PostgreSQL

Разработка приложений

Часть третья.

Возможности Postgres

- Views
- Массивы
- CTE
- Индексы
- Prepared statements
- Курсоры
- Разное

Пользователи

- Реально активно обычно 5-15% всех пользователей.
- Обмен с диском идет постранично.
- Один пользователь — одна страница
- При сохранении в кеше тысячи пользователей они будут занимать тысячу страниц; конкретно строка пользователя будет занимать 1/8 — 1/200 страницы.

Пользователи

- Таким образом, 1000 пользователей потребует 8M RAM, из которых будет реально использоваться только 1000*200байт ≈ **20КБ RAM.**
- Кроме того, RAM будет использоваться еще и под индекс.
  - Кстати, Postgres не работает мимо кеша ОС; следовательно, надо выделять или разумное количество памяти, или почти всю.
- Сложим всех активных в секцию usr_active, а неактивных — в usr_dormant

Union all

- ```
  select * from usr_active
  union all
  select * from usr_dormant
  ```
- От неактивных пользователей в кеше останется только индекс.
- Можно еще лучше — создать функцию.
  ```

```
create function usr_all(ns integer[]) returns table(n integer) as
$code$
declare
 v integer;
begin
  foreach v in array ns loop
    select a.n into usr_all.n from usr_active a where a.n=v;
    if found then
      return next;
    else
      select d.n into usr_all.n from usr_dormant d where d.n=v;
      if found then
        return next;
      end if;
    end if;
  end loop;
end;
$code$
language plpgsql
```

```
create function usr_all(p_usr_id int) returns table(n integer) as
$code$
begin
    select a.n into usr_all.n from usr_active a where a.n=p_usr_id;
    if found then
      return next;
    else
      select d.n into usr_all.n from usr_dormant d where d.n=p_usr_id;
      if found then
        return next;
      end if;
    end if;
end;
$code$
language plpgsql
```

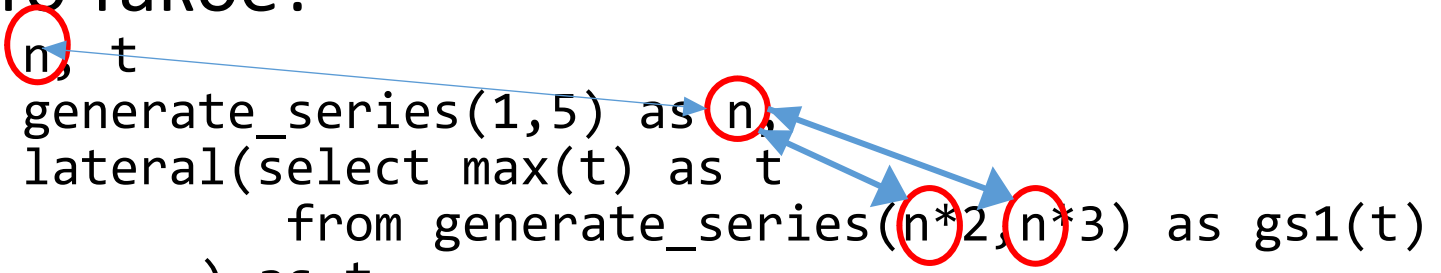
Сравнение

- View — удобно; накладные расходы на индекс
- Функция —удобно (относительно); отсутствие накладных расходов на индекс.
- Функция для LATERAL – удобно, отсутствие накладных расходов на индекс, наличие накладных расходов на вызов функции.
- Так ли уж это все надо?
  - Скорее всего, нет. Но про страничный обмен забывать все же не стоит.
  - pg_repack, pg_reorg

LATERAL

- Что это такое?

```
select n, t
  from generate_series(1,5) as n,
       lateral(select max(t) as t
                 from generate_series(n*2,n*3) as gs1(t)
               ) as t
```



```
 n | t
---+----
 1 |  3
 2 |  6
 3 |  9
 4 | 12
 5 | 15
```

LATERAL

- ```
  select n, t
    from generate_series(1,2) as n,
         lateral(select t as t
                   from generate_series(n*2,n*3) as gs1(t)
                 ) as t
  ```

```
 n | t
---+--
 1 | 2
 1 | 3
 2 | 4
 2 | 5
 2 | 6
```
```

LATERAL – зачем это надо?

- VIEW с параметрами
- Функции теперь first-class citizens
- Очевидное применение:

СЕКЦИОНИРОВАНИЕ

- В примере выше – неактивных пользователей можно хранить в массивах, построив индекс по range по их id.
- Не ограниченно только встроенными средствами – можно делать произвольно.

View с параметрами

- Часто хотят view с параметрами
- Например, для пользователей показывать, кто друг просматривающему; параметр — id просматривающего пользователя
- Что делать?
- Декартово произведение. Да, именно декартово произведение.
- N*1=N. Таблица параметров должна возвращать единственную строку.

View с параметрами

```
create or replace view usr_cart as
select p.id as viewer_id,
       u.*,
       exists(select * from friend f
                       where f.usr_id=viewer_id
                         and f.friend_id=u.id)
         as isfriend
from usr u, usr p
```

Используем:

```
select *
  from usr_cart
 where id in(...)
   and viewer_id=$1
```

View с параметрами

- А что делать, если надо и без viewer_id? Подставим несуществующий:
-
```
create or replace view usr_cart as
 select p.id as viewer_id,
        u.*,
        exists(select * from friend f
                where f.usr_id=viewer_id
                  and f.friend_id=u.id)

         as isfriend
  from usr u, (select 0 as id
                union all
               select id from usr) as p
```
- Используем:

```
select *
  from usr_cart
 where id in(...)
   and viewer_id=coalesce($1,0)
```

В отличие от CTE не создает временной таблицы и оптимизатор знает как это оптимизировать

Top N для M

- Например, 10 последних постов из самых популярных тем. Или 10 последних покупок в категориях.
- Очевидный запрос

```
select p.*
  from post p, topic t
 where p.id in (select p1.id
                  from post p1
                 where p1.topic_id=t.id
                 order by added desc
                 limit 10
                )
```

Top N для M

- К сожалению, не работает. Точнее, работает, но для больших объемов неприемлемо медленно. Обманываем оптимизатор

```
select p.*
  from post p, topic t
 where p.id=any(array(select p1.id
                        from post p1
                       where p1.topic_id=t.id
                       order by added desc
                       limit 10)
                      )
```

Top N для M

- Насколько эффективно это работает?
- Вообще говоря, вполне эффективно
- Если миллионы элементов, то все не так здорово

Common Table Expression

```
with query_name as(
  select * from tbl
)
select * from query_name
```

Зачем надо?

- Удобно: эдакое маленькое view в запросе
- Как и всякое view, позволяет отдельно описывать отдельные логические понятия и не дублировать код.
- Результаты живут в work_mem (или вытесняются на диск); по сути — временные таблицы.
- И, наконец, рекурсия.

Recursive CTE

```
with recursive tq as(
  select 1 as n --1
  union all
  select n+1 from tq where n<10 --2
)
select * from tq
```

Что получилось

```
 N
----
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
(10 строк)
```

Разворачиваем дерево

id	parent_id	name
1	null	Адам
2	1	Каин
3	1	Авель
4	2	Енох

Способ №1 или линейная рекурсия

```
with recursive
 tree(id, parent_id, name) as
 (values(1,null,'Адам'),
        (2,1,'Авель'),
        (3,1,'Каин'),
        (4,2,'Енох')
 ),
 with_parent as(
  select *, null as parent_name
    from tree
   where parent_id is null
  union all
  select tree.*, wp.name
    from tree, with_parent wp
   where tree.parent_id=wp.id
 )
select * from with_parent
```

```
 id | parent_id | name  | parent_name
----+-----------+-------+-------------
  1 |           | Адам  |
  3 |         1 | Каин  | Адам
  2 |         1 | Авель | Адам
  4 |         2 | Енох  | Авель
(4 строки)
```

Ну и в чем разница?

```
with recursive tree(id,
parent_id, name) as
(
  values(1,null,'Адам'),
        (2,1,'Авель'),
        (3,1,'Каин'),
        (4,2,'Енох')
)
select *,
    (select t.name
       from tree t
      where tree.parent_id=t.id
     ) as parent_name
from tree
```

```
 id | parent_id | name  | parent_name
----+-----------+-------+-------------
  1 |           | Адам  |
  3 |         1 | Каин  | Адам
  2 |         1 | Авель | Адам
  4 |         2 | Енох  | Авель
(4 строки)
```

А вот в чем

```
with recursive tree(id, parent_id, name)
as(
  values(1,null,'Адам'),
        (2,1,'Авель'),
        (3,1,'Каин'),
        (4,2,'Енох')),
 with_parent as(
  select *, 'Бог' as ancestors_names
    from tree
   where parent_id is null
  union all
  select tree.*,
         wp.ancestors_names||','||wp.name
    from tree, with_parent wp
   where tree.parent_id=wp.id)
```

•select * from with_parent

```
 id | parent_id | name  | ancestors_names
----+-----------+-------+-----------------
  1 |           | Адам  | Бог
  3 |         1 | Каин  | Бог,Адам
  2 |         1 | Авель | Бог,Адам
  4 |         2 | Енох  | Бог,Адам,Авель
(4 строки)
```

Нелинейная рекурсия

```
with recursive tree(id, parent_id, name) as(
  values(1,null,'Адам'),
        (2,1,'Авель'),(3,1,'Каин'),
        (4,2,'Енох'), (5,null,'Ева'),
        (6,5,'Дочь Евы'),(7,4,'Ирад')),
expand_tree as(
   select id,parent_id,name from tree
   union(
     with expand_tree as(
      select * from expand_tree
     )
     select e1.id, e1.parent_id, e2.name
       from expand_tree e1,
            expand_tree e2
      where e2.parent_id=e1.id
   )
)
select * from expand_tree e where id=1
```

Нелинейная рекурсия-2

```
 id | parent_id | name
----+-----------+-------
  1 |           | Адам
  1 |           | Каин
  1 |           | Авель
  1 |           | Енох
  1 |           | Ирад
(5 строк)
```

Нелинейная рекурсия-3
Как это, собственно, работает

```
 ...where name='Ирад'
 id | parent_id | name
----+-----------+------
  7 |         4 | Ирад
  4 |         2 | Ирад
  2 |         1 | Ирад
  1 |           | Ирад
(4 строки)
```

Путь в циклическом графе

- Создаем граф.
  - Создаем вершины:

```
create table vertex as
  select n
    from generate_series(1,10000) as gs(n);

alter table vertex add constraint vertex_pk
    primary key (n);
```

  - Создаем вершины:

```
 create table edge(f,t) as select distinct (random()
*10000)::integer as from, (random()*10000)::integer as to from
generate_series(1,100000) as gs(n);

 create index edge_ft on edge(f,t);
```

Ищем путь

- Весьма просто:

```
with recursive tq as(
  select 1 as pass, n, array[n] as path
    from vertex where n=1
  union all
   (with t as (select * from tq)
     select pass+1, v.n, t.path||v.n
       from vertex v, t, edge e
      where v.n=e.t and t.n=e.f
        and not (e.t=any(t.path))
        and pass<8
        and not exists(select * from t where n=5000)
    )
 )
select * from tq where n=5000
```

Ищем путь-2

```
 pass |  n   |           path
------+------+--------------------------
    5 | 5000 | {1,7000,6204,922,5000}
(1 строка)
```

- "CTE Scan on tq (cost=1314.97..1321.74 rows=2 width=40) (actual time=115.405..132.329 rows=1 loops=1)"

Что дальше

- CTE как конечный автомат. Со стеком! Разбор JSON'а.
  - Как?
    - Сделать одним запросом? Реально, но громоздко.
    - plpgsql
    - C-libraries
  - Уже встроенный тип
    - All art is quite useless
- Разбираем JSON. JSONPath навыворот
  - Пример: $.users.[100].id

Разбор JSON

```
with recursive
  tokens as(
   select s[1] as token, row_number() over() as n
     from regexp_matches($JS$
  {
    "data": [
     {"name": "User1 User1", "id": "768709679"},
     {"name": "User2 User2","id": "10604454123"}
    ]
    "paging": {
     "next": "https://graph.facebook.com/10000223"
    }
  }
$JS$,
```

Разбор JSON

```
  $RE$(
   (?:[\]\[\}\{])
   |
   (?:" (?:\\"|[^"])+ ")
   |
   \w+
   |
   \s+
   |
   [:,]
  )$RE$,
  'gx') as g(s)
  where s[1] !~ $RE$^[\s:,\s]+$$RE$
),
```

Разбор JSON. Промежуточный результат.

```
                  token                   | n
------------------------------------------+----
 {                                        |  1
 "data"                                   |  2
 [                                        |  3
 {                                        |  4
 "name"                                   |  5
 "User1 User1"                            |  6
 "id"                                     |  7
 "768709679"                              |  8
 }                                        |  9
 {                                        | 10
 "name"                                   | 11
 "User2 User2"                            | 12
 "id"                                     | 13
 "10604454123"                            | 14
 }                                        | 15
 ]                                        | 16
 "paging"                                 | 17
 {                                        | 18
 "next"                                   | 19
 "https://graph.facebook.com/10000223"    | 20
 }                                        | 21
 }                                        | 22
(22 строки)
```

Разбор JSON.

```
  parsed as(
    select n,
           token as token,
           array[token] as stack,
           array['$']::text[] as path,
           '' as jsp,
           array[0]::integer[] as poses
      from tokens t where n=1
    union all
    select t.n,
           t.token as token,
           case when t.token in (']','}') then p.stack[1:array_upper(p.stack,1)-1]
                when t.token in ('[','{') then p.stack || t.token
                else p.stack
           end,
           case when t.token in ('[','{') then p.path ||
                      case when p.stack[array_upper(p.stack,1)]='{'
                         then regexp_replace(p.token,'^"|"$','','g')
                         else '[' || (p.poses[array_upper(p.poses,1)]+1)::text || ']'
                      end
                when t.token in (']','}') then p.path[1:array_upper(p.path,1)-1]
                else p.path
            end,
           case when p.stack[array_upper(p.stack,1)]='{' then p.token
                when p.stack[array_upper(p.stack,1)]='[' then '[' ||
 (p.poses[array_upper(p.poses,1)]+1)::text || ']'
                else ''
           end,
           case when t.token in ('[','{') then p.poses[1:array_upper(p.poses,1)-1]||
(p.poses[array_upper(p.poses,1)]+1)||0
                when t.token in (']','}') then p.poses[1:array_upper(p.poses,1)-1]
                else p.poses[1:array_upper(p.poses,1)-1]||
(p.poses[array_upper(p.poses,1)]+1)
           end
      from parsed p, tokens t where t.n=p.n+1),
```

Разбор JSON.
Промежуточный результат.

n	token	stack	path	jsp	poses
1	{	{"{"}	{$}		{0}
2	"data"	{"{"}	{$}	{	{1}
3	[	{"{",[}	{$,data}	"data"	{2,0}
4	{	{"{",[,"{"}	{$,data,[1]}	[1]	{2,1,0}
5	"name"	{"{",[,"{"}	{$,data,[1]}	{	{2,1,1}
6	"User1 User1"	{"{",[,"{"}	{$,data,[1]}	"name"	{2,1,2}
7	"id"	{"{",[,"{"}	{$,data,[1]}	"User1 User1"	{2,1,3}
8	"768709679"	{"{",[,"{"}	{$,data,[1]}	"id"	{2,1,4}
9	}	{"{",[}	{$,data}	"768709679"	{2,1}
10	{	{"{",[,"{"}	{$,data,[2]}	[2]	{2,2,0}
11	"name"	{"{",[,"{"}	{$,data,[2]}	{	{2,2,1}
12	"User2 User2"	{"{",[,"{"}	{$,data,[2]}	"name"	{2,2,2}
13	"id"	{"{",[,"{"}	{$,data,[2]}	"User2 User2"	{2,2,3}
14	"10604454123"	{"{",[,"{"}	{$,data,[2]}	"id"	{2,2,4}
15	}	{"{",[}	{$,data}	"10604454123"	{2,2}
16	]	{"{"}	{$}	[3]	{2}
17	"paging"	{"{"}	{$}	]	{3}
18	{	{"{","{"}	{$,paging}	"paging"	{4,0}
19	"next"	{"{","{"}	{$,paging}	{	{4,1}
20	"https://graph.facebook.com/10000223"	{"{","{"}	{$,paging}	"next"	{4,2}
21	}	{"{"}	{$}	"https://graph.facebook.com/10000223"	{4}
22	}	{}	{}	}	{}

(22 строки)

Разбор JSON.

```
res as(
  select *
  from parsed
 where (stack[array_upper(stack,1)]='['
    or (stack[array_upper(stack,1)]='{'
   and poses[array_upper(poses,1)]%2=0)
   )
     and token not in ('{','[','}',']')
)
select array_to_string(path,'.')||'.'||
        regexp_replace(jsp,'^"|"$','','g')
               as json_path,
        regexp_replace(token,'^"|"$','','g')
               as json_value
  from res
```

Результат

```
    json_path     |                json_value
------------------+-----------------------------------
 $.data.[1].name | User1 User1
 $.data.[1].id   | 768709679
 $.data.[2].name | User2 User2
 $.data.[2].id   | 10604454123
 $.paging.next   | https://graph.facebook.com/10000223
(5 строк)
```

А зачем это надо?

- Вообще полезно знать.
- Иногда нет возможности установить что-то дополнительно
- Не только JSON — что угодно с относительно несложным синтаксисом.
- Можно предложить еще способ — например, строить дерево со ссылками на родителя, рекурсивно же разбирать аналог JSONPath и добираться до элементов.
  - И не только JSON — пути в дереве вообще.

```
with recursive fs(id,pid,name) as(
 values
   (1,null,null),
   (2,1,'usr'),
   (3,2,'local'),
   (4,3,'postgres'),
   (5,4,'bin')
),
path as(
 select p, row_number() over() as n
   from   regexp_split_to_table('/usr/local/postgres/bin','/')
   as v(p)
),
parsed as(
  select id, name,p,n
    from fs, path where fs.pid is null and path.p=''
  union all
  select fs.id, fs.name, path.p, path.n
    from fs, path, parsed
   where fs.pid=parsed.id and fs.name=path.p
     and path.n=parsed.n+1
)
select * from parsed order by n desc limit 1
```

Что получилось

```
 id | name |   p   | n
----+------+-------+---
  5 | bin  | bin | 5
(1 строка)
```

Prepared statements

- Увеличивают производительность — не требуется каждый раз
  - разбирать запрос
  - Нагружать каталог на предмет существования таблиц, индексов и т. п.
  - Проверять права доступа к объектам
  - А в случае широкого использования view все может оказаться еще интереснее
- prepare <sthname> as <real query>
- execute <sthname>(par1,par2...parN)

Prepared statements & plpgsql

- В plpgsql отсутствует аналог пакета DBMS_SQL — prepared statements недоступны.
  - Каждый статический запрос компилируется один раз при обращении или перекомпилируется при изменении объектов
    - Внимание: раньше была проблема, если объекты, на которые ссылается statement, в момент начала его выполнения еще существовали, но до обращения к ним были удалены. Теперь ее больше нет.
    - Типичный пример: переименование таблицы при высокой OLAP-нагрузке
    - Что делать?

Advisory locks

- Решение 1: рекомендательные блокировки
  - pg_advisory_lock, pg_advisory_xact_lock, _shared, pg_try_
- Решение 2: обычные блокировки. Таблица блокировок. select … for share.

Prepared statements

- А точно ли недоступны prepared statements в plpgsql?
- Конечно доступны: execute '….':

```
execute 'prepare sth1 for select 1';
execute 'execute sth1' into res;
```

- Execute 'execute' — а вот параметры, увы, действительно недоступны. `quote_literal()`, `quote_ident()`

Prepared statements

- Насколько это быстро?
  - Кстати: функции времени: now() возвращает дату и время начала транзакции. Используйте clock_timestamp()
  - Кстати - анонимные блоки:
    - do $$ code $$
  - Кстати: $$-quoting:
    - $$, RE, Im, $ANYTHING_ELSE$
      - `select $Re$^(\"|[^"])+$Re$`

Код

```
do $code$
declare
 i integer;
 ts1 timestamp with time zone;
 ts2 timestamp with time zone;
 res integer;
begin
  execute 'prepare sth1 as select 1';
  ts1:=clock_timestamp();
  for i in 1..100000 loop
--    execute 'execute sth1' into res;
--    select 1 into res;
--    execute 'select 1' into res;
-- Реальная таблица: select 1 as n into stht
--   execute 'prepare sth1 as select n from stht';
--   select n into res from stht
--   execute 'select n from stht' into res
  end loop;
  ts2:=clock_timestamp();
   raise notice 'diff=%', (ts2-ts1)::text;
end;
$code$
```

Результаты

	execute 'execute...'	select .. into	execute 'select...'
select 1	00:00:01.157	00:00:00.297	00:00:01.374
select n from stht	00:00:02.158	00:00:00.930	00:00:04.331

А зачем это, собственно, надо?

- Разбор
- Построение плана
- Выполнение
- Нередко построение плана > выполнение
- С динамическим выполнением все идет каждый раз по новой
- С prepared — только один раз
- pg_prepared_statements

Prepared statements

- Возвращаем резалтсет
  - return query execute
  - только обернув в функцию
- Что еще?
  - save_record('table_name',
    'column1_name','column2_value',
    'column2_name','column2_value'...
)
    - plpgsql — функции с переменным числом параметров

Кстати: upsert и вставка в несколько таблиц

Подход в лоб:

```
with
 input as
  (select n from generate_series(1,10) as gs(n)),
  upd as(
   update ups set val=i.n from input i
    where ups.n=i.n returning i.n
  ),
  ins as(
    insert into ups select * from input i
     where not exists(select * from upd u
                      where i.n=u.n)
     returning ups.n
  )
  select (select count(*) from upd) as updates,
         (select count(*) from ins) as inserts
```

Upsert

- Подход в лоб не работает — race condition: после обновления другой процесс может добавить новые строки, и при вставке возникнет ошибка.
- Как правильно:
  - Для каждой вставляемой строки
    - Попытаться обновить
    - Обновилась — хорошо, не обновилось — добавляем
    - При нарушении уникальности повторяем попытку добавления строки с самого начала

Кстати: plpgsql, savepoint и исключения

- В plpgsql нет явных savepoint
- В plpgsql при использовании исключения в блоке на входе в блок ставится неявный savepoint
- При возникновении исключения в блоке происходит откат к нему
- Пример:

Кстати: plpgsql, savepoint и исключения

```
•do $code$
begin
 create temporary table
    excdemo as select 1 as n;
 begin
   update excdemo set n=10;

   raise notice
    'BEFORE exception n=%',
      (select n from excdemo);
   raise sqlstate
         'EX001';
 exception
   when sqlstate 'EX001' then
    raise notice
      'AFTER exception n=%',
      (select n from excdemo);
 end;
end;
$code$
```

```
•ЗАМЕЧАНИЕ:  BEFORE exception
n=10

•ЗАМЕЧАНИЕ:  AFTER exception
n=1
```

Кстати: plpgsql, savepoint и исключения

- Позволяет делать логгирование
  - Не нужны автономные транзакции (ну, почти не нужны...)
- **Позволяет выполнять потенциально некорректный код!**
  - **Это хорошо, а не плохо!** Можно в живой боевой сервер на ходу добавлять код, не обваливая все по ошибке.
    - «Можно» - разумеется, не значит «нужно».

Курсоры

- Задача — resultset из фунции
  - funcname(..) returns table(colname coltype...)
  - select * from funcname(...)
- Задача расширилась — несколько resultset'ов из функции
- Задача еще более расширилась — МНОГО resultset'ов
- Причем переменное число
- Что делать?

Курсоры

- Курсоры существуют только в пределах транзакции
  - Начало транзакции
  - Вызов фунции
  - Fetch all from <cursor name1>
  - Fetch all from <cursor name2>
  - …
  - Fetch all from <cursor nameN>
  - commit

Курсоры

- refcursor

В сущности, курсор — это просто его имя; имя курсора — строка

```
declare
 cursname1 refcursor;
...
begin
...
 open cursname1 for select ...
...
 return array[cursname1,cursname2...]
```

Зачем надо

Дата продажи	Тип	Сумма
2012-10-01	Пиво	100
2012-10-01	Квас	50
2012-10-02	Вино	45
...	...	...

	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII
Пиво	100	80	100	100	120	150	160	150	120	100	90	80
Квас	...	...	...	...	...	...	...	...	...	...	...	...
Вино	...	...	...	...	...	...	...	...	...	...	...	…
Кефир	...	...	...	...	...	...	...	...	...	...	...	...

Курсоры

- Требуется поддержка на клиенте
- Для курсоров она реализуется несложно — в perl или php достаточно рекурсивно обойти resultset и посмотреть типы колонок.
- В Java — используя декоратор, создать класс, возвращающий ResultSet (как именно его создавать — отдельная история)
- Для json можно поступить аналогично
  - Отличть json от text на клиенте малореально. Хак: можно сделать domain для text и смотреть длину.

generic_cursor

```
CREATE OR REPLACE FUNCTION generic_cursor(IN sql text, VARIADIC param text[] DEFAULT
ARRAY[]::text[])
  RETURNS refcursor AS $BODY$
declare rv refcursor;
sthhash text := 'GENCURS'||md5(sql);
exec    text := 'execute ' || sthhash ||
        case when array_length(param,1) is null then ''
else '('||(select string_agg(coalesce(quote_literal(pv),'NULL'),',') from
unnest(param) as u(pv)) ||')' end;
begin
  begin
     open rv for execute exec;
     exception
        when sqlstate '26000' -- prepared statement does not exist
          or sqlstate '0A000' -- table definition had changed
          or sqlstate '42703' --- -/-
          or sqlstate '42P11'
        then
         if sqlstate='0A000' or sqlstate='42703' then
           execute 'deallocate '||sthhash;
         end if;
         execute 'prepare '||sthhash|| coalesce((select '('|| string_agg('text',',') |
| ')' from unnest(param) as u(pv)),'') ||' as '||sql;
         open rv for execute exec;
  end;
  return rv;
end; $BODY$ LANGUAGE plpgsql
```

generic_exec

```
CREATE OR REPLACE FUNCTION generic_exec(
    IN sql text,
    VARIADIC param text[] DEFAULT
                              ARRAY[]::text[])
RETURNS SETOF record AS $BODY$
declare
 cr refcursor:=
       generic_cursor(sql, variadic param);
begin
    return query execute 'fetch all from '||
                            quote_ident(cr::text);  end;
$BODY$ LANGUAGE plpgsql
```

Используем

```
select *
from
generic_exec('select n from

                generate_series(1,$1::integer)
                   as gs(n)',
                  20::text) as ge(n integer)
```

А оно надо? Да!

```
do $code$
declare
 rv tt;
 s1 timestamp;
 s2 timestamp;
 s3 timestamp;
 i integer;
begin
 s1:=clock_timestamp();
 for i in 1..1000 loop
      execute 'select t1.* from tt t1, tt t2, tt t3, tt t4, tt t5, tt t7, tt t8
    where t1.id=t2.id and t2.id=t3.id and t3.id=t4.id and t1.id=t5.id and t5.id=t7.id and
t5.id=t8.id  and t8.id=9999
    and t1.id between 1 and 10000 and t4.id between 9999 and 200000 and t3.id>9000 and
exists(select * from tt t6 where t6.id=t5.id)
    and t4.cnt=49 order by t4.cnt' into rv;
 end loop;

 s2:=clock_timestamp();
 for i in 1..1000 loop
    select * into rv from generic_exec('select t1.* from tt t1, tt t2, tt t3, tt t4, tt t5,
tt t7, tt t8
    where t1.id=t2.id and t2.id=t3.id and t3.id=t4.id and t1.id=t5.id and t5.id=t7.id and
t5.id=t8.id and t8.id=9999
    and t1.id between 1 and 10000 and t4.id between 9999 and 200000 and t3.id>9000 and
exists(select * from tt t6 where t6.id=t5.id)
    and t4.cnt=49 order by t4.cnt') as ge(n integer, cnt bigint);
 end loop;
 s3:=clock_timestamp();
 raise notice 'Run 1=%',s2-s1;
 raise notice 'Run 2=%',s3-s2;
end; $code$
```

Результат

- ЗАМЕЧАНИЕ: Run 1=00:00:04.867
- ЗАМЕЧАНИЕ: Run 2=00:00:00.207

То же самое, но многопоточно

- $ pgbench -M prepared -U postgres -t 500 -n -j 2 -c 10 -f generic_exec.pgb work
number of clients: 10
number of threads: 2
number of transactions per client: 500
number of transactions actually processed: 5000/5000
tps = **2340.001464** (including connections establishing)
tps = **2508.521577** (excluding connections establishing)

- $ pgbench -M prepared -U postgres -t 500 -n -j 2 -c 10 -f plain_execute.pgb work
query mode: prepared
number of clients: 10
number of threads: 2
number of transactions per client: 500
number of transactions actually processed: 5000/5000
tps = **146.373869** (including connections establishing)
tps = **147.008750** (excluding connections establishing)

Альтернатива

- JSON — в 9.2 row_to_json, array_to_json
  - ```
    select
      array_to_json(
        array(
          select row_to_json(c.*)
    from pg_class c
              )
       )
    ```
- В предыдущих версиях легко можно написать и самостоятельно, используя, например, hstore
- Можно просто возвращать массив hstore и разбирать на клиенте
    ```

Заключение

- И снова: а зачем все это, собственно, надо?
- Производительность
- Переносимость
- Вычурность
- Пробуйте!
- Вопросы?