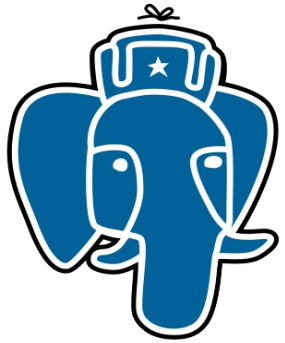


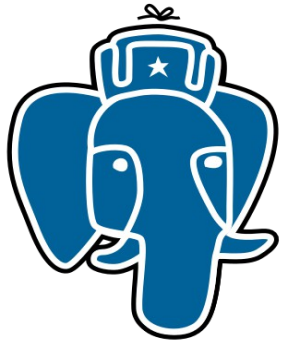
Слабо-структурированные данные в PostgreSQL и другие новости 9.4

Олег Бартунов,
ГАИШ МГУ, PostgreSQL Major Contributor

Олег Бартунов::Teodor Sigaev

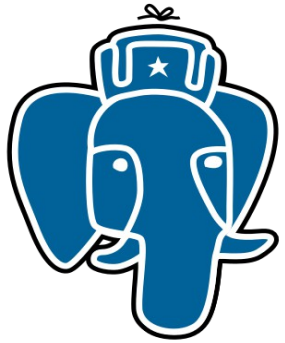
- Locale support
- Extensions:
  - intarray
  - pg_trgm
  - ltree
  - hstore, hstore v2.0 → jsonb
  - plantuner
- Full Text Search (FTS)
- Extendability (indexing)
  - GiST, GIN, SP-GiST

https://www.facebook.com/oleg.bartunov
obartunov@gmail.com



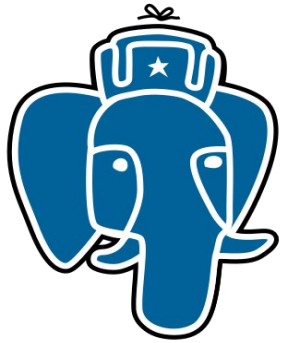
Agenda

- Слабо-структурированные данные в PostgreSQL
  - Что такое слабо-структурированные данные
  - NoSQL vs Relational
  - Hstore — key-value model
  - Json, Jsonb — document-oriented model
- Новости PostgreSQL 9.4 (осень, 2014)
  - Parallelism (Background workers)
  - View (Materialized)
  - Logical replication
  - Indexing
  - Ассорти



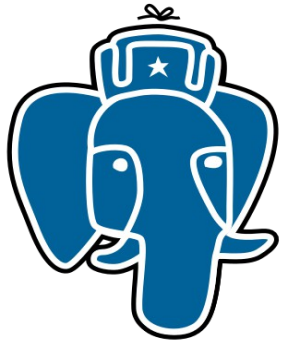
Слабо-структурированные данные

- Слабо-структурированные данные возникают от лени :)
- Агрегирование структурированных данных приводит к слабо-структурированным данным — разреженная матрица
- Все слабо-структурированные данные можно реализовать стандартными способами RDBMS
  - Неудобно, проблемы с производительностью
- json — жупел слабо-структурированных данных
- Реальная проблема — это schema-less данные
  - Реляционные СУБД трудно переживают изменение схемы
  - Key-value (NoSQL) хранилища таких проблем не имеют



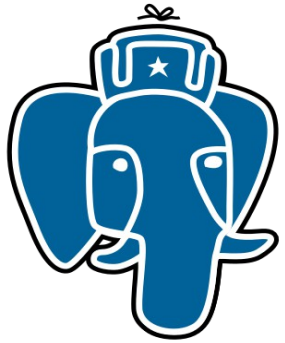
NoSQL (концептуальные предпосылки)

- Реляционные СУБД — интеграционные
  - Все приложения общаются через СУБД
  - SQL — универсальный язык работы с данными
  - Все изменения в СУБД доступны всем
  - Изменения схемы очень затратны, медл. релизы
  - Рассчитаны на интерактивную работу
    - Интересны агрегаты, а не сами данные, нужен SQL
    - SQL отслеживает транзакционность, ограничения целостности... вместо человека



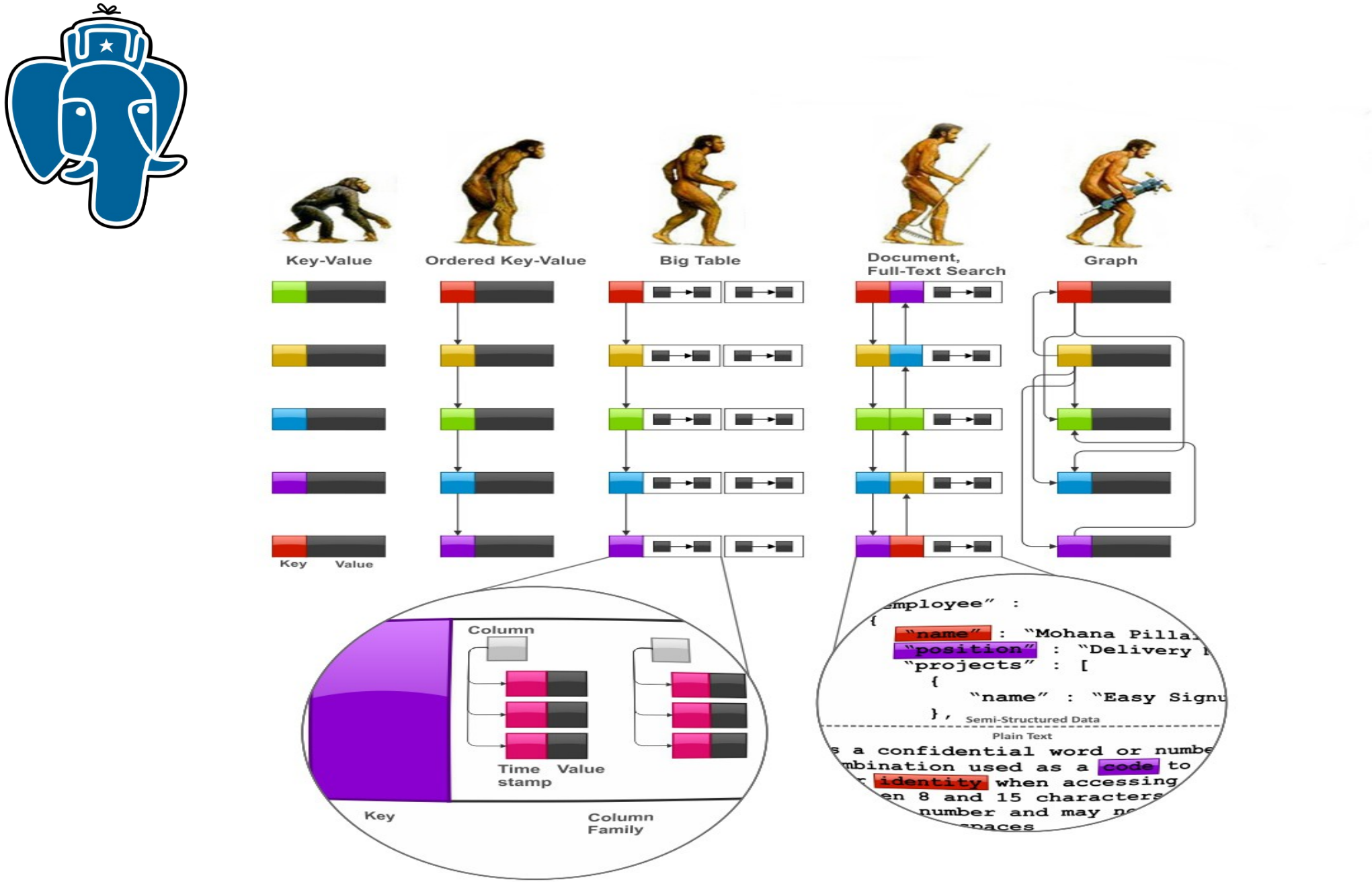
NoSQL (концептуальные предпосылки)

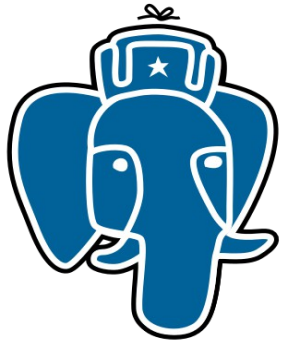
- Сервисная архитектура изменила подход к СУБД
  - Приложение состоит из сервисов, SQL->HTTP
  - Сервисам не нужна одна монолитная СУБД
  - Часто достаточно простых key-value СУБД
  - Схема меняется «на ходу», быстрые релизы
  - ACID → BASE
  - Сервисы — это программы, которые могут сами заниматься агрегированием
  - Сервисы могут сами следить за целостностью данных
- Много данных, аналитика, большое кол-во одновременных запросов
  - Распределенность - кластеры дешевых shared-nothing машин
- NoSQL —горизонтальная масштабируемость и производительность



NoSQL

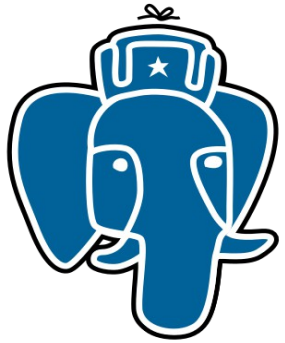
- Key-value databases
  - Ordered k-v для поддержки диапазонов
- Column family (column-oriented) stores
  - Big Table — value имеет структуру:
    - column families, columns, and timestamped versions (maps-of maps-of maps)
- Document databases
  - Value - произвольная сложность, индексы
    - Имена полей, FTS — значение полей
- Graph databases — эволюция ordered-kv


Key-Value
Ordered Key-Value
Big Table
Document,
Full-Text Search
Graph
Key
Value
Column
Time stamp
Value
Key
Column Family
"name" : "Mohana Pilla
"position" : "Delivery
"projects" : [
{
"name" : "Easy Signu
},
Semi-Structured Data
Plain Text
a confidential word or numbe
bination used as a code to
identity when accessing
en 8 and 15 characters
number and may n
paces



Челлендж !

- Полноценная работа со слабо-структурированными данными в реляционной СУБД
  - Хранение (тип данных для хранение key-value данных)
  - Поиск (операторы и функции)
  - Производительность (бинарное хранилище, индексы)



Introduction to hstore

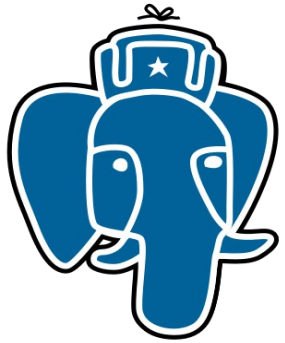
- Hstore — key/value storage (inspired by perl hash)

```
'a=>1, b=>2'::hstore
```

  - Key, value — strings
  - Get value for a key: hstore -> text
  - Operators with indexing support (GiST, GIN)

    Check for key: hstore ? text
    Contains: hstore @> hstore

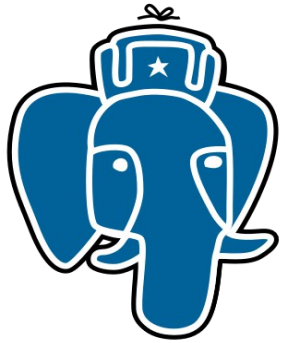
  - check documentations for more
  - Functions for hstore manipulations (akeys, avals, skeys, svals, each,.....)



History of hstore development

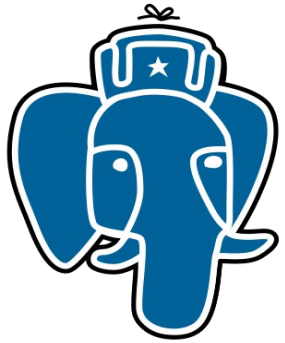
- May 16, 2003 — first version of hstore

```
Date: Fri, 16 May 2003 22:56:14 +0400
From: Teodor Sigaev <teodor@sigaev.ru>
To: Oleg Bartunov <oleg@sai.msu.su>, Alexey Slynko <slynko@tronet.ru>
Cc: E.Rodichev <er@sai.msu.su>
Subject: hash type (hstore)

Готова первайа версия:
zeus:~teodor/hstore.tgz

README написать не успел, поэтому здесь:
1 i/o типа hstore
2 операция hstore->text - извлечение значения по ключу text
select 'a=>q, b=>g'->'a';
 ?
------
 q

3 isexists(hstore), isdefined(hstore), delete(hstore,text) - полный перловый аналог
4 hstore || hstore - конкатенация, аналог в перле %a=( %b, %c );
5 text=>text - возвращает hstore
select 'a'=>'b';
 ?column?
----------
 "a"=>"b"


Все примеры есть в sql/hstore.sql
```



Introduction to hstore

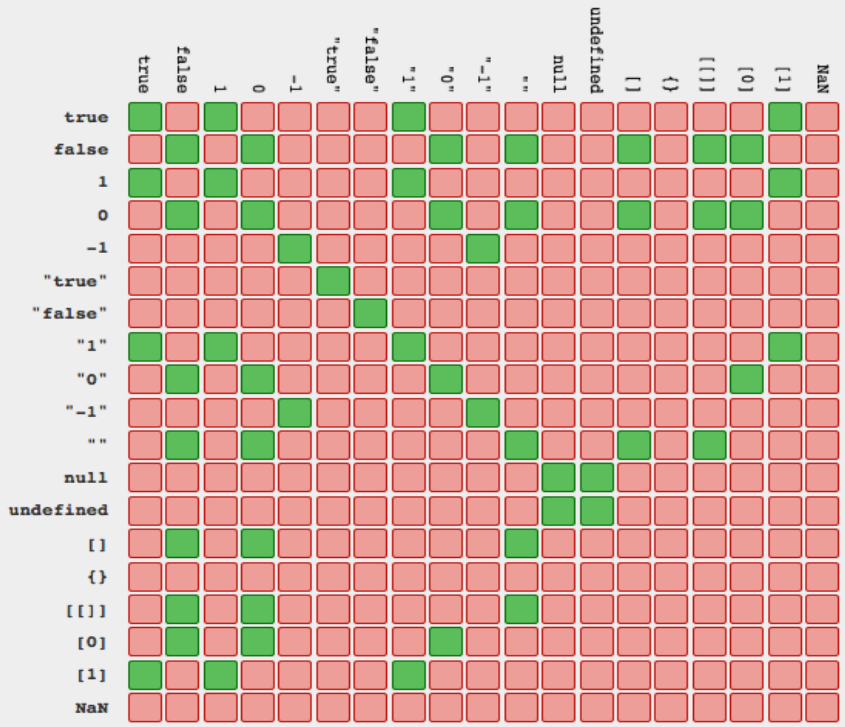
- Hstore benefits
  - In provides a flexible model for storing a semi-structured data in relational database
  - hstore has binary storage
- Hstore drawbacks
  - Too simple model !
    Hstore key-value model doesn't supports tree-like structures as json (introduced in 2006, 3 years after hstore)
- Json — popular and standartized (ECMA-404 The JSON Data Interchange Standard, JSON RFC-7159)
- Json — PostgreSQL 9.2, textual storage



Непростой Json :)

== (negated: !=)

When using two equals signs for JavaScript equality testing, some funky conversions

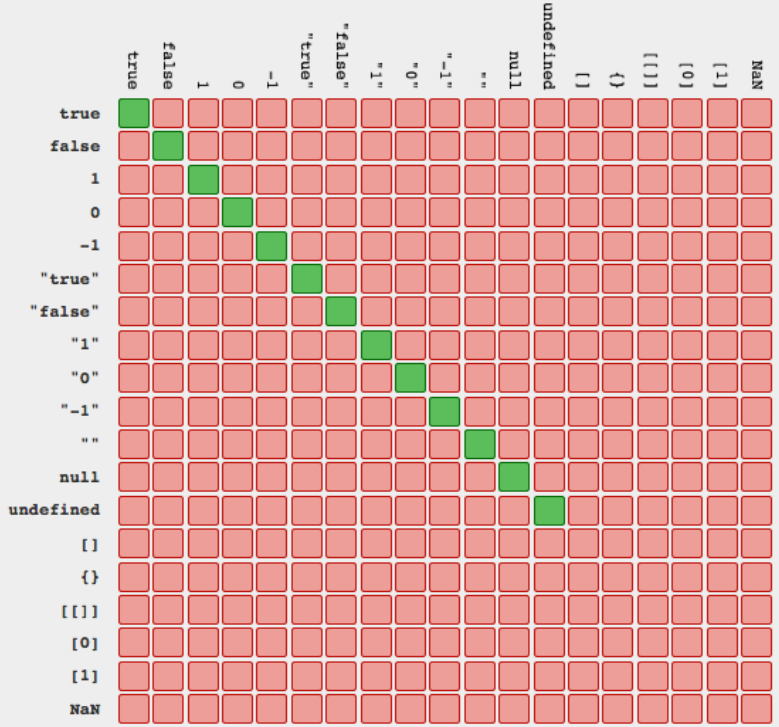


Moral of the story:

Use three equals unless you fully understand the conversions that take place for two

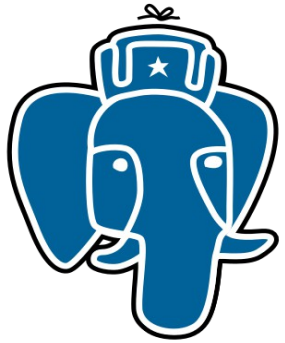
=== (negated: !==)

When using three equals signs for JavaScript equality testing, everything is as is. Nothing gets converted before being evaluated.



Moral of the story:

Use three equals unless you fully understand the conversions that take place for two-equals.

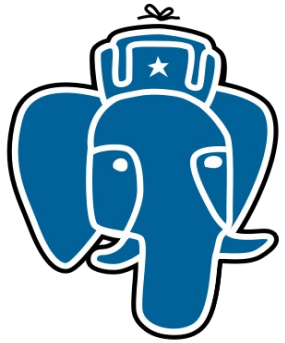


Hstore vs Json

- hstore явно быстрее json даже на простых данных

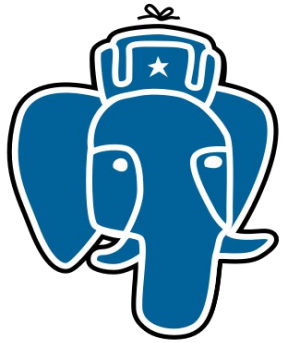
```
CREATE TABLE hstore_test AS (SELECT
'a=>1, b=>2, c=>3, d=>4, e=>5'::hstore AS v
FROM generate_series(1,1000000));

CREATE TABLE json_test AS (SELECT
'{"a":1, "b":2, "c":3, "d":4, "e":5}'::json AS v
FROM generate_series(1,1000000));
```

```
SELECT sum((v->'a')::text::int) FROM json_test;
851.012 ms

SELECT sum((v->'a')::int) FROM hstore_test;
330.027 ms
```



Hstore vs Json

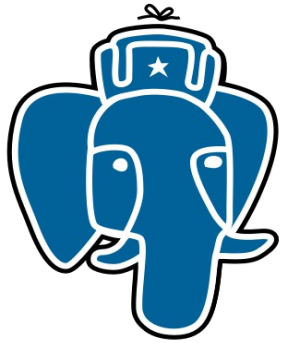
- PostgreSQL already has json since 9.2, which supports document-based model, but
  - It's slow, since it has no binary representation and needs to be parsed every time
  - Hstore is fast, thanks to binary representation and index support
  - It's possible to convert hstore to json and vice versa, but current hstore is limited to key-value
  - Need hstore with document-based model. Share it's binary representation with json !



History of hstore development

- May 16, 2003 - first (unpublished) version of hstore for PostgreSQL 7.3
- Dec, 05, 2006 - hstore is a part of PostgreSQL 8.2

  (thanks, Hubert Depesz Lubaczewski!)
- May 23, 2007 - GIN index for hstore, PostgreSQL 8.3
- Sep, 20, 2010 - Andrew Gierth improved hstore, PostgreSQL 9.0



Nested hstore

abstract

Oleg Bartunov <obartunov@gmail.com> 12/18/12

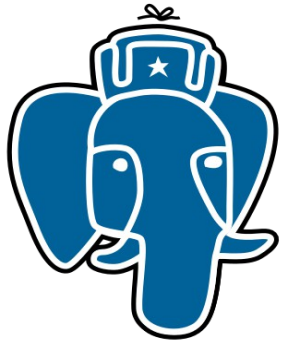
to Teodor

Поправь, дополни.

Title: One step forward true json data type. Nested hstore with array support.

We present a prototype of nested hstore data type with array support. We consider the new hstore as a step forward true json data type.

Recently, PostgreSQL got json data type, which basically is a string storage with validity checking for stored values and some related functions. To be a real data type, it has to have a binary representation, which could be a big project if started from scratch. Hstore is a popular data type, we developed years ago to facilitate working with semi-structured data in PostgreSQL. Our idea is to extend hstore to be nested (value can be hstore) data type and add support of arrays, so its binary representation can be shared with json. We present a working prototype of a new hstore data type and discuss some design and implementation issues.

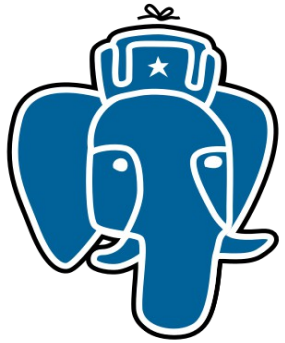


Nested hstore & jsonb

- Nested hstore был представлен на PGCon-2013, Оттава, Канада (24 мая) — спасибо Engine Yard за поддержку !

  One step forward true json data type.Nested hstore with arrays support

- Бинарное хранилище для вложенных структур было представлено на PGCon Europe — 2013, Дублин, Ирландия (29 октября)
  Binary storage for nested data structuresand application to hstore data type
- В ноябре бинарное хранилище было стандартизовано
  - nested hstore и jsonb — просто разные интерфейсы доступа к нему
- В начале января Andrew Dunstan начинает активно работать по jsonb
  - бинарное хранилище и основной функционал перемещается в ядро постгреса

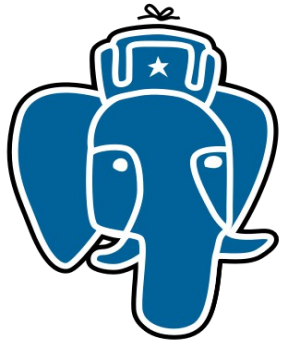


Nested hstore & jsonb

- В феврале-марте подключается Peter Geoghegan, мы принимаем решение оставить hstore как есть, чтобы избежать проблем с совместимостью
- 23 марта Andrew Dunstan закомитил jsonb в ветку 9.4 !
  pgsql: Introduce jsonb, a structured format for storing json.

  Introduce jsonb, a structured format for storing json.

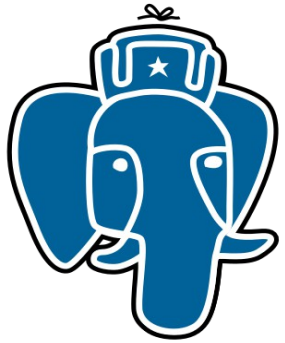
  The new format accepts exactly the same data as the json type. However, it is stored in a format that does not require reparsing the orgiginal text in order to process it, making it much more suitable for indexing and other operations. Insignificant whitespace is discarded, and the order of object keys is not preserved. Neither are duplicate object keys kept - the later value for a given key is the only one stored.



Jsonb vs Json

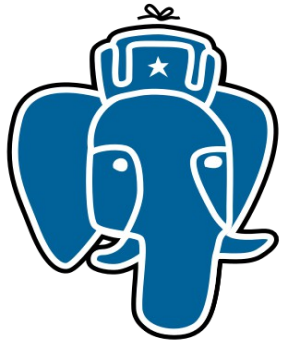
```
select '{"c":0,  "a":2,"a":1}'::json, '{"c":0,   "a":2,"a":1}'::jsonb;
         json           |      jsonb
------------------------+------------------
 {"c":0,  "a":2,"a":1} | {"a": 1, "c": 0}
(1 row)
```

- json: текстовое хранение «as is»
- jsonb: все пробелы (whitespace) убираются
- jsonb: дубликаты убираются, побеждает последний
- jsonb: все ключи сортируются



Jsonb vs Json

- Data
  - 1,252,973 bookmarks from Delicious in json format
  - The same bookmarks in jsonb format
  - The same bookmarks as text
- Server
  - MBA, 8 GB RAM, 256 GB SSD
- Test
  - Input performance - copy data to table
  - Access performance - get value by key
  - Search performance contains @> operator

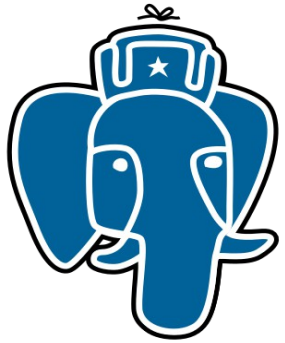


Jsonb vs Json

```
select count(js->>'title') from js;
  count
---------
 1252973
(1 row)

Time: 9215.143 ms
select count(jb->>'title') from jb;
  count
---------
 1252973
(1 row)

Time: 977.860 ms
```

```
select count(js->'tags'->1->'term') from js;
 count
--------
 796792
(1 row)

Time: 12352.468 ms
 select count(jb->'tags'->1->'term') from jb;
 count
--------
 796792
(1 row)

Time: 1080.460 ms
```

Jsonb в 10 раз быстрее Json !

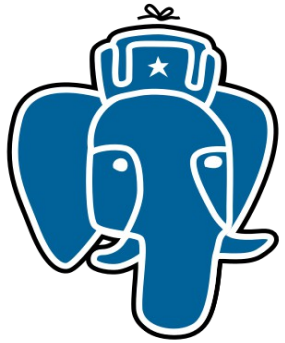


Jsonb vs Json

- Data
  - 1,252,973 bookmarks from Delicious in json format
  - The same bookmarks in jsonb format
  - The same bookmarks as text

```
=# \dt+
                    List of relations
 Schema | Name | Type  |  Owner   |  Size   | Description
--------+------+-------+----------+---------+-------------
 public | jb   | table | postgres | 1374 MB |
 public | js   | table | postgres | 1322 MB |
 public | tx   | table | postgres | 1322 MB |
```

оверхед бинарного хранилища < 4%



Jsonb vs Json

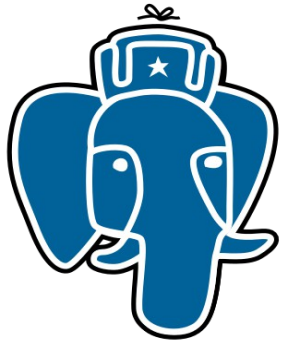
- Input performance (оверхед парсера)
  - Copy data (1,252,973 rows) as text, json,hstore

```
copy tt from  '/path/to/test.dump'
```

Text: 34 s

Json: 37 s

Jsonb: 43 s



Jsonb vs Json

- Access performance — get value by key
  - Base: select h from hs;
  - Jsonb: select j->>'updated' from jb;;
  - Json: select j->>'updated' from js;
  - Regexp: select (regexp_matches(j, '"updated":"([^"]*)"'))[1] from tx;

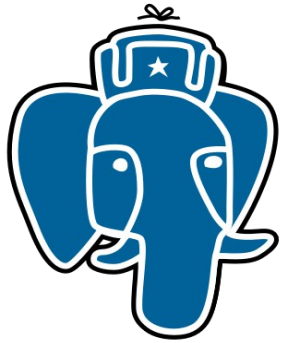
Base: 0.6 s

Jsonb: 1 s

Json: 9.6 s

regexp: 12.8 s

Jsonb 40X быстрее Json !
Спасибо, бинарное хранение

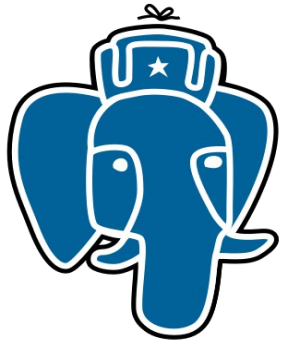


Jsonb vs Json

```
explain analyze select count(*) from js where js #>>'{tags,0,term}' = 'NYC';
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=187812.38..187812.39 rows=1 width=0)
(actual time=10054.602..10054.602 rows=1 loops=1)
   ->  Seq Scan on js  (cost=0.00..187796.88 rows=6201 width=0)
(actual time=0.030..10054.426 rows=123 loops=1)
         Filter: ((js #>> '{tags,0,term}'::text[]) = 'NYC'::text)
         Rows Removed by Filter: 1252850
 Planning time: 0.078 ms
 Total runtime: 10054.635 ms
(6 rows)
```

Json: нет оператора contains @>,

используем нижнюю оценку

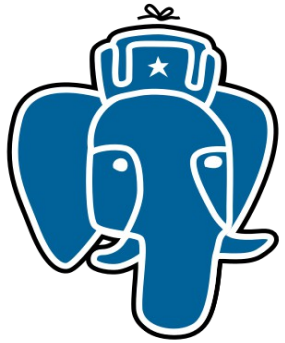


Jsonb vs Json

```
explain analyze select count(*) from jb where jb @> '{"tags":[{"term":"NYC"}]}'::jsonb;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=191521.30..191521.31 rows=1 width=0)
(actual time=1263.201..1263.201 rows=1 loops=1)
   ->  Seq Scan on jb  (cost=0.00..191518.16 rows=1253 width=0)
 (actual time=0.007..1263.065 rows=285 loops=1)
         Filter: (jb @> '{"tags": [{"term": "NYC"}]}'::jsonb)
         Rows Removed by Filter: 1252688
 Planning time: 0.065 ms
 Total runtime: 1263.225 ms
(6 rows)
```

Jsonb: оператор contains @> !

быстрее json ~ 10 раз



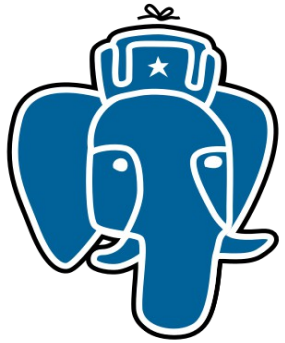
Jsonb vs Json

create index gin_jb_idx on jb using gin(jb);

```
explain analyze select count(*) from jb where jb @> '{"tags":[{"term":"NYC"}]}'::jsonb;
                                                        QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Aggregate  (cost=4772.72..4772.73 rows=1 width=0)
(actual time=8.486..8.486 rows=1 loops=1)
   ->  Bitmap Heap Scan on jb  (cost=73.71..4769.59 rows=1253 width=0)
(actual time=8.049..8.462 rows=285 loops=1)
         Recheck Cond: (jb @> '{"tags": [{"term": "NYC"}]}'::jsonb)
         Heap Blocks: exact=285
         ->  Bitmap Index Scan on gin_jb_idx  (cost=0.00..73.40 rows=1253 width=0)
(actual time=8.014..8.014 rows=285 loops=1)
               Index Cond: (jb @> '{"tags": [{"term": "NYC"}]}'::jsonb)
 Planning time: 0.115 ms
 Total runtime: 8.515 ms
(8 rows)
```

Jsonb: оператор contains @> !

ускоряется GIN индексом : 150X !! - keys && values

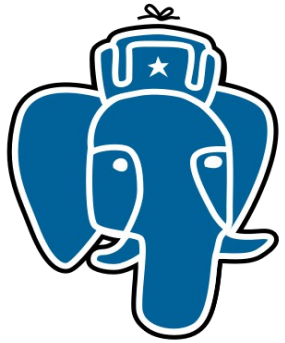


GIN hash index

- Idea: index hash(full paths to elements and values)

  {a=>{b=>{c=>1}}, d=>{1,2,3}}
  path-keys: a.b.c.1, d..1, d..2,d..3

  GIN: {hash(path-key)}



Jsonb vs Json

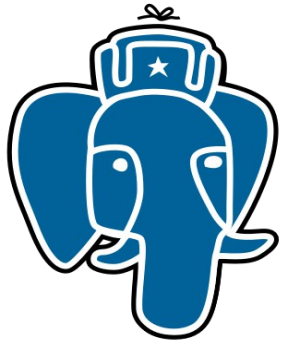
create index gin_jb_hash_idx on jb using gin(jb jsonb_hash_ops);

```
explain analyze select count(*) from jb where jb @> '{"tags":[{"term":"NYC"}]}'::jsonb;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=4732.72..4732.73 rows=1 width=0)
(actual time=0.644..0.644 rows=1 loops=1)
   ->  Bitmap Heap Scan on jb  (cost=33.71..4729.59 rows=1253 width=0)
(actual time=0.102..0.620 rows=285 loops=1)
         Recheck Cond: (jb @> '{"tags": [{"term": "NYC"}]}'::jsonb)
         Heap Blocks: exact=285
         ->  Bitmap Index Scan on gin_jb_hash_idx
(cost=0.00..33.40 rows=1253 width=0) (actual time=0.062..0.062 rows=285 loops=1)
               Index Cond: (jb @> '{"tags": [{"term": "NYC"}]}'::jsonb)
 Planning time: 0.056 ms
 Total runtime: 0.668 ms
(8 rows)
```

Jsonb: оператор contains @> !

ускоряется GIN индексом : 150X !! - keys && values

ускоряется GIN++ индексом: 1800X !!! - hash key.value



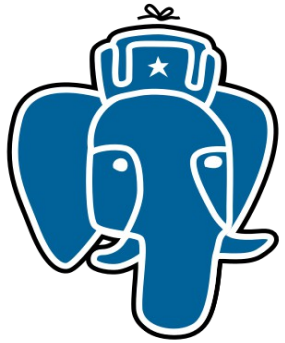
MongoDB 2.4.7

- Load data - ~8 min **SLOW !** **Jsonb 43 s**

```
mongoimport --host localhost -c js --type json < delicious-rss-1250k
Mon Oct 28 19:16:47.025                7400    2466/second
...
Mon Oct 28 19:24:38.030                1250800 2638/second
Mon Oct 28 19:24:38.902 check 9 1252973
Mon Oct 28 19:24:38.902 imported 1252973 objects
```

- Search - ~ 1s (seqscan) **THE SAME**

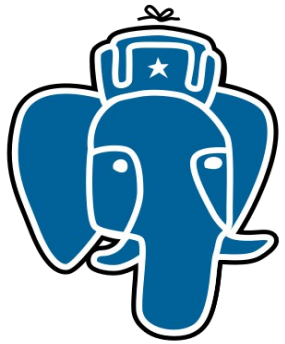
```
db.js.find({tags: {$elemMatch:{ term: "NYC"}}}).count()
285
-- 980 ms
```



MongoDB 2.4.7

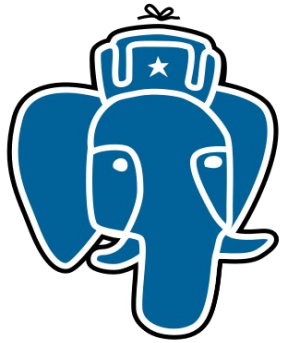
- Search — 1ms (index)

```
db.js.ensureIndex( {"tags.term" : 1} )
db.js.find({tags: {$elemMatch:{ term: "NYC"}}}).explain()
{
        "cursor" : "BtreeCursor tags.term_1",
        "isMultiKey" : true,
        "n" : 285,
        "nscannedObjects" : 285,
        "nscanned" : 285,
        "nscannedObjectsAllPlans" : 285,
  ..............................
        "millis" : 1,
        "indexBounds" : {
                "tags.term" : [
                        [
                                "NYC",
                                "NYC"
                        ]
                ]
```



Summary

- Оператор contains @>
  - json : 10 s seqscan
  - jsonb : 8.5 ms GIN
  - **jsonb : 0.7 ms GIN HASH opclass**
  - mongodb : 1.0 ms btree index

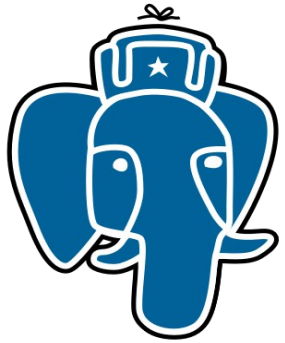


Jsonb

- Документация
  - JSON Types
  - JSON Functions and Operators
- Осталось портировать много функциональности из nested hstore
  - Это можно сделать расширением
- Очень большая работа над структурными запросами
  - «Хочется купить что-нибудь красное» - women oriented query
  - Проект VODKA — новый метод доступа вместо GIN

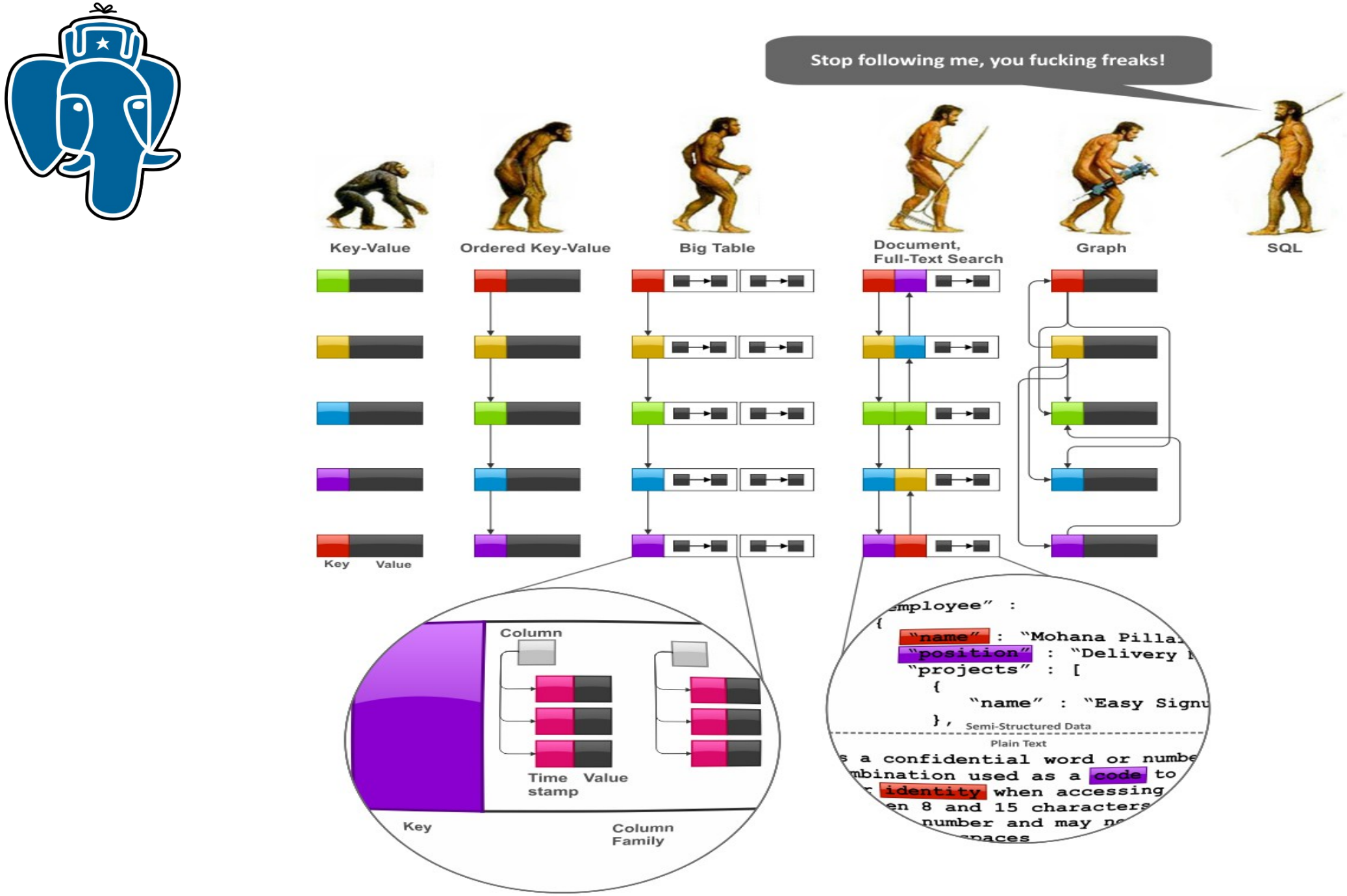
    Придумайте хорошую расшифровку для VODKA

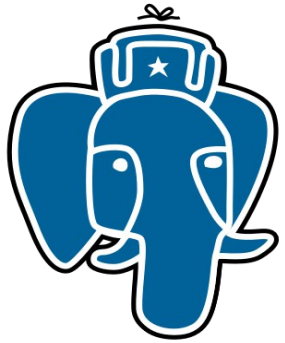
    CREATE INDEX … USING VODKA !



NoSQL vs Relational

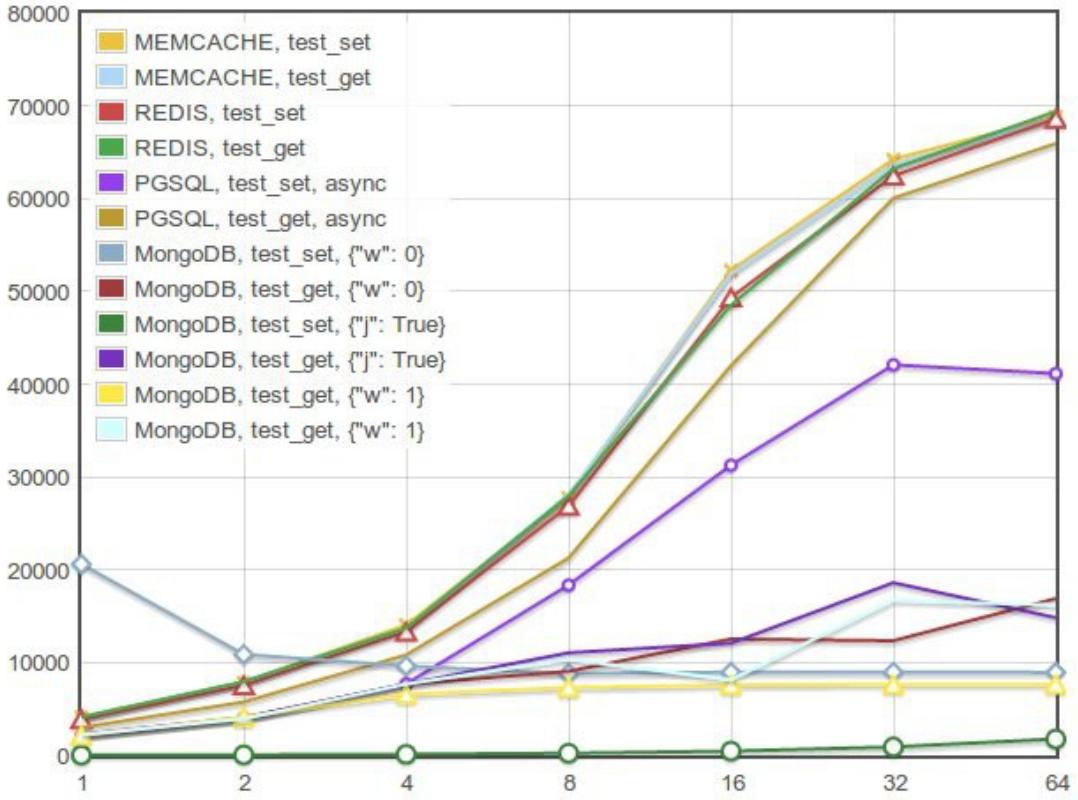
- PostgreSQL 9.4 — открытая реляционная СУБД с сильной поддержкой слабо-структурированных данных
  - Все плюсы реляционной модели
  - нормальный json с бинарным хранением и индексами
  - производительность не хуже MongoDB
  - Зачем использовать NoSQL !?
    - 0.1% проектов действительно нуждаются в NoSQL масштабируемости
    - NoSQL хорош для хранения несущественных данных — cache


Stop following me, you fucking freaks!
Key-Value
Ordered Key-Value
Big Table
Document,
Full-Text Search
Graph
SQL
Key
Value
Column
Time stamp
Value
Key
Column Family
"name" : "Mohana Pilla
"position" : "Delivery
"projects" : [
{
"name" : "Easy Signu
},
Semi-Structured Data
Plain Text
a confidential word or numbe
mbination used as a code to
identity when accessing
en 8 and 15 characters
number and may n
paces



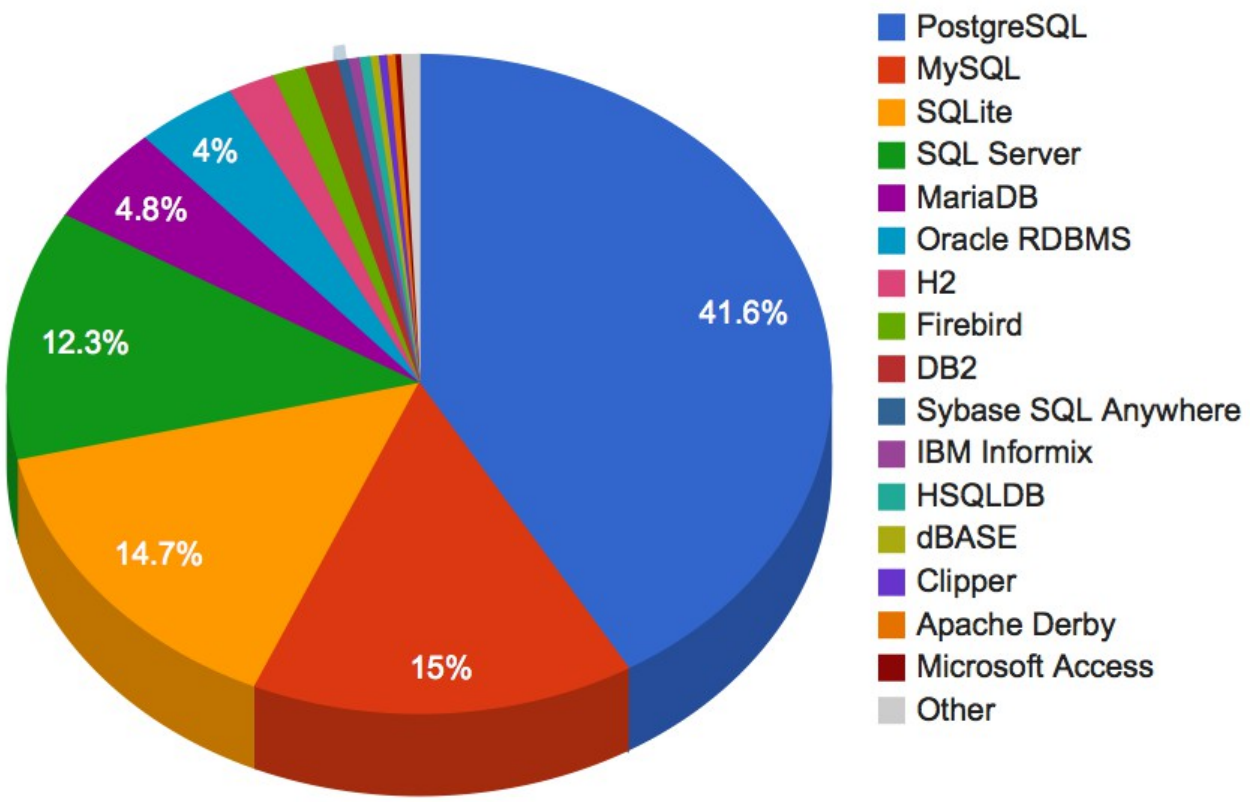
NoSQL vs Relatonal

4-core HT+ server, 1 client with 1..64 python scripts
async → synchronous_commit = of, json documens
data fits in memory

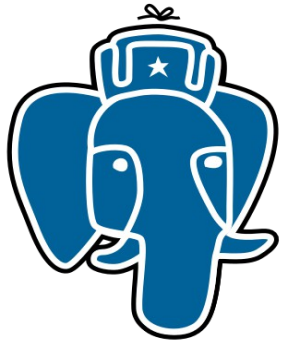


1 Server with 1 to 64 clients, Client(s) and server on separate host
minimum data size: 1188, max size: 2601, average size: 1874

PostgreSQL popularity - 2014



http://www.databasefriends.co/2014/03/favorite-relational-database.html

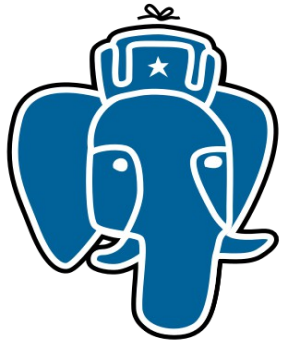


PostgreSQL popularity - 2014

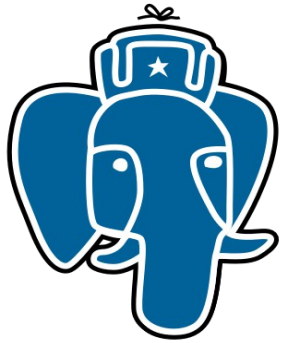
216 systems in ranking, March 2014

Rank	Last Month	DBMS	Database Model	Score	Changes
1.	1.	**Oracle**	**Relational DBMS**	**1491.80**	**-8.43**
2.	2.	**MySQL**	**Relational DBMS**	**1290.21**	**+1.83**
3.	3.	**Microsoft SQL Server**	**Relational DBMS**	**1205.28**	**-8.99**
4.	4.	**PostgreSQL**	**Relational DBMS**	**235.06**	**+4.61**
5.	5.	**MongoDB**	**Document store**	**199.99**	**+4.81**
6.	6.	**DB2**	**Relational DBMS**	**187.32**	**-1.14**
7.	7.	**Microsoft Access**	**Relational DBMS**	**146.48**	**-6.40**
8.	8.	**SQLite**	**Relational DBMS**	**92.98**	**-0.03**
9.	9.	**Sybase ASE**	**Relational DBMS**	**81.55**	**-6.33**
10.	10.	**Cassandra**	**Wide column store**	**78.09**	**-2.23**
11.	11.	Teradata	Relational DBMS	62.63	-1.18
12.	12.	Solr	Search engine	61.14	-1.56
13.	13.	Redis	Key-value store	53.46	-2.36
14.	14.	FileMaker	Relational DBMS	52.91	+1.01
15.	15.	Informix	Relational DBMS	37.20	+1.52
16.	16.	HBase	Wide column store	35.14	-0.01
17.	17.	Memcached	Key-value store	32.90	-1.83
18.	18.	Hive	Relational DBMS	30.21	+2.29
19.	↑ 20.	Elasticsearch	Search engine	26.17	+2.79
20.	↓ 19.	CouchDB	Document store	22.86	-0.57
21.	21.	Splunk	Search engine	22.43	+1.82
22.	22.	Neo4j	Graph DBMS	18.77	+0.83
23.	23.	Firebird	Relational DBMS	16.66	-1.05

http://db-engines.com/en/ranking



Другие новости PostgreSQL 9.4

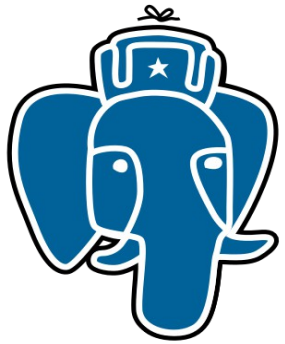


9.4: Development schedule

- June 14, 2013 - branch 9.3
- June 2013 - CF1
- September 2013 - CF2
- November 2013 - CF3
- January 2014 — CF4

  Начался 14 января ----> продолжается

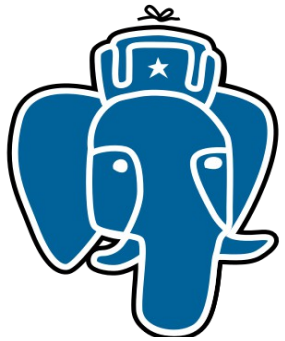
https://commitfest.postgresql.org/



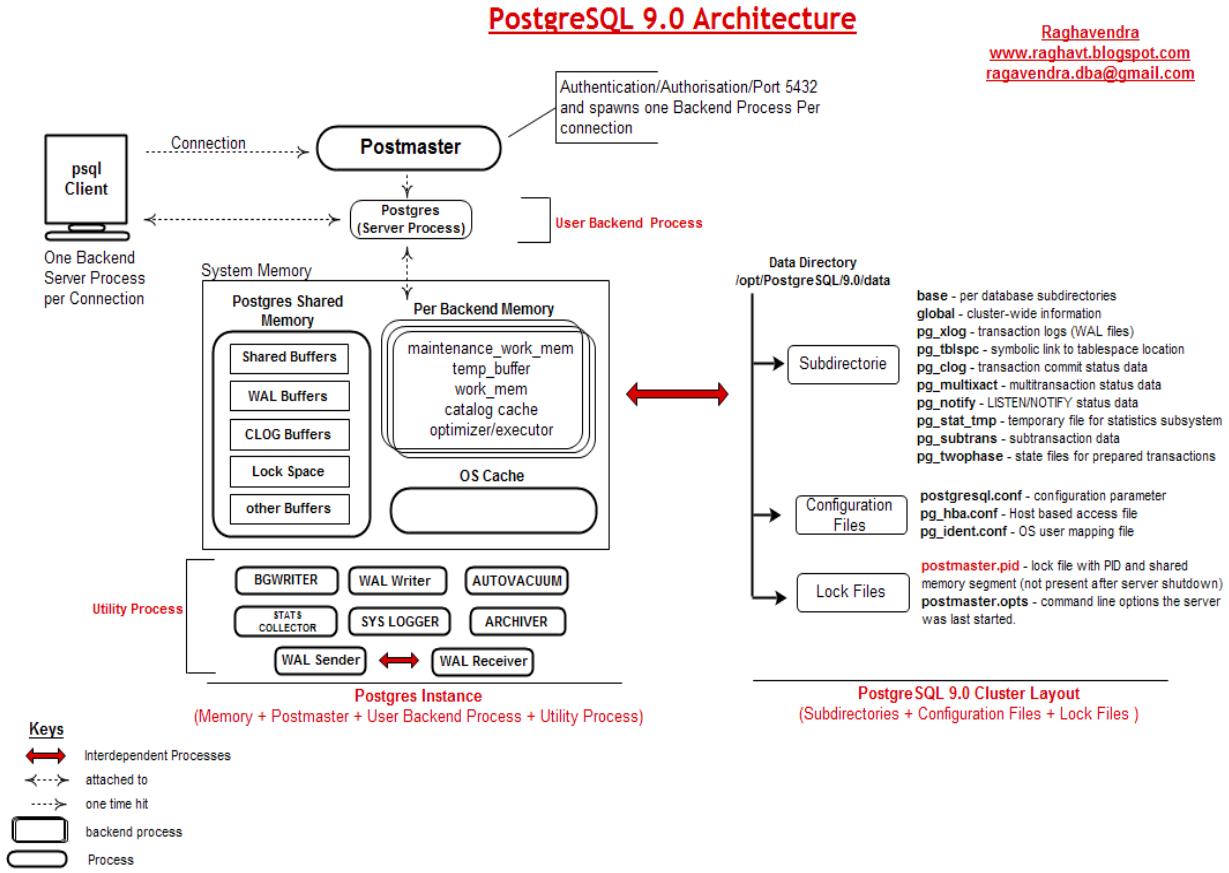
9.4: Parallelism in PostgreSQL

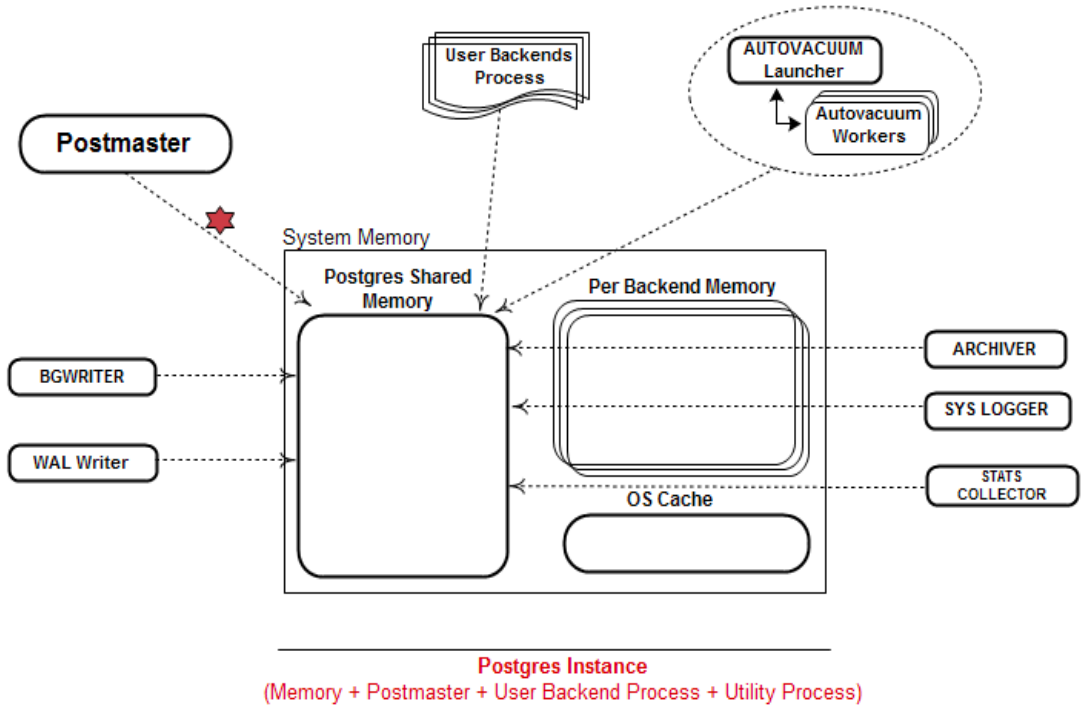
- «Implementing Parallelism in PostgreSQL. Where We Are Today, and What's On The Horizon».
  https://www.pgcon.org/2014/schedule/events/693.en.html

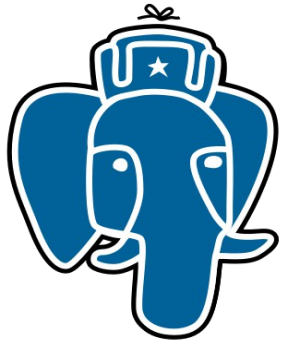
PostgreSQL's architecture is based heavily on the idea that each connection
is served by a single backend process, but CPU core counts are rising much f
aster than CPU speeds, and large data sets can't be efficiently processed serially.
Adding parallelism to PostgreSQL requires significant architectural changes to many areas of the system,
including background workers, shared memory, memory allocation, locking,
GUC, transactions, snapshots, and more.



9.4: Dynamic background workers

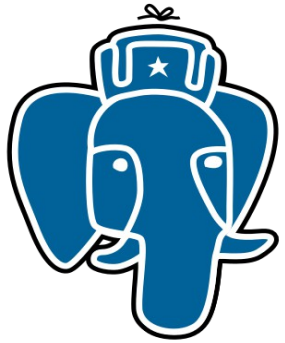






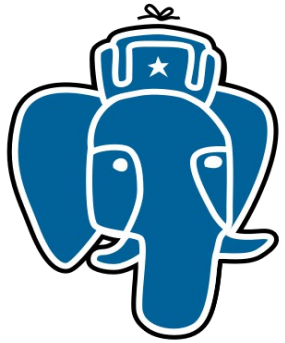
9.4: Dynamic background workers

- 9.3: Можно было регистрировать фоновые процессы
  - Демон, запускался при старте постмастера, и «умирал» вместе с ним.
  - Нельзя использовать для параллелизма
- Параллельное исполнение запроса, например, ORDER BY, может использовать разные ядра для работы воркеров, которые должны уметь запускаться динамически родительским бэкендом. Все эти процессы должны уметь обмениваться туплами, транзакционными снэпшотами, планами выполнения и прочей информацией. Также, для своей работы процессы должны уметь аллоцировать дополнительные сегменты разделяемой памяти (shared memory).



9.4: Dynamic background workers

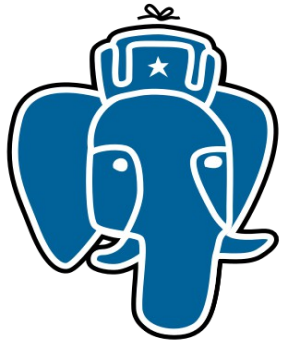
- 9.4: Allow background workers to be started dynamically.
  - Обычный пользовательский бэкенд может запустить фоновый процесс
  - Этот процесс может закончиться в любое время
  - пример: contrib/worker_spi
- 9.4: Allow dynamic allocation of shared memory segments.
  - Важно уметь брать у ОС память на время и освобождать после использования. Shared memory имеет фиксированный размер и ее неудобно использовать для выполнения запроса.
- 9.4: Shared Memory Message Queues
  - Кольцевой буфер в shared memory для обмена сообщениями(строка байтов произвольной длины) между бэкендами и фоновыми процессами.



9.4: Снято ограничение 256 Gb

- Allow using huge TLB pages on Linux (MAP_HUGETLB)
  - Память дешевеет, сервера с 1Tb памяти уже не редкость. Есть проекты, которые предполагают использование памяти
    «In-Memory Columnar Store extension for PostgreSQL», Константин Книжник
    https://www.pgcon.org/2014/schedule/events/643.en.html
    «IMCS is In-Memory Columnar Store for PostgreSQL. Vertical data model is more efficient for analytic queries performing operations on entire column. IMCS provides 10-100 times improvement in performance comparing with standard SQL queries «
  - Linux использует 4 Kb страницы памяти → ограничение 256 Gb
  - “Transaction Lookaside Buffers” (TLB), Huge Page Size — 2048 Kb !
    Только Linux.

    Cat /proc/meminfo | grep Hugepagesize

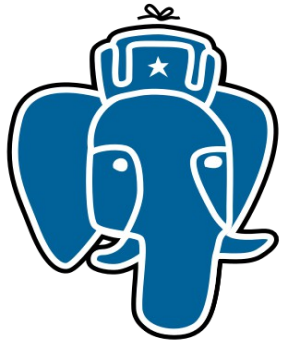


9.4: View (1)

- 9.4: CREATE VIEW .... **WITH [CASCADED | LOCAL) CHECK OPTION**
  - нельзя вставить данные (в auto-updatable VIEW), которые нельзя получить через VIEW

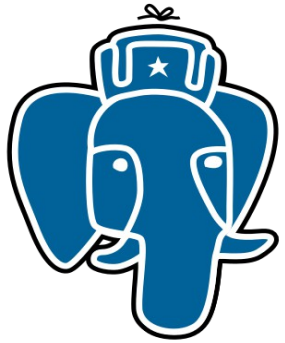
```
CREATE TABLE t1 (i integer);
INSERT INTO t1 VALUES(1),(-1);
CREATE VIEW vt1 AS SELECT * FROM t1 WHERE i > 0;
INSERT INTO vt1 VALUES(5),(-5);
=# SELECT * FROM t1;
 i
----
  1
 -1
  5
 -5
(4 rows)
=# SELECT * FROM vt1;
 i
---
 1
 5
(2 rows)
```

```
CREATE TABLE t1 (i integer);
INSERT INTO t1 VALUES(1),(-1);
CREATE VIEW vt1 AS SELECT * FROM t1 WHERE i > 0
            WITH CHECK OPTION;
INSERT INTO vt1 VALUES (5);
INSERT INTO vt1 VALUES (-5);
ERROR:  new row violates WITH CHECK OPTION for
view "vt1"
DETAIL:  Failing row contains (-5).
=# SELECT * FROM t1;
 i
----
  1
 -1
  5
(3 rows)
```



9.4: View (2)

- WITH CHECK OPTION
- LOCAL
  - проверяются условия только текущего VIEW
- CASCADE
  - Рекурсивно проверяются условия всех родительских VIEW
  - Используется по-умолчанию



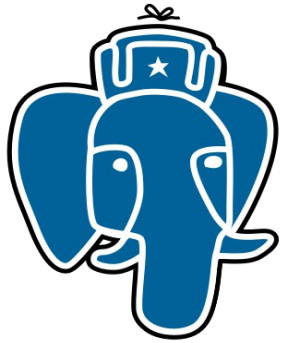
9.4: View (3)

- Allow only some columns of a view to be auto-updateable
- 9.3: Чтобы view было auto-updateable, все поля должны ссылаться напрямую на таблицу, иначе все поля становятся необновляемыми

```
create view vvt1 as select t1.*, (select avg(i) from t1) from t1;
SELECT column_name, is_updatable FROM information_schema.columns WHERE table_name = 'vvt1';
 column_name | is_updatable
-------------+--------------
 i           | NO
 avg         | NO
```

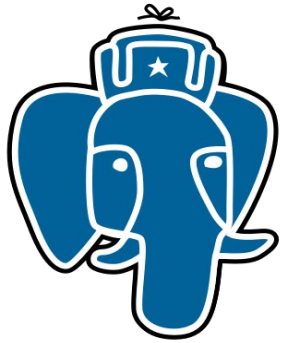
- 9.4: нет нужды писать тригеры :)

```
 SELECT column_name, is_updatable FROM information_schema.columns WHERE table_name = 'vvt1';
  column_name | is_updatable
 -------------+--------------
  i           | YES
  avg         | NO
```



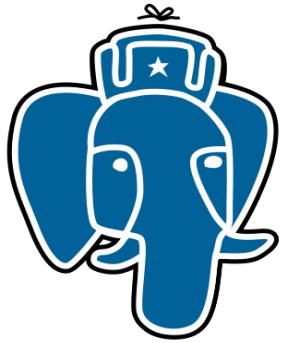
9.4: Materialized View

- 9.3: CREATE (ALTER, DROP, REFRESH) MATERIALIZED VIEW
  - не было автоматического обновления (кроме simple view)
  - при обновлении бралась блокировка (AccessExclusiveLock lock) таблицы
  - можно было вставить произвольные данные
  - «not quite usable»
- 9.4: REFRESH MATERIALIZED VIEW **CONCURRENTLY**
  - одновременное чтение и обновление (идет в фоне и более медленное)
  - требуется уникальный индекс
    (создается временная таблица с новыми материализованными данными, а потом делается full outer join со старыми данными, что невозможно при наличии дубликатов).



9.4: Logical Replication (1)

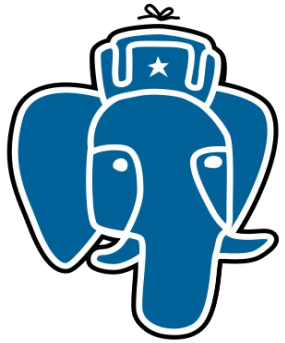
- Репликация в PostgreSQL осуществляется с помощью рассылки записей WAL
  - Физическая репликация — реплицируется весь кластер
  - Очень «дешевая», WAL все равно нужны для recovery
  - master-slave
  - Синхронная, асинхронная
- Существует множество сторонних утилит
  http://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling
  - Помимо физической репликации есть statement based:
    Slony, Bucardo, Pgpool, londiste



9.4: Logical Replication (2)

- Физическая репликация имеет ограничения (Robert Haas):
  http://rhaas.blogspot.ru/2011/02/case-for-logical-replication.html

  - You can't replication to a different major version of PostgreSQL.
  - You certainly can't replicate to a database other than PostgreSQL.
  - You can't replicate part of the database.
  - You can't write any data at all on the standby.
  - You certainly can't do multi-master replication.
  - MVCC bloat on the master propagates to the standby.
  - Anything that would bloat the standby causes query cancellations instead, or delays in recovery (or in 9.1, you'll be able to avoid the query cancellation by bloating the master).



9.4: Logical Replication (3)

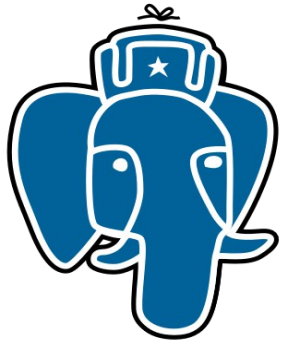
- Определить какие **туплы** (tuples, что это такое ?) изменились (inserted, updated, deleted)

  существующий хак — тригерами логировать изменения в таблицу, которая реплицируется и применять их на других узлах

  - Changeset extraction — 9.4+
  - WAL decoding — 9.4+
    изменения на физическом уровне разворачиваются в поток туплов и SQL команд, формат определяется приложением
  - postgresql.conf:

    wal_level = logical, max_replication_slots =

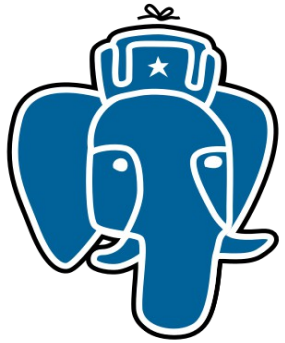
- Применить это знание к существующей копии базы
  - пока не реализовано в Core



9.4: Logical Replication (4)

- Это только первый шаг, потребуется несколько релизов
- API есть в ядре, сторонние приложения уже могут разрабатывать прототипы.
  Уже обсуждается замена триггеров на логическую репликация в SLONY -

  http://www.postgresql.org/message-id/BLU0-SMTP30D7892E90D1FF9E4EA77EDC570@phx.gbl



9.4: GIN (Generalized Inverted Index)

Report Index

A

abrasives, 27
acceleration measurement, 58
accelerometers, 5, 10, 25, 28, 30, 36, 58, 59, 61, 73, 74
actuators, 4, 37, 46, 49
adaptive Kalman filters, 60, 61
adhesion, 63, 64
adhesive bonding, 15
adsorption, 44
aerodynamics, 29
aerospace instrumentation, 61
aerospace propulsion, 52
aerospace robotics, 68
aluminium, 17
amorphous state, 67
angular velocity meas
antenna phased array
argon, 21
assembling, 22
atomic force microsco
atomic layer depositio
attitude control, 60, 6
attitude measurement
automatic test equipm
automatic testing, 24

B

backward wave oscillators, 45

compensation, 30, 68
compressive strength, 54
compressors, 29
computational fluid dynamics, 23, 29
computer games, 56
concurrent engineering, 14
contact resistance, 47, 66
convertors, 22
coplanar waveguide components, 40
Couette flow, 21
creep, 17
crystallisation, 64
current density, 13, 16

D

distributed feedback lasers, 38

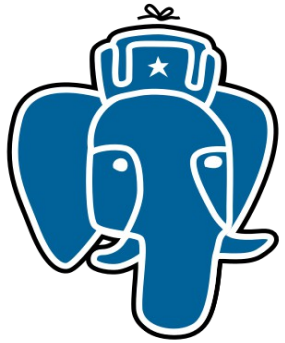
E

QUERY: compensation accelerometers

INDEX: accelerometers 5,10,25,28,**30**,36,58,59,61,73,74

compensation **30**,68

RESULT: **30**



9.4: GIN

- Posting list compression
  - 9.3 format: (4 bytes blockNumber , 2 bytes offset): 90 bytes

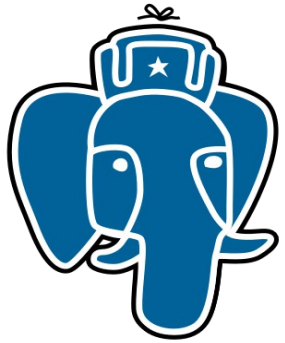
```
(0,8) (0,14) (0,17) (0,22) (0,26) (0,33) (0,34) (0,35) (0,45) (0,47)
  (0,48) (1,3) (1,4) (1,6) (1,8)
```

  - 9.4 format: compressed format, difference from previous item: 21 bytes

```
(0,8) +6 +3 +5 +4 +7 +1 +1 +10 +2 +1 +2051 +1+2 +2
```

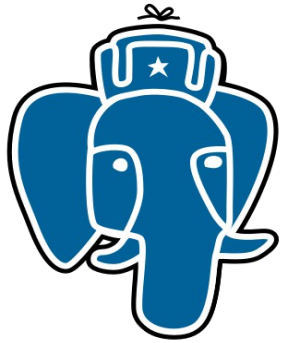
  - На 10,000,000 целых числах выигрыш существенный — 11 Mb vs 58 Mb !

```
SELECT g % 10 FROM generate_series(1,10000000) g;
```



9.4: GIN

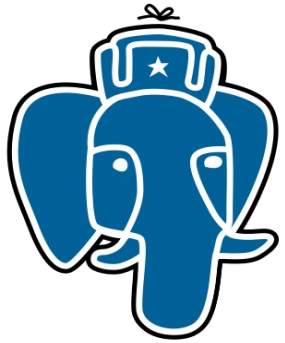
- FAST SCAN
  Оптимизация запросов rare & frequent (на самом деле сложнее)
  - 9.3: Читаем все постинг листы и потом пересекаем
    Time (frequent & rare) ~ Time(frequent)
  - 9.4: Читать начинаем с самого редкого с пропусками
    Time(frequent & rare) ~ Time(rare) !!!
  - 9.4: GIN индексирует jsonb: keys отдельное от values
    Key 'tags' очень частотное - 1138532,
    value '{{term=>NYC}}' редкое - 285
    выигрыш около 10X раз !



9.4: Ассорти

- Add recovery_target='immediate' option
  - Закончить recovery сразу по достижению consistent состояния (минимум информации из xlogs).
  - Удобно для получения standalone сервера (скинуть к себе на ноут)
- Add a pg_lsn data type, to represent an LSN
  - LSN - Log Sequence Number, 64-bit int, позиция в WAL
  - Операции (=, !=, <, >, <=, >=, -)

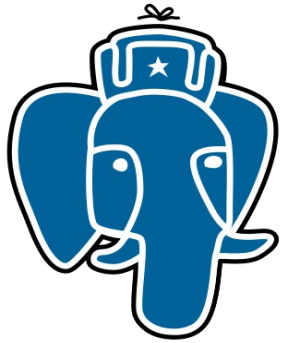
```
SELECT (pg_current_xlog_location())::pg_lsn;
 pg_current_xlog_location
--------------------------
 4/44123DE0
(1 row)
```



9.4: Ассорти

- Add pg_stat_archiver statistics view
  - Отображает текущее состояние архивирования WAL

```
\d pg_stat_archiver
            View "pg_catalog.pg_stat_archiver"
       Column        |           Type           | Modifiers
---------------------+--------------------------+-----------
 archived_count      | bigint                   |
 last_archived_wal   | text                     |
 last_archived_time  | timestamp with time zone |
 failed_count        | bigint                   |
 last_failed_wal     | text                     |
 last_failed_time    | timestamp with time zone |
 stats_reset         | timestamp with time zone |
```

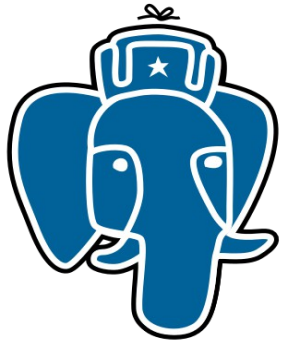


9.4: Ассорти

- Multiple-argument UNNEST (WITH ORDINALITY)

```
SELECT * FROM unnest(
array['a', 'b', 'c'],
array['d', 'e', 'f']
);
 unnest | unnest
--------+--------
 a      | d
 b      | e
 c      | f
(3 rows)
```

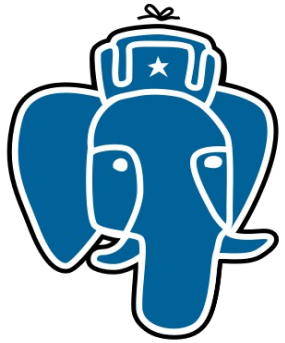
```
SELECT * FROM unnest(
array['a', 'b', 'c'],
array['d', 'e', 'f']
) WITH ORDINALITY;
 unnest | unnest | ordinality
--------+--------+------------
 a      | d      |          1
 b      | e      |          2
 c      | f      |          3
(3 rows)
```



9.4: Ассорти

- FILTER aggregates
  (раньше надо было использовать case-then-null :)

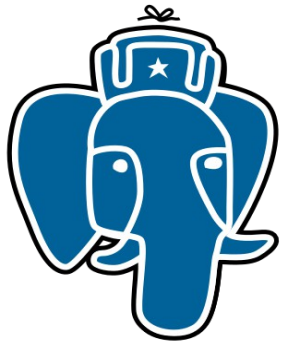
```
SELECT i, count(*), count(*) FILTER (WHERE i>0) FROM t1 GROUP BY i;
 i  | count | count
----+-------+-------
  1 |     1 |     1
 -1 |     1 |     0
  5 |     1 |     1
(3 rows)
```



9.4: Ассорти

- Improved EXPLAIN
  - Group Key в Agg и Group nodes
  - Время планирования запроса (может быть значительным)

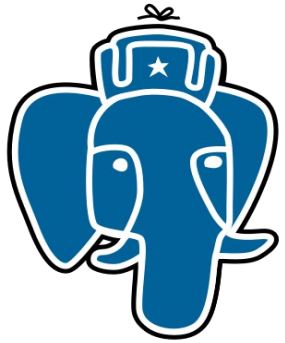
```
EXPLAIN ANALYZE SELECT * FROM t1 GROUP BY i;
                          QUERY PLAN
-----------------------------------------------------------
HashAggregate  (cost=40.00..42.00 rows=200 width=4)
                                  (actual time=0.013..0.013 rows=3 loops=1)
   Group Key: i
   ->  Seq Scan on t1  (cost=0.00..34.00 rows=2400 width=4)
                                  (actual time=0.005..0.007 rows=3 loops=1)
 Planning time: 0.023 ms  - не работает для prepared queries
 Total runtime: 0.042 ms
(5 rows)
```



9.4: Accopти

- Show exact/lossy pages in EXPLAIN ANALYZE for a bitmap heap scan

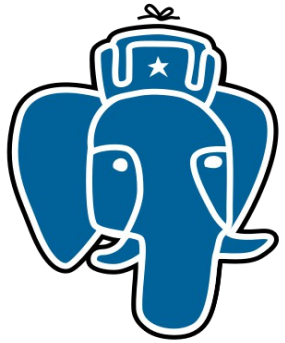
```
SET work_mem = '64kB';
SET
Time: 0.123 ms
postgres=# EXPLAIN ANALYZE SELECT * FROM aa WHERE a BETWEEN 100000 AND 200000;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on aa  (cost=1064.93..48166.70 rows=50000 width=4) (actual time=17.281..1087.696
   Recheck Cond: ((a >= 100000) AND (a <= 200000))
   Rows Removed by Index Recheck: 8797633
   Heap Blocks: exact=404 lossy=39367
   ->  Bitmap Index Scan on aai  (cost=0.00..1052.43 rows=50000 width=0) (actual time=17.147..17.1
         Index Cond: ((a >= 100000) AND (a <= 200000))
 Planning time: 0.178 ms
 Total runtime: 1093.238 ms
(8 rows)
```



9.4: Accopти

- Show exact/lossy pages in EXPLAIN ANALYZE for a bitmap heap scan

```
SET work_mem = '4MB';
SET
Time: 0.133 ms
postgres=# EXPLAIN ANALYZE SELECT * FROM aa WHERE a BETWEEN 100000 AND 200000;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on aa  (cost=2009.76..47676.28 rows=94568 width=4) (actual time=46.602..330.244
   Recheck Cond: ((a >= 100000) AND (a <= 200000))
   Heap Blocks: exact=39771
   ->  Bitmap Index Scan on aai  (cost=0.00..1986.12 rows=94568 width=0) (actual time=29.071..29.0
         Index Cond: ((a >= 100000) AND (a <= 200000))
 Planning time: 0.236 ms
 Total runtime: 344.044 ms
(7 rows)
```



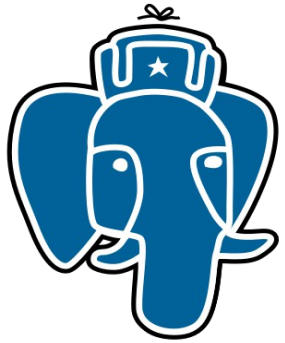
9.4: Ассорти

- Опции Tablespace можно задать при создании
  Раньше надо было создать и изменить (CREATE, ALTER)

```
CREATE TABLESPACE ssd
LOCATION '/ssd'
 WITH (random_page_cost = 1.1);
```

- Все объекты Tablespace можно перемещать одной командой

```
ALTER TABLESPACE pg_default
MOVE INDEXES TO ssd;
ALTER TABLESPACE ssd
MOVE ALL TO pg_default;
```



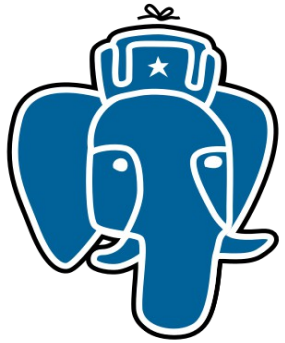
9.4: Ассорти

- «Разогреть кэш» - contrib/pg_prewarm
  - Весьма важно для разогревания реплики перед подключением вместо мастера (фэйловер)
  - Вместо популярного хака: find $PGDATA/base -type f -exec cat {} +
  - Можно использовать в SQL, не надо думать о структуре директорий

```
pg_prewarm(regclass, mode text default 'buffer', fork text default 'main',
           first_block int8 default null,
           last_block int8 default null) RETURNS int8
```

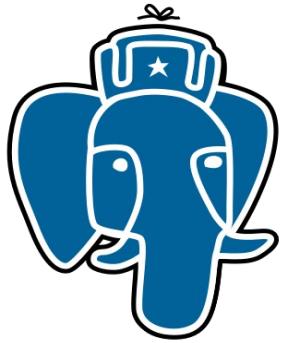
```
select pg_prewarm('jb');
 pg_prewarm
------------
     175856
(1 row)
```

```
'prefetch'- async., OS cache
'read'    - sync., OS cache
'buffer'  - shared buffers
```



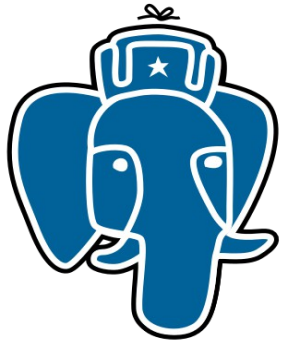
9.4: Ассорти

- ALTER SYSTEM SET - «редактирование» postgresql.conf в SQL
  - все изменения хранятся в postgresql.auto.conf
  - Нужен select pg_reload_conf();
  - бонус: не страшно ошибиться — просто SQL ошибка
- autovacuum_work_mem
  - отдельно от maintenance_work_mem
  - -1 умолчание (используется maintenance_work_mem)
  - maintenance_work_mem может быть очень большим, что ненужно для Autovacuum (их может быть много)



9.4: Ассорти

- wal_log_hints for consistent relation page tracking in WAL
  - hint bits содержатся в заголовке тупла, можно быстро проверять видимость без pg_clog (pg_subtrans). Некоторые утилиты используют эту информацию, например, pg_rewind (для быстрой перемотки
  - 9.3: нужно было обязательно использовать checksums, что вносит overhead и требует initdb.
  - 9.4: достаточно только wal_log_hints = on

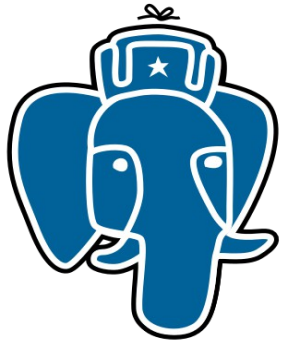


9.4: UPSERT ?

- Peter Geoghegan, Heroku.
  «UPSERT Why UPSERT is weird Counterintuitive lessons learned from the implementation effort»,

  http://www.pgcon.org/2014/schedule/events/661.en.html

  Talk that examines implementation process on the INSERT...ON DUPLICATE KEY LOCK FOR UPDATE feature proposed for PostgreSQL.
- Скорее всего, 9.5 (2015)



Это еще не все новости 9.4 !


Спасибо за Внимание