Postgres Unstructured 調査報告書

琉球大学理工学研究科 徳森海斗

2014 年 9 月 9 日

1 Postgres Unstructured 概要

Postgres Unstructured は EnterpriseDB 社による プレゼン [1] タイトル内の言葉で, NoSQL に対応した PostgreSQL を指す. Postgres 9.4 beta2 時点では HStore, json, jsonb という三つのデータ型が非構造なデータを扱うことを可能にしている. HStore は Postgres 8.2 から存在する Key-Value pair のデータ型であり, json 型, jsonb 型はそれぞれ 9.3, 9.4 で新たに加わった json を扱うデータ型である. 本報告書では jsonb 型 json 型について中心に述べる.

2 Postgres 9.4 beta2 環境設定

EnterpriseDB 社のページにインストーラが用意されているのでそれを利用する [2]. Mac OS X, Windows, Linux 向けにそれぞれ app, exe, run ファイルが用意されており, 実行するだけでインストールが完了する. インストーラにより, データベース用ディレクトリの作成, Postgres ユーザの追加までサポートされる. ただしパスの通ってないディレクトリをインストール先に指定した場合はパスを通す必要がある.

インストール後, createdb コマンドでデータベースを作成, psql [db name] でデータベースを操作できる. デフォルトではデータベースのユーザーには postgres しか存在しないので, ユーザーが postgres 出ない場合は -U オプションで postgres ユーザーを指定する必要がある.

サーバプロセスの管理は pg_ctl コマンドによって行う. こちらの操作も postgres ユーザーでしか行えない. 以下のように使用する.

```
% pg_ctl -D [database path] [start | stop | status | restart]
```

port や listen address の設定はデータベースに指定したディレクトリ以下の data/postgresql.conf を編集することで行える. それぞれ port=****,listen_addresses=****を任意の値に変更する. この設定は再起動するまで反映されない.

3 json, jsonb 型

jsonb 型は postgres の持つ json のバイナリ表現である. mongoDB の BSON とは異なり, 64bit 以上の数値を正確に表現することが可能という利点を持つ. jsonb は json の内部表現というわけではなく, json 型と jsonb 型はそれぞれ独立して存在する. jsonb 型を利用する際, 入力として与える値は json であるので, 型を指定する以外は特別 jsonb 型であることを意識する必要はない. jsonb 型を格納する際, 与えられた json を

jsonb へと変換する処理が入るので 値の格納に関しては json よりやや速度が劣る．しかし, jsonb のみが利用できる演算子の存在, 値の処理は jsonb の方が速いという利点が存在する．

その他の違いとして, json 型は文字列をそのまま格納するのに対し jsonb 型は一度変換を介すので, jsonb 型は整形されるという点がある．そのため, 1.230e-5 のように小数の表現に e を使ったものも 0.0000123 と直される．

また, json,jsonb 型に共通して, 数値に NaN, infinity を使用することができない．これらの値をもつ json は別の値に変換する必要がある．

4 json, jsonb とデーブルデータ間の変換

テーブルデータを json に変換するのには to_json 関数を用いる．この関数はテーブルの項目, 値をそれぞれキー, 要素とする json に変換する関数ある．以下に使用例と実行結果を示す．

```
=# SELECT to_json(table_name.*) FROM table_name;
                                    to_json
-------------------------------------------------------------------------------
 {‘‘city’’:’’okinawa’’,’’temp_lo’’:20,’’temp_hi’’:30,’’prcp’’:0.42,’’date’’:’’2014-08-30’’}
 {‘‘city’’:’’okinawa’’,’’temp_lo’’:20,’’temp_hi’’:30,’’prcp’’:0.42,’’date’’:null}
 {‘‘city’’:’’naha’’,’’temp_lo’’:20,’’temp_hi’’:30,’’prcp’’:0.42,’’date’’:null}
```

json をテーブルデータに変換するには json_populate_record 関数,json_populate_recordset 関数を用いる．この関数は引数としてベースとなるテーブルと json を求め, ベースとなるテーブルに与えられた json を対応させていく．このときベースに存在しない項目は無視され, json に存在しない項目には null が格納される．以下の使用例と実行結果を示す．

```
=# SELECT * FROM json_populate_recordset(NULL::table_name, (SELECT array_to_json(array_agg(json.data)
    ) FROM json_table_name));
  city   | temp_lo | temp_hi | prcp |    date
---------+---------+---------+------+------------
 okinawa |      20 |      30 | 0.42 | 2014-08-30
 okinawa |      20 |      30 | 0.42 |
 naha    |      20 |      30 | 0.42 |
```

それぞれの変換は全てのデータを読み, 変換して INSERT しているので変換以外の部分でも時間がかかる．上記のような項目のデータが十万件存在するデータを変換のみ行った場合, json からテーブルデータへの変換に 738.362 ミリ秒, テーブルデータから json への変換に 603.162 ミリ秒かかった．jsonb へ変換する場合はこれに加えて json から jsonb への変換が入るので, 合計で 1711.625 ミリ秒かかるという結果になった．

5 json, jsonb 型のフィールドへのアクセス

json, json のフィールドへのアクセスには’->’,’->>’ 演算子を用いる．前者はフィールドを単に取得するもので, 後者はテキストとして取得する．フィールドの値を SELECT に用いるのは容易で, 以下のように使用する．

```
=# SELECT json_data FROM json_jsonb;
                                      json_data
--------------------------------------------------------------------------------
 {‘‘string’’:’’Unstructuerd Postgres’’, ‘‘number’’:1.230e-5, ‘‘boolean’’: true, ‘‘null’’:null}
 {‘‘string’’:’’NULL CHECK’’, ‘‘number’’:1.230e-5, ‘‘boolean’’: true, ‘‘null’’:null}
 {‘‘number’’:2.345e+5,’’boolean’’:false,’’null’’:null,’’string’’:’’aua’’}
=# SELECT json_data->’string’ FROM json_jsonb;
        ?column?
```

```
------------------------
 ‘‘Unstructuerd Postgres’’
 ‘‘NULL CHECK’’
 ‘‘aua’’
```

WHERE の条件に利用する場合は型に気をつける必要があり, フィールドの型と比較先の値の型を揃える必要がある. 例えばフィールド内の数値と 100 を比較したい場合, フィールドの値は'->>' 演算子を用いてテキストとして取得し, 右辺の 100 はシングルクォートで囲い文字列扱いにする必要がある. '->' 演算子で取得した値は json 型になるので比較できない.

Listing 1　エラーが出る例

```
=# SELECT json_data->’string’ FROM json_jsonb WHERE jsonb_data->>’number’ = 234500;;
ERROR:  operator does not exist: text = integer
```

Listing 2　正しい例

```
ELECT json_data->’string’ FROM json_jsonb WHERE jsonb_data->>’number’ = ’234500’;
        ?column?
------------------------
 ‘‘Unstructuerd Postgres’’
```

6　json, jsonb 型のデータの更新

json 全体の UPDATE は通常の値を更新するのと同様に UPDATE を用いて行うことができる. しかし, 以下のように'->','->>' 演算子を利用してフィールドの値を直接更新することはできず, Postgres 公式ドキュメント [3] にもその方法は記載されていない.

Listing 3　エラーが出る例

```
=# UPDATE table_name SET json->’a’ = to_json(5) WHERE json->>’b’ = ’2’;
ERROR:  syntax error at or near ‘‘->’’
```

以下に示すリスト 4 のような関数を利用することで更新は可能であるが, この場合, 複数の json 内のフィールドの値を更新することはできない. (関数は stackoverflow より引用 [4]).

Listing 4　json_object_set_key

```
CREATE OR REPLACE FUNCTION ‘‘json_object_set_key’’(
  ‘‘json’’          json,
  ‘‘key_to_set’’    TEXT,
  ‘‘value_to_set’’  anyelement
)
  RETURNS json
  LANGUAGE sql
  IMMUTABLE
  STRICT
AS $function$
SELECT COALESCE(
  (SELECT (’{’ || string_agg(to_json(‘‘key’’) || ’:’ || ‘‘value’’, ’,’) || ’}’)
     FROM (SELECT *
             FROM json_each(‘‘json’’)
            WHERE ‘‘key’’ <> ‘‘key_to_set’’
            UNION ALL
           SELECT ‘‘key_to_set’’, to_json(‘‘value_to_set’’)) AS ‘‘fields’’),
  ’{}’
)::json
$function$;
```

この関数は次のように利用する.

```
UPDATE json_jsonb SET SELECT json_object_set_key((SELECT json_data FROM json_jsonb WHERE json_data->>
    'condition_key'='condition_value'),'key_name','new_value'::text)) WHERE json_data->>'
    condition_key'='condition_value';
```

7 json,jsonb 型での INDEX

json, jsonb 型どちらでもフィールドに対する index の作成が可能で, 作成のための CREATE INDEX 文は次の二通りの記述方法がある.

```
CREATE INDEX index_name ON table_name ((json_column_name->>'key'));
CREATE INDEX index_name ON table_name ((json_extract_path_text(json_column_name, 'key')));
```

json 型の場合, json データへのインデックスは作成できない. jsonb 型へはインデックスの作成が可能で, `@>`, `?`, `?&`, `?|` の 4 つの演算子がサポートされる.

```
CREATE INDEX index_name ON table_name USING GIN (jsonb_column_name);
```

また, 以下のようにすると `@>`のみをサポートする index を作ることができる.

```
CREATE INDEX index_name ON table_name USING GIN (jsonb_column_name jsonb_path_ops);
```

尚, 各演算子の意味は次のようになっている.

演算子	説明
`@>`	指定した json を含むかどうか.
`?`	指定したキーを持つかどうか.
`?&`	指定したキーをすべて持つかどうか.
`?|`	指定したキーのうちどれか一つでも持つか.

8 まとめ

Postgres は 9.4 beta2 時点で HStore, json, jsonb の三種の方により Unstructuerd data に対応している. 既存のテーブルデータと json, jsonb 間の変換が容易なので, json で受け取ったデータを RDB で管理するといった使い方もできる.

json, jsonb 型を用いて NoSQLDB として使う場合はフィールドの値を簡単に更新できないことが課題になるだろう. 先に作られた HStore 型の方はフィールドの値の更新に対応しているので, HStore 型で表現できるのならば HStore 型を用いるのが良い.

また, '`?`','`?|`','`?&`' の三つの演算子はドキュメントでは要素の検索も可能となっているが実際にはキーの検索しかできず, ドキュメント側のミスなのか開発段階なのかの判断ができない.

同様に json を扱う mongoDB と比較した場合, EnterpriseDB 社の発表 [1] によると処理速度, データベースサイズ等のパフォーマンス面では Postgres が勝るようだが, 前述した json のフィールドの更新が未対応である点に使いづらさを感じた.

この点を考えると, json を扱うためのデータベースとして利用するにはまだ早いと感じた. NoSQL と SQL の両方が利用できるという強みを活かせる使い方をするのがいいだろう.

参考文献

[1] NoSQL on ACID - Meet Unstructured Postgres.
“http://www.slideshare.net/EnterpriseDB/no-37327319”

[2] Download PostgreSQL.
“http://www.enterprisedb.com/products-services-training/pgdownload”

[3] PostgreSQL 9.4beta2 Documentation.
“http://www.postgresql.org/docs/9.4/static/”

[4] stackoverflow - How do I modify fields inside the new PostgreSQL JSON datatype?.
“http://stackoverflow.com/questions/18209625/how-do-i-modify-fields-inside-the-new-postgresql-json-datatype”