Breizh C@mp
Mix de technologies

Not only SQL avec PostgreSQL

#pgnosql

Pierre-Alban DEWITTE - @__pad__

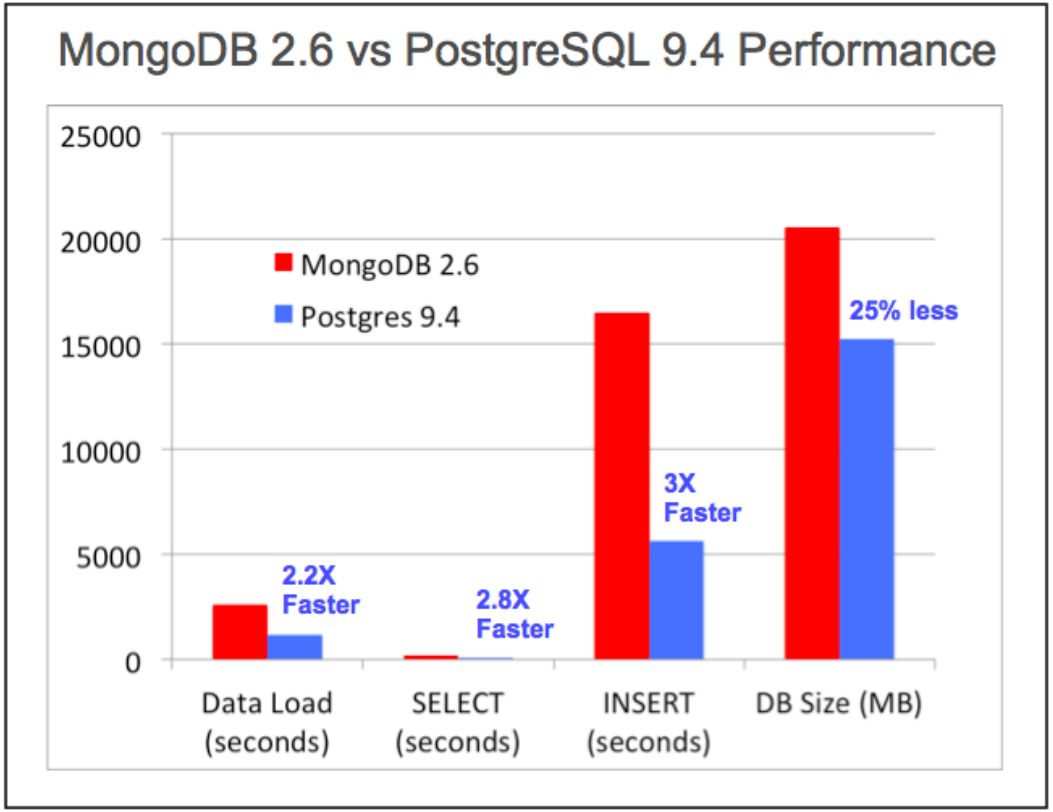
dev.myscript.com

Windows

7

Mac

JSONB et Postgresql



Source : http://www.enterprisedb.com/nosql-for-enterprise

Histoire du type JSONB

- 2006 – HSTORE – Stockage clef/valeur
- 2012 – JSON - JavaScript Object Notation
- 2014 – JSONB – JSON Binaire

Introduit par Oleg Bartunov, Teodor Sigaev, Peter Geoghegan and Andrew Dunstan dans Postgresql 9.4

Caractéristiques du type JSONB

- Permet de stocker un document JSON synthaxiquement valide
- Stocker en interne sous une forme binaire permettant l'indexation

En comparaison du format JSON

- Léger surcout à l'insertion
- Attention, contrairement à un chaine de charactère, l'ordre des objets n'est pas conservé
- Les attributs avec le même nom ne sont pas conservés

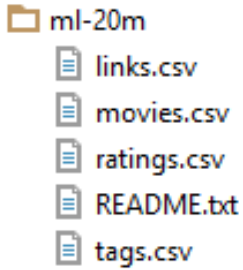
Modélisation

Possibilités offertes

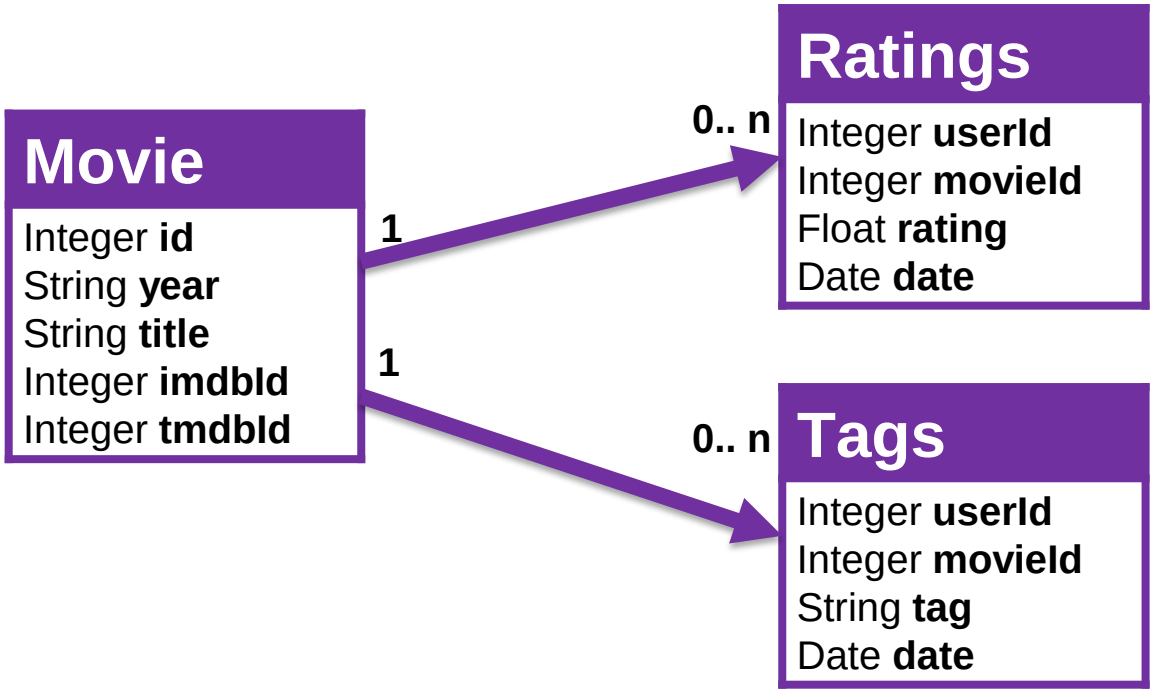
- Ajouter simplement de la flexibilité au schéma
- Modéliser les données en fonction des use cases
- Regroupement dans une même entités l’ensemble des informations constituant sont identité

Application de démo

- Utilisation des données MovieLens
  http://grouplens.org/datasets/movielens/



Modèle Conceptuel de Données



Modèle orienté document

```
{ "id" : 0,                 //Integer
  "year" : "...",           //String
  "title" : "...",          //String
  "imdbId" : "...",         //Integer
  "tmdbId" : "...",         //Integer
  "tags" : [{               //Array of tags
      "userId" : 0,         //Integer
      "tag" : "...",        //String
      "date" : "..."        //String
    }],
  "ratings" : [{            //Array of ratings
      "userId" : 0,         //Integer
      "rating" : "...",     //String
      "date" : "..."        //String
    }]
}
```

Modèle hybride – PGsql 9.4

Movie

Integer **id**
String **year**
String **title**
Integer **imdbId**
Integer **tmdbId**
Jsonb **otherInformations**

```
{
  "tags" : [{
      "userId" : 0,
      "tag" : "...",
      "date" : "..."
    }],
  "ratings" : [{
      "userId" : 0,
      "rating" : "...",
      "date" : "..."
    }]
}
```

Créer le schéma

Colonne JSONB

```
CREATE TABLE movies
(
  id integer primary key,
  name character varying(200),
  year character varying(4),
  imdbid integer,
  tmdbid integer,
  otherInformations jsonb
);
```

```
ALTER TABLE books
   ADD COLUMN "otherInformations" jsonb;
```

Insérer un enregistrement

```
INSERT INTO movies
        (id, name, year, imdbid, tmdbid,
otherInformations)
VALUES (666666, 'Diabolic movie', 1966, 666, 6666,
'{"ratings" : [
        {"userId" : 4,"rating" : 3},
        {"userId" : 5,"rating" : 5},
    ]
}'
);
```

- **Insertion des champs JSON dans un chaine de charactère.**
- **Validation synthaxique effectuée par le moteur**

Construire un champ JSON

```
SELECT row_to_json(movies.*)
FROM movies
LIMIT 1
```

```
-- Résultats
{"id":370,"name":"Naked Gun 33 1/3: The Final Insult
","year":"1994","imdbid":110622,"tmdbid":36593,"otherin
formations":{"tags": [{"tag": "foqam", "date":
1139200469000, "userId": 14260}, {"tag": "Leslie
Neilsen", "date": 1158438794000, "userId": 18390},
{"tag": "nazi", "date": 1161646656000, "userId":
18390}, {"tag": "sequel", "date": 1354555994000,
```

Fonctions de construction JSON

- **to_json :** convertit une valeur
- **row_to_json :** convertit un tuple
- **json_build_array :** convertit un tableau
- **json_build_object :** construit un object JSON depuis un “varidic”
- **json_object :** construit à partir d’une chaine
- **array_to_json :** convertit un tableau

Requêtes

Lecture brute d'un champ

```
-- Lecture brute d'un champ JSON
SELECT otherinformations
FROM movies
WHERE otherinformations IS NOT NULL
LIMIT 2
```

```
-- Résultats
{"tags": [{"tag": "foqam", "date": 1139200469000,
"userId": 14260}, {"tag": "Leslie Neilsen", "date":
1158438794000, "userId": 18390}, {"tag": "nazi",
"date": 1161646656000, "userId": 18390}, {"tag":
"sequel", "date": 1354555994000, "userId": 42640},
{"tag": "Comedy", "date": 1248035546000, "userId":
60710}, {"tag": "parody", "date": 1248035543000,
```

Ce n’est pas du texte

```
-- Requête avec like
SELECT otherinformations
FROM movies
WHERE otherinformations LIKE '%tags%'
LIMIT 2
```

```
//Résultats
Could not execute SQL statement. ERROR: operator does
not exist: jsonb ~~ unknown
  Indice : No operator matches the given name and
argument type(s). You might need to add explicit type
casts.
```

~~Ce n'est pas du texte~~

```
-- Requête avec like et cast
SELECT otherinformations
FROM movies
WHERE otherinformations::text LIKE '%tags%'
LIMIT 2
```

```
-- Résultats
{"tags": [{"tag": "foqam", "date": 1139200469000,
"userId": 14260}, {"tag": "Leslie Neilsen", "date":
1158438794000, "userId": 18390}, {"tag": "nazi",
"date": 1161646656000, "userId": 18390}, {"tag":
"sequel", "date": 1354555994000, "userId": 42640},
{"tag": "Comedy", "date": 1248035546000, "userId":
60710}, {"tag": "parody", "date": 1248035543000,
```

Lecture d'un champ

```
--Lecture d'un champ
SELECT otherinformations->'averageRating'
FROM movies
WHERE (otherinformations->>'averageRating')::float > 4
```

```
-- Résultats
4.00200100050025
4.081620314389359
4.089572192513369
4.216110019646365
4.126740947075209
4.078947368421052
```

Lecture d’un tableau (texte)

```
-- Lecture texte d'un élément de tableau
SELECT otherinformations::jsonb->'tags'->>3
FROM movies
LIMIT 2
```

```
-- Résultats
{"tag": "sequel", "date": 1354555994000, "userId": 42640}
{"tag": "Nudity (Full Frontal)", "date": 1298520169000, "userId": 33860}
```

Lecture d’un tableau (JSON)

```
-- Lecture JSON d'un élément de tableau
SELECT otherinformations::jsonb->'tags'->3->'tag'
FROM movies
LIMIT 2
```

```
-- Résultats
"sequel"
"Nudity (Full Frontal)"
```

Tester l'existence d'un champ

```
-- Tester l'existence d'un champ
SELECT count(*)
FROM movies
WHERE otherinformations ? 'nbOfRatings'
```

```
-- Résultats
1715
```

Recherche JSON

```
-- Recherche JSON
SELECT id, otherinformations->'nbOfRatings'
FROM movies
WHERE
  otherinformations @> '{"nbOfRatings" : 50}'::jsonb
```

```
-- Résultats
1180    50
4000    50
8690    50
```

Lecture avec chemin

```
-- Lecture avec chemin en pseudo JSON
-- Recherche du champ tag du cinquième élément du
tableau tags
-- Equivalent jsonb_extract_path et
jsonb_extract_path_text

SELECT otherinformations #> '{tags, 5, tag}'
FROM movies
LIMIT 2
```

```
-- Résultats
"parody"
"Monty Python"
```

Fonctions de manipulation JSON

- **jsonb_array_length** : nb d'éléments d'un tableau JSONB
- **jsonb_each & jsonb_each_text** : transforme chaque clé valeur en une ligne
- **jsonb_object_keys :** liste des clefs d'un object json

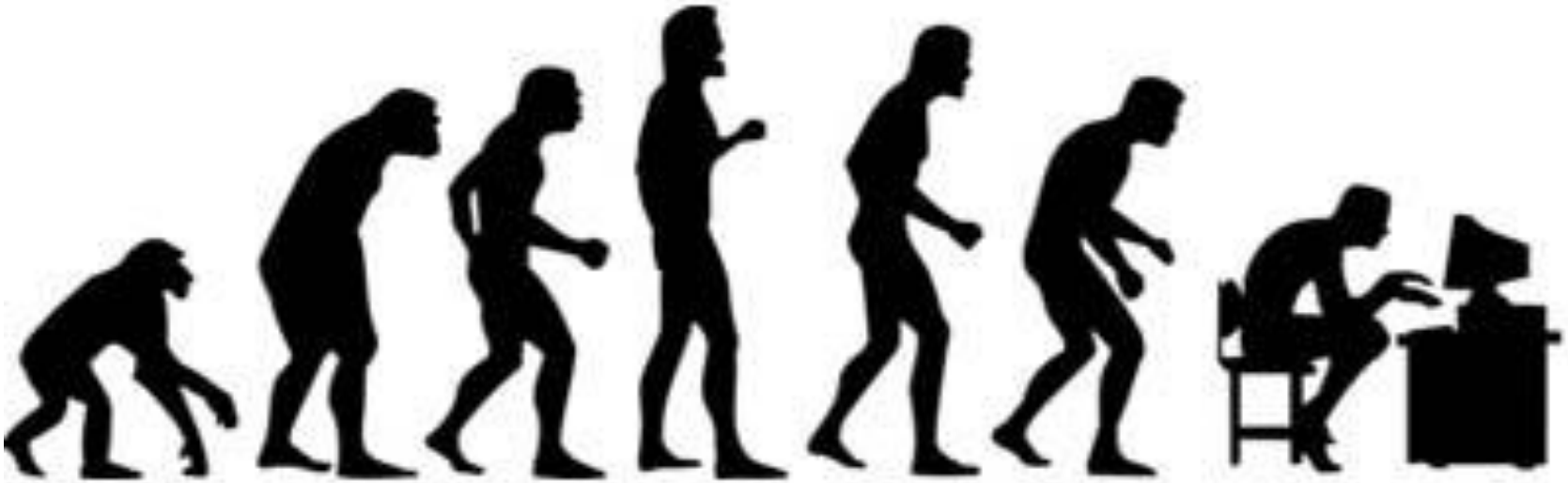
Fonctions de manipulation JSON

- **jsonb_array_elements & jsonb_array_elements_text** : transform un tableau JSON en tuple d’objets
- **jsonb_typeof :** indique le type parmi object, array, string, number, boolean et null.

Fonctions de manipulation JSON

- **jsonb_populate_record & jsonb_populate_recordset** : construit implicitement un tuple
- **json_to_record & json_to_recordset** : construit explicitement un tuple

Démo



Et les indexes ?

Et les indexes ?

```
-- Création d'un index sur l'ensemble du document
CREATE INDEX IDX_GIN_otherInformations
ON movies
USING GIN (otherInformations);
```

Un index gin améliore les requêtes suivantes :

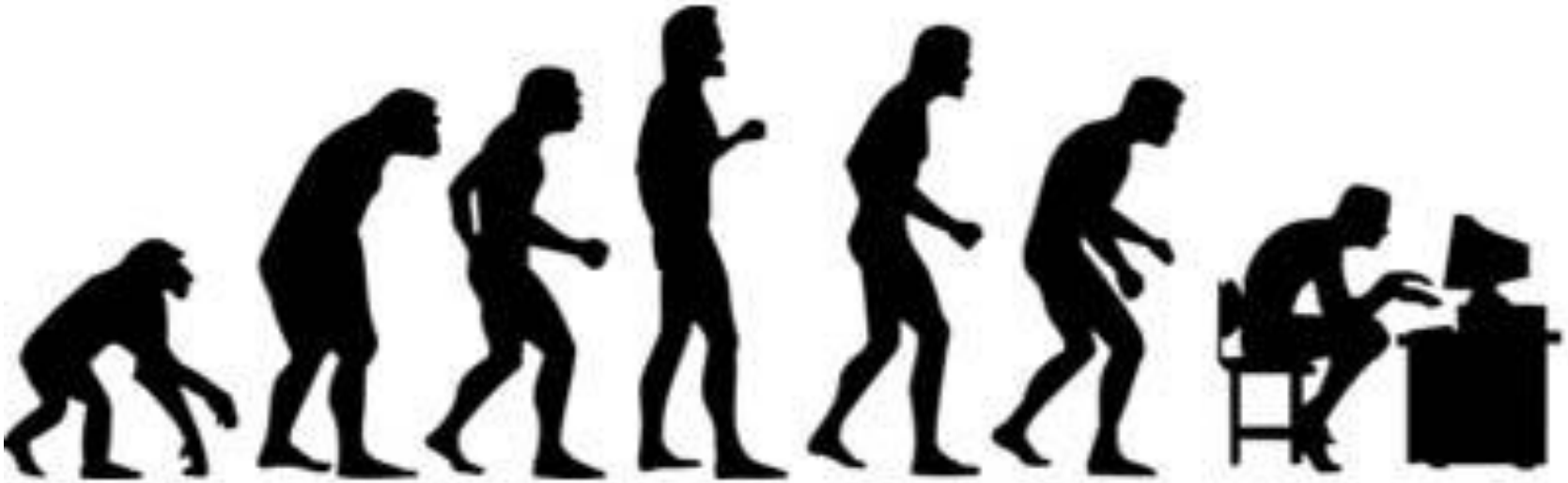
- **JSON @> JSON is a subset**
- **JSON ? TEXT contains a value**
- **JSON ?& TEXT[] contains all the values**
- **JSON ?| TEXT[] contains at least one value**

Et les indexes ?

```
-- Création d'un index sur l'ensemble du document
CREATE INDEX IDX_GIN_otherInformations_opt
ON movies
USING GIN (otherInformations jsonb_path_ops);
```

Paramètre jsonb_path_ops optimise uniquement les recherches avec @>

Démo

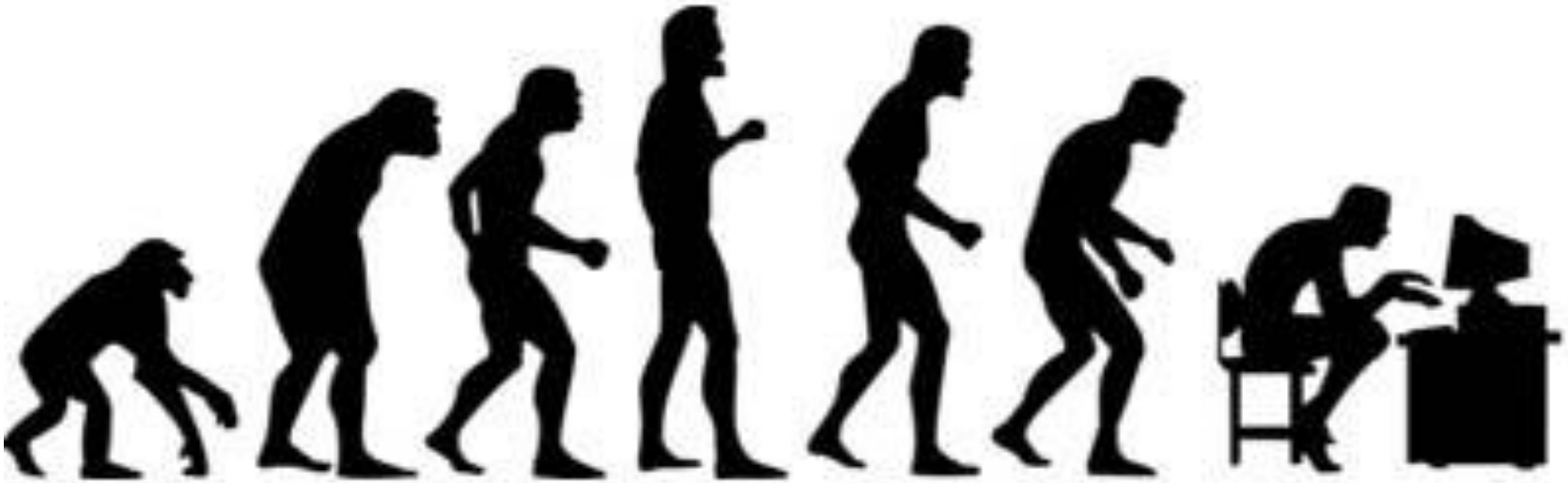


Et en JAVA ?

JDBC

- Un champ JSONB se manipule comme un DBObject ou comme du TEXT
- Pas de particularité dans le driver JDBC PostgreSQL
- Nécessité d’utiliser un mapper objet / JSON

Démo



DAO Insert

```
//Using Jackson to convert additional field to JSON
String ratingsAsString = new
ObjectMapper().writeValueAsString(overallRating);
… pstSetter = new PreparedStatementSetter() {
public void setValues(PreparedStatement preparedStatement)
throws SQLException {
        //Using a PGObject to store otherInformations
        PGobject dataObject = new PGobject();
        dataObject.setType("jsonb");
        dataObject.setValue(ratingsAsString);
        preparedStatement.setObject(1, dataObject);
        …
    }
};
jdbcTemplateObject.update("UPDATE movies SET
otherInformations = ? WHERE id = ?", pstSetter);
```

DAO Select

```
jdbcTemplateObject.query("SELECT id, otherinformations::text
as text_otherinformations FROM movies WHERE otherinformations
is not null limit 2", new RowCallbackHandler() {

public void processRow(ResultSet resultSet) {
  while (resultSet.next()) {
    …
    movie.setId(resultSet.getLong("id"));
    …
    movie.setOther( mapper.readValue(
      resultSet.getString("text_otherinformations"),
        OtherInformations.class));
    …
}}});
```

Avec un ORM

https://github.com/pires/hibernate-postgres-jsonb

Update

Et un update ?

- Postgresql 9.4 ne possède pas nativement d’opérateur permettant de manipuler un champ json
- Nécessité de lire / modifier / updater

Manipulation avec plv8

```
CREATE OR REPLACE FUNCTION json_data_update(data json,
field text,
value text)
RETURNS jsonb
LANGUAGE plv8 STABLE STRICT
 AS $$
 var data = data;
 var val = value;
 data[field] = val;
 return JSON.stringify(data);
 $$;
```

Jsonbx

- Extension jsonbx en cours de portage dans PG 9.5

```
-- Update avec chemin en pseudo JSON

UPDATE movies

SET otherinformations =
jsonb_replace(otherinformations, '{"tags",3,"tag"}',
'{"SUPER"}'::jsonb)
WHERE id = 1
```

Nouveautés PGSql 9.5

```
-- Add values to a jsonb object:
SELECT '{"name": "Joe", "age": 30}'::jsonb || '{"town":
"London"}'::jsonb;

-- Replace
SELECT '{"town": "Dataville", "population":
4096}'::jsonb || '{"population": 8192}'::jsonb;
```

```
-- Résultats
{"age": 30, "name": "Joe", "town": "London"}

{"town": "Dataville", "population": 8192}
```

Nouveautés PGSql 9.5

```
-- Remove a key
SELECT '{"name": "James", "email":
"james@localhost"}'::jsonb - 'email';
```

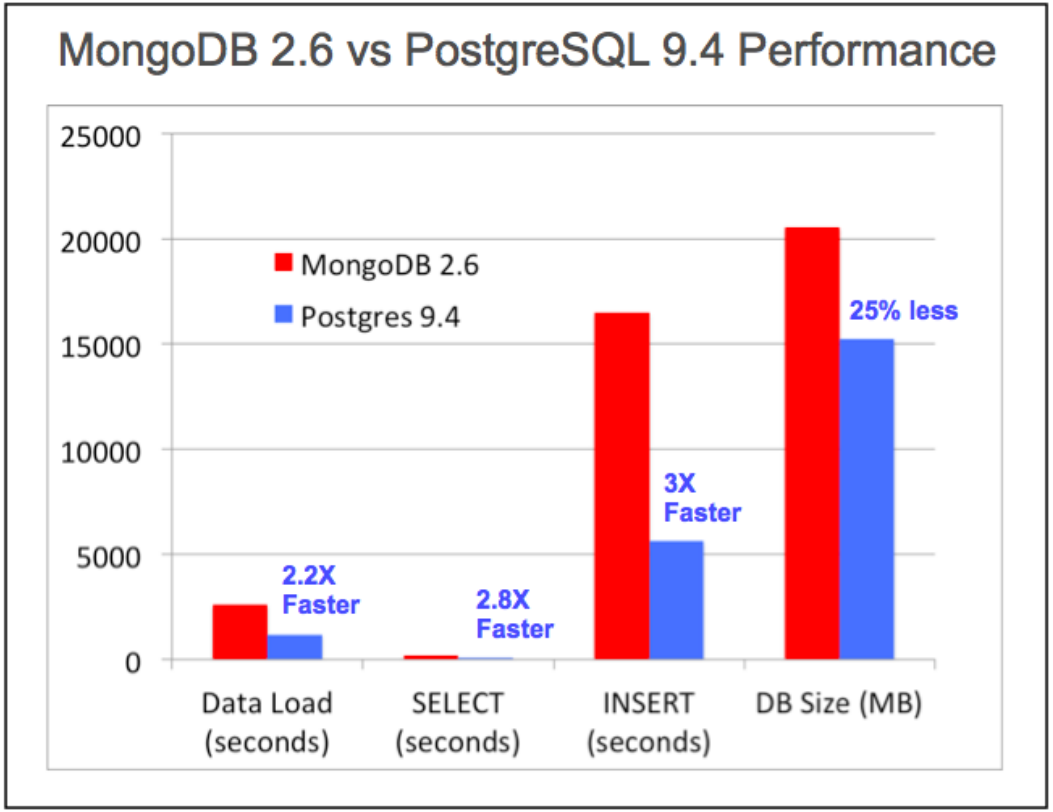
```
-- Résultats
{"name": "James"}
```

PG vs MongoDB

PostgreSQL

mongoDB

JSONB et Postgresql



Source : http://www.enterprisedb.com/nosql-for-enterprise

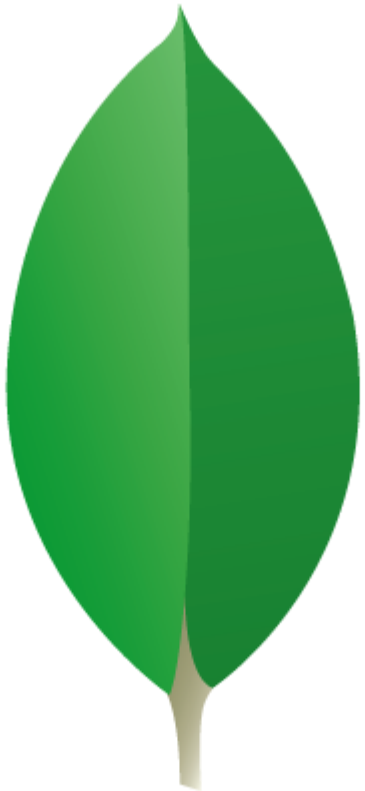
PG vs MongoDB

PG 9.4 répond aux besoins en lecture
Puissance du SQL sous réserve de bien maitriser les fonctions de convertion

PG ne permet pas simplement de mettre à jour un object JSON

PG vs MongoDB ?

Sharding plus facile
Haute disponibilité



Suivre l’actualité

@tapoueh
@petervgeoghegan
@g_lelarge
@EnterpriseDB
@craigkerstiens

http://postgresweekly.com/

http://www.craigkerstiens.com/

https://planet.postgresql.org/

http://blog.2ndquadrant.com/

Questions

#pgnosql

@__pad__

https://github.com/padewitte/bzhcamp_pgjson

Sources

Requete :

- http://www.postgresql.org/docs/9.4/static/functions-json.html
- http://adpgtech.blogspot.fr/2015/05/new-jsonb-features-for-95.html

Index

- http://blog.2ndquadrant.com/jsonb-type-performance-postgresql-9-4/

JSONB et Postgresql

- http://blog.2ndquadrant.com/postgresql-anti-patterns-unnecessary-jsonhstore-dynamic-columns/
- http://www.pgcon.org/2014/schedule/attachments/313_xml-hstore-json.pdf
- https://www.compose.io/articles/is-postgresql-your-next-json-database/