Viva, the NoSQL Postgres!

Oleg Bartunov
Lomonosov Moscow University, Postgres Professional
Oleg Bartunov
Major PostgreSQL contributor
CEO, Postgres Professional
Research scientist at
Lomonosov Moscow University
obartunov@postgrespro.ru

Since 1995
Five concepts in 15 minutes

1) PostgreSQL is a COOL universal database
2) NoSQL in PostgreSQL is a MATURE feature
3) NoSQL PostgreSQL is fast
4) NoSQL PostgreSQL has GOOD roadmap
5) ALL YOU NEED IS Postgres!
PostgreSQL Forks: OLTP, MPP, OLAP, CLOUD, GIS, STREAM, TIMESERIES, GPU

- PostgreSQL
- OLTP
- MPP
- OLAP
- CLOUD
- GIS
- STREAM
- TIMESERIES
- GPU

- Commercial
- Open Source
NoSQL PostgreSQL is MATURE

- HSTORE — binary key-value storage, index support
- 2003 — initial release
- 2006 — part of PostgreSQL
Two JSON data types !!!

Textual storage «as is»

Binary storage, index support

A lot of functionality!
4.46  JSON data handling in SQL ................................................................. 174
4.46.1 Introduction .................................................................................. 174
4.46.2 Implied JSON data model ............................................................ 175
4.46.3 SQL/JSON data model ................................................................. 176
4.46.4 SQL/JSON functions ................................................................. 177
4.46.5 Overview of SQL/JSON path language ......................................... 178

5  Lexical elements .................................................................................. 181
5.1  <SQL terminal character> ........................................................... 181
5.2  <token> and <separator> ............................................................. 185
SQL/JSON in PostgreSQL

• PostgreSQL implementation (1 year of development)
  • Uses native data types JSON, JSONB
  • JSONPATH data type for SQL/JSON path language
  • Nine functions SQL/JSON functions for constructing:
    • JSON_OBJECT, JSON_ARRAY,
      JSON_OBJECTAGG, JSON_ARRAYAGG
  • and retrieving
    • JSON_VALUE, JSON_QUERY, JSON_TABLE,
      IS [NOT] JSON, JSON_EXISTS
• Extensions: more methods, JSONB op JSONPATH
SQL/JSON in PostgreSQL

• SQL-2016 path language specifies the parts (the projection) of JSON data to be retrieved by path engine for SQL/JSON functions.

• Jsonpath — the binary data type for SQL/JSON path expression to effective query JSON data.

SELECT JSON_QUERY(js,  
'$.floor[*]? (@.level >1).apt[*]? (@.area>$min && @.area < $max).no' 
PASSING 40 AS min, 90 AS max)  
FROM house;
Visual guide on jsonpath

```json
{
    "address": {
        "city": "Moscow",
        "street": "Ulyanova, 7A"
    },
    "lift": false,
    "floor": [
        { "level": 1,
          "apt": [
            { "no": 1, "area": 40, "rooms": 1 },
            { "no": 2, "area": 80, "rooms": 3 },
            { "no": 3, "area": 50, "rooms": 2 }
          ]
        },
        { "level": 2,
          "apt": [
            { "no": 4, "area": 100, "rooms": 3 },
            { "no": 5, "area": 60, "rooms": 2 }
          ]
        }
    ]
}
```
2-floors house
$.floor[0, 1].apt[1 to last]
$.floor[*]?(@.level > 1).apt[*]? (@.area > 40 && @.area < 90).no
SELECT JSON_QUERY(js, '$.floor[0, 1].apt[1 to last]' WITH WRAPPER) FROM house;

[{
"no": 2, "area": 80, "rooms": 3
}, {
"no": 3, "area": 50, "rooms": 2
}, {
"no": 5, "area": 60, "rooms": 2
}]

(1 row)
### JSON_TABLE — relational view of json

```sql
SELECT apt.*
FROM
  house,
  JSON_TABLE(js, '$.floor[0, 1]' COLUMNS (  
    level int,
    NESTED PATH '$.apt[1 to last]' COLUMNS (  
      no int,
      area int,
      rooms int
    )
  )
)) apt;
```

<table>
<thead>
<tr>
<th>level</th>
<th>no</th>
<th>area</th>
<th>rooms</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>80</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>60</td>
<td>2</td>
</tr>
</tbody>
</table>

(3 rows)
SQL/JSON indexing

- Uses existing GIN indexes

  ```sql
  CREATE INDEX  ON bookmarks USING gin
  (JSON_QUERY(js, '$.tags.term' WITH WRAPPER) jsonb_path_ops);
  ```

- Index only selected parts of json (parameters for opclass, PG 11-12)

  ```sql
  CREATE INDEX ON bookmarks USING gin
  (js jsonb_path_ops(projection='$.tags[*].term'));
  Index size: 33Mb vs 292 Mb (full json)
  ```

It is possible to index several paths:

  ```sql
  CREATE INDEX ON bookmarks USING gin
  (js jsonb_path_ops(projection='$.tags[*].term, $.id, $.links'));
  ```
SQL/JSON availability

- Currently under review for PG 11
- Github Postgres Professional repository
  https://github.com/postgrespro/sqljson
- WEB-interface to play with SQL/JSON
  http://sqlfiddle.postgrespro.ru/#!21/0/1819
- Technical Report (SQL/JSON) - available for free
• JSONB is a «fat» data type — keys could be up to $2^{28}$, 256 Mb!

«loooooooooooooooooooooooooooong_key1»:1,
«veeeeeeeeeeery_loooooooooooooooooooong_key2»:2

• Dictionary compression using
  CUSTOM Compression API (PG 11)
jsonb compression: table size
jsonb compression: summary

- jsonbc can reduce jsonb column size to its relational equivalent size
- jsonbc has a very low CPU overhead over jsonb and sometimes can be even faster than jsonb
- jsonbc compression ratio is significantly lower than in page level compression methods
- Availability:
  - Under review for PG 11
NoSQL Postgres is fast!

- Yahoo! Cloud Serving Benchmark (YCSB) - de-facto standard benchmark for NoSQL databases
- We run YCSB for PostgreSQL 10, MongoDB 3.4.5
  - 1 server with 72 cores, 3 TB RAM, 2 TB SSD for clients
  - 1 server with 72 cores, 3 TB RAM, 2 TB SSD for database
  - 10Gbps switch
- In most practical cases PostgreSQL is faster MongoDB
- PostgreSQL performance degrades in high-contention writes (zipfian distribution of queries, high number backends >100)
- Avoid high-contention with built-in pool of connections (PG 12)
Built-in pool of connections helps!!!
JSONB subscripting syntax (PG11)

- Based on «Generic type subscripting» on commitfest https://commitfest.postgresql.org/15/1062/
  Extends array syntax to support other types

```sql
UPDATE test_table set ARR[1] = 100;

SELECT JS['a']['a1']['a2'] FROM test_table;

UPDATE test_table SET JS['a']['b'] = '2'::jsonb;
```
PERFORMANCE

STANDARD

2003-2006

HSTORE

2012

JSON

2014

JSONB

SQL/JSON

SQL/JSON++

Custom types support
smart indexing
update, delete

SQL 2016 support
Jsonb compression
subscripting syntax

2019 ?

2018

HSTORE

PERFORMANCE
Summary

• Postgres is already a good NoSQL database + clear roadmap
• Move from NoSQL to Postgres to avoid nightmare!
• SQL/JSON provides better flexibility and interoperability (PG 11)
• JSONB dictionary compression is really useful (PG 11)
• In most practical cases PostgreSQL is faster MongoDB
• PostgreSQL performance degrades in high-contention writes (zipfian distribution of queries, high number backends >100)
• Avoid high-contention with built-in pool of connections (PG 12)
• More slides: https://goo.gl/3XVzQD