# PostgreSQL 9.x: une nouvelle ère!

Jean-Paul Argudo
Dalibo
*L'Expertise* PostgreSQL
Paris, France

http://www.solutionslinux.fr/

DALIBO
*L'Expertise* PostgreSQL

# Agenda

# À propos ... de moi

- Fondateur du site PostgreSQL.fr (février 2004)
- Co-fondateur et secrétaire de PostgreSQLFr (février 2005)
- Co-fondateur et trésorier de PostgreSQL Europe (depuis 2008)
- Gérant de Dalibo SARL

# À propos ... de ma société

- Dalibo, Expertise PostgreSQL en France depuis 2005
- Plus de 250 clients, des dotcoms aux très grosses sociétés, dont des institutions publiques (ministères, etc)
- 3 métiers principaux:
  - conseil
  - formation
  - support
- Quelques clients


DALIBO
L'Expertise PostgreSQL

# Introduction

- PostgreSQL et NoSQL ?
- Avant la 9
- Version 9 et suivantes
- Récents cas d'utilisation
- Conclusion

# NoSQL

- Not-Only-SQL ou No-SQL ?
- ACID-less
- Big Data ?
- Cassandra, BigTable, MongoDB, CouchDB et d'autres

# clé/valeur: c'est vieux chez PostgreSQL

```
                    Release 8.2
     Release Date: 2006-12-05
[...]
Contrib Changes
[...]
     * Add hstore module (Oleg, Teodor)
```

# hstore: installation

```
montest=# create extension hstore;
CREATE EXTENSION
montest=# select 'a=>b, b=>1, c=>NULL'::hstore
@> 'a=>b' as resultat_test;

 resultat_test
---------------
 t
(1 row)
```

# hstore est complet

- plus de 10 opérateurs
- plus de 20 fonctions
- indexable avec GiST et GIN
  - `CREATE INDEX hidx_gist ON matable USING GIST (h);`
  - `CREATE INDEX hidx_gin ON matable USING GIN (h);`

# Exemples avec hstore (1/5)

Ajouter une clé, ou mettre à jour une clé existante avec une nouvelle valeur:

```
UPDATE tab SET h = h || hstore('c', '3');
```

Supprimer une clé:

```
UPDATE tab SET h = delete(h, 'k1');
```

# Exemples avec hstore (2/5)

Convertir un type record en un hstore:

```
CREATE TABLE test (col1 integer, col2 text,
col3 text);
INSERT INTO test VALUES (123, 'foo', 'bar');
SELECT hstore(t) FROM test AS t;

                    hstore
-----------------------------------------
 "col1"=>"123", "col2"=>"foo", "col3"=>"bar"
(1 row)
```

# Exemples avec hstore (3/5)

Convertir un type hstore en un type record prédéfini:

```
CREATE TABLE test (col1 integer, col2 text,
col3 text);
SELECT * FROM populate_record(null::test,
'"col1"=>"456", "col2"=>"zzz"');
```

```
 col1 | col2 | col3
------+------+------
  456 | zzz  |
(1 row)
```

# Exemples avec hstore (4/5)

Modifier une enregistrement existant en utilisant les valeurs provenant d'un hstore:

```
CREATE TABLE test (col1 integer, col2 text,
col3 text);
INSERT INTO test VALUES (123, 'foo', 'bar');
SELECT (r).* FROM (SELECT t #= '"col3"=>"baz"'
AS r FROM test t) s;

 col1 | col2 | col3
------+------+------
  123 | foo  | baz
(1 row)
```

# Exemples avec hstore (5/5)

Statistiques en ligne:

```
SELECT key, count(*) FROM
(SELECT (each(h)).key FROM testhstore) AS stat
GROUP BY key
ORDER BY count DESC, key;
```

```
    key     | count
------------+-------
 line       |   883
 node       |   202
 title      |   190
 org        |   189
 [...]
```

# JSON 1/2

- type de données natif (validation des données. . . )
- avant: `text`
- inclus dans PostgreSQL 9.2

# JSON 2/2

- ```
  select ('{"a":123}')::json;
  -[ RECORD 1 ]---
  json | {"a":123}
  ```
- ```
  select array_to_json(array(
       select z
       from (select i, i*2 as j
       from generate_series(1,4) i) as z));
                        array_to_json
  ------------------------------------------------------------
   [{"i":1,"j":2},{"i":2,"j":4},{"i":3,"j":6},{"i":4,"j":8}]
  (1 row)
  ```
- ```
  $ select *, row_to_json(r) from t as r;
   i | t |    a    |          row_to_json
  ---+---+---------+-------------------------------
   1 | a | {1,2,3} | {"i":1,"t":"a","a":[1,2,3]}
   2 | b | {2,3,4} | {"i":2,"t":"b","a":[2,3,4]}
  (2 rows)
  ```
  ```

# PL/v8 1/3

- basé sur le V8 Engine de Google
- procs. stocks. rapides en Javascript
- création d'attributs d'index ad-hoc sur des données JSON
- inclus dans PostgreSQL 9.2

# PL/v8 2/3

```
CREATE TYPE rec AS (i integer, t text);
CREATE FUNCTION set_of_records() RETURNS SETOF rec AS
$$
    // plv8.return_next() stores records in an internal tuplestore,
    // and return all of them at the end of function.
    plv8.return_next( { "i": 1, "t": "a" } );
    plv8.return_next( { "i": 2, "t": "b" } );

    // You can also return records with an array of JSON.
    return [ { "i": 3, "t": "c" }, { "i": 4, "t": "d" } ];
$$
LANGUAGE plv8;

SELECT * FROM set_of_records();
 i | t
---+---
 1 | a
 2 | b
 3 | c
 4 | d
(4 rows)
```

# PL/v8 3/3

## PL/v8 et JSON en action dans PostgreSQL:

```
CREATE FUNCTION to_jsontext(keys text[], vals text[]) RETURNS text AS
$$
        var o = {};
        for (var i = 0; i < keys.length; i++)
                o[keys[i]] = vals[i];
        return JSON.stringify(o);
$$
LANGUAGE plv8 IMMUTABLE STRICT;

SELECT to_jsontext(ARRAY['age', 'sex'], ARRAY['21', 'female']);
         to_jsontext
-----------------------------
 {"age":"21","sex":"female"}
(1 row)
```

# PostgreSQL 8.4: Backend

- version stable et éprouvée
- archivage et Point In Time Recovery
- Warm Standby
  - maître en lecture/écriture
  - esclave fermé
  - replication par rejeu des journaux de transaction

# PostgreSQL 8.4: Performances et Internationalisation

- amélioration de GIN
- restauration en parallèle
- collation par base de données

# PostgreSQL 8.4: Divers

- statistiques sur les fonctions
- trigger sur TRUNCATE
- Window Functions et CTE

et bien d'autres choses encore!

# PostgreSQL 9.0: Backend

- Hot Standby: ouverture de l'esclave aux lectures seules
- Streaming Replication: réplication en quasi temps réel
- Amélioration sur Vacuum Full

# PostgreSQL 9.0: Performances

- version Windows 64 bits
- jointures inutiles supprimées (ORMs)
- IS NOT NULL peut utiliser les index

# PostgreSQL 9.0: Divers

- améliorations importantes sur les langages procéduraux
- configuration par couple utilisateur / base de données
- sortie XML, JSON et YAML pour EXPLAIN
- authentification via RADIUS

La liste complète des améliorations sur la page wiki du projet:
http://wiki.postgresql.org/wiki/What's_new_in_PostgreSQL_9.0

# PostgreSQL 9.1: Backend

- Streaming Replication désormais synchrone
- UNLOGGED table
- Extension

# PostgreSQL 9.1: Performances

- Indexation KNN (K-Nearest-Neighbor)
- amélioration des performances en écriture (Synchronous writes)
- amélioration sur les performances des tables partitionnées

# PostgreSQL 9.1: Divers

- SE-Postgres
- PGXN
- SQL/MED et Foreign Data Wrappers
- auto-tunning des wal_buffers
- option ENCODING sur COPY TO/FROM

La liste complète des améliorations sur la page wiki du projet:
http://wiki.postgresql.org/wiki/What's_new_in_PostgreSQL_9.1

# PostgreSQL 9.2: Backend

- Index-only scans (select count(*)...)
- Cascading Replication
- fonctions de monitoring intégrées

# PostgreSQL 9.2: Performances

- scalabilité pour systèmes multi-processeurs
- amélioration des tris en mémoire
- SP-GiST (Space GiST)

# PostgreSQL 9.2: Divers

- type de données RANGE
- type de données JSON (array-to-json, row-to-json, etc)
- réduction de la consommation de ressources (électricité..)

La liste complète des améliorations sur la page wiki du projet:
http://wiki.postgresql.org/wiki/What's_new_in_PostgreSQL_9.2

# Le moteur d'Orange

Indexation de documents pour le moteur de recherche

- 5 milliards de tuples répartis sur
- 160 machines qui abritent 800 serveurs PostgreSQL
- volumétrie totale de 24 téra-octets
- répartition géographique

Cas d'utilisation complet sur:
http://www.postgresql.fr/temoignages:moteur_orange

# Le Bon Coin

Site de vente de particulier à particulier.
*Un seul* serveur abrite tout.

- DL980
  - 160 coeurs
  - 1 téra-octet de RAM
- baie 3PAR V800
  - 192 disques (dont 32 SSD et 160 fiber-channel)
  - 400 giga-octets de cache

Cas d'utilisation complet sur:
http://www.postgresql.fr/temoignages:le_bon_coin

# La CNAF

Migration de Bull RFM sous Gecos 8 vers PostgreSQL sous Red-Hat Linux

- 168 bases PostgreSQL
- volume global de 4 To de données.
- plus d'un milliard de requêtes SQL par jour

Cas d'utilisation complet sur: http://www.boursier.com/actions/actualites/news/bull-modernise-les-applications-coeur-de-metier-de-la-cnaf-401344.html

# Conclusion

- PostgreSQL est prêt pour NoSQL
- . . . avec la garantie d'ACID pour vos données!
- un projet très actif
- des fonctionalités inédites
- des utilisateurs sérieux