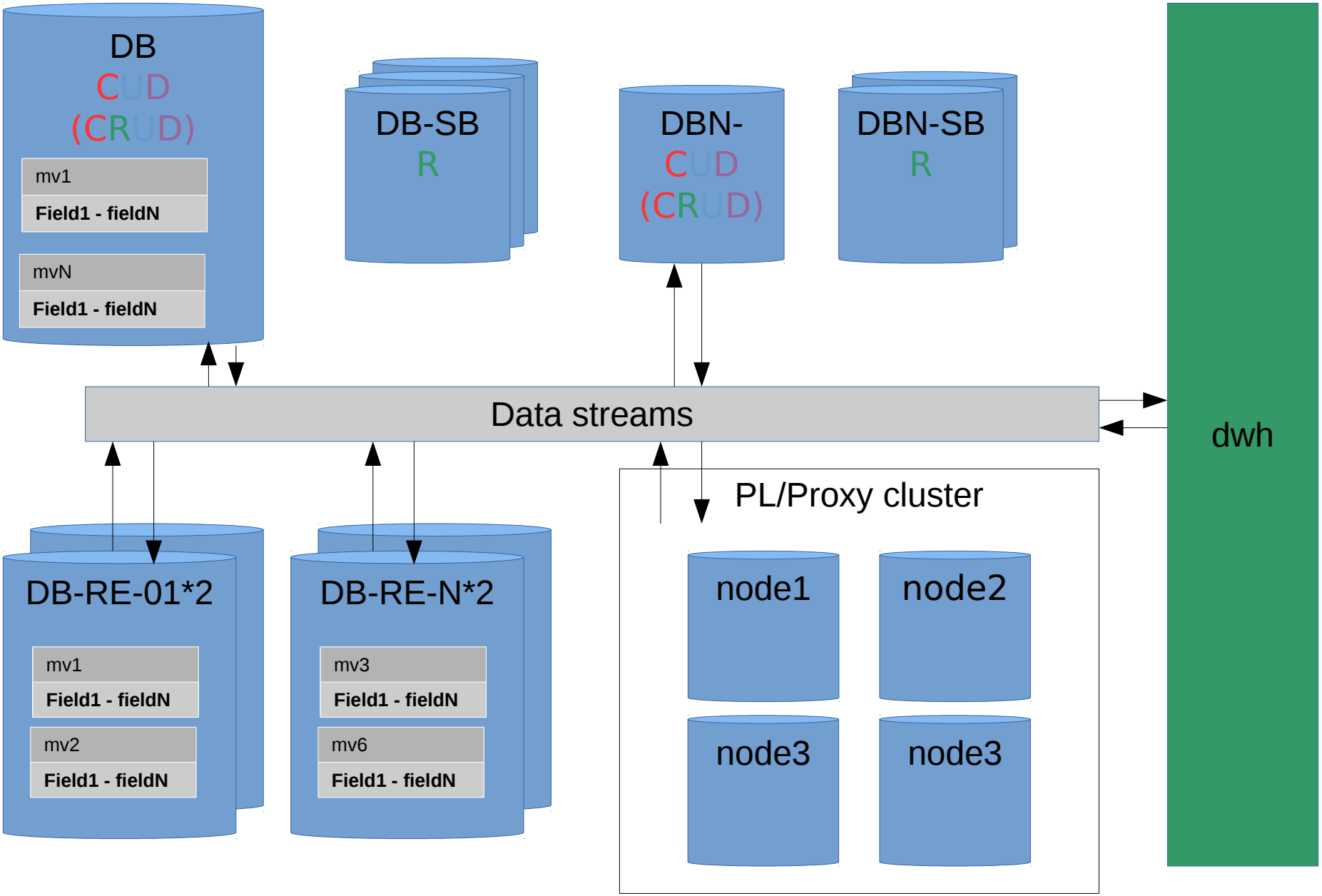


http://www.slideshare.net/pavlushko/sphinx-10460333
http://www.pgcon.org/2008/schedule/attachments/64_BSDCan2008-MaterializedViews-paper.pdf
http://www.pgcon.org/2008/schedule/attachments/63_BSDCan2008-MaterializedViews.pdf

It looks like

1. Command and Query Responsibility Segregation (CQRS) или Command-query separation (CQS)
2. Event Sourcing
3. Eventual consistency
4. **Lambda Architecture**
5. **Kappa Architecture**

http://lambda-architecture.net/
http://milinda.pathirage.org/kappa-architecture.com/


DB
CUD
(CRUD)
mv1
Field1 - fieldN
mvN
Field1 - fieldN
DB-SB
R
DBN-
CUD
(CRUD)
DBN-SB
R
dwh
Data streams
PL/Proxy cluster
node1
node2
node3
node3
DB-RE-01*2
mv1
Field1 - fieldN
mv2
Field1 - fieldN
DB-RE-N*2
mv3
Field1 - fieldN
mv6
Field1 - fieldN

Выгрузка по времени

```
select
    hstore(x)
from
    table x
where
    txtime > max_time_batch(n-1) - longest_trans_duration()
```

\+ работает

− есть вопросы

Выгрузка данных batch-ами


```
#pg_current_xlog_insert_location()
#Get current transaction log insert location
master_pos=$(psql_ping_db -c "select force_wal(pg_current_xlog_insert_location())")

#pg_xlog_location_diff(location text, location text)   numeric
#Calculate the difference between two transaction log locations
psql -c "select pg_xlog_location_diff(pg_last_xlog_replay_location(), '${master_pos}')")
```

https://github.com/eshkinkot/pgday2016/tree/master/ping_DB

Ticker & pgq

2008 год! https://www.pgcon.org/2008/schedule/attachments/55_pgq.pdf

https://www.postgresql.org/docs/9.4/static/functions-info.html#FUNCTIONS-TXID-SNAPSHOT

https://github.com/markokr/skytools/tree/skytools_2_1_stable/python

https://github.com/markokr/skytools/blob/skytools_2_1_stable/sql/pgq/functions/pgq.ticker.sql

http://www.pgcon.org/2009/schedule/attachments/91_pgq.pdf

MVCC (https://www.postgresql.org/docs/8.3/static/release-8-3.html)

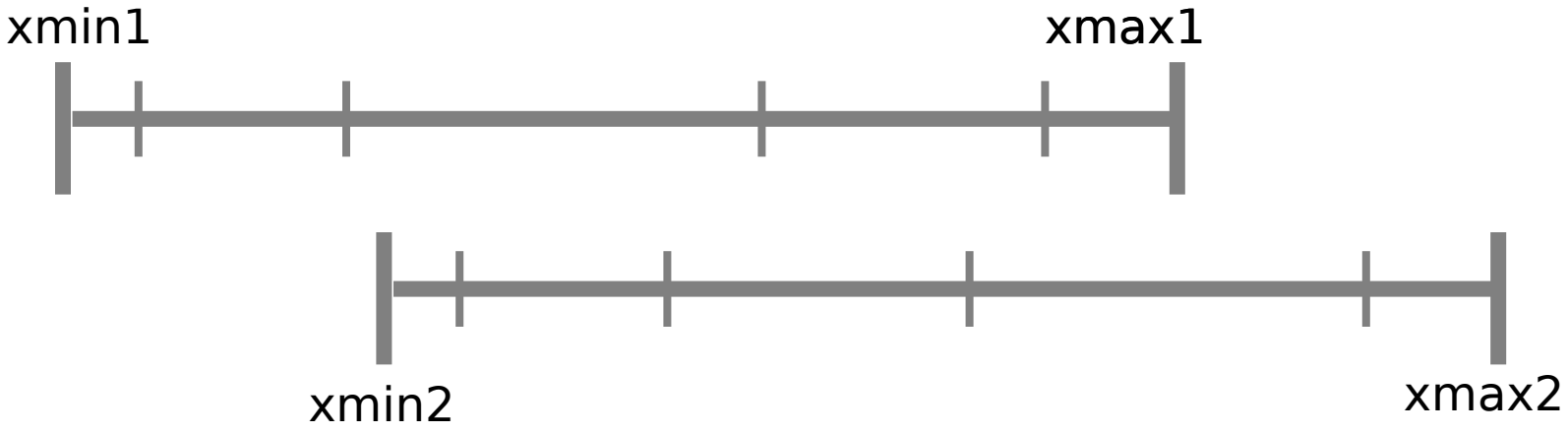
Add several txid_*() functions to query active transaction IDs (Jan)
This is useful for various replication solutions.

Table 9-56. Transaction IDs and Snapshots

Name	Return Type	Description
`txid_current()`	`bigint`	get current transaction ID
`txid_current_snapshot()`	`txid_snapshot`	get current snapshot
`txid_snapshot_xip(txid_snapshot)`	`setof bigint`	get in-progress transaction IDs in snapshot
`txid_snapshot_xmax(txid_snapshot)`	`bigint`	get xmax of snapshot
`txid_snapshot_xmin(txid_snapshot)`	`bigint`	get xmin of snapshot
`txid_visible_in_snapshot(bigint, txid_snapshot)`	`boolean`	is transaction ID visible in snapshot? (do not use with subtransaction ids)

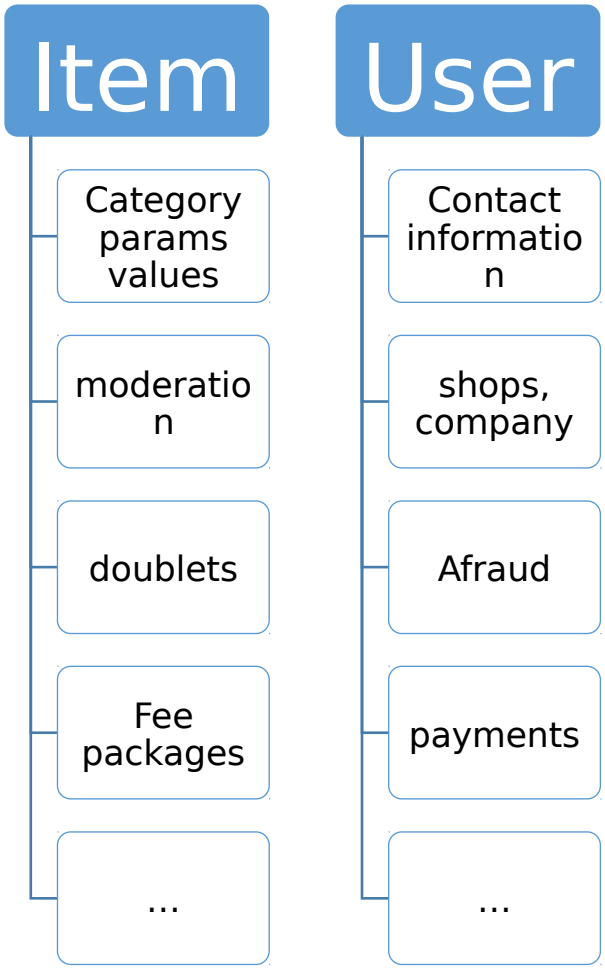
Name	Description
`xmin`	Earliest transaction ID (txid) that is still active. All earlier transactions will either be committed and visible, or rolled back and dead.
`xmax`	First as-yet-unassigned txid. All txids greater than or equal to this are not yet started as of the time of the snapshot, and thus invisible.
`xip_list`	Active txids at the time of the snapshot. The list includes only those active txids between `xmin` and `xmax`; there might be active txids higher than `xmax`. A txid that is `xmin <= txid < xmax` and not in this list was already completed at the time of the snapshot, and thus either visible or dead according to its commit status. The list does not include txids of subtransactions.

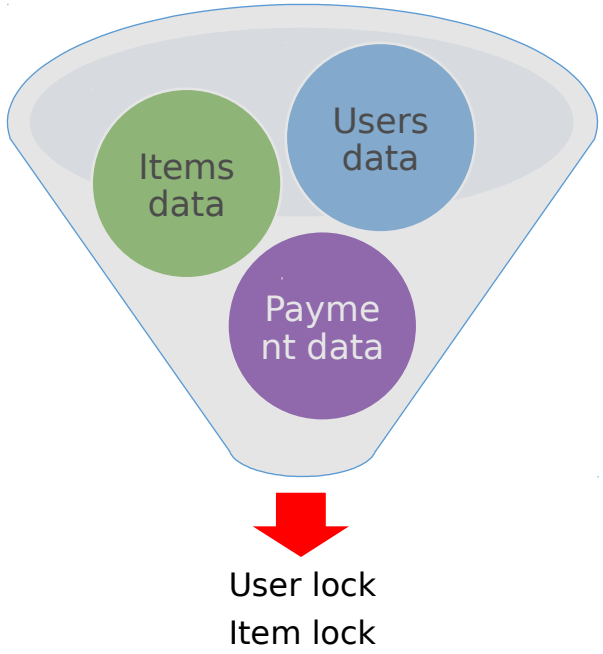
How to select txids that are between snapshots




```
SELECT * FROM queue q WHERE
/*вспомогательный скан-фильтр*/
q.ev_txid BETWEEN xmin1 AND xmax2
-- xmin1 = txid_snapshot_xmin(snap1),
-- xmax2 = txid_snapshot_xmax(snap2)
/*вспомогательный скан-фильтр*/
/*суть*/
AND NOT is_visible( q.ev_txid, snap1 ) -- txid_visible_in_snapshot
AND     is_visible( q.ev_txid, snap2 ) -- txid_visible_in_snapshot
/*суть*/
```

Все изменения данных под единой блокировкой объекта(row level)

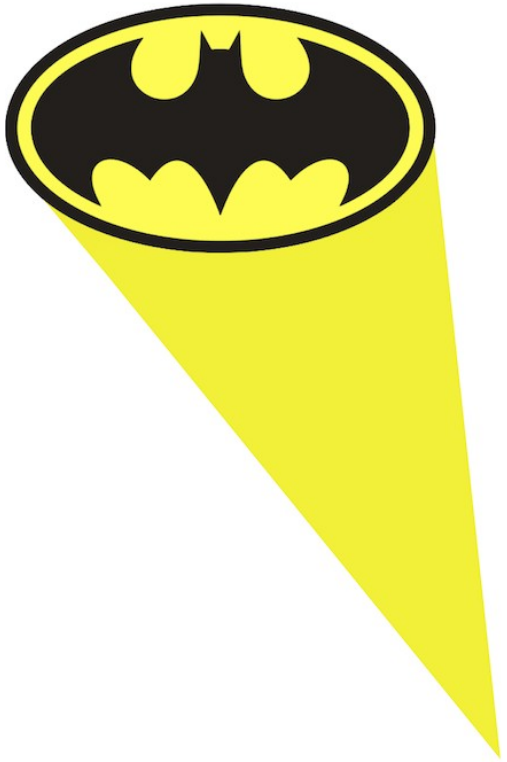




Под блокировкой понимаем цепочку блокировок:
select item_id from items for update;

Сессионные переменные “signal”

Name	Return Type	Description
current_setting(setting_name)	text	get current value of setting
set_config(setting_name, new_value, is_local)	text	set parameter and return new value



Достоинства:

1. Достаточно дешево по ресурсам
2. Не нужно разбирать цепочку вызовов процедур, и добавлять входные/выходные параметры

Сессионные переменные, групповые действия, подводные камни …

В 1 транзакции может меняться несколько объектов, а событие относится не ко всем.
Решение:Используем массив по unique key array ‘{}’

```
select current_setting('avito.var')::int[] into v_items;
  if not ( NEW.item_id = any (v_items) ) then
    if coalesce(array_length(v_items, 1), 0) = 100 then
      raise exception 'More then 100 items updating in one transaction ';
    end if;
    v_items := v_items || NEW.item_id::int;
    perform set_config('avito.var', v_items::text, true);
  end if;
```

Init remote mv or counter

https://www.depesz.com/2016/06/14/incrementing-counters-in-database/

1) Создать временную таблицу на стороне подписчика
2) Создать принимающую процедуру
   - временную для инициализации
   - реальную
3) Начинаем слать события
4) Запускаем инициализацию со стендбая(дождаться прихода события, с которого начали заполнять временную таблицу)
5) Переключаем на реальную процедуру приема
6) Пересчитываем под блокировкой через очередь

Init remote mv or counter

https://github.com/eshkinkot/pgday2016/tree/master/remote_cnt

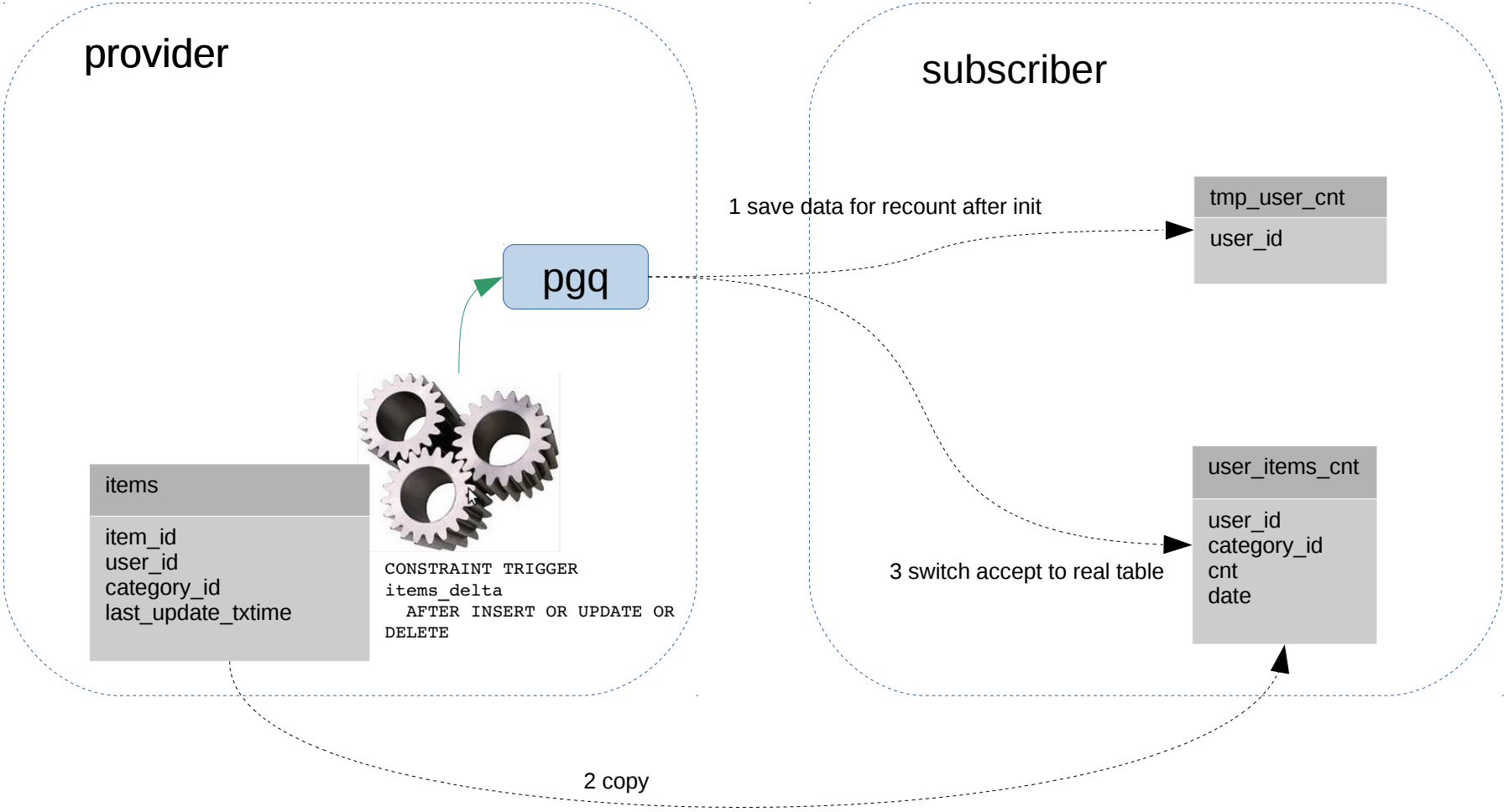


Table and trigger function on provider's side

```
---Table and trigger function on provider's side
create table items
(
    item_id int,
    user_id int,
    category_id,
    last_update_txtime timestamp without time zone
)
```

```
CREATE CONSTRAINT TRIGGER items_delta
  AFTER INSERT OR UPDATE
  ON items
  DEFERRABLE INITIALLY DEFERRED
  FOR EACH ROW
  EXECUTE PROCEDURE items_delta_trg();
```

```
--- deferred trigger
CREATE OR REPLACE FUNCTION items_delta_trg()
  RETURNS trigger AS
$BODY$
declare
    data hstore;
begin
  if TG_OP = 'INSERT' then
        Data := (select hstore(i) || hstore('old_category_id',null::text) from items i where i.item_id = NEW.item_id);
  elsif TG_OP = 'UPDATE' then
    if OLD.last_update_txtime is distinct from NEW.last_update_txtime then
        data := (select hstore(i)|| hstore('old_category_id',OLD.category_id::text) from items i where i.item_id = NEW.item_id);
    end if;
  end if;

  if data is not null then
        perform xrpc._call(xrpc.x_qname('q_items_dt'), 'consumer_db', 'accept_item', data);
  end if;
  return NULL; -- deferred trigger
end
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

Tables on subscriber's side
Accept function

```
-------SUBSCRIBER
CREATE TABLE user_items_cnt
(
  user_id integer,
  category_id integer,
  cnt integer DEFAULT 0,
  date timestamp without time zone DEFAULT now()
)
```

```
CREATE TABLE tmp_user_cnt
(
  user_id integer
)
```

```
CREATE OR REPLACE FUNCTION accept_item(i_args hstore)
  RETURNS text AS
$BODY$
Begin
    if ((i_args→'category_id') is distinct from (i_args→'category_id_old')) then
        insert into user_items_cnt(user_id, category_id,cnt,date)
        values((i_args→'user_id')::Int, (i_args→'category_id')::Int,
             1, now());
        if (is_args→'old_category_id') is distinct from null then
        insert into user_items_cnt(user_id, category_id,cnt,date)
        values((i_args→'user_id')::Int, (i_args→'old_category_id')::Int,
             -1, now());
        end if;

    --TMP
    /*
     if not exists (
         select user_id from tmp_user_cnt where user_id= (i_args→'user_id')::Int
         ) then
            insert into tmp_user_cnt (user_id)
                values((i_args→'user_id')::Int);
     end if;
    */
    --TMP
    end if;
    return 'OK';

end
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

Provider init function

```
CREATE OR REPLACE FUNCTION init_user_cnt(IN i_user_id integer, OUT error_code)
  RETURNS integer AS
$BODY$
begin
    error_code := 0;

    perform user_id from users where user_id = i_user_id for update;

    perform xrpc._call(xrpc.x_qname('q_items_dt'), 'consumer_db', 'delete_user_cnt', hstore('user_id', i_user_id::text))

    perform
        xrpc._call(xrpc.x_qname('q_items_dt'), 'consumer_db', 'init_cnt', hstore(i))
    from
        (
        select
            user_id,category_id,count(*)
        from
            items
        where
            user_id = i_user_id
        group by user_id, category_id
        ) i;
end;
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

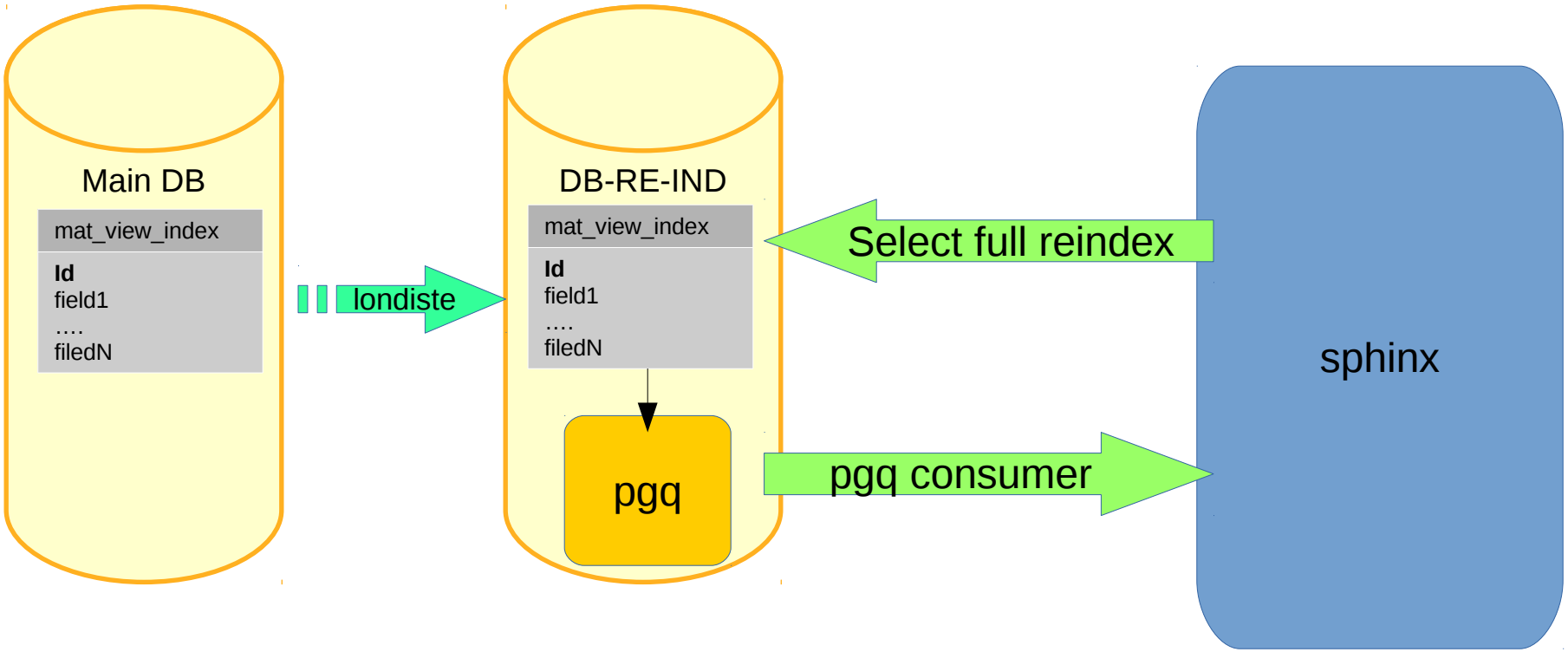
Subscriber's init functions

```
CREATE OR REPLACE FUNCTION delete_user_cnt(i_args hstore)
  RETURNS text AS
$BODY$
begin
    delete from user_items_cnt where user_id = (i_args->'user_id')::int;
    return 'OK';
end
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;


CREATE OR REPLACE FUNCTION init_cnt(i_args hstore)
  RETURNS text AS
$BODY$
begin
    insert into user_items_cnt(user_id, category_id,cnt,date)
        values((i_args->'user_id')::Int, (i_args->'category_id')::Int,
             (i_args->'count')::Int, now());
    return 'OK';

end
$BODY$
  LANGUAGE plpgsql VOLATILE
  COST 100;
```

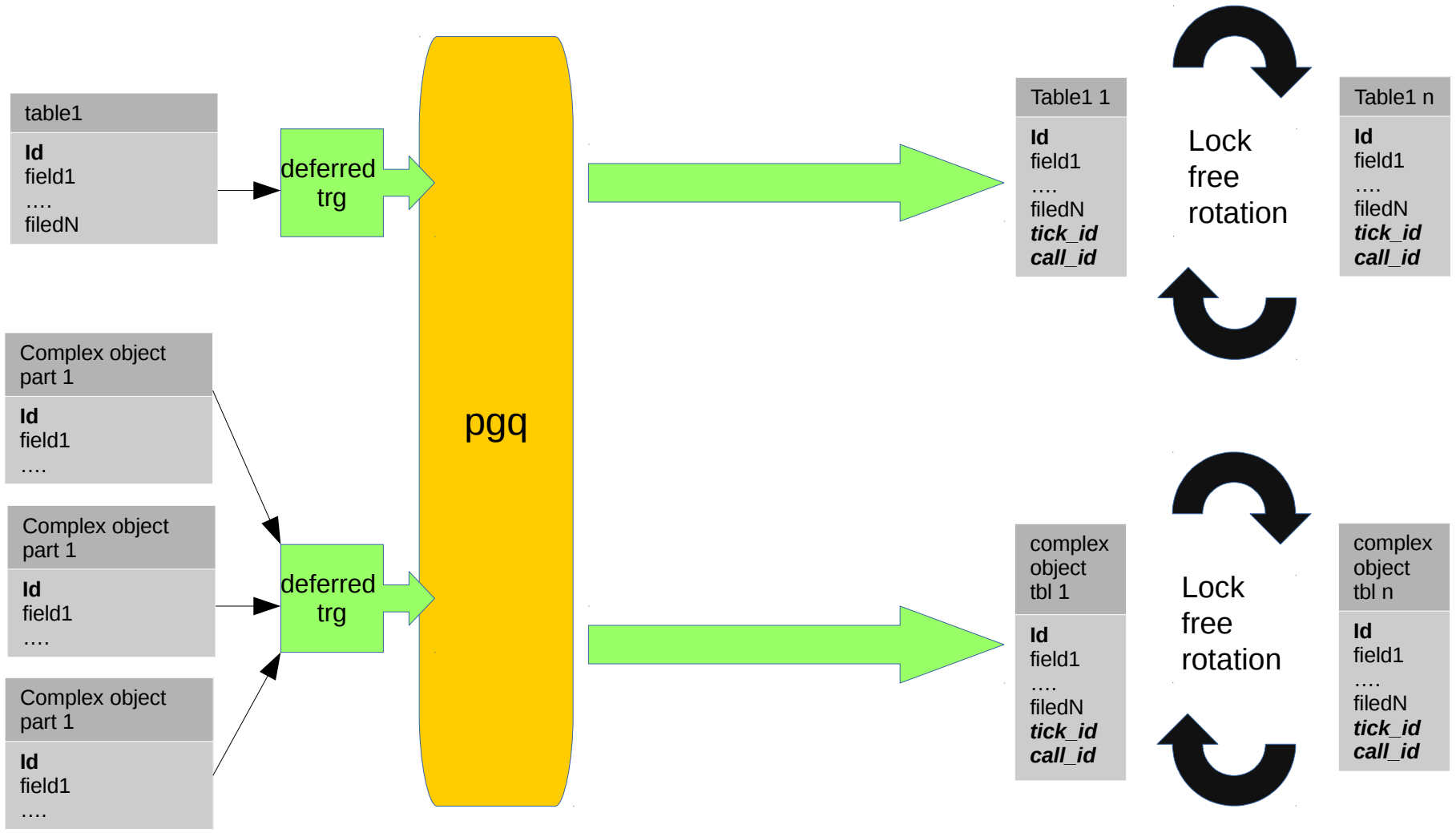
Maintenance

```
    set local work_mem = '4 GB';

    with src as (
        --- найдём строки для удаления (агрегации)
        select
            cnt.user_id,
            cnt.category_id
        from
            user_items_cnt cnt
        group by
            cnt.user_id, cnt.category_id
        having
            count(*) > 1
    ), del as (
        --- удаляем и возвращаем что успешно удалилось на группировку
        delete from
            user_items_cnt cnt
        using
            src
        where
            cnt.user_id = src.user_id and cnt.category_id = src.category_id
        returning cnt.*
    ), agg as (
        --- суммируем только успешно удалённые (заблокированные) строки
        --- защита от конкурентного вызова по ошибке этой же функции
        select
            del.user_id,
            del.category_id,
            sum(del.cnt)::integer  as cnt
        from
            del
        group by
            del.user_id, del.category_id
    )
insert into user_items_cnt(user_id, category_id, cnt, date)
  select agg.user_id, agg.category_id, agg.cnt, now() from agg where agg.cnt_total > 0;
```

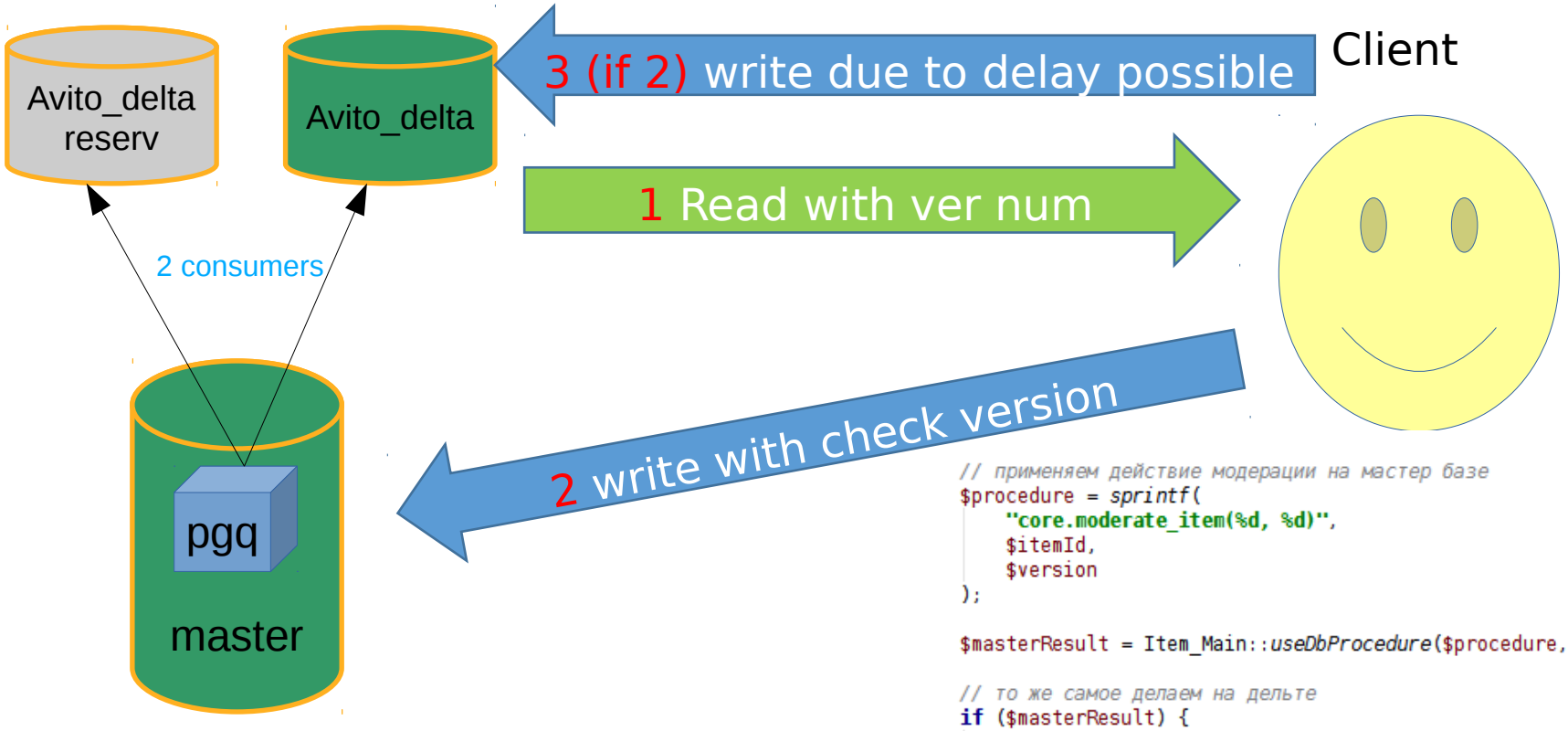
Real time sphinx index



Persistent queue

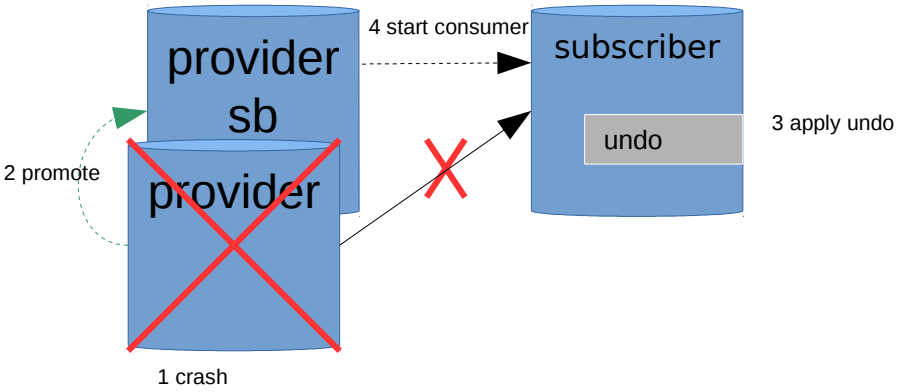


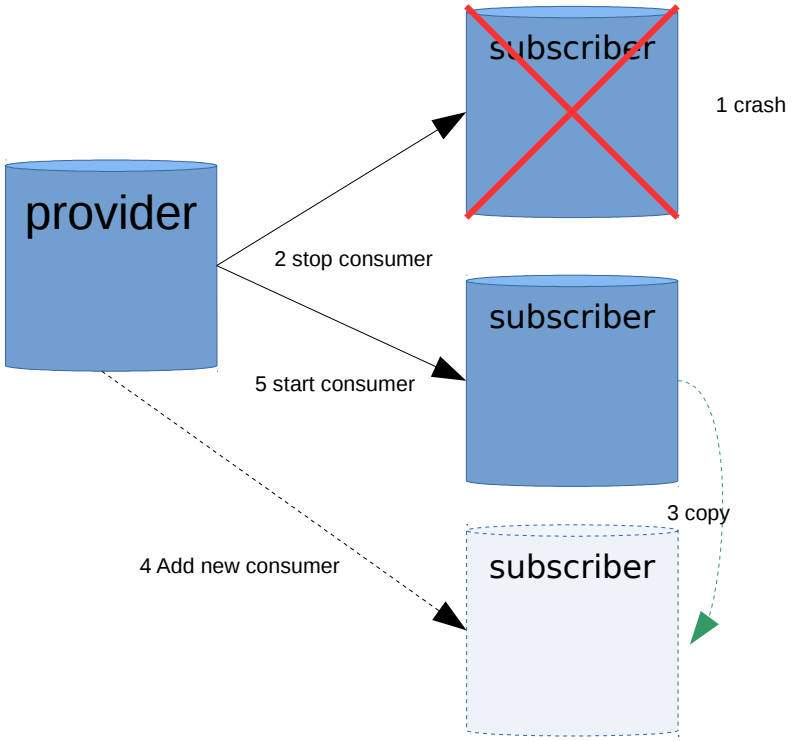
Особенности согласования данных
Eventual consistency




```php
// применяем действие модерации на мастер базе
$procedure = sprintf(
    "core.moderate_item(%d, %d)",
    $itemId,
    $version
);

$masterResult = Item_Main::useDbProcedure($procedure, $itemId);

// то же самое делаем на дельте
if ($masterResult) {
    App::db('delta')->execSP(
        "select * from moderate_item@ver@(%d, %d, %d)",
        $itemId,
        $version,
        $masterResult['o_action_success_flg']
    );
}
```

Восстановление после аварий





Спасибо за внимание!

Константин Евтеев
kevteev@avito.ru

https://hh.ru/vacancy/10795267
https://hh.ru/vacancy/11463461