NoSQL + SQL = PostgreSQL

timbira

A empresa brasileira de PostgreSQL

DBA Brasil 1.0, São Paulo, 2016-04-16

Palestrante

timbira

- **Fabrízio de Royes Mello**
  - Desenvolvedor PostgreSQL
  - Líder do PostgreSQL Brasil
  - Pós-Graduando Uniritter (Agile)
  - @fabriziomello
  - `http://fabriziomello.github.io`
- **Timbira**
  - Consultor/Mentor/Coach
  - A empresa brasileira de PostgreSQL
  - Consultoria
  - Desenvolvimento
  - Suporte 24x7
  - Treinamento

Background

timbira

Top Skills

65	PostgreSQL
51	SQL
40	Databases
29	PHP
29	Linux
28	Software Development
25	Operating Systems
23	Agile Methodologies
13	Ubuntu
13	Open Source

Sobre este material

timbira

- esta apresentação está disponível em: `http://www.timbira.com.br/material`
- esta apresentação está sob licença *Creative Commons Atribuição 3.0 Brasil*: `http://www.creativecommons.org/licenses/by/3.0/br`
- Embora este material tenha sido elaborado com toda precaução, os autores não assumem quaisquer responsabilidades por erros, omissões ou danos resultantes da utilização das informações aqui contidas.
- Se você encontrar qualquer erro, por favor reporte-o a `contato@timbira.com.br`

Resumo

timbira

Teorema CAP

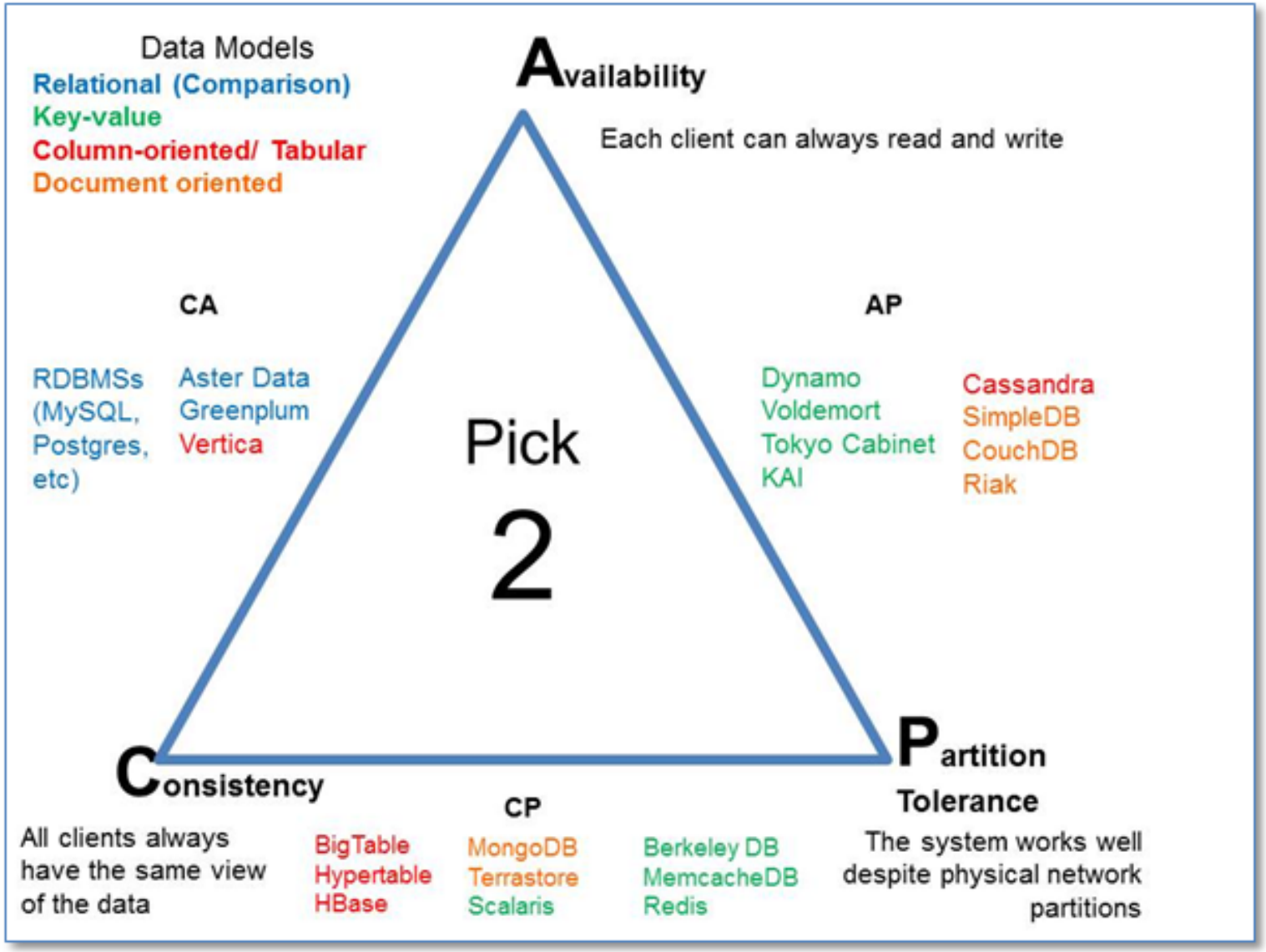
timbira

Em Ciências da Computação, o teorema CAP, também conhecido como Teorema de Brewer, afirma que é impossível para um sistema de computador distribuído fornecer simultaneamente todas as três das seguintes garantias:

- **Consistência** (Consistency): todos nós enxergam os mesmos dados ao mesmo tempo;
- **Disponibilidade** (Availability): garantia que todas requisições recebem uma resposta de falha ou sucesso;
- **Tolerância a Particionamento** (Partition Tolerance): o sistema continua operando apesar de perda de mensagem ou falha em parte dele;

Teorema CAP

timbira



Resumo

timbira

SQL

timbira

- *Structured Query Language*
- Linguagem Declarativa
- DML, DDL, DCL
- Joins, Anti-Joins, Semi-Joins, ...

Resumo

timbira

NoSQL

timbira

"Um banco de dados NoSQL ou Not Only SQL fornece um mecanismo para armazenamento e recuperação de dados que são modelados de forma diferente das relações (tabular) dos bancos de dados relacionais.
Motivações para essa abordagem incluem: design simples, escalabilidade horizontal e controle mais fino sobre disponibilidade." (Wikipedia)

NoSQL: História

timbira

- introduzido em 1998 por Carlos Strozzi como nome de um banco de dados que não tinha interface SQL
- re-introduzido em 2009 por Eric Evans (Rackspace) quando Johan Oskarsson (Last.fm) queria organizar um evento sobre bancos de dados open-source distribuidos

NoSQL: Classificação

timbira

- Key-Value (PostgreSQL, Dynamo, FoundationDB, MemcacheDB, Redis, Riak, ...)
- Document (PostgreSQL, CouchDB, MongoDB, ...)
- Column (PostgreSQL??, Cassandra, HBase, ...)
- Graph (PostgreSQL??, Allegro, Neo4j, OrientDB, ...)

NoSQL: ?????

timbira

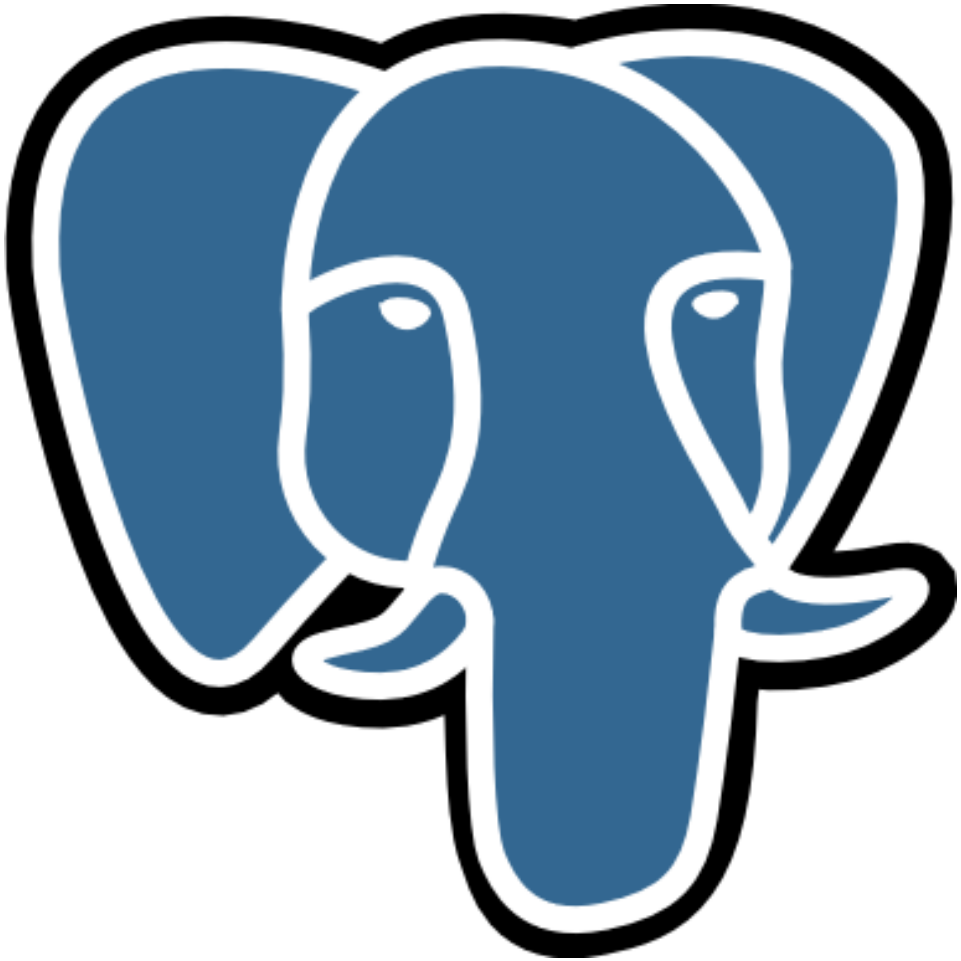
WTF, PostgreSQL em todas classificações NoSQL????

Resumo

timbira

PostgreSQL: O que é?

timbira



The world's most advanced open source database.

PostgreSQL: História

timbira

- Antes : Oriundo do INGRES
- 1986 : Início Projeto (Berkley)
- 1987 : Primeira versão do Postgres
- 1991 : versão 3 com principais funcionalidades atuais
- 1993 : versão 4.2, última lançada pela Berkley
- 1994 : Andrew Yu e Jolly Chen lançam Postgre95 com interpretador para linguagem SQL
- 1997 : Nome muda para PostgreSQL, versão 6 lançada
- 2000 : versão 7 lançada com suporte a FK
- 2005 : versão 8 lançada com versão nativa Windows, Tablespaces, Savepoints, Point-In-Time-Recovery
- 2005 : versão 8.1 Commit Tho Phases, Roles
- 2006 : versão 8.2 (Insert, Update, Delete) Returning, melhora performance OLTP e BI

PostgreSQL: História (continuação)

timbira

- 2008 : versão 8.3 debug PL/PgSQL, Tsearch2 (XML) no core
- 2009 : versão 8.4 Windowing Functions, Common Table Expressions and Recursive Queries, Parallel Restore, "pg_upgrade"
- 2010 : versão 9.0 Hot Standby and Streaming Replication
- 2011 : versão 9.1 Synchronous Replicacion, FDW (SQL/MED), CREATE EXTENSION, Unlogged Tables
- 2012 : versão 9.2 Index-only Scans, Cascading Replication, JSON, Range Types
- 2013 : versão 9.3 Materialized Views, Lateral Join, writable FDW, Event Triggers, Background Workers
- 2014 : versão 9.4 JSONB, Logical Decoding, Dynamic Background Workers
- 2015 : versão 9.5 UPSERT, RLS, BRIN, GROUPING SETS/ROLLUP, ALTER TABLE SET LOGGED/UNLOGGED
- 2016 : versão 9.6 Parallel SeqScan e Aggregate, Custom Access Method, and more...

Extensão HSTORE presente desde 8.2 (2006) - pacote 'contrib'

```
CREATE EXTENSION hstore;

CREATE TABLE users (
  email VARCHAR(255) PRIMARY KEY,
  data HSTORE
);

INSERT INTO
  users (email, data)
VALUES
 ('fabriziomello@gmail.com','sex=>"M",birthdate=>"1979-08-08"'),
  ('fabio.telles@gmail.com','sex=>"M",state=>"SP"');
```

PostgreSQL: Key/Value Database

timbira

```
SELECT * FROM users;
          email           |                    data
--------------------------+-------------------------------------
 fabriziomello@gmail.com | "sex"=>"M", "birthdate"=>"1979-08-08"
 fabio.telles@gmail.com  | "sex"=>"M", "state"=>"SP"
(2 rows)

SELECT * FROM users WHERE data->'state' = 'SP';
         email           |            data
-------------------------+----------------------------
 fabio.telles@gmail.com | "sex"=>"M", "state"=>"SP"
(1 row)

SELECT * FROM users WHERE data->'birthdate' = '1979-08-08';
          email           |                    data
--------------------------+-------------------------------------
 fabriziomello@gmail.com | "sex"=>"M", "birthdate"=>"1979-08-08"
(1 row)
```

Conseguimos transformar uma tupla inteira em HStore ;-)

```
SELECT hstore(users.*) FROM users;

                        hstore
-------------------------------------------------------------
 "data"=>"\"sex\"=>\"M\", \"birthdate\"=>\"1979-08-08\"",
     "email"=>"fabriziomello@gmail.com"
 "data"=>"\"sex\"=>\"M\", \"state\"=>\"SP\"",
     "email"=>"fabio.telles@gmail.com"
(2 rows)
```

E ainda um HStore em JSON ;-)

```
SELECT hstore_to_json(data) FROM users;

             hstore_to_json
-----------------------------------------
 {"sex": "M", "birthdate": "1979-08-08"}
 {"sex": "M", "state": "SP"}
(2 rows)
```

PostgreSQL: Document Database

timbira

Nativo apartir 9.2 (2012) e na 9.4 JSONB (2014)

```
CREATE TABLE users (
  email VARCHAR(255) PRIMARY KEY,
  data JSON
);

INSERT INTO
  users (email, data)
VALUES
 ('fabriziomello@gmail.com','{"sex":"M", "birthdate": "1979-08-08
  ('fabio.telles@gmail.com','{"sex":"M", "state":"SP"}');
```

PostgreSQL: Document Database

timbira

```
SELECT * FROM users;
          email           |                    data
--------------------------+-------------------------------------
 fabriziomello@gmail.com | {"sex":"M", "birthdate": "1979-08-08"}
 fabio.telles@gmail.com  | {"sex":"M", "state":"SP"}
(2 rows)

SELECT * FROM users WHERE data->>'state' = 'SP';
         email           |            data
-------------------------+----------------------------
 fabio.telles@gmail.com | {"sex":"M", "state":"SP"}
(1 row)

SELECT * FROM users WHERE data->>'birthdate' = '1979-08-08';
          email           |                    data
--------------------------+-------------------------------------
 fabriziomello@gmail.com | {"sex":"M", "birthdate": "1979-08-08"}
(1 row)
```

Conseguimos transformar uma tupla inteira em JSON/JSONB ;-)

```
SELECT row_to_json(users.*) FROM users;
                  row_to_json
-----------------------------------------------------
 {"email":"fabriziomello@gmail.com",
     "data":{"sex":"M", "birthdate": "1979-08-08"}}
 {"email":"fabio.telles@gmail.com",
     "data":{"sex":"M", "state":"SP"}}
(2 rows)
```

PostgreSQL: Column Database

timbira

Através de FDW (Foreign Data Wrappers)

- cassandra_fdw: acessar cassandra dentro do PostgreSQL (https://github.com/disqus/cassandra_fdw)
- cstore_fdw: columnar store for PostgreSQL (https://github.com/citusdata/cstore_fdw)
- hadoop_fdw: acessar dados do Hadoop dentro do PostgreSQL (http://www.bigsql.org/se/hadoopfdw)

PostgreSQL: Graph Database

timbira

Até dá, é difícil, e conheço apenas 2 formas possíveis:

- Modelando + CTE
  - modelar "nodes" e "edges" (nós e arestas)
  - utilizar CTE (Common Table Expressions) recursivas (WITH RECURSIVE ...)
- Utilizando neo4j_fdw (https://github.com/nuko-yokohama/neo4j_fdw)
- Implementando o SEU FDW... OrientDB??

Exemplo sequencia fibonacci

```
WITH RECURSIVE fibonacci(i, j) AS (
  SELECT 0, 1
  UNION ALL
  SELECT GREATEST(i, j), (i + j) AS i
    FROM fibonacci
   WHERE j < 13
)
SELECT i FROM fibonacci;
 i
---
 0
 1
 1
 2
 3
 5
 8
(7 rows)
```

PostgreSQL: E tem mais...

timbira

Stable:

- Extensão PL/V8 - funções em javascript e coffescript dentro do seu banco (http://code.google.com/p/plv8js/wiki/PLV8)
- mongo_fdw - FDW (leitura e escrita - 9.3) para MongoDB (https://github.com/EnterpriseDB/mongo_fdw)
- PgREST - API REST (http://pgre.st)
- PostgREST - API REST (http://postgrest.com)
- ToroDB - MongoDB compatible on Top of PostgreSQL (http://www.torodb.com)

Prototype:

- MongoLike - operações do mongodb no PostgreSQL (https://github.com/JerrySievert/mongolike)
- Mongres - background worker que conversa com protocolo do mongo (https://github.com/umitanuki/mongres)

PostgreSQL: Extensibilidade

timbira

- Custom Data Types, Operators, Casts, Aggregate, Collation, Encoding, Operator Class, Language Handlers
- Extensions (http://www.pgxn.org)
- Background Workers
- Hooks
- Foreign Data Wrappers (SQL/MED)
- Custom Access Methods
- Logical Decoding

PostgreSQL: Escalabilidade Horizontal

timbira

- PL/Proxy - https://wiki.postgresql.org/wiki/PL/Proxy
- Postgres-XL - http://www.postgres-xl.org
- BDR (Bidirectional Replication) - http://wiki.postgresql.org/wiki/BDR_Project
- CitusData - https://github.com/citusdata/citus, https://www.citusdata.com
- Custom - i.e. Instagram (http://media.postgresql.org/sfpug/instagram_sfpug.pdf)

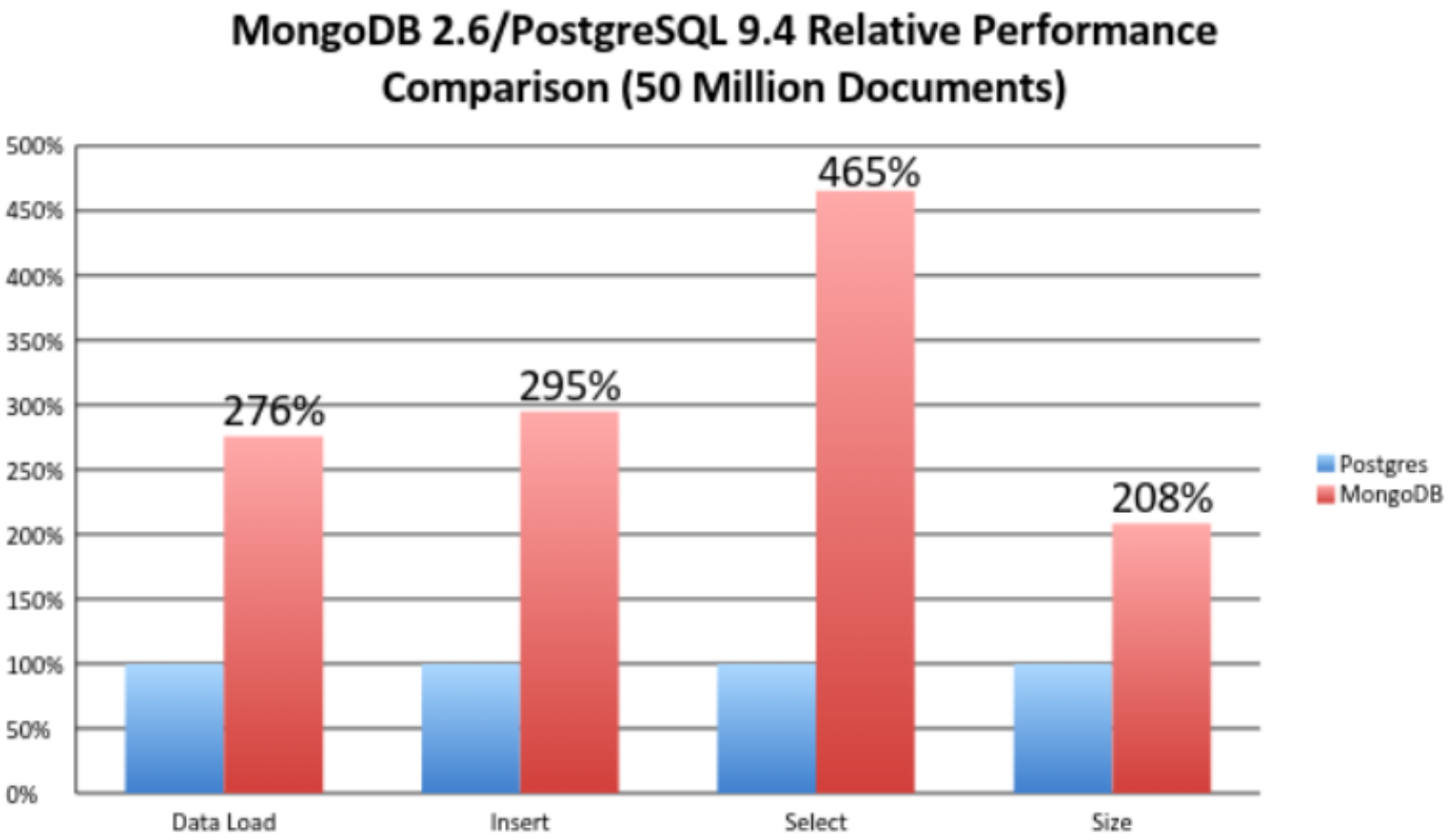
PostgreSQL: Consistência Eventual

timbira

- Via UNLOGGED TABLES
- Escrita extremamente rápida
- Não geram registros no WAL (Write Ahead Log)
- Não são crash-safe, seu conteúdo é zerado após um crash
- Seu conteúdo não é replicado via Streaming Replication

PostgreSQL 9.4 (JSONB) vs MongoDB

timbira



PostgreSQL 9.4 (JSONB) vs MongoDB

timbira

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

PostgreSQL 9.4 (JSONB) vs MongoDB

timbira

- Benchmark desenvolvido e executado pela EnterpriseDB (`www.enterprisedb.com.br`)
- Scripts disponíveis em `https://github.com/EnterpriseDB/pg_nosql_benchmark`
- Post com mais detalhes publicado em `http://bit.ly/1t2jG1r`

Resumo

timbira

Considerações Finais

timbira

- O PostgreSQL tem o melhor dos 2 mundos, mas ...
- ... pense fora da "caixinha"
- ... "nem todo problema é prego para precisar usar martelo"
- ... quem sabe vc precisa é de persistência poliglota

?

Fabrízio de Royes Mello
@fabriziomello
fabrizio@timbira.com.br
fabriziomello@gmail.com
http://www.timbira.com.br
http://slideshare.net/fabriziomello