JSON AND POSTGRES: BETTER TOGETHER

Vibhor Kumar
V.P. Performance Engineering

EDB™

AGENDA

- Introduction to EDB
- Intro to JSON and HSTORE
- JSON History in Postgres
- JSON Data Types, Operators and Functions
- JSON and JSONB – when to use which one?
- JSONB and Node.JS – easy as pie
- NoSQL Performance in Postgres – fast as greased lightning
- Say ‘Yes’ to ‘Not only SQL’
- Useful resources

THE WAY FORWARD

EDB™

Accelerate your
open source transformation

WHO WE ARE FOR

Business leaders who see data as strategic

Technologists who want the most versatile technologies

The money managers who want to free up budget

EDB™

The heartbeat of Postgres

75% of F500 Postgres customers

Most strategic usage

30% of Postgres code contributed

Source of innovation

1400+ Global customers

Customer requirements influencing the future direction of Postgres

>300 Dedicated Postgres engineers

Unparalleled expertise

3 of 7 Postgres Core Team Members

Central source of community influence and expertise

We know closing the gap requires a **Postgres Acceleration Strategy** for powering modern enterprise applications

Enterprise-Grade DBMS Capabilities

Power your enterprise APPS

Flexible Deployment Choice

Own your data

Risk Mitigation

Go as fast as you can

We know closing the gap requires a **Postgres Acceleration Strategy** for powering modern enterprise applications

Enterprise-Grade DBMS Capabilities

Power your enterprise apps

Extreme high availability
High performance at scale
Advanced security
Migration automation

Flexible Deployment Choice

Own your data

Deploy anywhere:
multi-cloud, on prem, hybrid
Containers, VM’s

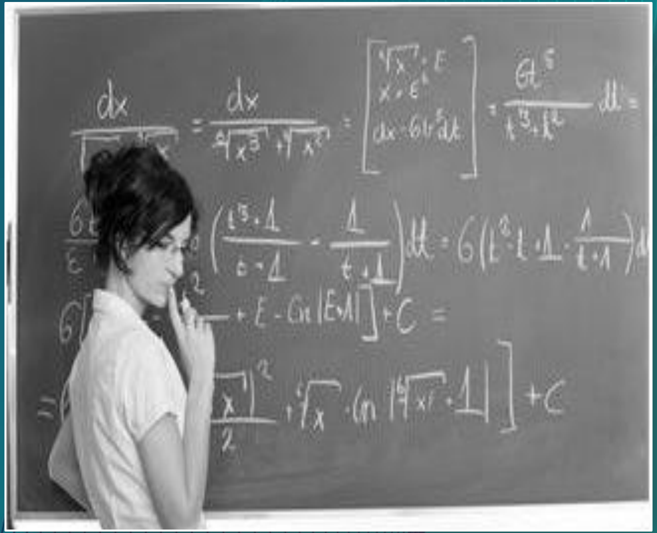
Risk Mitigation

Go as fast as you can

Hire the best Postgres expertise
Proven best practices

POSTGRES - FLEXIBLE DATA TYPES

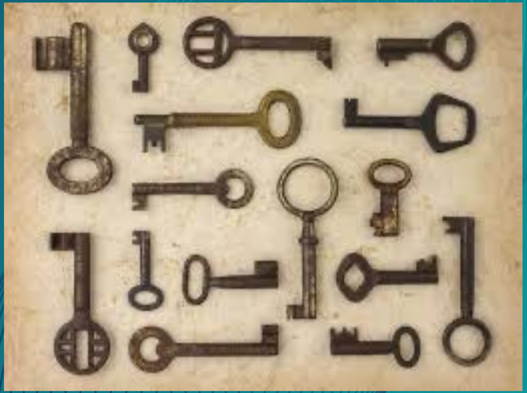
- HSTORE
  - Key-value pair
  - Simple, fast and easy
  - Postgres v 8.2 – pre-dates many NoSQL-only solutions
  - Ideal for flat data structures that are sparsely populated
- JSON
  - Hierarchical document model
  - Introduced in Postgres 9.2
- JSONB
  - Binary version of JSON
  - Faster, more operators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



EDB

POSTGRES: KEY-VALUE STORE

- Supported since 2006, the HStore contrib module enables storing key/value pairs within a single column
- Allows you to create a schema-less, ACID compliant data store within Postgres
- Create single HStore column and include, for each row, only those keys which pertain to the record
- Add attributes to a table and query without advance planning
- Combines flexibility with ACID compliance



HSTORE EXAMPLES

- Create a table with HSTORE field

```
CREATE TABLE hstore_data (data HSTORE);
```

- Insert a record into `hstore_data`

```
INSERT INTO hstore_data (data) VALUES (‘
"cost"=>"500",
"product"=>"iphone",
"provider"=>"apple"');
```

- Select data from `hstore_data`

```
SELECT data FROM hstore_data ;
----------------------------------------------------
"cost"=>"500”,"product"=>"iphone”,"provider"=>"Apple"
(1 row)
```

POSTGRES – DOCUMENT STORE

- JSON is the most popular data-interchange format on the web
- Derived from the ECMAScript Programming Language Standard (European Computer Manufacturers Association).
- Supported by virtually every programming language
- New supporting technologies continue to expand JSON’s
    - Node.js
- Postgres has a native JSON data type (v9.2) and a JSON parser and a variety of JSON functions (v9.3)
- Postgres have a JSONB data type with binary storage and indexing (more capability coming in v15)

JSON EXAMPLES

- Creating a table with a JSONB field

```
CREATE TABLE json_data (data JSONB);
```

- Simple JSON data element:

```
{"name": "Apple Phone", "type": "phone", "brand": "ACME", "price": 200,
"available": true, "warranty_years": 1}
```

- Inserting this data element into the table json_data

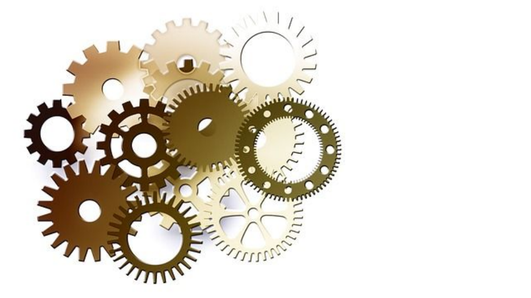
```
INSERT INTO json_data (data)  VALUES
     (’ {     "name": "Apple Phone",
         "type": "phone",
         "brand": "ACME",
         "price": 200,
         "available": true,
         "warranty_years": 1
     } ')
```

A QUERY THAT RETURN JSON DATA

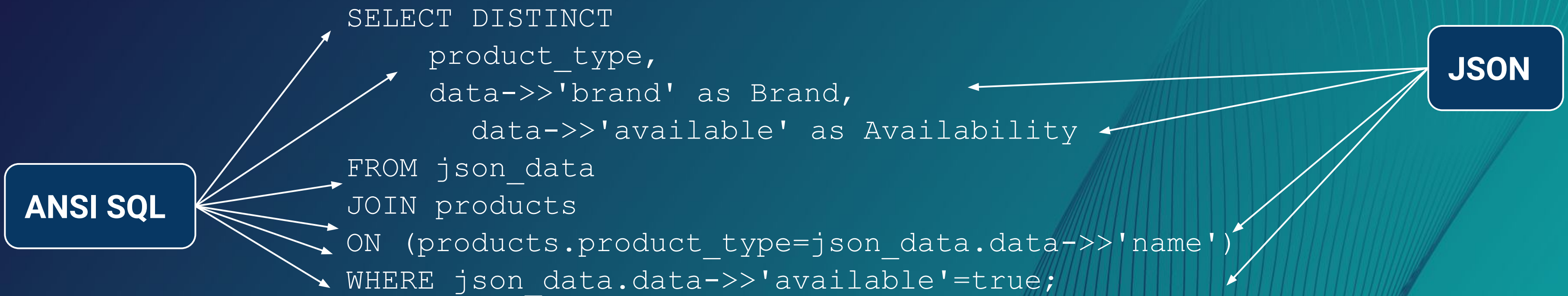
```
SELECT data FROM json_data;
data
------------------------------------------
 {"name": "Apple Phone", "type": "phone", "brand": "ACME", "price": 200,
"available": true, "warranty_years": 1}
```

JSON(B) AND ANSI SQL IN POSTGRES – A NATURAL FIT

- JSON is naturally integrated with ANSI SQL in Postgres
- JSON and SQL queries use
  the same language, the same planner, and the same ACID compliant transaction framework
- JSON and HSTORE are elegant and easy to use extensions of the underlying object-relational model



JSON AND ANSI SQL – EXAMPLE



```
SELECT DISTINCT
     product_type,
     data->>'brand' as Brand,
       data->>'available' as Availability
FROM json_data
JOIN products
ON (products.product_type=json_data.data->>'name')
WHERE json_data.data->>'available'=true;
```

```
 product_type | brand     | availability
----------------------------+-----------+------------
--
 AC3 Phone         | ACME  | true
```

No need for programmatic logic to combine SQL and NoSQL in the application – Postgres does it all

BRIDGING BETWEEN SQL AND JSON

Simple ANSI SQL Table Definition

```
CREATE TABLE products (id integer, product_name text );
```

Select query returning standard data set

```
SELECT * FROM products;

 id | product_name
----+---------------
  1 | iPhone
  2 | Samsung
  3 | Nokia
```

Select query returning the same result as a JSON data set

```
SELECT ROW_TO_JSON(products) FROM products;

 {"id":1,"product_name":"iPhone"}
 {"id":2,"product_name":"Samsung"}
 {"id":3,"product_name":"Nokia”}
```

JSON DATA TYPES

JSON is defined per RFC – 7159
For more detail please refer
http://tools.ietf.org/html/rfc7159

- Number:
  - Signed decimal number that may contain a fractional part and may use exponential notation.
  - No distinction between integer and floating-point
- String
  - A sequence of zero or more Unicode characters.
  - Strings are delimited with double-quotation mark
  - Supports a backslash escaping syntax.
- Boolean
  - Either of the values true or false.
- Array
  - An ordered list of zero or more values,
  - Each values may be of any type.
  - Arrays use square bracket notation with elements being comma-separated.
- Object
  - An unordered associative array (name/value pairs).
  - Objects are delimited with curly brackets
  - Commas to separate each pair
  - Each pair the colon ':' character separates the key or name from its value.
  - All keys must be strings and should be distinct from each other within that object.
- Null
  - An empty value, using the word null

JSON DATA TYPES EXAMPLE

```
{
  "firstName": "John",           -- String Type
  "lastName": "Smith",           -- String Type
  "isAlive": true,               -- Boolean Type
  "age": 25,                     -- Number Type
  "height_cm": 167.6,            -- Number Type
  "address": {                   -- Object Type
    "streetAddress": "21 2nd Street”,
    "city": "New York”,
    "state": "NY”,
    "postalCode": "10021-3100”
  },
  "phoneNumbers": [        // Object Array
    {                     // Object
      "type": "home”,
      "number": "212 555-1234”
    },
    {
      "type": "office”,
      "number": "646 555-4567”
    }
  ],
  "children": [],
  "spouse": null        // Null
}
```

POSTGRES JSONB TIMELINE (KEY HIGHLIGHTS)

- PG 9.2: Introduction of JSON (JSON text; no indexes)
- PG 9.4: Introduction of JSONB (canonical binary format; indexes)
- PG 9.5: jsonb_set() jsonb_pretty()to_jsonb(), jsonb_object(), jsonb_build_object(), jsonb_build_array(), jsonb_agg(), and jsonb_object_agg(), jsonb || operator, jsonb_strip_nulls() ...
- PG 9.6: jsonb_insert()
- PG 10: Full text search support for JSONB
- PG 11: jsonb_plpython
- PG 12: json_path (like xpath in XML; part of SQL Standard 2016)
- PG 13: jsonpath.datetime()
- PG 14: JSONB subscripting can be used to extract and assign to portions of JSONB.

POSTGRES JSONB CAPABILITIES

- Integration into same transactional context
- Fully ACID compliant
- Rich set of indexing technology (GIN, B-TREE, GIST, Trigram, Hash --- select the right index for the right operation)
- Rich set of functions and operators
- JSONPATH (similar to XPATH)
- Outperforms MongoDB in many use cases

EXAMPLE 1 – USING JSON FOR PERSONALIZATION

Flexible address specification

Array of phone numbers

```
INSERT INTO USERS (customer_nbr, details)
      VALUES(
            1,'
            {
            "firstName": "John", "lastName": "Smith",
            "isAlive": true,
            "height_cm": 165.6,
            "address":
                   {
                   "streetAddress": "21 2nd Street",
                   "city": "New York",
                   "state": "NY",
                   "ZIPCode": "10021-3100",
                   "country": "USA"
                   },
            "phoneNumbers":
                  [
                         {
                         "type": "home",
                         "number": "212 234-5678"
                         },
                         {
                         "type":"office",
                         "number": "646 555-4567"
                         }
                  ]
            }
       ');
```

EXAMPLE 1 – USING JSON FOR PERSONALIZATION

```
INSERT INTO USERS (customer_nbr, details)
        VALUES (
                2,'
                 {
                 "firstName": "Joan", "lastName": "of Arc",
                 "isAlive": false,
                 "height_cm": 162,
                 "address":
                         {
                         "city": "Rouen",
                         "codePostal": "7600",
                         "country": "FRANCE"
                         },
                 "phoneNumbers":[]
        }
         ');
```

Flexible address specification

Array of phone numbers

EDB

EXAMPLE 2: USING JSON_PATH

Do our records have a phone number for Joan of Arc?

```
SELECT jsonb_path_query(
     details, '$.phoneNumbers'
     )
FROM users
WHERE
      jsonb_path_exists (
          details, '$.lastName
               ?
          (@ == "of Arc")');
```

- Dots to move into the tree
- Brackets access a given array member coupled with a position.
- Variables, with ‘$’ representing a JSON text and ‘@’ for result path evaluations.
- ? applies a filter

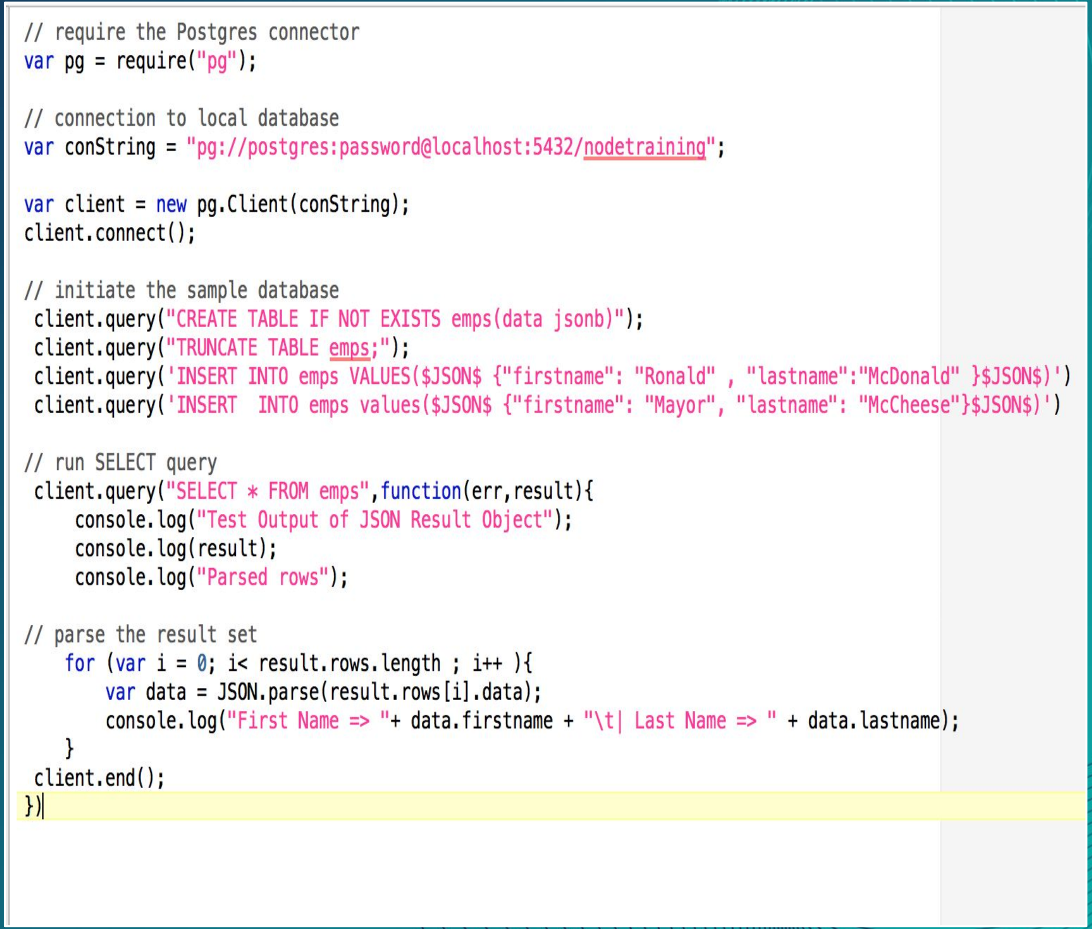
EXAMPLE 2: USING JSON_PATH

Get the phone numbers for customer ‘Smith’

```
SELECT jsonb_pretty
(jsonb_path_query(
details,
'$.phoneNumbers'))
FROM users
WHERE
jsonb_path_exists (
details,
'$.lastName ? (@ == "Smith")'
);
```

```
Result:
[
    {
        "type": "home",
        "number": "212 234-5678"
    },
    {
        "type": "office",
        "number": "646 555-4567"
    }
]
```

JSONB AND Node.js - EASY AS π



```
// require the Postgres connector
var pg = require("pg");

// connection to local database
var conString = "pg://postgres:password@localhost:5432/nodetraining";

var client = new pg.Client(conString);
client.connect();

// initiate the sample database
 client.query("CREATE TABLE IF NOT EXISTS emps(data jsonb)");
 client.query("TRUNCATE TABLE emps;");
 client.query('INSERT INTO emps VALUES($JSON$ {"firstname": "Ronald" , "lastname":"McDonald" }$JSON$)')
 client.query('INSERT  INTO emps values($JSON$ {"firstname": "Mayor", "lastname": "McCheese"}$JSON$)')

// run SELECT query
 client.query("SELECT * FROM emps",function(err,result){
     console.log("Test Output of JSON Result Object");
     console.log(result);
     console.log("Parsed rows");

// parse the result set
    for (var i = 0; i< result.rows.length ; i++ ){
        var data = JSON.parse(result.rows[i].data);
        console.log("First Name => "+ data.firstname + "\t| Last Name => " + data.lastname);
    }
 client.end();
})
```

JSON OR JSONB?

- JSON/JSONB is more versatile than HSTORE
- JSON or JSONB?
  - if you need any of the following, use JSON
    - Storage of validated json, without processing or indexing it
    - Preservation of white space in json text
    - Preservation of object key order Preservation of duplicate object keys
    - Maximum input/output speed
- For any other case, use JSONB

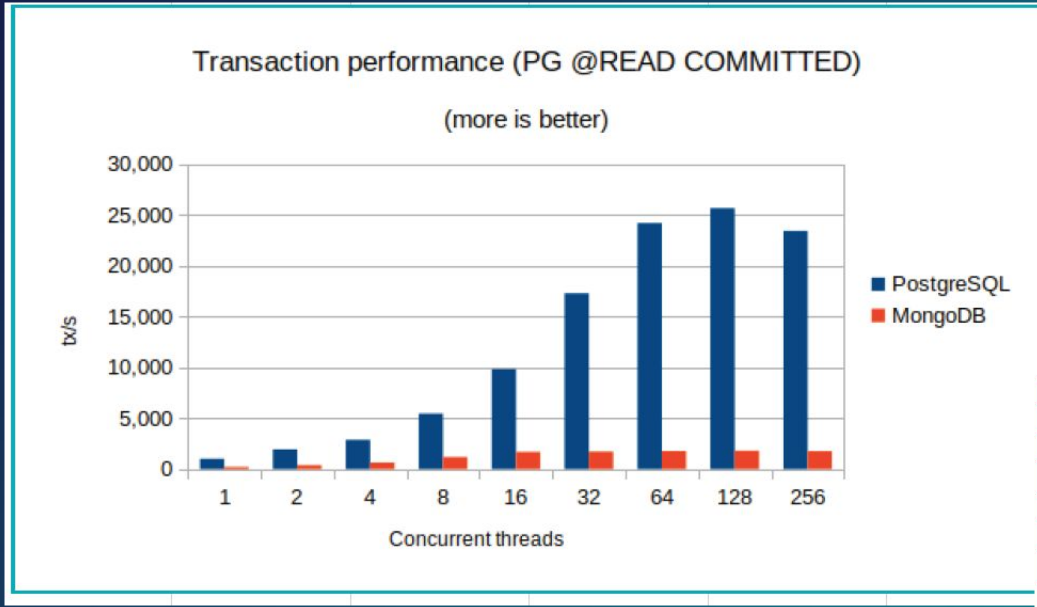
PERFORMANCE COMPARISON: POSTGRESQL 11/MongoDB 4.0

- Benchmarks published in June 2019 (follow up to similar analysis in 2014)
- Executed by ongres.com
- Using m5.4xlarge (16 vcores) on AWS EC2
- Details at (including the code and the engine to verify the findings)

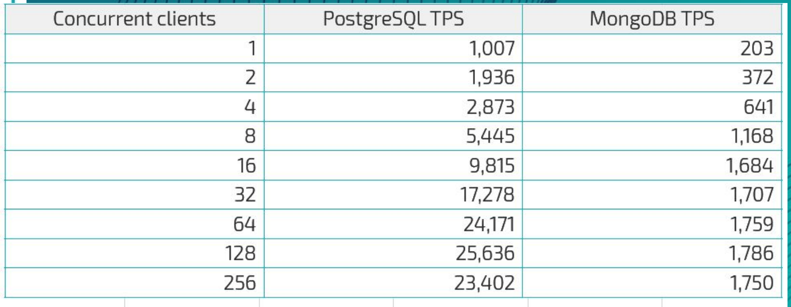
  http://info.enterprisedb.com/Performance-Benchmarks-PostgreSQL-vs-MongoDB.html

PERFORMANCE COMPARISON: POSTGRESQL 11/MongoDB 4.0

Complex Transaction Set



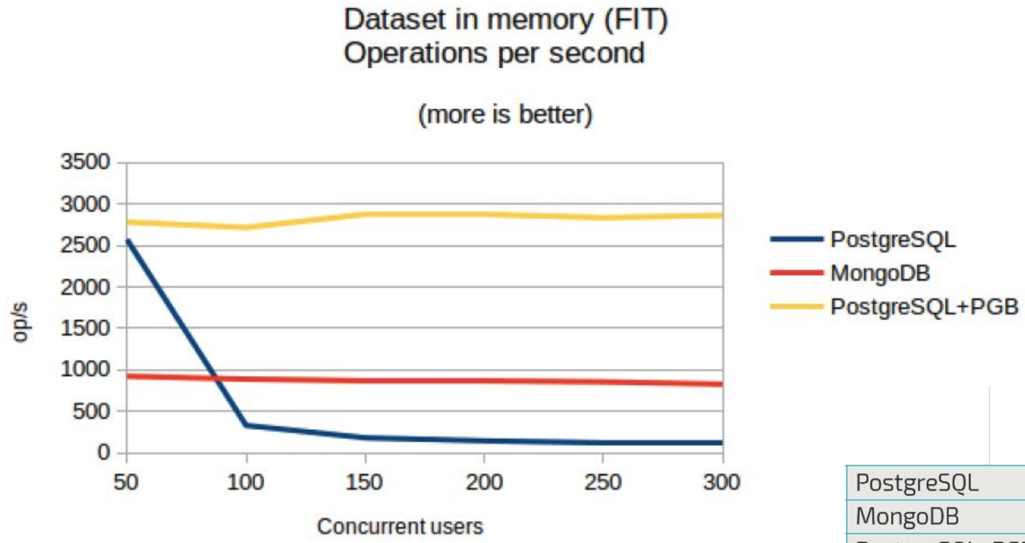
PostgreSQL is 4-15 times faster than MongoDB



Concurrent clients	PostgreSQL TPS	MongoDB TPS
1	1,007	203
2	1,936	372
4	2,873	641
8	5,445	1,168
16	9,815	1,684
32	17,278	1,707
64	24,171	1,759
128	25,636	1,786
256	23,402	1,750

PERFORMANCE COMPARISON: POSTGRESQL 11/MongoDB 4.0

OLTP (sysbench) - Many Small Transactions - Small Data Set

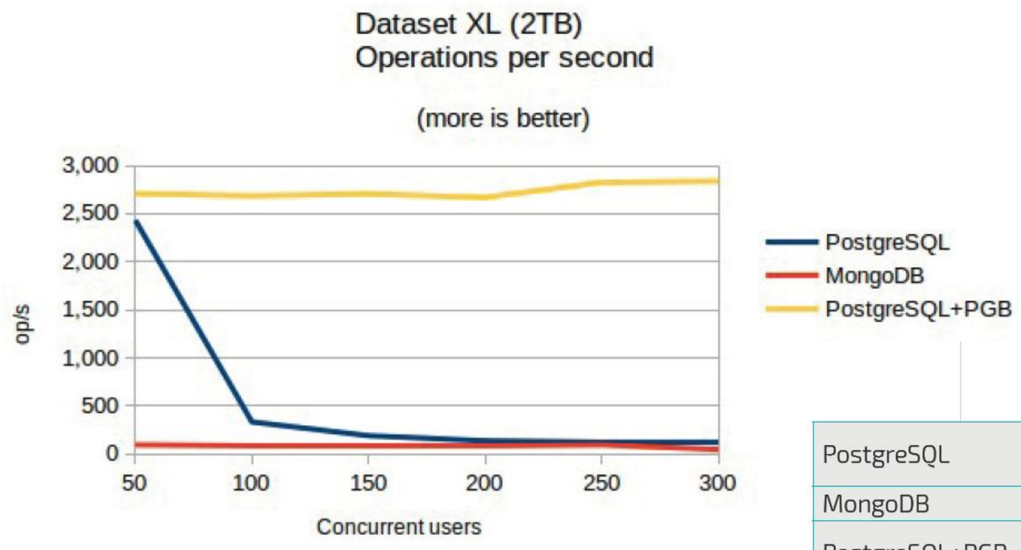


PGBouncer (connection pooler) is key to manage highly concurrent access

	Concurrent connections					
	50	100	150	200	250	300
PostgreSQL	2,569	332	183	147	127	121
MongoDB	924	889	872	867	856	828
PostgreSQL+PGB	2,779	2,714	2,880	2,881	2,832	2,860

PERFORMANCE COMPARISON: POSTGRESQL 11/MongoDB 4.0

OLTP (sysbench) - Many Small Transactions - Large Data Set



PGBouncer (connection pooler) is key to manage highly concurrent access

	Concurrent connections					
	50	100	150	200	250	300
PostgreSQL	2,433	334	191	138	116	123
MongoDB	96	81	82	86	96	48
PostgreSQL+PGB	2,709	2,686	2,707	2,670	2,827	2,839

ULTIMATE FLEXIBILITY WITH POSTGRES

Database Development
(PL/SQL, PL/pgSQL, PL/Tcl, PL/Perl …)

Cloud Deployment

Structured Data

Unstructured Data

On-Premise Deployment

Web 2.0/3.0

SAY YES TO 'NOT ONLY SQL'

- Postgres overcomes many of the standard objections "It can't be done with a conventional database system"
- Postgres
  - Combines structured data and unstructured data (ANSI SQL and JSON/HSTORE)
  - Is faster (for many workloads) than than the leading NoSQL-only solution
  - Integrates easily with Web 2.0 application development environments
  - Can be deployed on-premise or in the cloud

Do more with Postgres – the Enterprise NoSQL Solution

MICHAEL STONEBRAKER & MongoDB

3 blogs @ EDB from the original creator of PostgreSQL

- **“Schema Later” Considered Harmful**: *If you have data that will require a schema at some point, you are way better off doing the work up front to avoid data debt, because the cost of schema later is a lot higher*
- **Comparison of JOINS: MongoDB vs. PostgreSQL:** *The conclusion is that MongoDB joins are very brittle (when things change, application programs must be extensively recoded), and often MongoDB offers very poor performance, relative to Postgres*
- **Those Who Forget the Past Are Doomed to Repeat It**: *If you want to insulate yourself from the changes that business conditions dictate, use a relational DBMS. If you want the successor to the successor to your job to thank you for your wise decision, use a relational model.*

USEFUL RESOURCES

- The Postgres and MongoDB Report
- The CRUD of JSON in Postgres
- Building JSON Documents from Relational Tables
- Building JSON document relational tables
- Hear From EDB Customers Why Postgres is Their Preferred DBMS
- Liquibase and EDB extend CI/CD to EDB PostgreSQL Advanced Server
- Managing Data Changes to your PostgreSQL Database with Liquibase

THANK YOU