Data stream in Avito

Константин Евтеев
kevteev@avito.ru

Всем привет!

Константин Евтеев, Avito.ru

- Разработчик баз данных
- С PostgreSQL с 2009 с версий PostgreSQL 7.4/8.3.1, занимался миграциями с MS SQL Server и администрированием обеих СУБД
- В Авито с 2014: разработка распределенных систем обработки данных

Streams in avito

Data stream

1. Логический поток изменений
2. Производное от данных
3. Deffered trigger
4. Trigger
5. Снимаем overhead на источнике

Click stream

1. Нет понятия транзакций
2. Часть событий не идет в click stream

Выгрузка по времени отдельных таблиц в dwh

Задачи решаемые с помощью data stream

Выгрузка данных аналитикам (dwh)

Хранение истории за 4 дня для

1. Возможности повторной отправки в DWH
2. Анализа и отладки жизненного цикла объявлений
3. Используются для получения текущего состояния объявления(модерация)

Заполнение распределенного хранилища

Различная статистика и счетчики

Модерация объявлений

Анализ дублей

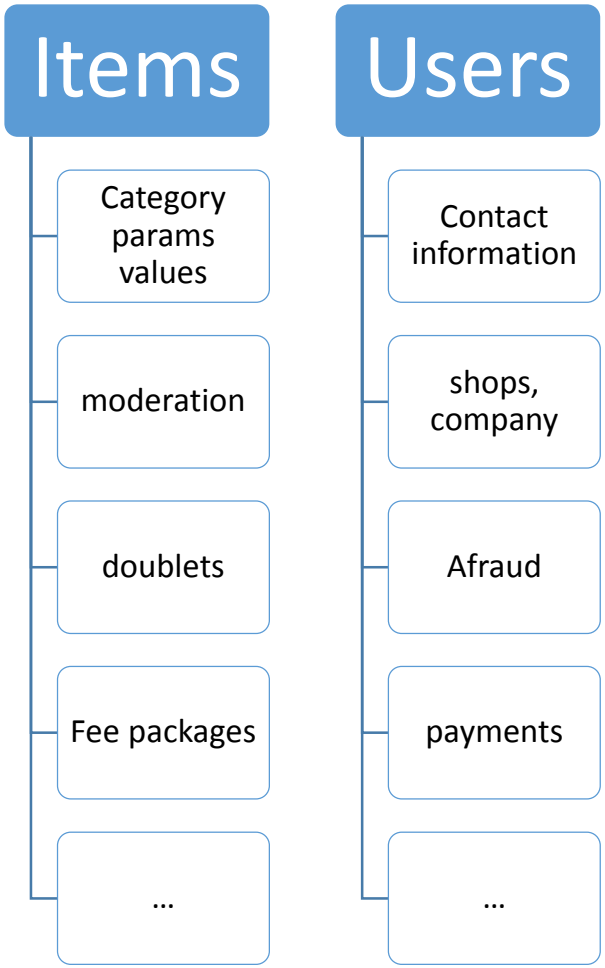
Карма пользователей

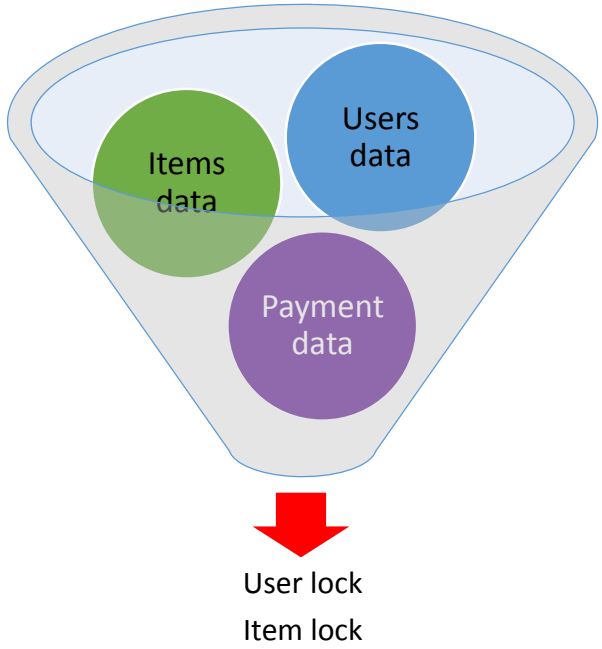
Создание файлового архива

Предпосылки создания: выгрузка данных в различные подсистемы

На данном этапе как раз появляется необходимость создания data stream: сериализовывать данные, а получатель сам выберет, что нужно из данного потока
+ возникает необходимость
1 Реал тайм события
2 История изменений

Все изменения данных под единой блокировкой объекта(row level)





Под блокировкой понимаем цепочку блокировок:
select item_id from update;
....

SELECT ... FROM a JOIN b FOR UPDATE a; BREAKS SNAPSHOT

```
SELECT ... FROM a JOIN b FOR UPDATE a; BREAKS SNAPSHOT

Select of 2-nd transaction will return new value from a and old value from b

CREATE TABLE a
(
  id integer,
  val1 integer
);

CREATE TABLE b
(
  id integer,
  val1 integer
);

insert into a select 1,2;
insert into b select 1,3;

--FIRST TRANSACTION
begin

select * from a inner join b
on
    a.id = b.id
where
    a.id =1 for update of a;

update a set val1 = 0 where id = 1;
update b set val1 =0 where id = 1;

end

-- SECOND TRANSACTION (starts during executing of first transaction)
 select * from a
inner join
b on a.id = b.id
 where a.id =1 for update of a

 --result:
 1;0;1;3
```

Bug report

Your bug report has been received, and given id #13769. It has been posted to the pgsql-bugs mailinglist and will show up there as soon as it has cleared the moderator queue.

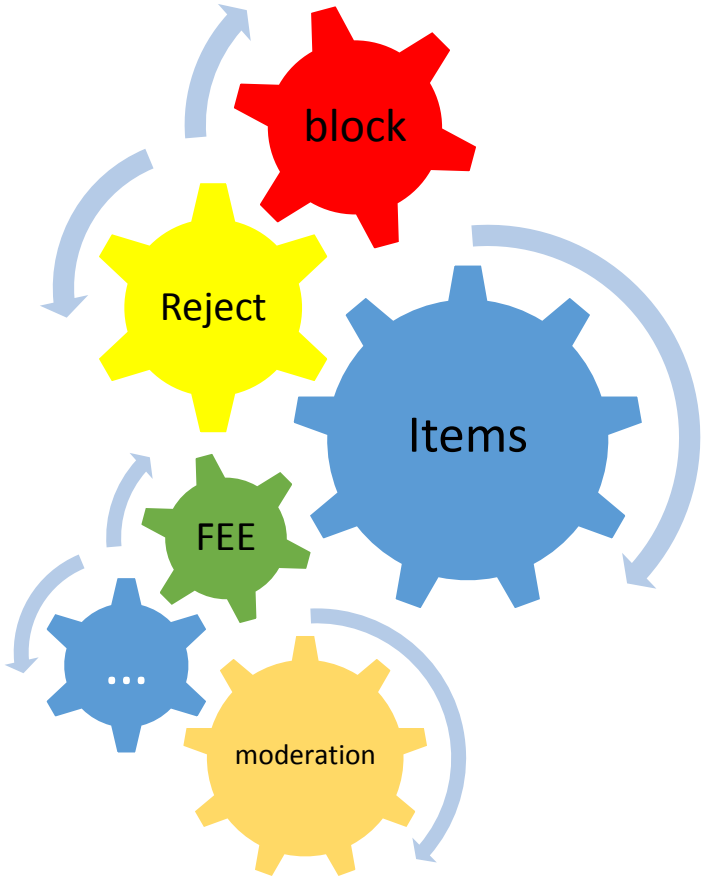
Магия *DEFERRED trigger*

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

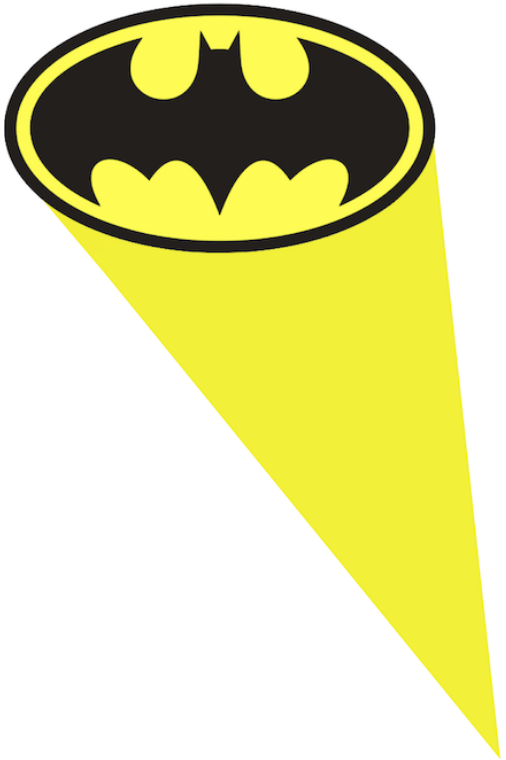
```
CREATE OR REPLACE FUNCTION delta_trg_proc()
  RETURNS trigger AS
$BODY$
Begin
…
if OLD.last_update_txtime is distinct from NEW.last_update_txtime then
        select hstore(i) from items i where i.item_id = NEW.item_id
...
```

Как быть с растущим функционалом?

1. Рост функционала => рост количества таблиц => рост количества join => рост cpu
2. Много событий которые достаточно редки
3. Сериализация удобна, но мы cpu bound



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

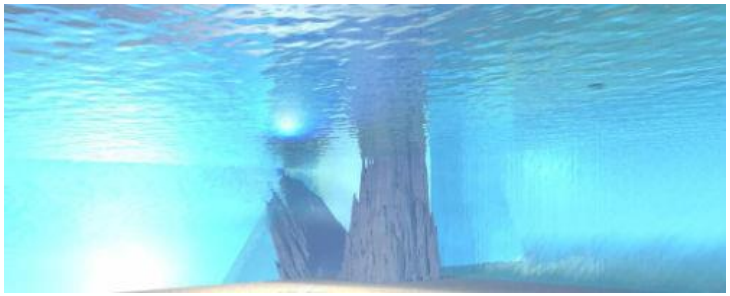
Достоинства:

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

Сессионные переменные, групповые действия, подводные камни …

1. Нужно объявлять переменную в postgresql.conf, чтобы избежать exception (скоро поправят)
2. В 1 транзакции может меняться несколько объектов, а событие относится не ко всем. Решение:Используем массив по unique key array ‘{}’
3. Убираем проблемы рекурсивного зацикливания deffered trigger

```
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;
```



Доставка событий

1. Успешный опыт использования логической системы асинхронной репликации Londiste, предопредил выбор в сторону PgQ, на базе которой написана сама Londiste. *https://www.pgcon.org/2008/schedule/attachments/55_pgq.pdf*

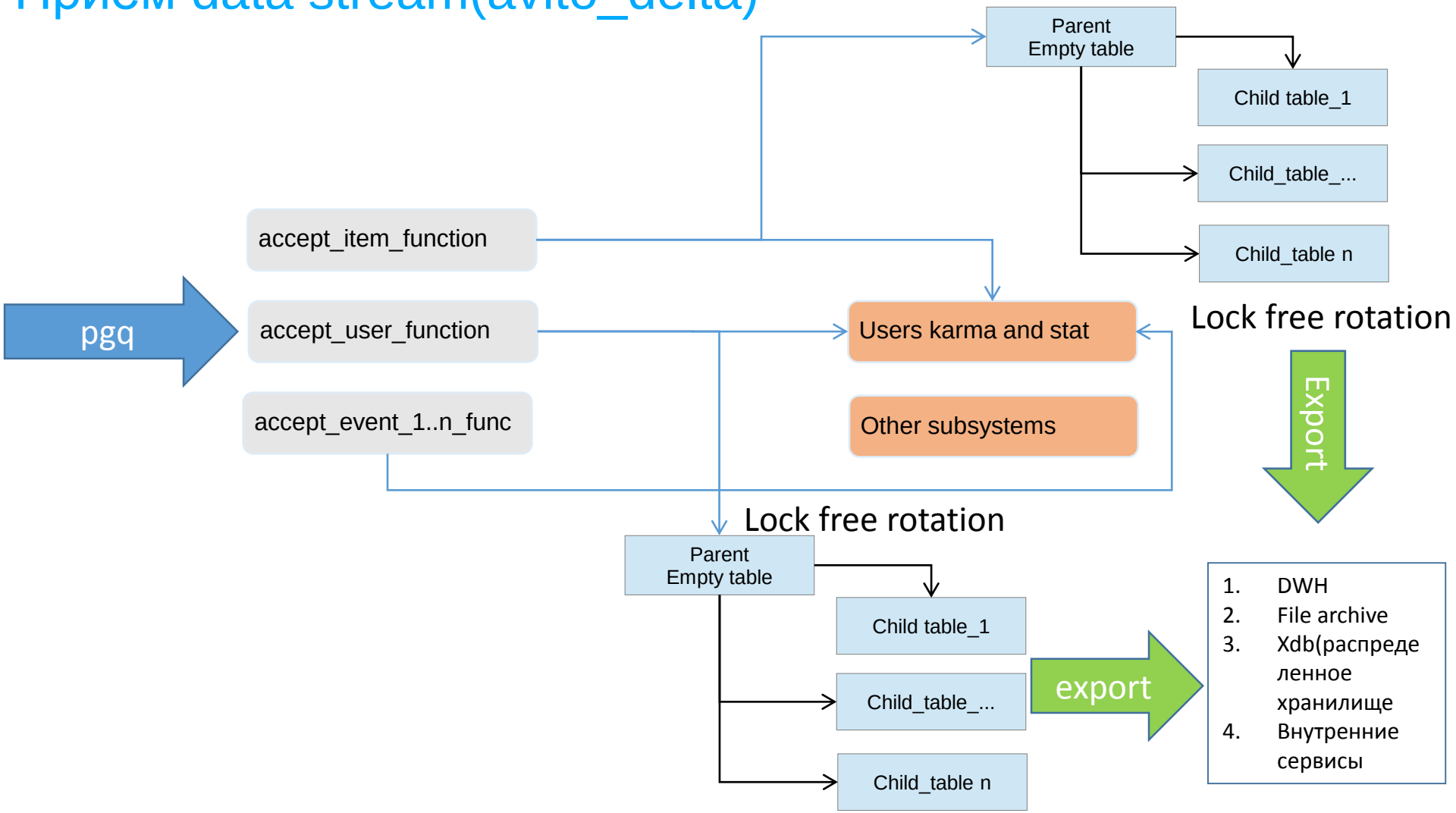
2. PgQ — транзакционная очередь, что гарантирует, что вы увидите каждой событие.
3. Нужно чтобы подписчик очереди работал быстрее отправителя(можно положить в память или делать unlogged таблицы и дублировать получателей на случай аварий)

http://skytools.projects.pgfoundry.org/doc/
https://wiki.postgresql.org/wiki/PGQ_Tutorial

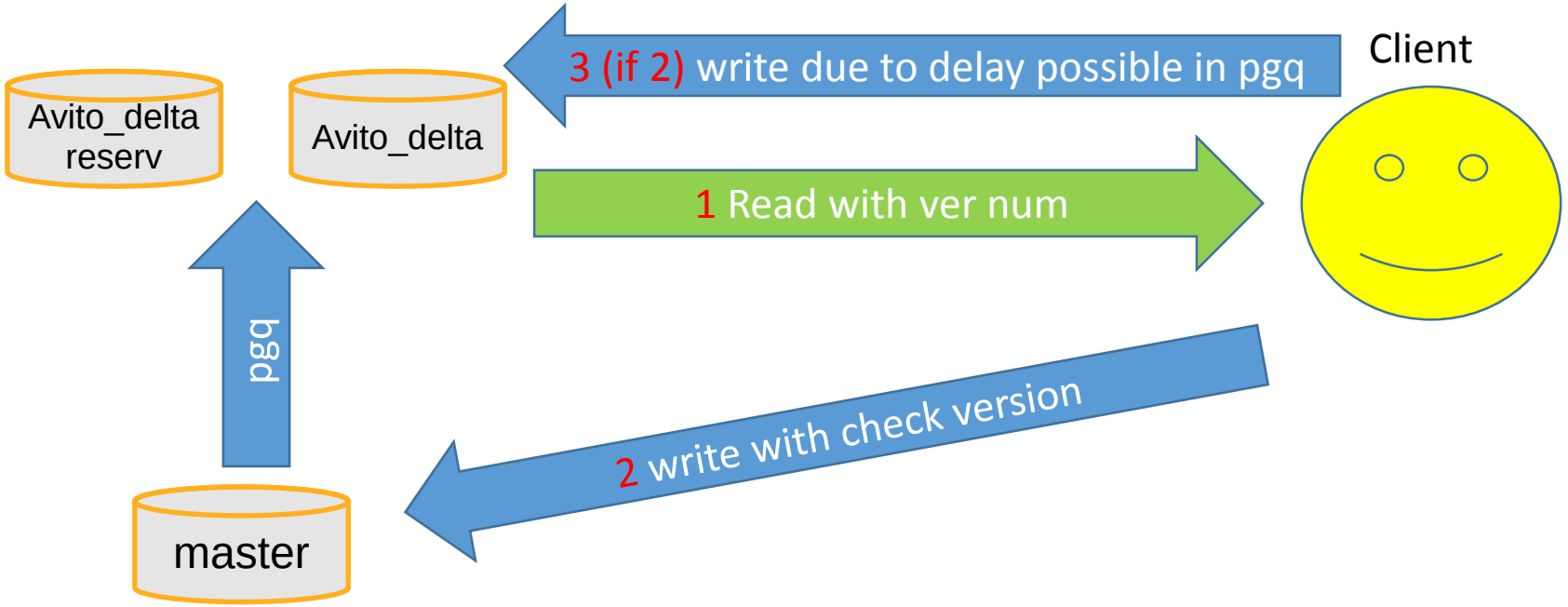
Вызов xrpc

```
data := select hstore(i) from items i where i.item_id = NEW.item_id

…

if current_setting('avito.items_to_moderation') <> '{}' then
        if NEW.item_id = any (
        current_setting('avito.items_to_moderation')::int[]
        ) then
                data := data || hstore('to_moderation', 'true'::text);
        end if;
end if;

…

perform xrpc._call(xrpc.x_qname('items_log'), 'avito_delta',
'collector.accept_item', data);
```

Прием data stream(avito_delta)



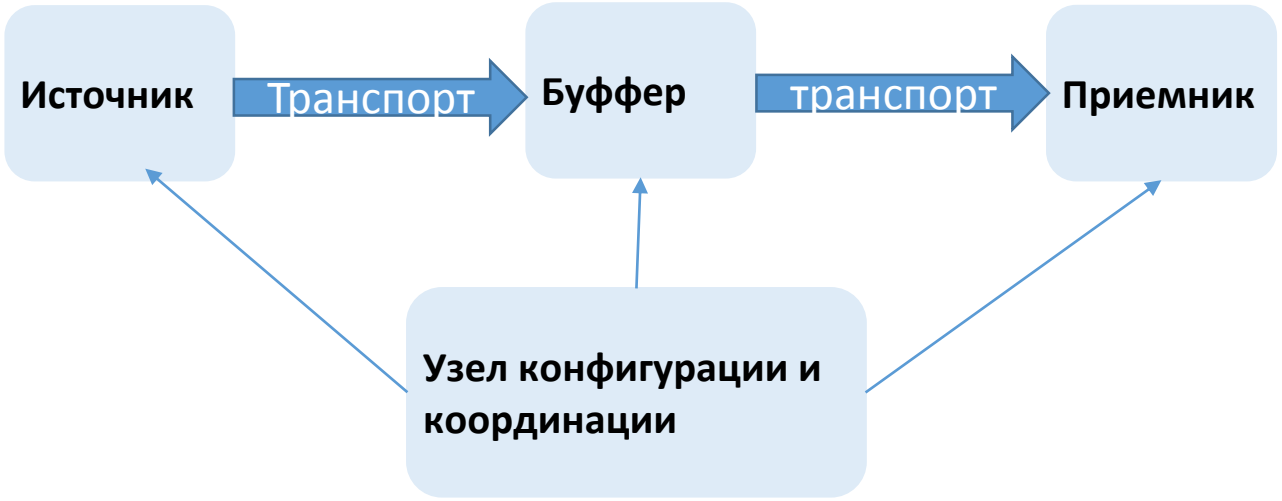
Особенности согласования данных



Moderate item

```php
$procedure = sprintf(
    "core.moderate_item_new@ver@(%d, %d, %d, '%s', '%s', %s, '%s', '%s', %d, '%s', %d)",
    $itemId,
    $version,
    $action,
    \Core\DB::quote($user->login),
    \Core\DB::toTimestamp($moderationTime),
    \Core\DB::toArray($reasons),
    \Core\DB::quote($customReason),
    \Core\DB::quote($comment),
    $accept,
    $reasonExtra ?
    \Core\DB::quote(json_encode($reasonExtra, JSON_UNESCAPED_SLASHES | JSON_UNESCAPED_UNICODE))
    : '{}',
    $eventLogId
);
$masterResult = Item_Main::useDbProcedure($procedure, $itemId);

// то же самое делаем на дельте
if ($masterResult) {
    App::db('delta')->execSP(
        "select * from moderate_item@ver@(%d, %d, %d, '%s', '%s', %d)",
        $itemId,
        $version,
        $action,
        \Core\DB::quote($user->login),
        \Core\DB::toTimestamp($moderationTime),
        $masterResult['o_action_success_flg']
    );
}

return $masterResult;
```

Выгрузка данных в dwh



pg_current_xlog_insert_location()
select pg_xlog_location_diff(pg_last_xlog_replay_location(), '$1')")
select force_wal(pg_current_xlog_insert_location())

Profit

1. Сняли читающую нагрузку с основной базы(делаем предварительные выборки и далее по primary key)
2. Минимум(в большинстве случаев никаких) усилий для отправки в dwh новых полей таблиц, а также возможность перевыгрузить данные
3. Файловой архив позволяющий разобрать любую ситуацию в прошлом(очень хорошо подходит для аудита)
4. Имеем узел на котором выполнять длительные запросы для модерации (до 1 сек)
5. Считаем карму и статистику пользователей, а также без последствий проводим нормализацию данных

Спасибо за внимание!