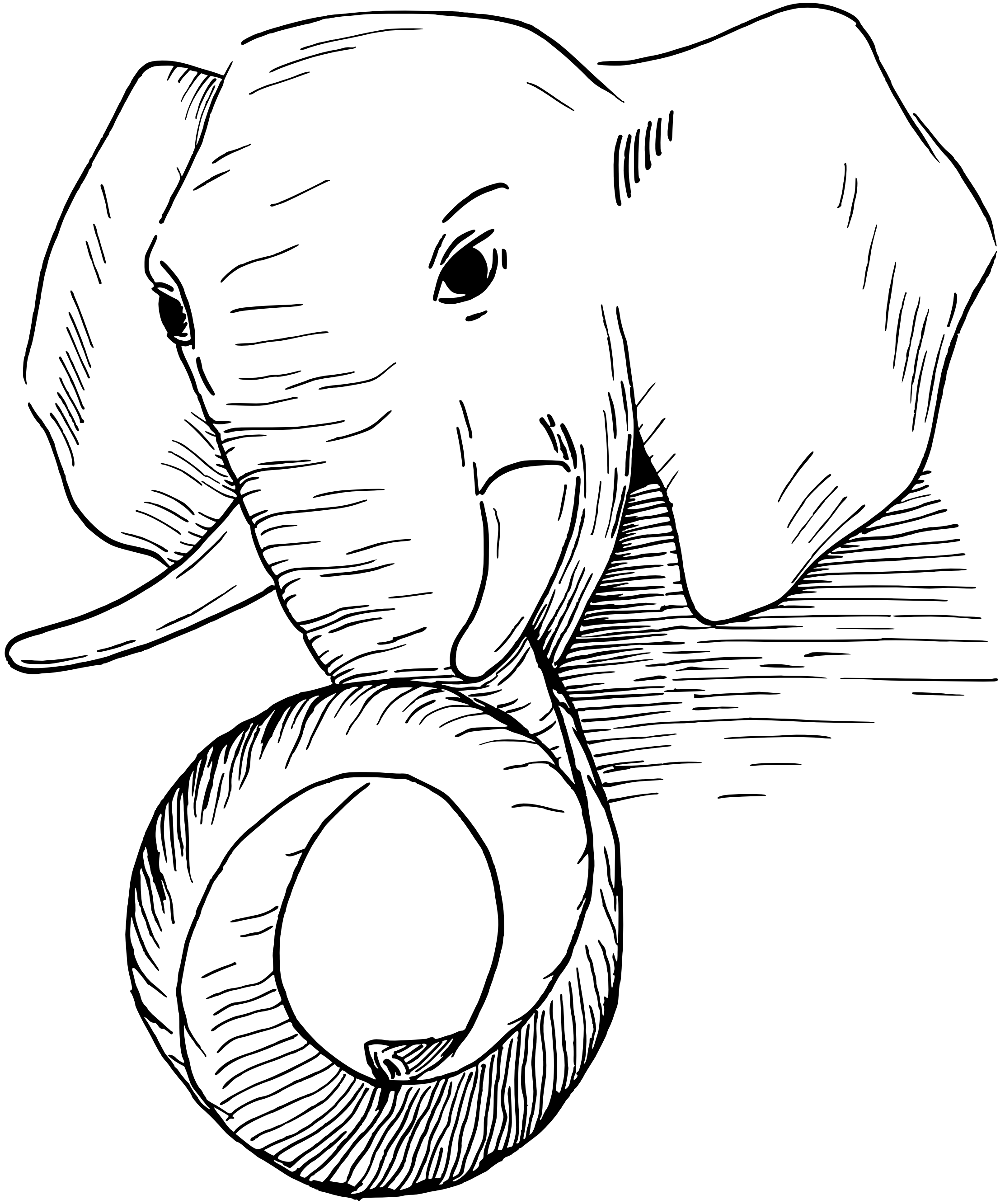


PostgreSQL

JSON Roadmap

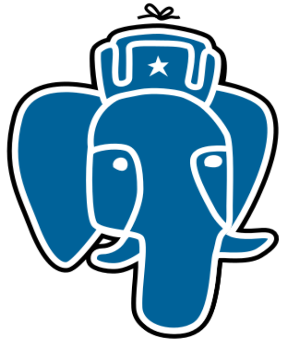
Oleg Bartunov
Postgres Professional

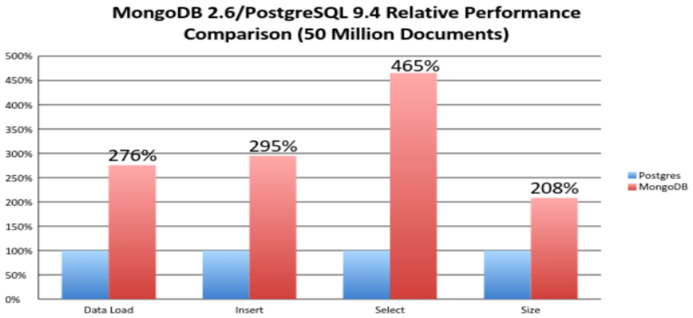
March 17, 2017, Moscow

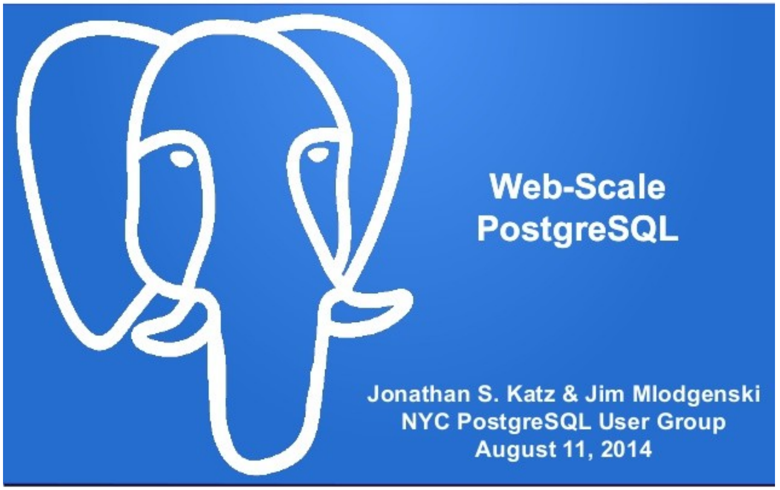
NoSQL Postgres briefly

- 2003 — hstore
- 2006 — hstore as illustration of GIN
- 2012 (sep) — JSON in 9.2
- 2012 (dec) — nested hstore proposal
- 2013 — PGCon talk about nested hstore
- 2013 — PGCon.eu talk about binary storage for nested data
- 2013 (nov) — nested hstore & jsonb
- 2014 (feb-mar) — forger nested hstore for jsonb
- Mar 23, 2014 — jsonb committed for 9.4

18 декабря 2014

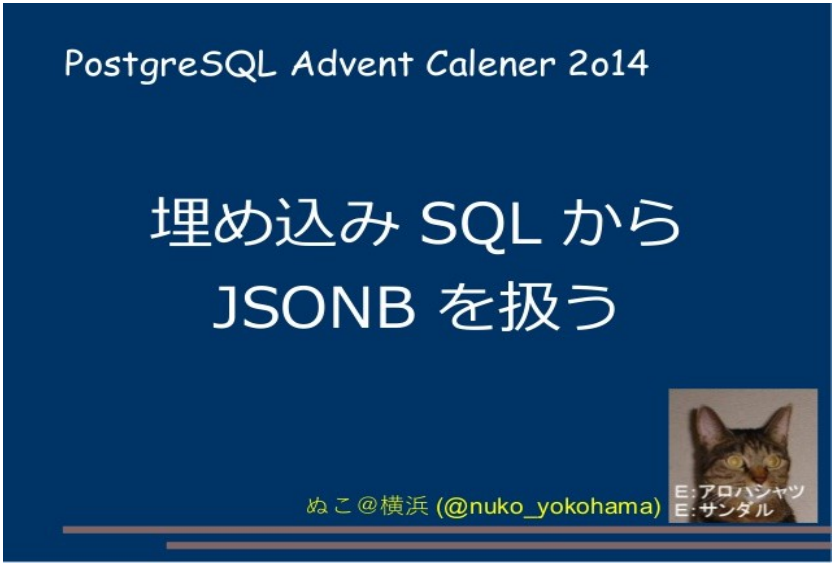






Web-Scale PostgreSQL

Jonathan S. Katz & Jim Mlodgenski
NYC PostgreSQL User Group
August 11, 2014



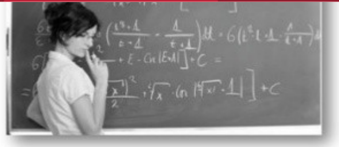
PostgreSQL Advent Calener 2014

埋め込み SQL から JSONB を扱う

ぬこ＠横浜 (@nuko_yokohama)

E:アロハシャツ
E:サンダル

Postgres’ NoSQ
Capabilities



- HSTORE
  - Key-value pair
  - Simple, fast and easy
  - Postgres v 8.2 – pre-dates many NoSQL-only solutions
  - Ideal for flat data structures that are sparsely populated
- JSON
  - Hierarchical document model
  - Introduced in Postgres 9.2, perfected in 9.3
- JSONB
  - Binary version of JSON
  - Faster, more operators and even more robust
  - Postgres 9.4

Postgres Unstructured
NoSQL with ACID

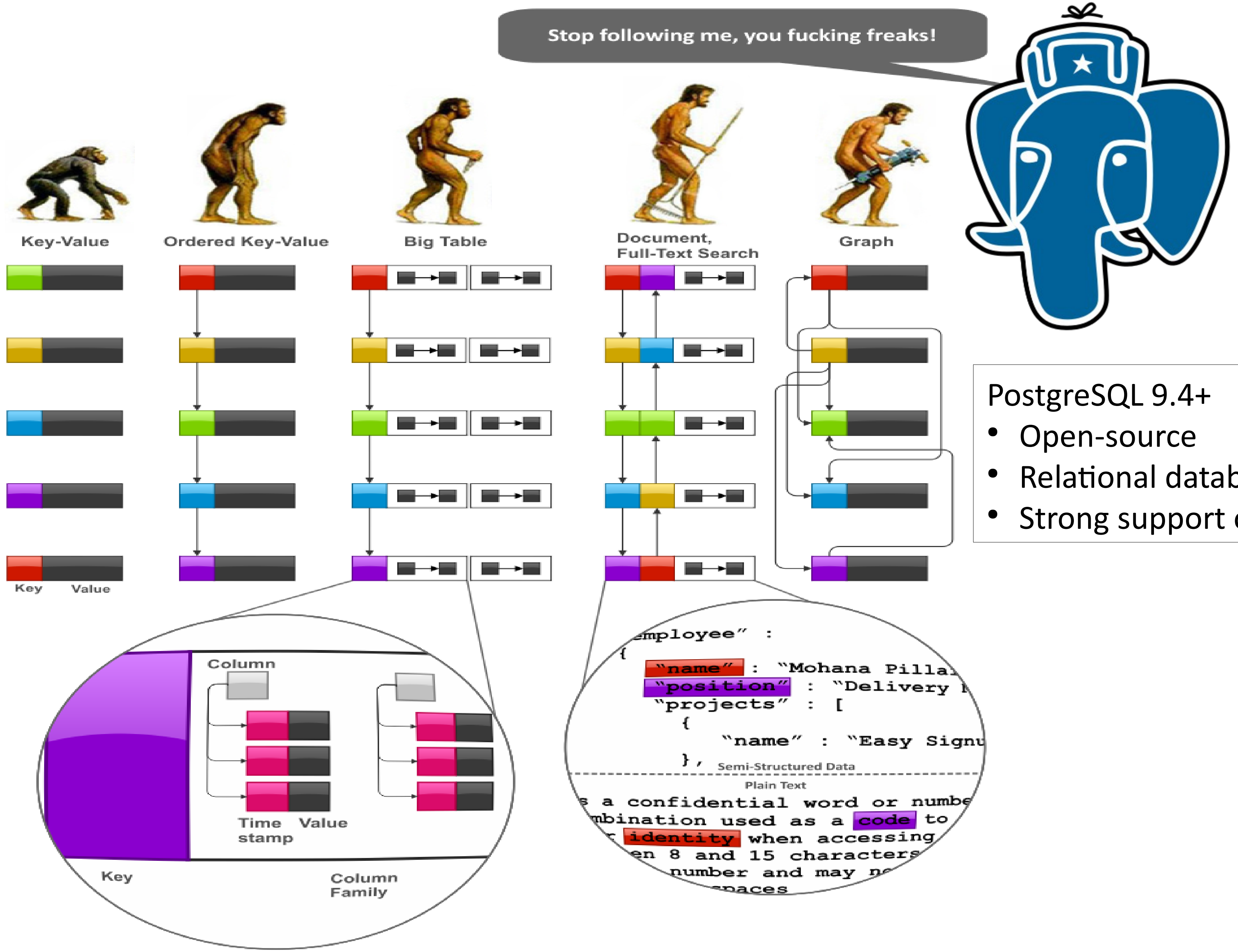
	Postgres	MongoDB
Data Load (s)	4,732	13,046
Insert (s)	29,236	86,253
Select (s)	594	2,763
Size (GB)	69	145

JSONB Features

- Equality operator
  - `SELECT '{“a”: 1, “b”: 2}'::jsonb = '{“b”:2, “a”:1}'::jsonb`
- Containment operator (Softserve)
  - `SELECT '{“a”: 1, “b”: 2}'::jsonb @> {“b”:2}::jsonb`
- Existence
  - `SELECT '{“a”: 1, “b”: 2}'::jsonb ? 'b';`

```
serve works as well)
}'::jsonb = '{“a”:[1,2]}'::jsonb
```

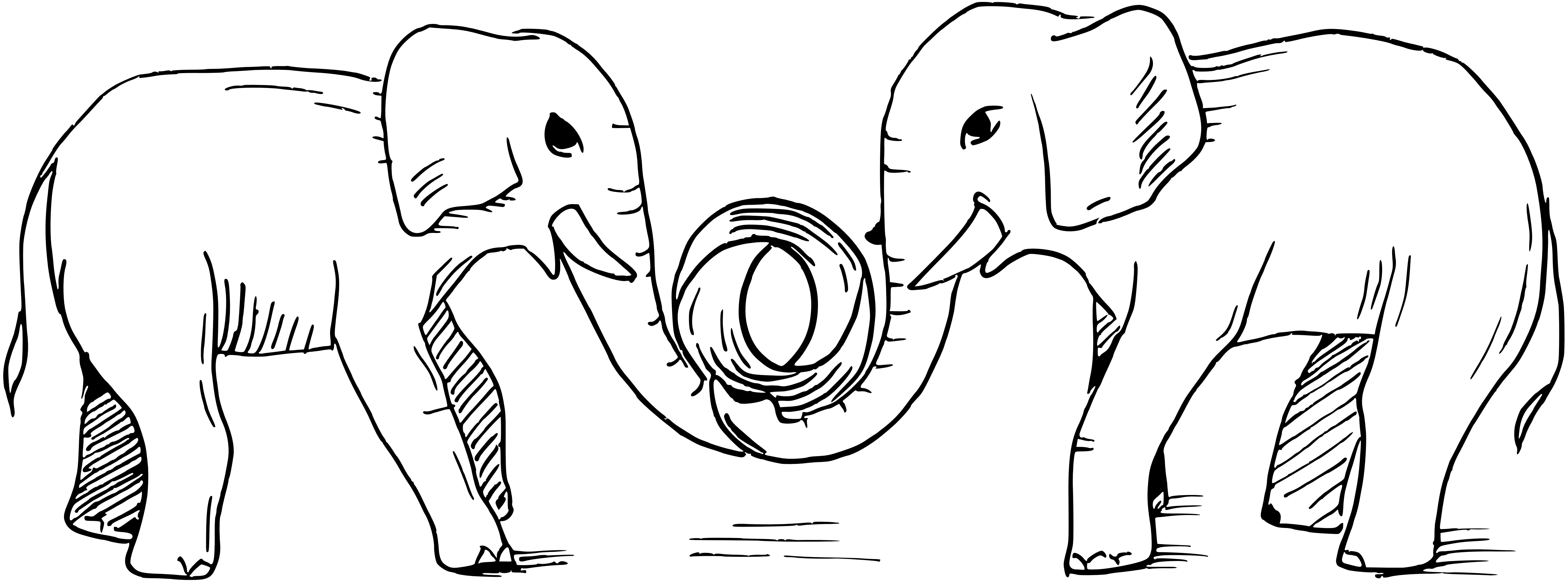
PROFESSIONAL Postgres



PostgreSQL 9.4+

- Open-source
- Relational database
- Strong support of json

Two JSON data types !!!



Jsonb vs Json

```
SELECT j::json AS json, j::jsonb AS jsonb FROM
(SELECT '{"cc":0, "aa":  2, "aa":1,"b":1}' AS j) AS foo;
               json               |           jsonb
----------------------------------+----------------------------
 {"cc":0, "aa":  2, "aa":1,"b":1} | {"b": 1, "aa": 1, "cc": 0}
(1 row)
```

- json: textual storage «as is»
- jsonb: no whitespaces
- jsonb: no duplicate keys, last key win
- jsonb: keys are sorted by (length, key)
- jsonb has a binary storage: no need to parse, has index support
- FORGET about json !

Summary: PostgreSQL 9.4 vs Mongo 2.6.0

- Search key=value (contains @>)
  - json : 10 s seqscan
  - jsonb : 8.5 ms GIN jsonb_ops
  - **jsonb : 0.7 ms GIN jsonb_path_ops**
  - mongo : 1.0 ms btree index
- Index size
  - jsonb_ops - 636 Mb (no compression, 815Mb)
    jsonb_path_ops - 295 Mb
  - jsonb_path_ops (tags) - 44 Mb USING gin((jb->'tags') jsonb_path_ops
  - mongo (tags) - 387 Mb
    mongo (tags.term) - 100 Mb

- Table size
  - postgres : 1.3Gb
  - mongo : 1.8Gb
- Input performance:
  - Text : 34 s
  - Json : 37 s
  - Jsonb : 43 s
  - mongo : 13 m

JSONB is Great, BUT

No good query language —

jsonb is a «black box» for SQL

Find something «red»

```
•      Table "public.js_test"
 Column |  Type   | Modifiers
--------+---------+-----------
 id     | integer | not null
 value  | jsonb   |

select * from js_test;

 id |                                 value
----+-----------------------------------------------------------------------
  1 | [1, "a", true, {"b": "c", "f": false}]
  2 | {"a": "blue", "t": [{"color": "red", "width": 100}]}
  3 | [{"color": "red", "width": 100}]
  4 | {"color": "red", "width": 100}
  5 | {"a": "blue", "t": [{"color": "red", "width": 100}], "color": "red"}
  6 | {"a": "blue", "t": [{"color": "blue", "width": 100}], "color": "red"}
  7 | {"a": "blue", "t": [{"color": "blue", "width": 100}], "colr": "red"}
  8 | {"a": "blue", "t": [{"color": "green", "width": 100}]}
  9 | {"color": "green", "value": "red", "width": 100}
(9 rows)
```

Find something «red»

```
WITH RECURSIVE t(id, value) AS ( SELECT * FROM
js_test
 UNION ALL
    (
      SELECT
        t.id,
        COALESCE(kv.value, e.value) AS value
      FROM
        t
        LEFT JOIN LATERAL
jsonb_each(
CASE WHEN jsonb_typeof(t.value) =
'object' THEN t.value
          ELSE NULL END) kv ON true
        LEFT JOIN LATERAL
jsonb_array_elements(
          CASE WHEN
jsonb_typeof(t.value) = 'array' THEN t.value
          ELSE NULL END) e ON true
        WHERE
          kv.value IS NOT NULL OR e.value IS
NOT NULL
     )
)
```

```
SELECT
  js_test.*
FROM
  (SELECT id FROM t WHERE value @> '{"color":
"red"}' GROUP BY id) x
  JOIN js_test ON js_test.id = x.id;
```

- Not easy !

PGCon-2014, Ottawa

Find something «red»

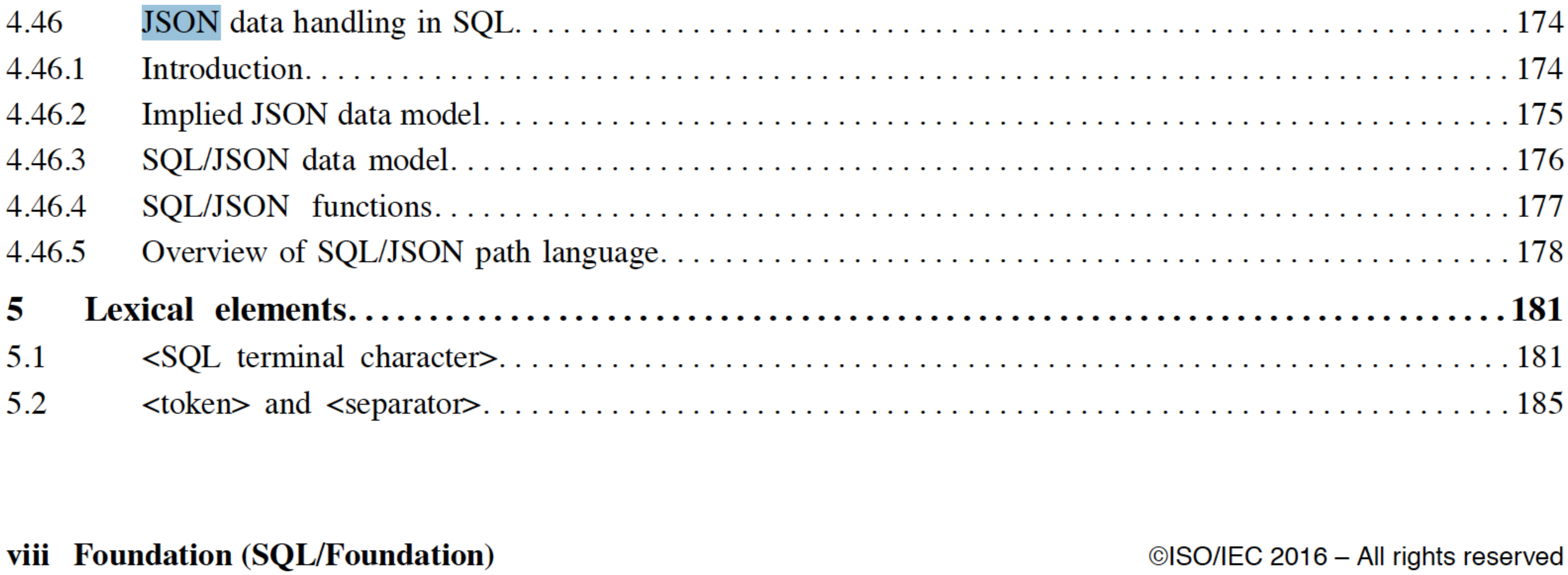
-
```
WITH RECURSIVE t(id, value) AS ( SELECT * FROM
js_test
 UNION ALL
    (
      SELECT
        t.id,
        COALESCE(kv.value, e.value) AS value
      FROM
        t
        LEFT JOIN LATERAL
jsonb_each(
CASE WHEN jsonb_typeof(t.value) =
'object' THEN t.value
          ELSE NULL END) kv ON true
        LEFT JOIN LATERAL
jsonb_array_elements(
          CASE WHEN
jsonb_typeof(t.value) = 'array' THEN t.value
          ELSE NULL END) e ON true
        WHERE
          kv.value IS NOT NULL OR e.value IS
NOT NULL
     )
)
```

```
SELECT
  js_test.*
FROM
  (SELECT id FROM t WHERE value @> '{"color":
"red"}' GROUP BY id) x
  JOIN js_test ON js_test.id = x.id;
```

- **Jsquery**

```
SELECT * FROM js_test
 WHERE
 value @@  '*.color = "red"';
```

JSON in SQL-2016



JSON in SQL-2016

- ISO/IEC 9075-2:2016(E) - https://www.iso.org/standard/63556.html
- BNF https://github.com/elliotchance/sqltest/blob/master/standards/2016/bnf.txt
- Discussed at Developers meeting Jan 28, 2017 in Brussels
- Post -hackers, Feb 28, 2017 (March commitfest)
  «Attached patch is an implementation of SQL/JSON data model from SQL-2016 standard (ISO/IEC 9075-2:2016(E)), which was published 2016-12-15 ...»
- Patch was too big (now about 16,000 loc) and too late for Postgres 10 :(

SQL/JSON in PostgreSQL

- It“s not a new data type, it“s a JSON data model for SQL
- PostgreSQL implementation is a subset of standard:
  - JSONB - ORDERED and UNIQUE KEYS
  - jsonpath data type for SQL/JSON path language
  - nine functions, implemented as SQL CLAUSEs

SQL/JSON in PostgreSQL

- **Jsonpath** provides an ability to operate (in standard specified way) with json structure at SQL-language level
  - Dot notation — $a.b.c
  - Array - [*]
  - Filter ? - $a.b.c ? (@.x > 10)
  - Methods - $a.b.c.x.type()

```
SELECT * FROM js WHERE JSON_EXISTS(js, 'strict $.tags[*] ? (@.term ==
"NYC")');

SELECT * FROM js WHERE js @> '{"tags": [{"term": "NYC"}]}';
```

SQL/JSON in PostgreSQL

```
SELECT JSON_EXISTS(jsonb '{"a": 1, "b": 2}', '$.* ? (@ > $x && @ < $y)'
            PASSING 0 AS x, 2 AS y);
 ?column?
----------
 t
(1 row)

SELECT JSON_EXISTS(jsonb '{"a": 1, "b": 2}', '$.* ? (@ > $x && @ < $y)'
            PASSING 0 AS x, 1 AS y);
```

SQL/JSON in PostgreSQL

- The SQL/JSON **construction** functions:
  - JSON_OBJECT - serialization of an JSON object.
    - json[b]_build_object()
  - JSON_ARRAY - serialization of an JSON array.
    - json[b]_build_array()
  - JSON_ARRAYAGG - serialization of an JSON object from aggregation of SQL data
    - json[b]_agg()
  - JSON_OBJECTAGG - serialization of an JSON array from aggregation of SQL data
    - json[b]_object_agg()

SQL/JSON in PostgreSQL

- The SQL/JSON **retrieval** functions:
  - JSON_VALUE - Extract an SQL value of a predefined type from a JSON value.
  - JSON_QUERY - Extract a JSON text from a JSON text using an SQL/JSON path expression.
  - JSON_TABLE - Query a JSON text and present it as a relational table.
  - IS [NOT] JSON - test whether a string value is a JSON text.
  - JSON_EXISTS - test whether a JSON path expression returns any SQL/JSON items

SQL/JSON in PostgreSQL

```
SELECT
x,
JSON_VALUE(
jsonb '{"a": 1, "b": 2}',
'$.* ? (@ > $x)' PASSING x AS x
RETURNING int
DEFAULT -1 ON EMPTY
DEFAULT -2 ON ERROR
) y
FROM
generate_series(0, 2) x;
 x | y
---+----
 0 | -2
 1 |  2
 2 | -1
(3 rows)
```

SQL/JSON in PostgreSQL

```
SELECT
        JSON_QUERY(js FORMAT JSONB, '$'),
        JSON_QUERY(js FORMAT JSONB, '$' WITHOUT WRAPPER),
        JSON_QUERY(js FORMAT JSONB, '$' WITH CONDITIONAL WRAPPER),
        JSON_QUERY(js FORMAT JSONB, '$' WITH UNCONDITIONAL ARRAY WRAPPER),
        JSON_QUERY(js FORMAT JSONB, '$' WITH ARRAY WRAPPER)
FROM
        (VALUES
                ('null'),
                ('12.3'),
                ('true'),
                ('"aaa"'),
                ('[1, null, "2"]'),
                ('{"a": 1, "b": [2]}')
        ) foo(js);
```

SQL/JSON in PostgreSQL

```
CREATE TABLE test_json_constraints (
       js text,
       i int,
       x jsonb DEFAULT JSON_QUERY(jsonb '[1,2]', '$[*]' WITH WRAPPER)
       CONSTRAINT test_json_constraint1
             CHECK (js IS JSON)
       CONSTRAINT test_json_constraint2
CHECK (JSON_EXISTS(js FORMAT JSONB, '$.a' PASSING i + 5 AS int, i::text AS txt))
       CONSTRAINT test_json_constraint3
CHECK (JSON_VALUE(js::jsonb, '$.a' RETURNING int DEFAULT ('12' || i)::int
       ON EMPTY ERROR ON ERROR) > i)
       CONSTRAINT test_json_constraint4
             CHECK (JSON_QUERY(js FORMAT JSONB, '$.a'
WITH CONDITIONAL WRAPPER EMPTY OBJECT ON ERROR) < jsonb '[10]')
);
```

Find something «red»

-
  ```
WITH RECURSIVE t(id, value) AS ( SELECT * FROM
js_test
 UNION ALL
  (
   SELECT
    t.id,
    COALESCE(kv.value, e.value) AS value
   FROM
    t
    LEFT JOIN LATERAL
jsonb_each(
CASE WHEN jsonb_typeof(t.value) =
'object' THEN t.value
     ELSE NULL END) kv ON true
    LEFT JOIN LATERAL
jsonb_array_elements(
     CASE WHEN
jsonb_typeof(t.value) = 'array' THEN t.value
     ELSE NULL END) e ON true
    WHERE
     kv.value IS NOT NULL OR e.value IS
NOT NULL
  )
)
```

```
SELECT
 js_test.*
FROM
 (SELECT id FROM t WHERE value @> '{"color":
"red"}' GROUP BY id) x
 JOIN js_test ON js_test.id = x.id;
```

- Jsquery

```
SELECT * FROM js_test
 WHERE
 value @@  '*.color = "red"';
```

- **SQL/JSON 2016**

```
SELECT * FROM js_test WHERE
JSON_EXISTS( value,'$.* *.color ?
(@ == "red")');
```

SQL/JSON availability

- Github Postgres Professional repository https://github.com/postgrespro/sqljson
- We need your feedback, bug reports and suggestions
- Help us writing documentation !

JSON Roadmap

- Push SQL/JSON to Postgres 11 (Postgres Pro 10)
- Dictionary compression to Postgres 11 (Postgres Pro 10)

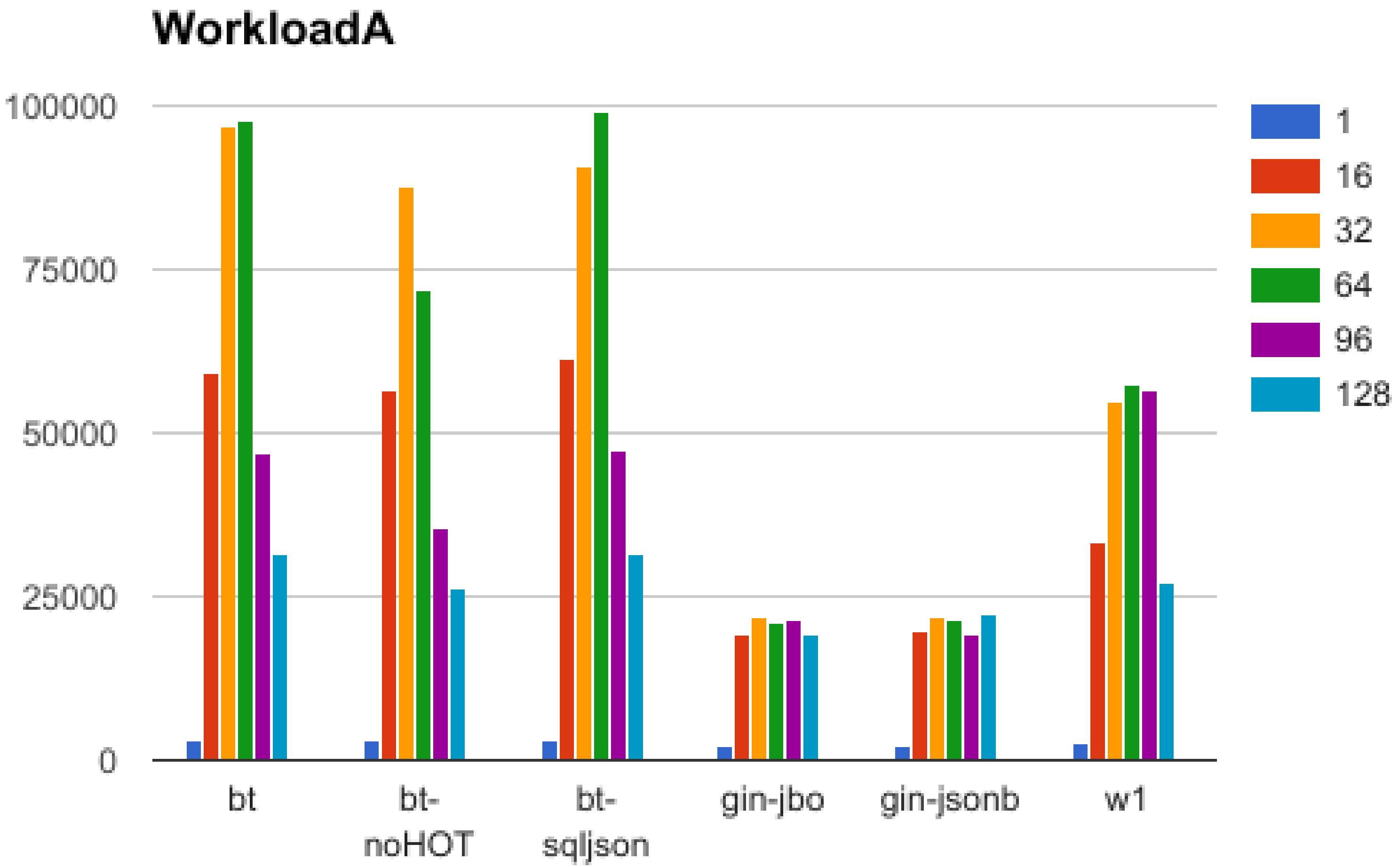
- Need sharding to be a real NoSQL database !

WHAT“S ABOUT MONGO !?!?!?

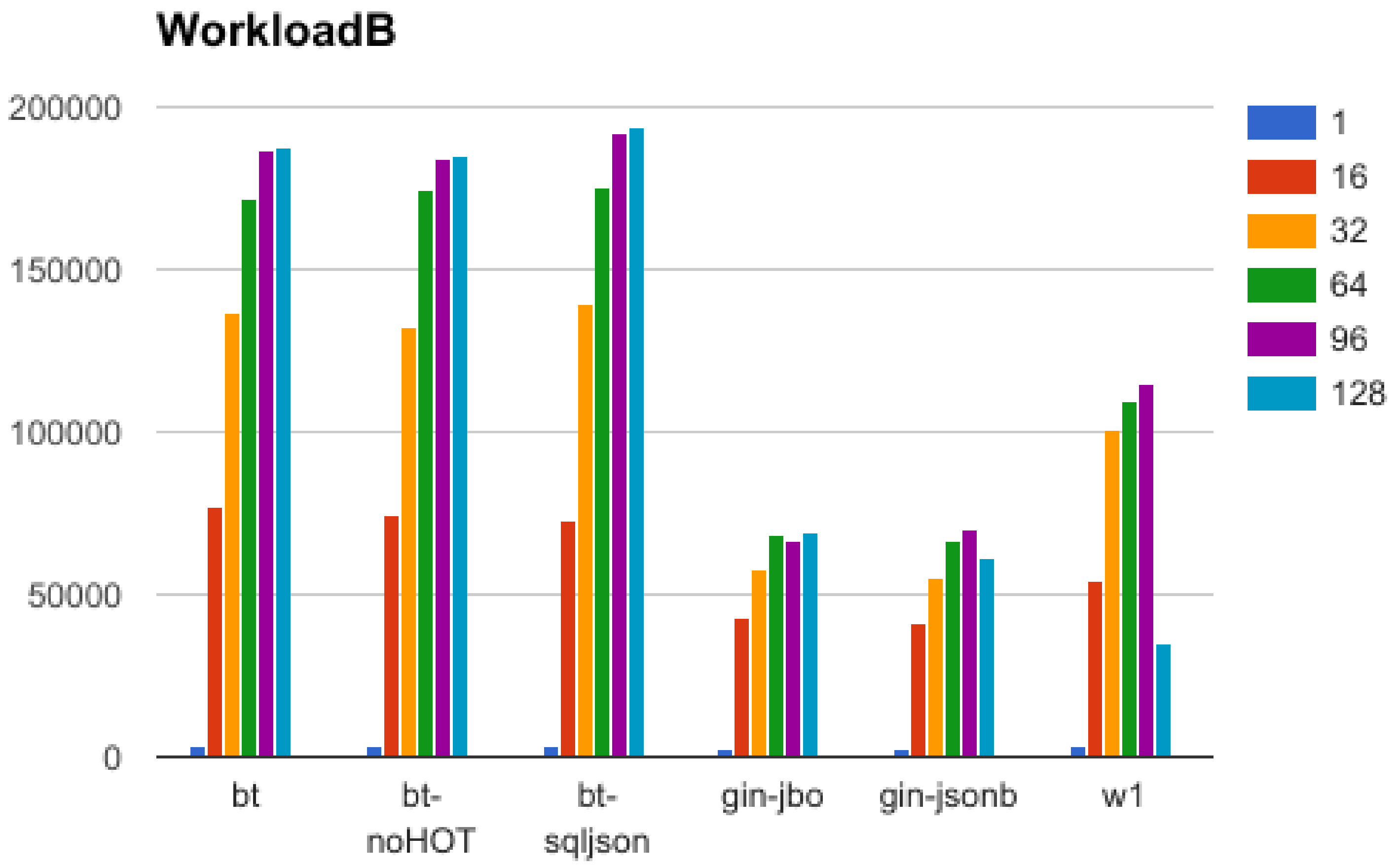
YCSB Benchmark

- Yahoo! Cloud Serving Benchmark - https://github.com/brianfrankcooper/YCSB/wiki
- De-facto standard benchmark for NoSQL databases
- Scientific paper «Benchmarking Cloud Serving Systems with YCSB» https://www.cs.duke.edu/courses/fall13/cps296.4/838-CloudPapers/ycsb.pdf
- We run YCBS for Postgres master and MongoDB 3.4.2
  - 1 server with 24 cores, 48 GB RAM for clients
  - 1 server with 24 cores, 48 GB RAM for database
  - Postgres tuned (asycnhronous commit off)
  - Mongodb (w1, j0)

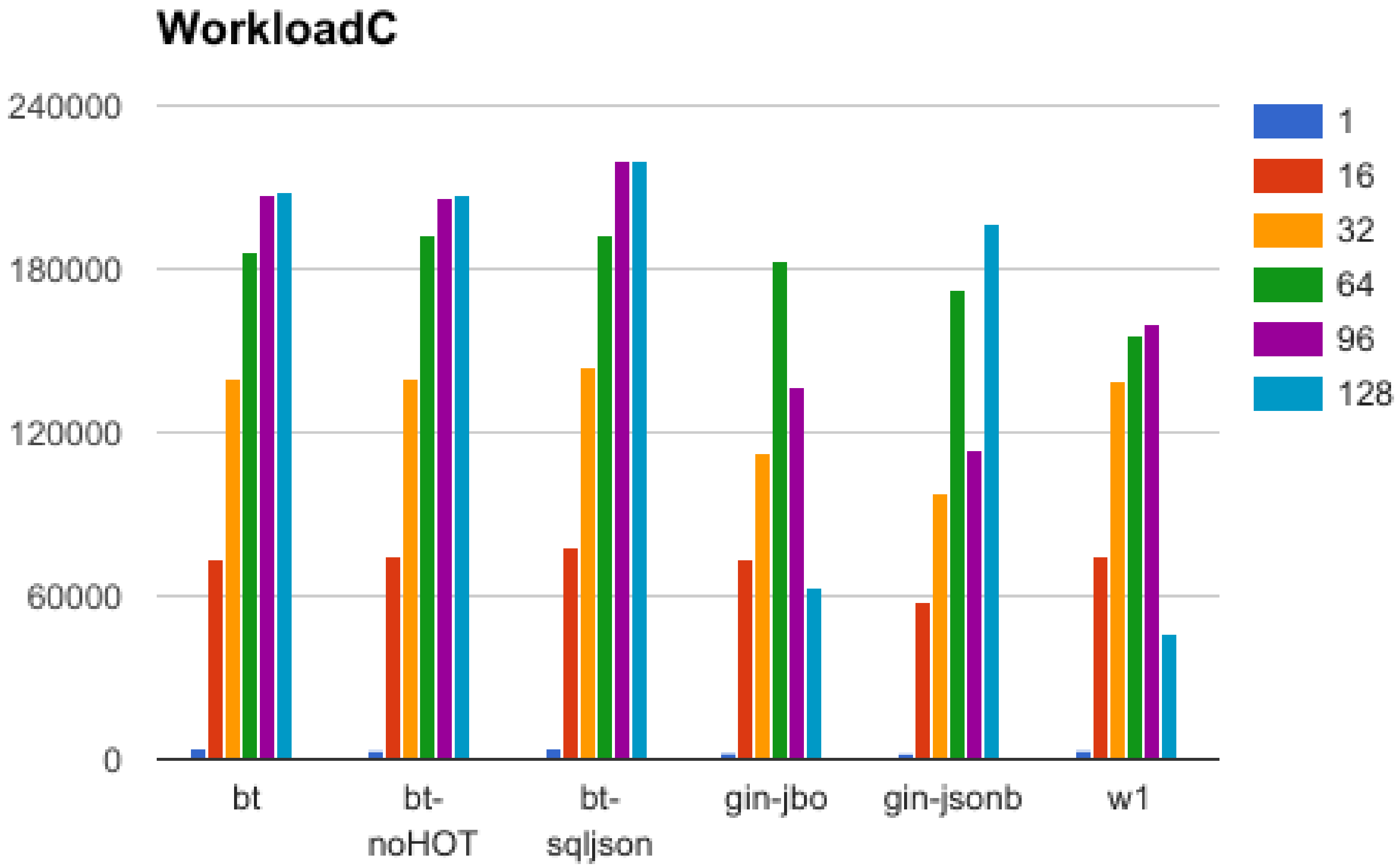
YCSB Benchmark — read 50%, update 50%



YCSB Benchmark — Read 95%, update 5%

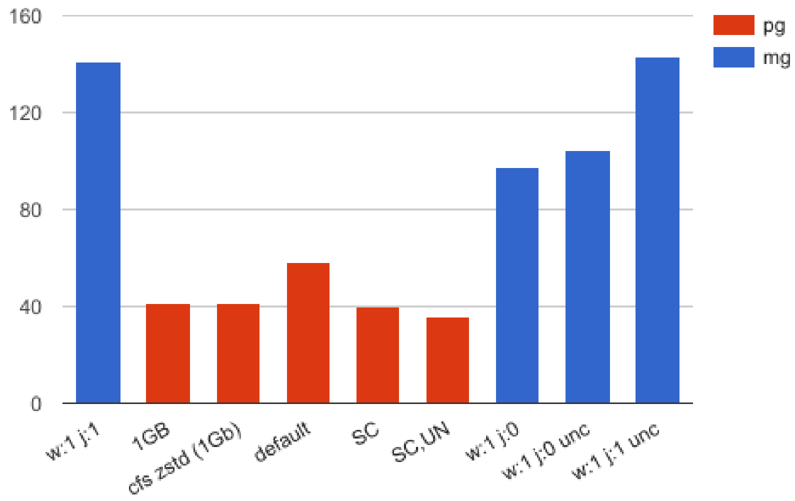


YCSB Benchmark — Read 100%

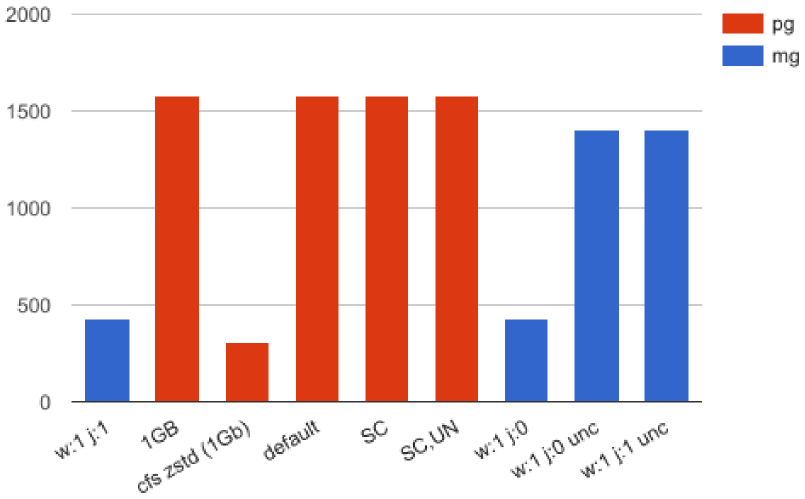


Load data 3 mln Citus dataset

Load time (s) 3 mln rows. Pg 10, Mongo 3.4.12



Size (Mb) 3 mln rows. Pg 10, Mongo 3.4.12



I see no reason to use Mongodb,

PostgreSQL still beats Mongodb !

Thanks !