PARIS OPEN SOURCE SUMMIT

DALIBO

Quand utiliser les fonctionnalités NoSQL de PostgreSQL ?

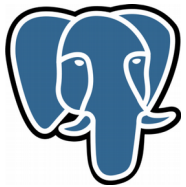
7 dec. 2017

Philippe BEAUDOIN
Consultant

L'expertise PostgreSQL

Antinomique, non ?

Postgre SQL



No SQL

Relationnel et mouvement « NoSQL »

Relationnel

- Réponse aux besoins de
  - Gestion de données très structurées
    - Tables, lignes x colonnes
  - Gestion de l'intégrité
    - propriétés ACID
- Langage SQL efficace

NoSQL

- Réponse aux besoins de
  - Gestion de données peu/non structurées
    - Clés / valeurs
    - Documents complexes
- Privilégie la performance en écriture et la scalabilité à l'intégrité

Et si on prenait le meilleur des 2 mondes ?

Coopération Relationnel ↔ NoSQL

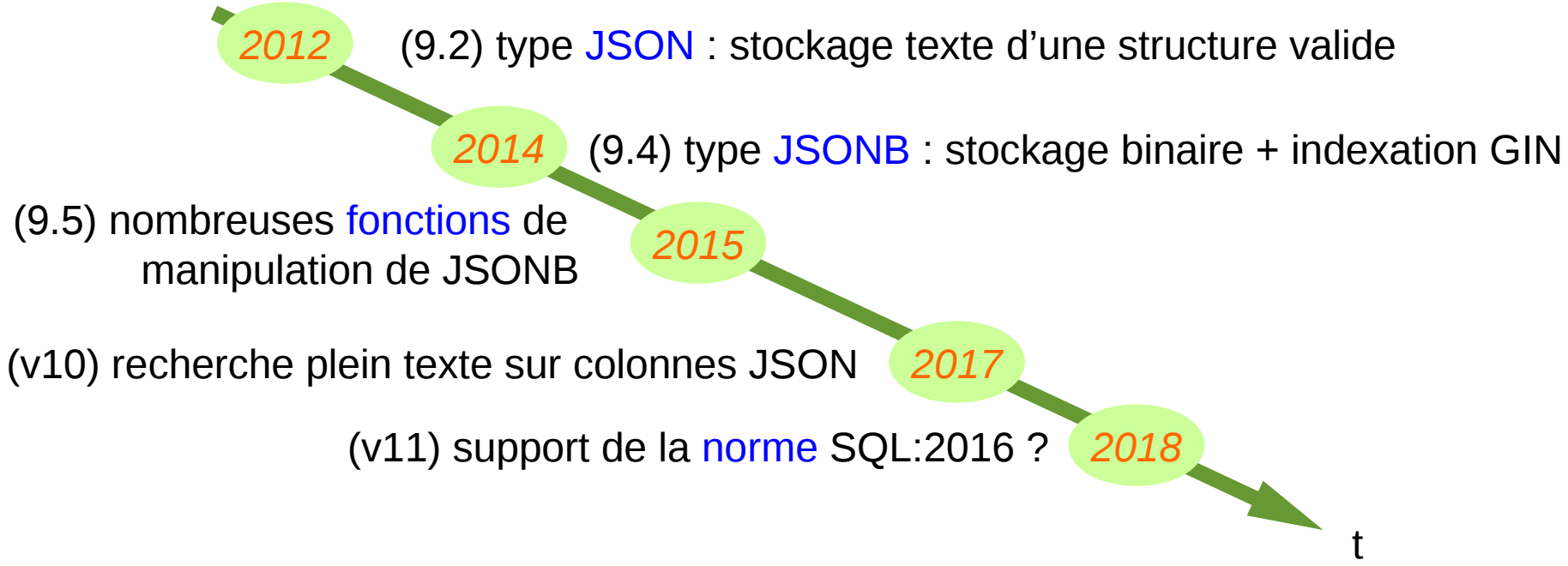
YeSQL ⟵ ***N**ot **O**nly SQL* ⟶ *NoSQL*

Moteur relationnel pur	Moteur relationnel avec des fonctionnalités NoSQL	Moteur NoSQL connecté à un moteur relationnel	Moteur NoSQL pur
			
		Foreign Data Wrappers	

Ce qui existe dans PostgreSQL de longue date

- ARRAY : Une colonne peut contenir un tableau de valeurs
  - *ma_colonne INT [ ]*
  - Le type peut aussi être une structure : *ma_colonne mon_type [ ]*
- Type XML
  - Natif depuis 8.3 (2008)
- Extension HSTORE
  - Stockage clé / valeur
  - Simple et efficace
  - Depuis 8.2
  - Idéal pour des grandes structures avec peu de valeurs

Puis vint le support de JSON...



- 15 opérateurs et 24 fonctions de manipulation de JSON et JSONB

Un exemple JSON : structure et insertion

Créer une table

```
CREATE TABLE tbl_json (…, col_json JSONB,  ...);
```

Insertion de donnée JSON dans la table

```
BEGIN;
INSERT INTO tbl_json (…, col_json, ...) VALUES
   (…, ' { "nom": "Apple Phone", "type": "phone", "marque": "Apple",
          "prix": 500, "disponibilite": true, "garantie_annees": 1 } ', ...);
COMMIT;
```

Créer un index GIN sur la colonne JSON (pour une indexation vraiment schema-less)

```
CREATE INDEX idx_j ON tbl_json USING GIN (col_json jsonb_path_ops);
```

Créer un index UNIQUE de type BTREE sur l'attribut 'nom'

```
CREATE UNIQUE INDEX uidx_nom ON tbl_json ((col_json->'nom'));
```

Ajouter une contrainte sur l'attribut 'type'

```
ALTER TABLE tbl_json ADD CONSTRAINT type_existe CHECK (col_json ? 'type');
```

Un exemple JSON : utilisation des données

SQL

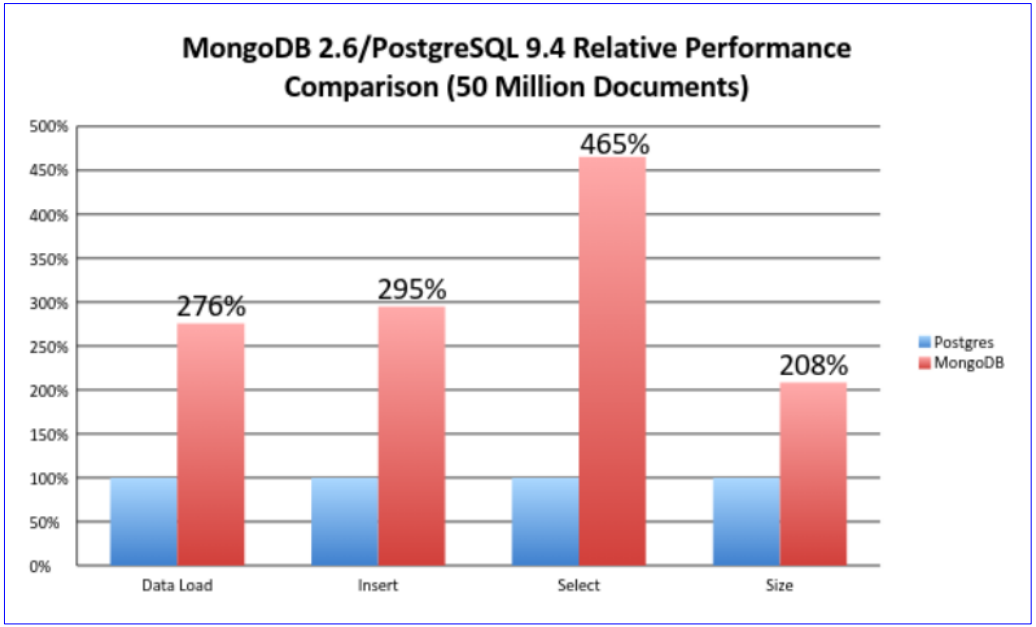
```
SELECT DISTINCT
   nom_produit,
   data->>'marque' as Marque,
   data->>'disponibilite' as Dispo
FROM tbl_json
JOIN produit
   ON (produit.nom_produit = col_json->>'nom')
WHERE col_json ->>'disponibilite' = true;
```

JSON

```
nom_produit         | Marque     | Dispo
--------------------+------------+-------
Apple Phone         | Apple      | true
```

Et les performances ?

- 2014 - Benchmark EnterpriseDB => PostgreSQL meilleur que MongoDB (1 nœud) sur les temps d'accès et l'espace disque
  - ★ Pas d'inquiétude sur la performance intrinsèque du moteur



https://www.enterprisedb.com/node/3441

- La réalisation des jointures par le SGBD évite des transferts de données coûteux avec l'application

Intérêt du JSON dans des tables relationnelles

- Un modèle de données où cohabitent données très structurées et données peu structurées
- Souplesse dans la conception des applications / bases de données
  - Structuration progressive sans rupture de technologie
- Pas besoin d'ajouter de la logique dans les applications pour gérer la complexité des accès
- Intégrité des données
  - Les propriétés ACID
  - Les contraintes (unicité, check, exclusion)
- On conserve le langage SQL
  - Efficace, connu et normalisé

Quand ne pas utiliser PostgreSQL pour le NoSQL ?

- Que des données non structurées et pas besoin d'intégrité
- Très gros volumes ou très grosses charges pour lesquels la scalabilité horizontale est indispensable
  - Mais la scalabilité verticale de PostgreSQL est importante

SELECT questions FROM audience;

Philippe.beaudoin@dalibo.com
+33 7 69 14 67 21

www.dalibo.com