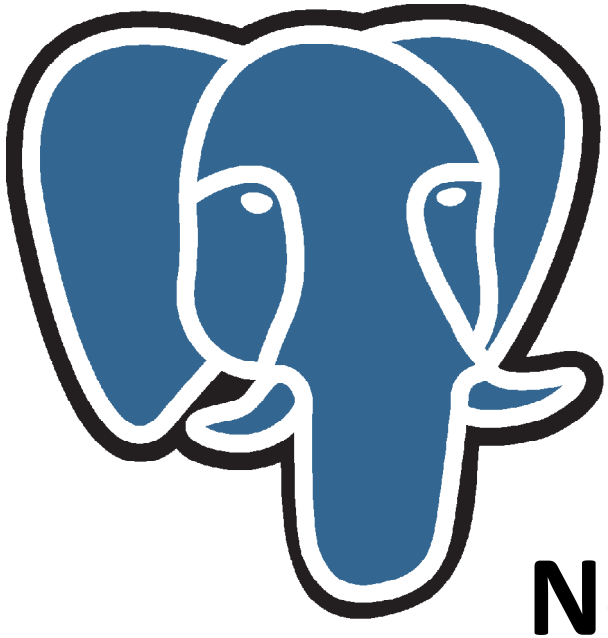


NoSQL в PostgreSQL

Чувашов Иван

Старший инженер-разработчик

АО «ОТС»

SONY

Чем PostgreSQL лучше других свободных СУБД

HITACHI

Яндекс

pg_barman

plproxy

Типы данных

bigint, bigserial, bit **[ ]**, bit varying **[ ]**, boolean, box, bytea, character **[ ]**, character varying **[ ]**, cidr, circle, date, double precision, inet, integer, interval [поля] **[ ]**, **JSON**, **JSONB**, line, lseg, macaddr, money, numeric **[ ]**, path, pg_lsn, point, polygon, real, smallint, smallserial, serial, text, time **[ ]**, time **[ ]** with time zone, timestamp **[ ]** , timestamp **[ ]** with time zone, tsquery, tsvector, txid_snapshot, uuid, xml

Размеры данных

Максимальный размер базы данных	Неограничен
Максимальный размер таблицы	32 TB
Максимальный размер строки	1.6 TB
Максимальный размер поля	1 GB
Максимальное количество строк в таблице	Неограничено
Максимальное количество столбцов в таблице	250-1600 в зависимости от типа столбца
Максимальное количество индексов в таблице	Неограничено

Индексирование

- **Частичные индексы**

```
--Частичный индекс по двум битовым полям
CREATE INDEX ixf__tender__publication_date
  ON public.tender
  USING btree
  (publication_date)
  WHERE is_gov_tender = 0::bit(1) AND rejection_flag = 0::bit(1);
```

Индексирование

- **Функциональные индексы**

```
--создание индекса URL в нижнем регистре
create index ix__tender__tender_href on tender( lower(tender_href) );
```

```
--создание индекса по полному ФИО пользователя
create index ix__contact_info__first_name__last_name ON
       spider.contact_info ((last_name || ' ' || first_name || ' ' || middle_name));
```

Индексирование

- **GIST** (геометрические типы, сетевые адреса, диапазоны)
- **GIN** (массивов, JSONB, tsvector)
- **BRIN** (не точный, быстрый, маленький)

Возможности запросов

- **Объединение запросов**

  UNION, INTERSECT и EXCEPT

- **Оконные функции**

  ROW_NUMBER(), RANK(), DENSE_RANK() и PERCENT_RANK()

Возможности запросов

- **Латеральные вложенные запросы**

```
from spider.contract c
 left join lateral
 (
   select contract_id, id, register_in_rf_tax_bodies__inn as inn, null kpp
   from spider.individual_person_foreign_state a
   where a.contract_id = c.id
   union all
   select contract_id, id, inn, null kpp
   from spider.individual_person_rf a
   where a.contract_id = c.id
   union all
   select contract_id, id, register_in_rf_tax_bodies__inn as inn, register_in_rf_tax_bodies__kpp as kpp
   from spider.legal_entity_foreign_state a
   where a.contract_id = c.id
   union all
   select contract_id, id, inn, kpp
   from spider.legal_entity_rf a
   where a.contract_id = c.id
 ) supplier on 1 = 1
) t;
```

Языковые расширения

PL/pgSQL, PL/Perl, PL/Python и PL/Tcl,

PL/Java, PL/PHP, PL/Py, PL/R, PL/Ruby, PL/Schem e, PL/sh, PL/V8 и C

NoSQL данные в PostgreSQL

- Массивы

```
CREATE TABLE sal_emp (
    name            text,
    pay_by_quarter  integer[],
    schedule        text[][]
);
```

```
select * from unnest(array[1, 2, 3]);
unnest
--------
1
2
3
(3 rows)
```

массив в таблицу

```
select string_to_array('a|b|c|d', '|');
string_to_array
-----------------
{a,b,c,d}
(1 row)
```

строку в массив

```
select array_agg(a) from unnest(array[1, 2, 3, 4, 5]) a
where a = any(array[4, 5, 6, 7, 8]);
array_agg
-----------
{4,5}
(1 row)
```

найти пересечение
двух массивов

```
select array_agg(distinct a)
from unnest(array[1, 2, 3, 4, 5] || array[4, 5, 6, 7, 8]) a;
array_agg
-------------------
{1,2,3,4,5,6,7,8}
(1 row)
```

сложить
два массива

```
select array_agg(a) from unnest(array[1, 2, 3, 4, 5]) a
where a <> all(array[4, 5, 6, 7, 8]);
array_agg
-----------
{1,2,3}
(1 row)
```

вычитание
двух массивов

Пример работы с массивами

```
--таблица продуктов
create table product
(
  id bigserial,
  product_name text not null,
  constraint pk_product primary key(id)
);

--таблица магазинов
create table shop
(
  id bigserial,
  shop_name text not null,
  constraint pk_shop primary key(id)
);

--таблица связи
create table shop_product
(
  id bigserial,
  product_id bigint not null,
  shop_id bigint not null,
  constraint pk_shop_product primary key(id),
  constraint fk_shop_product__shop__shop_id foreign key (shop_id)
      references shop (id) match simple
      on update no action on delete no action,
  constraint fk_shop_product__product__product_id foreign key (product_id)
      references product (id) match simple
      on update no action on delete no action
);

--Создаем индексы
create /*drop */ index ix__shop_product__shop_id__product_id on shop_product (shop_id, product_id);
create /*drop */ index ix__product__product_name on product (product_name);
```

Пример работы с массивами

```
--заполняем таблицы
insert into product(id, name)
select a, 'Продукт №' || a :: text from generate_series (1,10000000) a;

insert into shop(id, name)
select a, 'Магазин №' || a :: text from generate_series (1,10000) a;

insert into shop_product(product_id, shop_id)
select case when product_id = 0
            then 1
            else product_id
       end as product_id,
       case when shop_id = 0
            then 1
            else shop_id
       end as shop_id
from
(
  select (random() * 10000000) as product_id, (random() * 10000) as shop_id
  from generate_series (1,10000000) a
) t;
```

Пример работы с массивами

```
--таблица магазинов с массивами
create table shop_mass
(
  id bigserial,
  shop_name text not null,
  product_id bigint[] not null,
  constraint pk_shop_mass primary key(id)
);

--заполняем данными
insert into shop_mass(shop_name, product_id)
select s.shop_name, array(select sp.product_id from shop_product sp where sp.shop_id = s.id)
from shop s;

--стоим индекс
create index ix__shop_mass__product_id on shop_mass using gin (product_id);
```

Пример работы с массивами

```
--таблица магазинов с массивами
create table shop_mass
(
  id bigserial,
  shop_name text not null,
  product_id bigint[] not null,
  constraint pk_shop_mass primary key(id)
);

--заполняем данными
insert into shop_mass(shop_name, product_id)
select s.shop_name, array(select sp.product_id from shop_product sp where sp.shop_id = s.id)
from shop s;

--стоим индекс
create index ix__shop_mass__product_id on shop_mass using gin (product_id);
```

Пример работы с массивами

```
select s.shop_name
from product p
  join shop_product sp on p.id = sp.product_id
  join shop s on s.id = sp.shop_id
where p.product_name like '%₽50000%'
order by s.shop_name;
```

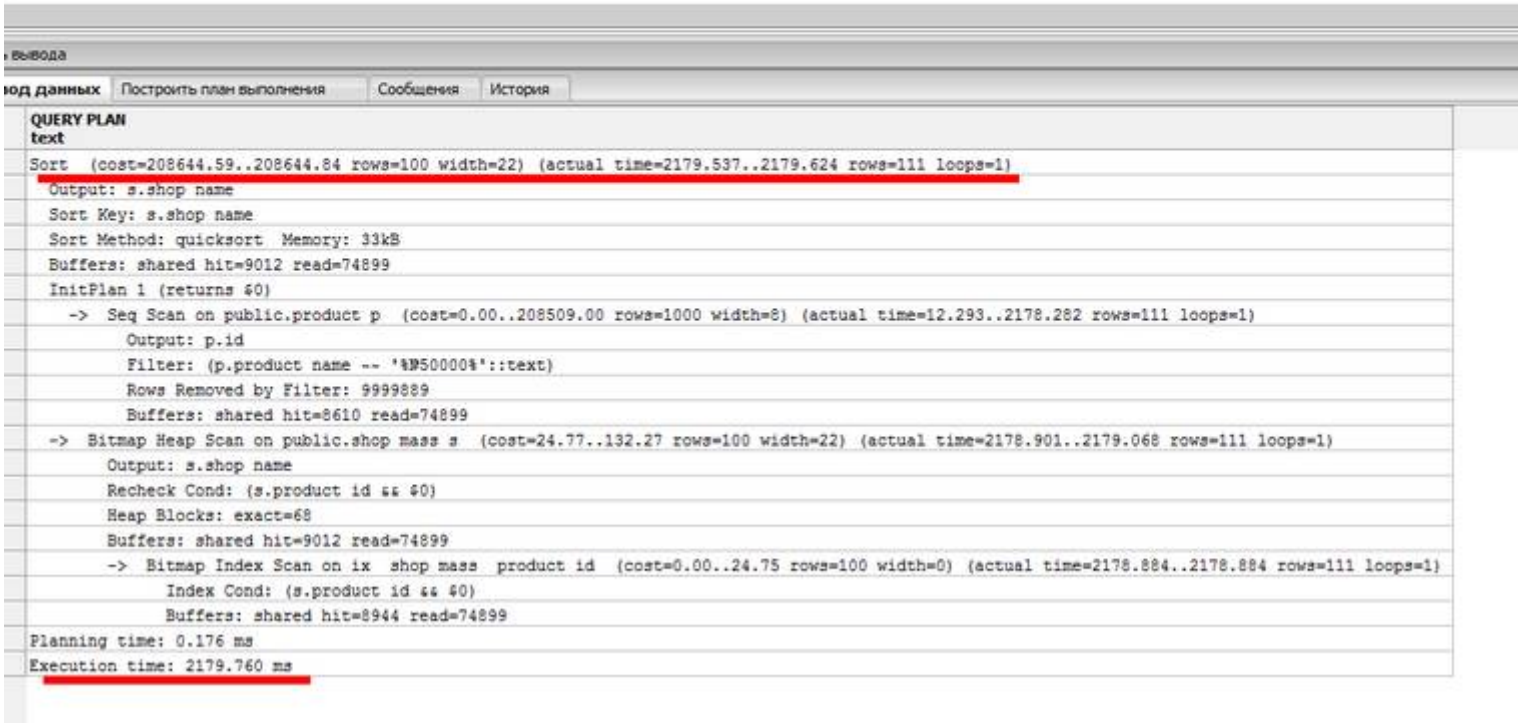
вывода

од данных | Построить план выполнения | Сообщения | История

```
QUERY PLAN
text
Sort  (cost=638739.11..638741.61 rows=1000 width=22) (actual time=23085.195..23085.317 rows=111 loops=1)
  Output: s.shop_name
  Sort Key: s.shop_name
  Sort Method: quicksort  Memory: 33kB
  Buffers: shared hit=8979 read=335673
  ->  Nested Loop  (cost=208521.79..638689.28 rows=1000 width=22) (actual time=8131.717..23084.681 rows=111 loops=1)
        Output: s.shop_name
        Buffers: shared hit=8979 read=335673
        ->  Hash Join  (cost=208521.50..638376.74 rows=1000 width=8) (actual time=8131.697..23083.689 rows=111 loops=1)
              Output: sp.shop_id
              Hash Cond: (sp.product_id = p.id)
              Buffers: shared hit=8645 read=335673
              ->  Seq Scan on public.shop_product sp  (cost=0.00..383744.45 rows=12293545 width=16) (actual time=0.004..11150.497 rows=10000000 loops=1)
                    Output: sp.id, sp.product_id, sp.shop_id
                    Buffers: shared hit=67 read=260742
              ->  Hash  (cost=208509.00..208509.00 rows=1000 width=8) (actual time=2303.594..2303.594 rows=111 loops=1)
                    Output: p.id
                    Buckets: 1024  Batches: 1  Memory Usage: 13kB
                    Buffers: shared hit=8578 read=74931
                    ->  Seq Scan on public.product p  (cost=0.00..208509.00 rows=1000 width=8) (actual time=12.449..2303.464 rows=111 loops=1)
                          Output: p.id
                          Filter: (p.product_name ~~ '%₽50000%'::text)
                          Rows Removed by Filter: 9999889
                          Buffers: shared hit=8578 read=74931
        ->  Index Scan using pk_shop on public.shop s  (cost=0.29..0.30 rows=1 width=30) (actual time=0.004..0.005 rows=1 loops=111)
              Output: s.id, s.shop_name
              Index Cond: (s.id = sp.shop_id)
              Buffers: shared hit=334
Planning time: 0.425 ms
Execution time: 23085.469 ms
```

Пример работы с массивами

```
select s.shop_name
from shop_mass s
where s.product_id && (array(select id from product p where p.product_name like '%P50000%'))
order by s.shop_name
```



вывода

вод данных | Построить план выполнения | Сообщения | История

```
QUERY PLAN
text
Sort  (cost=208644.59..208644.84 rows=100 width=22) (actual time=2179.537..2179.624 rows=111 loops=1)
  Output: s.shop_name
  Sort Key: s.shop_name
  Sort Method: quicksort  Memory: 33kB
  Buffers: shared hit=9012 read=74899
  InitPlan 1 (returns $0)
    ->  Seq Scan on public.product p  (cost=0.00..208509.00 rows=1000 width=8) (actual time=12.293..2178.282 rows=111 loops=1)
          Output: p.id
          Filter: (p.product_name ~~ '%P50000%'::text)
          Rows Removed by Filter: 9999889
          Buffers: shared hit=8610 read=74899
  ->  Bitmap Heap Scan on public.shop_mass s  (cost=24.77..132.27 rows=100 width=22) (actual time=2178.901..2179.068 rows=111 loops=1)
        Output: s.shop_name
        Recheck Cond: (s.product_id && $0)
        Heap Blocks: exact=68
        Buffers: shared hit=9012 read=74899
        ->  Bitmap Index Scan on ix_shop_mass_product_id  (cost=0.00..24.75 rows=100 width=0) (actual time=2178.884..2178.884 rows=111 loops=1)
              Index Cond: (s.product_id && $0)
              Buffers: shared hit=8944 read=74899
Planning time: 0.176 ms
Execution time: 2179.760 ms
```

Пример работы с массивами

	product	**shop**	**shop_product**	**shop_mass**
Размер таблицы, Мб	652	0.5	2038	0.8
Размер индексов, Мб	688	0.2	1733	415
Всего, Мб	1340	0.7	3771	416

Производительность

1. По скорости в **10 раз**
2. По размеру в **9 раз**

NoSQL данные в PostgreSQL

- **JSON и JSONB**

```
--создаем табличку
create table test_json
(
  id bigserial,
  data_json jsonb,
  constraint pk_test_json primary key(id)
);

--заночим данные
insert into test_json(data_json)
values ('{}'),
       ('{"a" : 1 }'),
       ('{"a" : 2, "b" : ["c", "d"] }'),
       ('{"a" : 1, "b" : {"c": "d", "e": true} }'),
       ('{"b": 2}');
```

```
select * from test_json where data_json = '{"a":1}';

id    |data_json
bigint|jsonb
------|-------------
   2  | {"a": 1}
      |
```

Равенство

```
select * from test_json where data_json @> '{"a":1}';

id    |data_json
bigint|jsonb
------|-------------
   2  | {"a": 1}
   4  | {"a": 1, "b": {"c": "d", "e": true}}
```

Ограничения

```
select * from test_json where data_json ?| array['a', 'b']

id    |data_json
bigint|jsonb
------|-------------
   2  | {"a": 1}
   3  | {"a": 2, "b": ["c", "d"]}
   4  | {"a": 1, "b": {"c": "d", "e": true}}
   5  | {"b": 2}
```

Фильтрация по ключам

Фильтрация по ключам и значениям

```
select * from test_json where data_json ->> 'a' > '1';

id    |data_json
bigint|jsonb
------|-------------
   3  | {"a": 2, "b": ["c", "d"]}
```

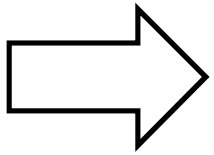
Пример работы с JSON

```
l_cnt int = 1000000;
l_json json = '[
       {"first name":"Sergey","last name":"Olontsev","age":32,"skills":["SQL Server 2016","T-SQL","JSON"]},
       {"first name":"John","last name":"Smith","sex":"m","skills":["SQL Server 2014","In-Memory OLTP"]},
       {"first name":"Mary","last name":"Brown","age":25,"skills":["SQL Server 2016","In-Memory OLTP"]}]';
l_jsonb jsonb = '[
       {"first name":"Sergey","last name":"Olontsev","age":32,"skills":["SQL Server 2016","T-SQL","JSON"]},
       {"first name":"John","last name":"Smith","sex":"m","skills":["SQL Server 2014","In-Memory OLTP"]},
       {"first name":"Mary","last name":"Brown","age":25,"skills":["SQL Server 2016","In-Memory OLTP"]}]';
l_xml xml = '<?xml version="1.0" encoding="UTF-8" ?>
<root><row>
<firstname>Sergey</firstname>
<lastname>Olontsev</lastname>
<age>32</age></row>
<skills>SQLServer2016</skills>
<skills>T-SQL</skills>
<skills>JSON</skills>
<row>
<firstname>John</firstname>
<lastname>Smith</lastname>
<sex>m</sex>
<skills>SQLServer2014</skills>
<skills>In-MemoryOLTP</skills>
</row>
<row>
<firstname>Mary</firstname>
<lastname>Brown</lastname>
<age>25</age>
<skills>SQLServer2016</skills>
<skills>In-MemoryOLTP</skills>
</row>
</root>';
```

Пример работы с JSON

д данных | Построить план выполнения | Сообщения | История

date_type text	test_id integer	path_expression text	returned_type text	elapsed_time_ms integer
json	1	$[0].age	int	8541
json	2	$[2].first name	varchar	8535
json	3	$[2].skills[0]	varchar	11070
jsonb	4	$[0].age	int	4125
jsonb	5	$[2].first name	varchar	4059
jsonb	6	$[2].skills[0]	varchar	7110
xml	7	(/root/row/age)[1]	int	59253
xml	8	(/root/row/@firstname)[3]	varchar	61815
xml	9	(/root[1]/row[3]/skills[1]	varchar	126357

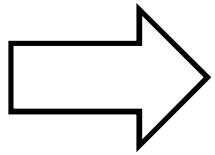


PostgreSQL

http://olontsev.ru/2016/05/sql-server-2016-json-part-6-compare-performance-with-xml/

Results | Messages

	data_type	test_id	path_expression	returned_type	elapsed_time_ms
1	json	1	$[0].age	int	11234
2	json	2	$[2]."first name"	varchar	16373
3	json	3	$[2].skills[0]	varchar	18027
4	json u	1	$[0].age	int	9580
5	json u	2	$[2]."first name"	varchar	15800
6	json u	3	$[2].skills[0]	varchar	16326
7	xml	1	(/root/rec/age)[1]	int	32613
8	xml	2	(/root/rec/@first_name)[3]	varchar	28563
9	xml	3	/root[1]/rec[3]/skills[1]/skill[1]	varchar	154300

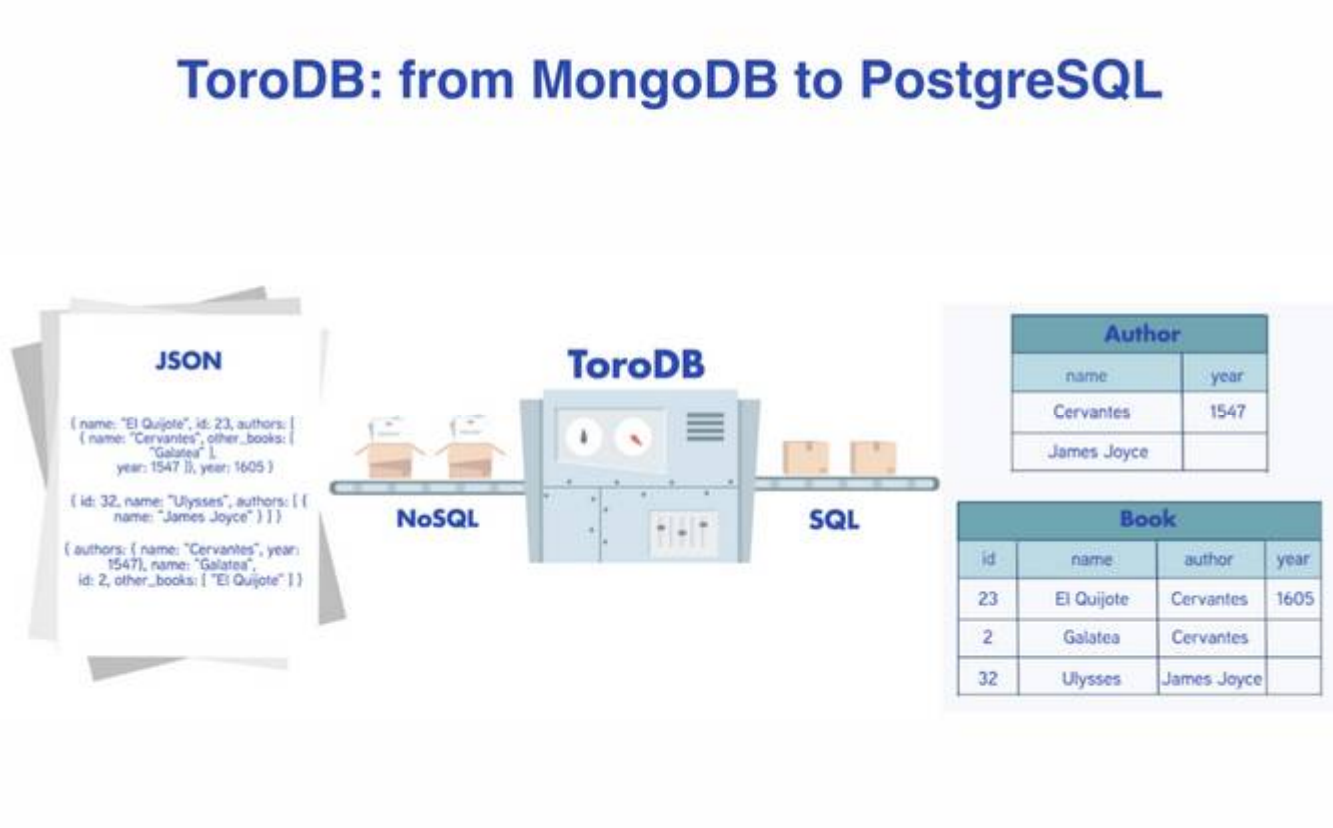


MSSQL

Как мигрировать на PostgreSQL

Миграция с MongoDB на PostgreSQL

ToroDB (www.torodb.com)



Как мигрировать на PostgreSQL

Миграция с MySQL на PostgreSQL

Mysql2postgres

https://github.com/maxlapshin/mysql2postgres

mysql_fdw

https://github.com/EnterpriseDB/mysql_fdw

Как мигрировать на PostgreSQL

Миграция с MSSQL на PostgreSQL

tds_fdw

https://github.com/tds-fdw/tds_fdw

Bcp -> psql

Остались вопросы?

Пишите: chuvashovin@gmail.com

Звоните: 8-903-954-85-18