

何が違うのか？ PostGISと最新版MySQLの GIS機能を徹底比較

2015年11月27日 PostgreSQLカンファレンス2015

国府田 諭 Satoshi Koda

(埼玉大学大学院 理工学研究科 博士後期課程, satkouda@gmail.com)

本資料の掲載場所 <http://kenpg.bitbucket.org/blog/201511/29.html>

本日の内容

- それぞれの最新版でGeoJSONデータを処理しながら比較
 - 対象 : MySQL Community Server 5.7.9
PostgreSQL 9.5 Beta 2 + PostGIS 2.2.0
 - 使うデータ : 「MAPZEN」の世界の行政界から、日本のもの
 - さらに、
 - » PostgreSQL 9.5 で便利になった外部テーブルのインポート例
 - » 統計処理言語Rを組み込むPL/Rの利用例
- … 時間があれば紹介

用語等について

- GIS : Geographic Information System 地理情報システム
- 情報の呼び方はいろいろあるけど、今日は特に区別しません
 - ・ GISデータ
 - ・ 地理データ
 - ・ 空間データ
 - ・ 位置情報データ etc.
- MySQL [のGIS機能](#) と毎回言うのは面倒なので、省略する場合があります
- 処理時間を示した箇所は、だいたい10回くらい繰り返して同様の結果になったものから、ある1回を例示してます

最初に、PostGIS と MySQL のあらし

PostGISとは

- PostgreSQLの拡張機能
GIS用のデータ型、関数、若干のテーブルやビューを追加
- 2001年：最初のバージョン0.1
2005年：1.0.0リリース（当時、PostgreSQLは8.0）
…… だいたい1年おきに 1.1 -> 1.2 -> 1.3 -> 1.4 -> 1.5
2012年：大幅に進化したバージョン2.0へ
…… その後もバージョンアップ。2015年10月、2.2リリース
- 詳しくは：Introduction to PostGIS 第1章（日本語）
<http://workshops.boundlessgeo.com/postgis-intro-jp/introduction.html>
- 最新2.2開発版のマニュアル日本語訳
<http://www.finds.jp/docs/pgisman/2.2.0/index.html>

MySQL の Extensions for Spatial Data

- 拡張機能でなく、ビルトインされている GIS用のデータ型と関数
- Extensionsというのは、標準的なSQLやRDBMSに対しての意味。
詳しくは：

MySQL 5.6 Reference Manual - 空間データの拡張（日本語）

<http://dev.mysql.com/doc/refman/5.6/ja/spatial-extensions.html>

- 2004年：MySQL 4.1で追加
……（略）……
2014年：MySQL 5.7.5（GeoJSON用関数など追加）
2015年：MySQL 5.7.6（2点の経緯度から距離を出す関数など）
- データの内部形式がPostGISと異なる → ダンプファイルでの互換性なし

PostGISが、MySQLへの拡張でなかった理由

- 先ほど紹介の ↓ いわく

Introduction to PostGIS 第1章（日本語）

<http://workshops.boundlessgeo.com/postgis-intro-jp/introduction.html>

オープンソースデータベースをよく知る人々からよく聞かれるのが、“なぜPostGISをMySQL上で構築しなかったのか？”です。

(... PostgreSQLの特徴を列挙 ...)

総じて、PostgreSQLにより、新しい空間型の追加が容易となる開発工程を経ることができると言えます。

(... 中略 ...)

MySQLが、バージョン4.1から基本的な空間型をリリースしたとき、PostGISチームはそのコードを見て、そしてまた実際に使ってみて、改めてPostgreSQLを使用しようという決意を強めました。

最新版ではどうなのか？ 実際に使ってみる

実行環境

- ハードウェア（ストレージ：Crucial M500 SSD）

Rating:

7.1

Windows Experience Index

Processor:

Intel(R) Core(TM) i7-3720QM CPU @ 2.60GHz 2.60 GHz

Installed memory (RAM):

16.0 GB

System type:

64-bit Operating System

- Windows 7 64bit

MySQL Community Server 5.7.9

PostgreSQL 9.5 Beta 2 + PostGIS 2.2.0

- ともにZIP版を使い、バッチファイルでサーバ起動（サービスでない）

設定はデフォルトのまま

「チューニングしないと使えない」では、GISユーザにとって敷居が高すぎ

PostGIS : まずインストールから

- Linux … パッケージ管理システムで楽々
- Mac … postgres.appなら最初から入ってる
- Windows … スタックビルダが使える

ただしPostgreSQLがサービスになっている前提らしい

手動インストール用ファイル → http://postgis.net/windows_downloads/



Windows: Winnie Bot PostGIS and pgRouting Experimental Builds

- [for PostgreSQL 9.5beta2 32-bit/64-bit](#) compiled against PostgreSQL 9.5beta2 for PostGIS 2.2, 2.3 (built with SFCGAL support) (also pgrouting 2.1) (osm2pgrouting 2.1), (in extras folder: [pointcloud](#))
- [for PostgreSQL 9.4 32-bit, 64-bit](#) compiled against PostgreSQL 9.4 for PostGIS 2.2, 2.3 (built with SFCGAL support) (also pgrouting 2.0, 2.1) (osm2pgrouting 2.1)
- [for PostgreSQL 9.3 32-bit, 64-bit](#) compiled against

PostGIS : データベース作成と準備

```
create database test_gis
  with encoding 'UTF8'
  lc_collate = 'C'
  lc_ctype = 'C';
```

create extension postgis; -- この DB に PostGIS を導入

```
select postgis_full_version(); -- バージョン確認
```

```
-----
POSTGIS="2.2.0 r14208" GEOS="3.5.0-CAPI-1.9.0 r4090" PROJ
="Rel. 4.9.1, 04 March 2015" GDAL="GDAL 2.0.1, released 2
015/09/15" LIBXML="2.7.8" LIBJSON="0.12" RASTER
(1 row)
```

```
select version(); -- 参考まで、本体のバージョン
```

```
-----
PostgreSQL 9.5beta2, compiled by Visual C++ build 1800, 64-bit
```

データ

- MAPZENというサイトで配布されているBorders -> Japan
<https://mapzen.com/data/borders/>

The screenshot shows the Mapzen website on the left and a file explorer window on the right. The website has a navigation bar with links for PROJECTS, DATA, DOCUMENTATION, BLOG, and DEVELOPERS. Below the navigation bar, there is a paragraph of text explaining the data structure: "Each country link contains a zipped directory of geojson files, each describing increasingly detailed administrative boundaries. The hierarchy varies by country but in the United States, admin layer 2 describes the national boundary, layer 4 is states, layer 6 is counties, and layer 8 is cities. The OSM wiki has specifics for each country. Check out the OSM wiki for more information on admin levels. Full planet file available here. Last updated on September 26, 2015 at 9:11am". Below the text is a world map with various countries colored in different shades. The file explorer window shows a directory named "R:\japan_geojson.tgz" with a list of files. The files are named "admin_level_0.geojson" through "admin_level_10.geojson". The table below shows the details of these files.

Name	Size	Date modified
admin_level_0.geojson	1 KB	Sep 26 6:15
admin_level_1.geojson	1 KB	Sep 26 6:15
admin_level_2.geojson	247 KB	Sep 26 6:19
admin_level_3.geojson	1 KB	Sep 26 6:16
admin_level_4.geojson	18,989 KB	Sep 26 6:21
admin_level_5.geojson	5,952 KB	Sep 26 6:17
admin_level_6.geojson	52,134 KB	Sep 26 6:24
admin_level_7.geojson	159,681 KB	Sep 26 6:23
admin_level_8.geojson	8,474 KB	Sep 26 6:38
admin_level_9.geojson	1,136 KB	Sep 26 6:18
admin_level_10.geojson	304 KB	Sep 26 6:25

- OpenStreetMapから、世界各国の行政界を抽出
 ただし海岸線はなく「領海までの区域」に拡張されている
- ファイル: 国別のGeoJSON (Japan 59.2 MB)、世界一括もあり

GeoJSON とは

- JSON 形式で GIS データを格納するオープンな規格

Contents

- 1. Introduction
 - 1.1. Examples
 - 1.2. Definitions
- 2. GeoJSON Objects
 - 2.1. Geometry Objects
 - 2.1.1. Positions
 - 2.1.2. Point
 - 2.1.3. MultiPoint
 - 2.1.4. LineString
 - 2.1.5. MultiLineString
 - 2.1.6. Polygon
 - 2.1.7. MultiPolygon
 - 2.1.8. Geometry Collection
 - 2.2. Feature Objects

The GeoJSON Format Specification

Authors Howard Butler (Hobu Inc.), Martin Daly (Cadcorp), Allan Doyle (MIT), Sean Gillies (UNC-Chapel Hill), Tim Schaub (OpenGeo), Christopher Schmidt (MetaCarta)

Revision 1.0

Date 16 June 2008

Abstract GeoJSON is a geospatial data interchange format based on JavaScript Object Notation (JSON).

Copyright Copyright © 2008 by the Authors. This work is licensed under a [Creative Commons Attribution 3.0 United States License](#).

1. Introduction

GeoJSON is a format for encoding a variety of geographic data structures. A GeoJSON object may represent a geometry, a feature, or a collection of features. GeoJSON supports the following geometry types: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, and GeometryCollection. Features in GeoJSON contain a geometry object and additional properties, and a feature collection represents a list of features.

- 公式ウェブサイト <http://geojson.org/>
仕様（日本語） <http://s.kitazaki.name/docs/geojson-spec-ja.html>
- PostGIS … 出力はバージョン1.5以降で可、入力には2.0以降
MySQL … JSON自体を含め、5.7で初めて対応

GeoJSON を「大きな文字列」として読み込む

```
create table borders_raw as
select *
from unnest(array[4, 7, 10]) as admin_level,
     format('1127_borders/admin_level_%s.geojson',
           admin_level) as fname,
     pg_read_file(fname) as raw;

-- 3 rows affected, 6926 ms execution time.
```

	admin_level integer	fname text	raw text
1	4	1127_borders/admin_level_4.geojson	{"type":"FeatureCollection","geocod
2	7	1127_borders/admin_level_7.geojson	{"type":"FeatureCollection","geocod
3	10	1127_borders/admin_level_10.geojson	{"type":"FeatureCollection","geocod

- データフォルダの下 or シンボリックリンク先にファイルを置き、
pg_read_file()で一つの文字列として読み込む

GeoJSON → 1行ずつのテーブルにするクエリ例

```
create table borders_geom as
select admin_level,
       row_number() over(partition by admin_level) as gid,
       json->'properties' as prop,
       ST_GeomFromGeoJson(json->>'geometry') as geom -- 地理データ
from borders_raw,
     json_array_elements(raw::json->'features') as json;

-- 2050 rows affected, 24133 ms execution time.
```

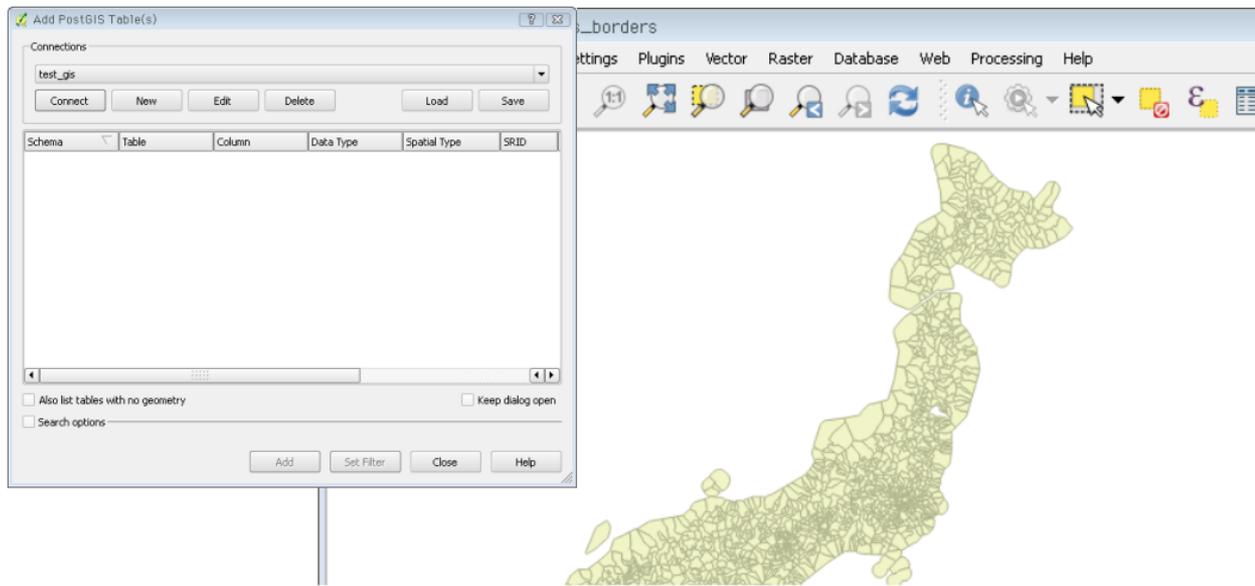
	admin_level integer	gid bigint	prop json	geom geometry
1	4	1	{"name":"広島県","source":"KSJ2",'	01060000000100000
2	4	2	{"name":"大阪府","name:en":"Osaka	01060000000100000
3	4	3	{"name":"香川県","source":"KSJ2",'	01060000000100000
4	4	4	{"name":"徳島県","source":"KSJ2",'	01060000000100000
5	4	5	{"name":"和歌山県","note":"飛び地	01060000000300000
6	4	6	{"name":"奈良県","note":"和歌山県	01060000000100000

以上2クエリの振り返り (GeoJSONのテーブル化)

1. GeoJSONファイルを、`pg_read_file()`で読める場所に置く
2. ファイル名の列などを付け、1ファイル1文字列として一時テーブル化
3. 一つのGeoJSONは、トップ -> 'features' で配列になる。
その各要素を `json_array_elements()` で一行ずつにバラす。
各行のJSONから、キー-propertiesで属性を、`geometry`で地理データを取り出し、PostGISのジオメトリ型に変換。
4. キー-propertiesにIDが入ってない場合 (今回もそう)、
`row_number()`とかを使って適宜IDを振る。
5. 以上のクエリ結果を `CREATE TABLE ... AS` でテーブル化。
`SELECT ... INTO` でもいいと思います。

インポート結果の確認 (QGIS)

- QGIS : オープンソースのデスクトップ型GISソフトの代表格
PostGISをはじめ様々なデータベースに対応



- これで今回のテスト用テーブルが出来上り

MySQL : PostGISと違う点

1. 拡張機能ではない → インストールや DB への設定が不要
2. pg_read_file()に相当する関数がない → LOAD DATA コマンドで、GeoJSONを「1行1列のデータ」として読み込む

```
load data infile './1127_borders/admin_level_4.geojson'
into table borders_raw
lines terminated by '\\';

-- Query OK, 1 row affected (16.43 sec)
```

admin_level	fname	raw
10	./1127_borders/admin_level_10.geojson	{"type":"FeatureCollection","geocoding":{"creation_date":"20
4	./1127_borders/admin_level_4.geojson	{"type":"FeatureCollection","geocoding":{"creation_date":"20
7	./1127_borders/admin_level_7.geojson	{"type":"FeatureCollection","geocoding":{"creation_date":"20

※ GeoJSONの列 (raw) 以外は、別途 UPDATE クエリで追加

MySQL : PostGISと違う点 (続)

3. PostGISと同名の関数 ST_GeomFromGeoJSON() で 丸ごと読める

```
alter table borders_raw add geom GEOMETRY;  
update borders_raw  
  set geom = ST_GeomFromGeoJson(raw)  
  where admin_level = 4;  
  
-- Query OK, 1 row affected (1 min 24.21 sec)
```

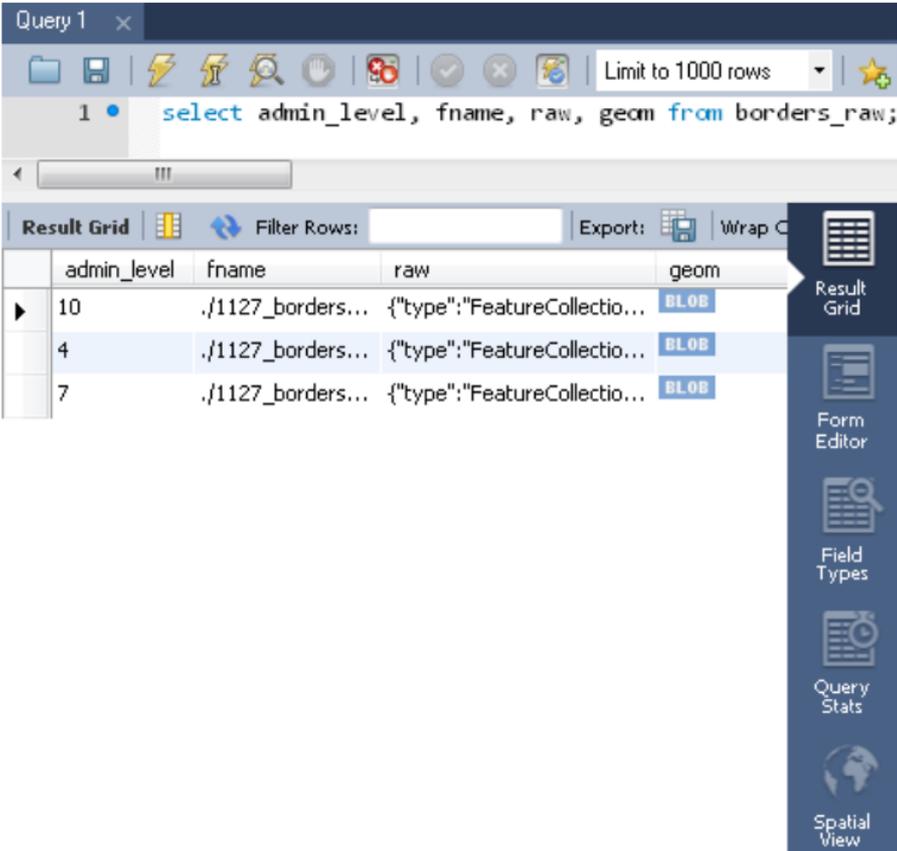
4. 読み込んだGeoJSONデータを MySQL Workbench で確認できる (バージョン6.2以降。画面は次頁)

詳しくは :

The MySQL Workbench Team Blog - Workbench 6.2: Spatial
Data

<http://mysqlworkbench.org/2014/09/mysql-workbench-6-2-spatial-data/>

MySQL Workbench : Spatial View



The screenshot shows the MySQL Workbench interface. At the top, a query editor window titled "Query 1" contains the following SQL query:

```
select admin_level, fname, raw, geom from borders_raw;
```

Below the query editor, the results are displayed in a table view. The table has four columns: `admin_level`, `fname`, `raw`, and `geom`. The `geom` column is highlighted in blue and labeled "BLOB". The table contains three rows of data:

	admin_level	fname	raw	geom
▶	10	./1127_borders...	{"type":"FeatureCollectio...	BLOB
	4	./1127_borders...	{"type":"FeatureCollectio...	BLOB
	7	./1127_borders...	{"type":"FeatureCollectio...	BLOB

On the right side of the interface, there is a vertical toolbar with several icons. The "Spatial View" icon at the bottom is highlighted, indicating that the spatial data is being viewed in a spatial context.

MySQL Workbench : Spatial View

The screenshot displays the MySQL Workbench interface in Spatial View mode. At the top, a query window titled "Query 1" contains the following SQL statement:

```
1 • select admin_level, fname, raw, geom from borders_raw;
```

Below the query editor, the Spatial View toolbar shows the "Projection" set to "Robinson". The main canvas area is currently blank with the text "Repainting...".

At the bottom of the interface, a status message reads: "Rendering spatial data, please wait." Below this, it indicates "Rendering 3 objects in layer 1..." and features a green progress bar that is approximately 60% full.

On the right side, a vertical toolbar contains icons for "Result Grid", "Form Editor", "Field Types", "Query Stats", and "Spatial View".

MySQL Workbench : Spatial View

The screenshot displays the MySQL Workbench interface in Spatial View. The main window shows a map of Japan with a grid overlay and administrative boundaries. The query editor at the top contains the following SQL statement:

```
1 • select admin_level, fname, raw, geom from borders_raw;
```

The interface includes a toolbar with various tools and a right-hand sidebar with the following components:

- Layer Source:** A table listing the layers displayed on the map.
- Result Grid:** A button to view the query results in a grid format.
- Form Editor:** A button to edit the form of the selected feature.
- Field Types:** A button to view the field types of the selected feature.
- Query Stats:** A button to view the query execution statistics.
- Spatial View:** A button to toggle between Spatial View and Table View.
- Execution Plan:** A button to view the execution plan of the query.

The Layer Source table is as follows:

Layer	Source
<input checked="" type="checkbox"/> Grid	
<input checked="" type="checkbox"/> geom	borders_raw 3

The map shows the geographical outline of Japan with a grid overlay. The projection is set to Robinson. The coordinates of the selected feature are:

Lat: 43d53' 9,50"N
Lon: 158d18'35,29"E

Click a feature to view its record

ただし、JSON の処理に難あり

```
create table borders_json (admin_level int, js json);
```

```
insert borders_json
  select admin_level, cast(raw as JSON)
  from borders_raw
  where admin_level = 4;
```

```
ERROR 1301 (HY000): Result of json_binary::serialize()
was larger than max_allowed_packet (4194304) - truncated
```

- 305 KBの小さなGeoJSON文字列 → JSON型に変換できた
しかし19.0 MBのGeoJSON（日本の都道府県境界） → 変換できず

※ max_allowed_packetの値を大きく設定しても、今回は解決せず

- JSONにできない → 「1行ずつの普通のテーブル」にできない

仕方ないので、PostGISからダンプして読み込み

```
-- ON PostGIS
copy (
  select admin_level, gid, prop, ST_AsGeoJson(geom)
  from borders_geom
) to 'R:/tmp/dump_postgis.tsv';
-- 2050 rows affected, 10093 ms execution time.

-- ON MySQL
create table borders_geom
  (admin_level int, gid int, prop JSON, geojson JSON);

load data infile './1127_borders/dump_postgis.tsv'
  into table borders_geom;
-- Query OK, 2050 rows affected (27.68 sec)

alter table borders_geom add geom GEOMETRY;
update borders_geom set geom = ST_GeomFromGeoJson(geojson);
-- Query OK, 2050 rows affected (54.74 sec)
```

ここまでの作業と処理時間

- 読み込み対象 : GeoJSON * 3ファイル (304 KB, 19.0 MB, 160 MB)
- PostGIS : ファイル読み込みクエリ ... 7.0 sec
テーブルへの変換クエリ ... 24 sec
ダンプ出力 ... 10 sec
- MySQL : (処理内容が違うので、参考まで)
ダンプファイル読み込み ... 28 sec
GISデータへの変換クエリ ... 55 sec
- MySQL ... GeoJSONを丸ごと読み込めるのは利点
しかし、属性を抽出しGISデータと合わせて普通のテーブルにするなら
結局JSONとしてのパース & 操作が必要。。。
PostgreSQL ... JSONの操作 (配列を行にばらす等) が豊富で便利 !

同じテーブルができたので、何か関数を使ってみる

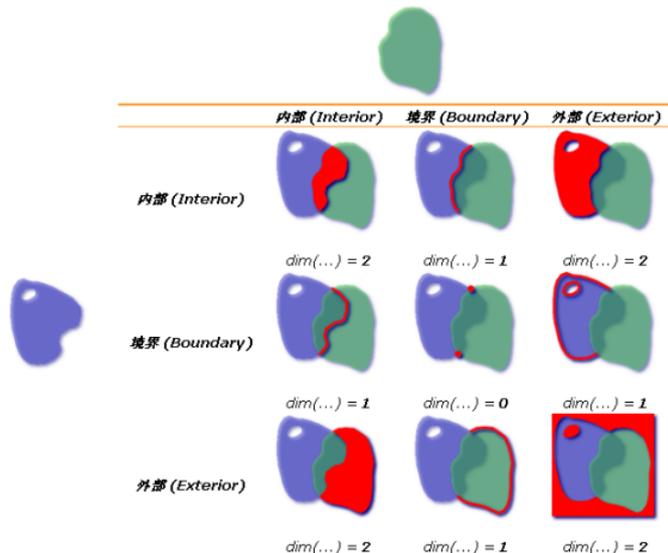
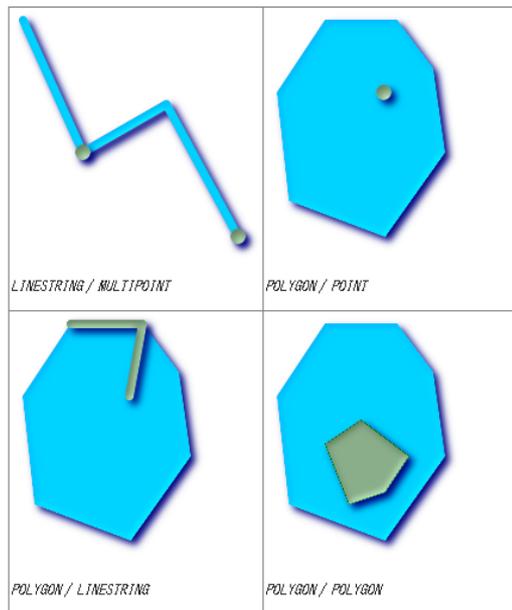
地理データどうしの「重なり」とかを調べる

- 今回の対象は、行政界で囲まれた「面」(ポリゴン)
ポリゴンどうしの 「空間的な関係」 を調べられる関数、いろいろ
- MySQL と PostGIS 両方にあるもの：
ST_Contains(), ST_Crosses(), ST_Disjoint(), ST_Equals()
ST_Intersects(), ST_Overlaps(), ST_Touches(), ST_Within()
- PostGIS だけにあるもの：
ST_ContainsProperly(), ST_Covers(), ST_CoveredBy()
ST_DWithin(), ST_Relate(), … 他にも多数

※ 関数だけでなく演算子も多数。&&, &&&, <<, @, 等々
- MySQL だけにあるもの：ポリゴンを囲む矩形(MBR)で判断する関数群

空間的な関係を調べる、とは

次に示す図全てで、ST_ContainsはTRUEを返します。



左から右、上から下に読むと、次元行列は'212101212'と表現されます。

1つ目の例である、2線が線上でインタセクトする場合の関係行列は'1*1***1***'となります。

左 : http://www.finds.jp/docs/pgisman/2.2.0/ST_Contains.html

右 : <http://www.finds.jp/docs/pgisman/2.2.0>

[/using_postgis_dbmanagement.html](http://www.finds.jp/docs/pgisman/2.2.0/using_postgis_dbmanagement.html)

何を調べるか：今回のデータで欠けている「県」

```
select admin_level, count(*)
  from borders_geom
  group by admin_level;
```

```
admin_level | count
```

```
-----+-----
```

```
4 | 46
```

<-- 都道府県らしいが、一つ足りない

```
7 | 1743
```

<-- 市町村

```
10 | 261
```

<-- よく分からない

```
(3 rows)
```

- 調べる方法 (1) 市町村で「どの都道府県とも重ならない」ものを検索。
→ 見つければ、その市町村名から「足りない県」を推測できる
- 調べる方法 (2) 46都道府県を一つのポリゴンに融合。それと重ならない市町村を検索する。→ 見つければ (同上)

方法 (1) PostGISで

```
select cities.prop :: text
       -- JSON のままだと except 使えないので文字にキャスト

from borders_geom as cities
     where admin_level = 7 -- 全ての市区町村

except all

select cities.prop :: text
       from borders_geom as prefs, borders_geom as cities
     where prefs.admin_level = 4
           and cities.admin_level = 7
           and st_intersects(prefs.geom, cities.geom);

       -- 都道府県と市区町村で「重なる」組を列挙、そして引く
```

→ 地理データの列 (geom) にインデクス付け、explainする (次頁)

方法 (1) PostGISのクエリプラン

QUERY PLAN

```
-----
HashSetOp Except All (cost=0.00..566.45 rows=1743 width=32)
-> Append (cost=0.00..558.50 rows=3180 width=32)
    -> Subquery Scan on "*SELECT* 1" (cost=0.00..155.77 rows=1743 width=32)
        -> Seq Scan on borders_geom cities (cost=0.00..138.34 rows=1743 width=32)
            Filter: (admin_level = 7)
    -> Subquery Scan on "*SELECT* 2" (cost=0.14..402.73 rows=1437 width=32)
        -> Nested Loop (cost=0.14..388.36 rows=1437 width=32)
            -> Seq Scan on borders_geom prefs (cost=0.00..129.63 rows=1437 width=32)
                Filter: (admin_level = 4)
            -> Index Scan using borders_geom_geom_idx on borders_geom (cost=0.14..258.73 rows=1437 width=32)
                Index Cond: (prefs.geom && geom)
                Filter: ((admin_level = 7) AND _st_intersects(prefs.geom, geom))
(12 rows)
```

→ 地理データのインデクスが使われる。結果は次頁

方法 (1) PostGISでの結果

11	rows removed by filter: 2004
12	-> Index Scan using borders_geom_geom_idx on borders_g
13	Index Cond: (prefs.geom && geom)
14	Filter: ((admin_level = 7) AND _st_intersects(pre
15	Rows Removed by Filter: 51
16	Planning time: 0.261 ms
17	Execution time: 3970.445 ms

	prop text
1	{"name":"西ノ島町","source":"KSJ2/N03","name:en":"Nishinoshima","name:ja":
2	{"name":"海士町","source":"KSJ2/N03","name:en":"Ama","name:ja":"海士町","
3	{"name":"出雲市","source":"KSJ2/N03","name:en":"Izumo","name:ja":"出雲市"
4	{"name":"隠岐の島町","source":"KSJ2/N03","name:en":"Okinoshima","name:ja"
5	{"name":"大田市","source":"KSJ2/N03","name:en":"Ooda","name:ja":"大田市",
6	{"name":"知夫村","source":"KSJ2/N03","name:en":"Chibu","name:ja":"知夫村"

→ 約 4 秒で結果取得。(島根県がなかった)

方法 (1) MySQLで、同様のクエリ

```
select x.prop
  from borders_geom as x
  where admin_level = 7 and not exists (
    select cities.prop
      from borders_geom as prefs, borders_geom as cities
      where prefs.admin_level = 4
            and cities.admin_level = 7
            and st_intersects(prefs.geom, cities.geom)

            and x.prop = cities.prop
            -- PostgreSQLでは文字型にキャストして比較
  );
```

- » EXCEPT句を使えないので NOT EXISTS で代用
- » PostGISで実行したら、先ほどと同様の結果
- » MySQLでは、JSON型をそのまま = 演算子で比較できる

方法 (1) MySQLのクエリプラン (抜粋) と実行結果

```
explain ( 前頁のクエリ )
```

table	partitions	type	possible_keys	key	key_len	ref	rows
x	NULL	ALL	NULL	NULL	NULL	NULL	20
cities	NULL	ALL	geom	NULL	NULL	NULL	20
prefs	NULL	ALL	geom	NULL	NULL	NULL	20

» 地理データのインデクスは「候補」になるが使われない。所要44秒

```
| prop
```

```
| {"name": "川本町", "source": "KSJ2/N03", "name:en": "Kawamoto", "name:ja": "川本町", "boundary": "administrative"},
| {"name": "江津市", "source": "KSJ2/N03", "name:en": "Goutsu", "name:ja": "江津市", "boundary": "administrative"},
| {"name": "大田市", "source": "KSJ2/N03", "name:en": "Ooda", "name:ja": "大田市", "boundary": "administrative"},
| {"name": "出雲市", "source": "KSJ2/N03", "name:en": "Izumo", "name:ja": "出雲市", "boundary": "administrative"},
| {"name": "知夫村", "source": "KSJ2/N03", "name:en": "Chibu", "name:ja": "知夫村", "boundary": "administrative"},
| {"name": "西ノ島町", "source": "KSJ2/N03", "name:en": "Nishinoshima", "name:ja": "西ノ島町", "name:ru": "Nishinoshima"},
| {"name": "海士町", "source": "KSJ2/N03", "name:en": "Ama", "name:ja": "海士町", "boundary": "administrative"},
| {"name": "隠岐の島町", "source": "KSJ2/N03", "name:en": "Okinoshima", "name:ja": "隠岐の島町", "boundary": "administrative"}
```

```
8 rows in set (44.29 sec)
```

方法（2）都道府県を融合。PostGISの場合

```

-- 都道府県を融合した一つのポリゴンを、別テーブルにする
create table union_prefs as
select ST_Union(geom) as geom
      from borders_geom
      where admin_level = 4;
-- one row affected, 15273 ms execution time.

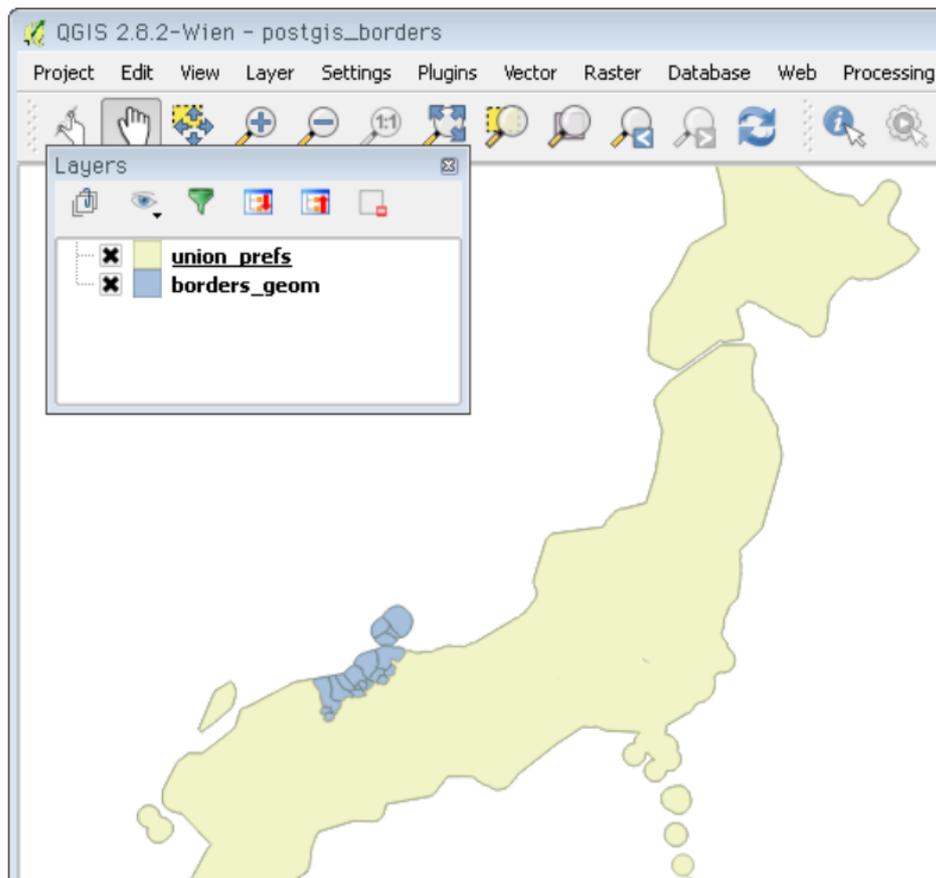
-- 検索
select cities.prop
      from borders_geom as cities, union_prefs as prefs
      where cities.admin_level = 7
            and not ST_intersects(cities.geom, prefs.geom);

```

4	-> Seq Scan on union_prefs prefs (cost=0.00..1.01 rows=1 width=391128) (actual time=0.004..0.00)
5	-> Seq Scan on borders_geom cities (cost=0.00..129.63 rows=1743 width=46361) (actual time=0.011..0.011)
6	Filter: (admin_level = 7)
7	Rows Removed by Filter: 307
8	Planning time: 0.270 ms
9	Execution time: 805.506 ms

フルスキャンになったが、約0.8秒

結果を QGIS で確認



方法（2）MySQLでは厳しい、その理由

- 複数ポリゴンを一括して融合する「集約関数」がない

```
mysql> create table union_prefs as
-> select ST_Union(geom) as geom
->       from borders_geom
->       where admin_level = 4;
```

```
ERROR 1582 (42000): Incorrect parameter count
in the call to native function 'ST_Union'
```

- 関数 ST_Union() は、2つの地理データを融合するのみ
例えば ST_Union(東京都, 埼玉県) のように。
- 数値で言えば「2つの数の加算はできても、sumがない」状態
- ストアドを書けば可能かもしれないが。。。

融合 (Union) に限らず、 MySQL には「地理データの集約関数」がない

- 地理データを RDBMS で扱うメリットが半減してしまう
- PostGIS の集約関数 (ST_Union 以外) :
 - ST_Accum() … 配列型での array_agg に当たるもの
 - ST_Collect() … 融合ではなく「集合」を返す (境界線を残す)
 - ST_Extent() … 全体を囲む矩形を返す
 - ST_Polygonize() など
- 実質的な集約関数 (複数の地理データ → 一つのデータに変換) :
 - ST_LineMerge() … 線分を一本につなげる (経路探索で重宝)
 - ST_BuildArea() など多数

その他、MySQLの主な制約

- 経緯度データ → メートル座標系へ変換できない
 - だから、経緯度データに対して「線の長さ」や「面積」を出せない
(計算はできるが、角度から求めた無意味な値になる)
 - ただし 2点間の距離 だけは、バージョン5.7.6から可能
- 地球が「球」でなく「回転楕円体」であることは考慮外
 - 地球は「南北に少しつぶれた」形をしている
(赤道上の1点から、東へ90度行った距離 > 北へ90度行った距離)
 - PostGISでは、どちらも選べる → 計算の速い「球」
 - 遅いけど実際に近い「回転楕円体」
- 一般的なGISデータ (Shapefile) のインポートツールが付属していない

両方で比較できるもの : kNN 検索 (k 最近傍検索)

kNN 検索 (k 最近傍検索) とは

- NN : Nearest Neighbour (最近傍) の略
- ある点から、個数 k の NN を探す → kNN
例 : 今いる場所から、近い順にATMを 10ヶ所ピックアップしたい
- SQL自体は単純 (下の関数名は仮)

```
select atm from tb_atms -- 検索対象テーブル
      order by distance(atm, point(139, 35)) -- 今いる場所
      limit 10; -- k = 10
```

- 問題 : 検索対象が膨大な場合の実行コスト

上記クエリだと ① 検索対象の全行につき、今いる場所との距離を算出
② 距離の短い順にソートし、先頭10件を取り出す

従来の一般的な改善策

- 全行につき距離を出さず、ある範囲の行に絞ってから距離を測る

```
select atm from (  
  
    select atm from tb_atms  
    where inteseects(atm, box(138, 34, 140, 36))  
    -- 検索対象を絞り込むサブクエリ。boxは適当に決める  
    -- 上の関数名は仮  
  
) foo  
order by distance(atm, point(139, 35)) -- 今いる場所  
limit 10; -- k = 10
```

- 上記サブクエリでインデクスを有効にすれば、速い
- 問題：適切な絞り込み範囲は、どうやって決めればよいのか

PostgreSQL 9.5 + PostGIS 2.2はもっと簡単

- 演算子 <-> で、事前の絞り込みなく高速に kNN 検索できる

```
select atm
  from tb_atms
 order by atm <-> 'POINT(138 35)' :: geography
 limit 10;
```

- テスト用に約63万の点を作成（都道府県ポリゴンの周縁の構成点）

```
create table fringes as
select prop->>'name' as pname,
       (st_dumppoints(geom)).geom
  from borders_geom
 where admin_level = 4;
-- 629456 rows affected, 6599 ms execution time.
```

最初の単純なSQLだと

```
select pname from fringes -- 検索対象テーブル
  order by st_distance_sphere(geom,
    'SRID=4326; POINT(138 35):: geometry)
 limit 10;
```

- 全63万行に対して、メートル単位の距離を算出するので遅い（46秒）

	QUERY PLAN text
1	Limit (cost=506944.88..506944.90 rows=10 width=42) (actual time=46399.539 ms)
2	-> Sort (cost=506944.88..508518.52 rows=629456 width=42) (actual time=46399.539 ms)
3	Sort Key: (st_distance_sphere(geom, '0101000020E61000000000000000000000'))
4	Sort Method: top-N heapsort Memory: 25kB
5	-> Seq Scan on fringes (cost=0.00..493342.56 rows=629456 width=42)
6	Planning time: 0.094 ms
7	Execution time: 46399.539 ms

経緯度からメートル距離を出しやすくするため、 検索対象の点を **geography** 型で準備

```
alter table fringes add geog geography;  
update fringes set geog = geom::geography;  
-- 629456 rows affected, 3525 ms execution time.  
  
-- インデクス付加  
create index on fringes using gist (geog);  
-- Query returned successfully with no result in 6786 ms.
```

- 準備はこれだけ、約10秒
- クエリの explain analyze 結果：次頁

演算子 <-> を使うとインデクスが有効になる

```
select pname, st_distance(
    'POINT(138 35)' :: geography, geog) as dist
from fringes
order by geog <-> 'POINT(138 35)' :: geography
limit 10;
```

- 46399 ms → 5 ms 信じられないくらい速い!

	QUERY PLAN text
1	Limit (cost=0.29..6.79 rows=10 width=66) (actual time=4.599..4.651 rows=10 loop=1)
2	-> Index Scan using fringes_geog_idx on fringes (cost=0.29..409367.77 rows=6)
3	Order By: (geog <-> '0101000020E610000000000000004061400000000000804140')
4	Planning time: 0.363 ms
5	Execution time: 5.464 ms

- クエリ自体の結果：次頁

演算子 <-> を使ったクエリ結果

```
select pname, st_distance(
    'POINT(138 35)' :: geography, geog) as dist
from fringes
order by geog <-> 'POINT(138 35)' :: geography
limit 10;
```

	pname text	dist double precision
1	静岡県	21613.42874473
2	愛知県	21613.42874473
3	静岡県	21613.91219715
4	愛知県	21613.91219715
5	愛知県	21614.18871788
6	静岡県	21614.18871788
7	静岡県	21615.73452799

- kNN検索は地球を「球」として行い、最近傍10件への距離（列 dist）はより正確な「回転楕円体」で算出。
- 演算子 <-> は以前からあったが、距離の精度が低かった。それが最新 PostGIS 2.2で改善された

→ http://postgis.net/docs/manual-2.2/geometry_distance_knn.html

MySQLでのkNN検索（準備）

- テスト用63万の点テーブルを作成
PostGISのように「ポリゴンを構成する全点を一括抽出」はできない

```
create table centroids as
select st_centroid(geom) as geom from borders_geom;
set @n := 0;

-- 以下を繰り返し。点の数が約63万個になるまで
set @n := @n + 1;
insert into centroids
select ST_PointN(
    ST_ExteriorRing(
        ST_GeometryN(geom, 1)), @n)
from borders_geom
where ST_NumPoints(
    ST_ExteriorRing(
        ST_GeometryN(geom, 1))) >= @n;
select count(*) from centroids;
```

MySQLでのkNN検索（続）

- 対象テーブルの行数と、explainの抜粋（インデクスは使われない）

```
select count(*) from centroids; -- 対象の点テーブルの行数
```

```
+-----+
| count(*) |
+-----+
|   630123 |
+-----+
```

```
explain
```

```
select ST_AsText(geom) from centroids
       order by st_distance_sphere(geom,
                                     ST_GeomFromText('POINT(138 35)', 4326))
       limit 10;
```

table	partitions	type	possible_keys	key	key_len	r
centroids	NULL	ALL	NULL	NULL	NULL	1 N

MySQLでのkNN検索（結構速い）

```
select ST_AsText(geom) from centroids
       order by st_distance_sphere(geom,
                                   ST_GeomFromText('POINT(138 35)', 4326))
       limit 10;
```

```
+-----+
| ST_AsText(geom) |
+-----+
| POINT(138.044186 34.923237) |
| POINT(138.044175 34.92313) |
| POINT(138.044118 34.923002) |
| POINT(138.044084 34.922856) |
| POINT(138.044055 34.922699) |
| POINT(138.044031 34.922631) |
| POINT(138.043952 34.922403) |
| POINT(138.043864 34.922192) |
| POINT(138.04378 34.922065) |
| POINT(138.043666 34.921936) |
+-----+
10 rows in set (1.31 sec)
```

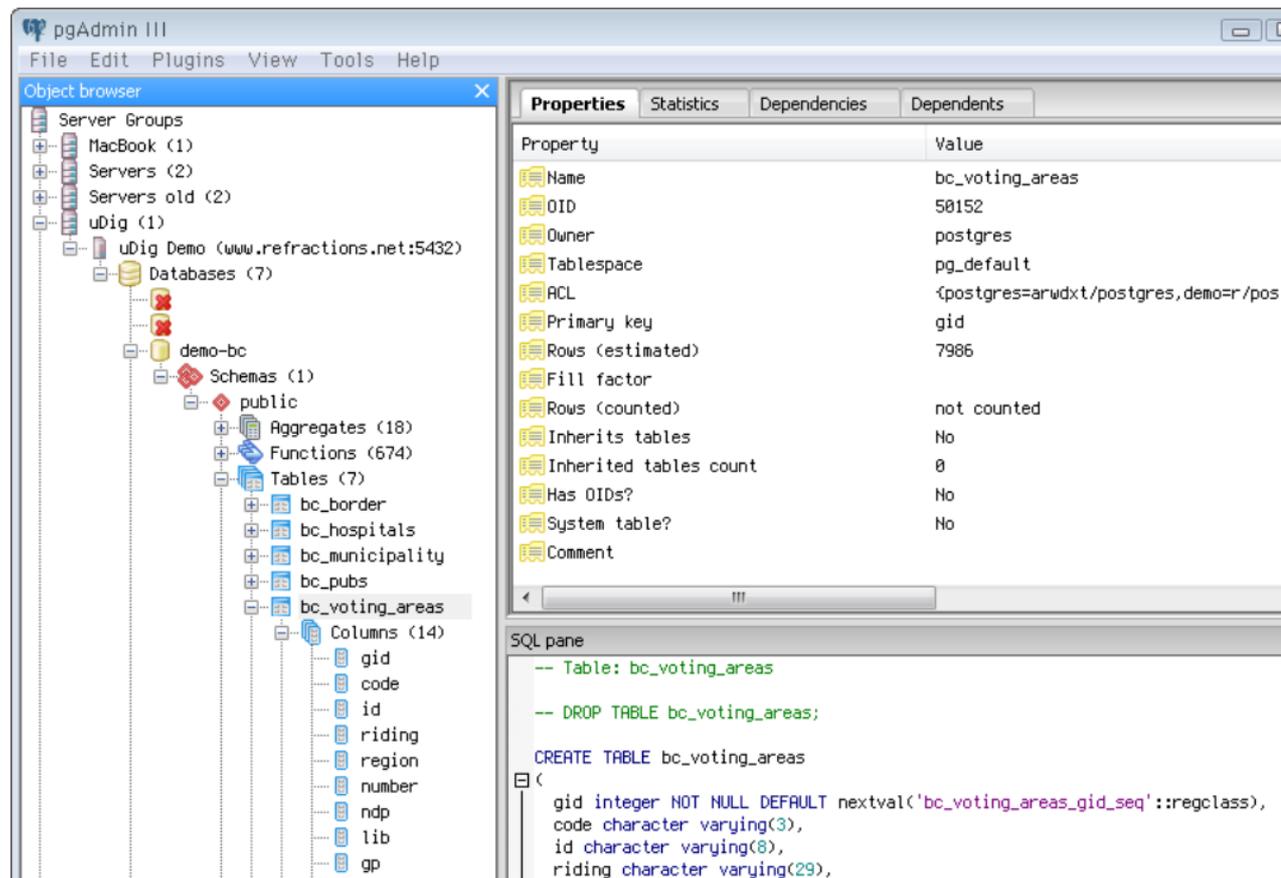
- 最初の単純なクエリで、1秒余りと結構速い。同じクエリでPostGISは40数秒かかった。
- 特段の準備やチューニングせずこの性能なので、kNN検索には合っているかも

PostgreSQLのFDW（外部データラッパ）と PostGIS

例：外部のPostGISデータに、必要な時だけアクセス

- 必要なもの
 - ・ postgres_fdw（標準的な拡張機能の一つ）
 - ・ 外部で稼動しているPostGISサーバ
- PostGISを開発したRefractions Research社が、同社のオープンソースGIS「uDig」のチュートリアル用にPostGISサーバを公開
http://udig.refrations.net/files/docs/latest/user/getting_started/walkthrough1/postgis.html
- このチュートリアル用テーブルを、postgres_fdwで外部テーブル化
その際、Postgres 9.5 新機能 [import foreign schema](#) がとても便利
- 外部テーブルをマテリアライズドビューにして、必要時だけアクセス

uDig チュートリアル用サーバに接続テスト



The screenshot shows the pgAdmin III interface. The left pane displays the Object browser tree, showing the connection to 'uDig Demo' and the 'demo-bc' database. The 'public' schema contains several tables, with 'bc_voting_areas' selected. The right pane shows the Properties tab for this table.

Property	Value
Name	bc_voting_areas
OID	58152
Owner	postgres
Tablespace	pg_default
ACL	{postgres=arwdxt/postgres,demo=r/postgres}
Primary key	gid
Rows (estimated)	7986
Fill factor	
Rows (counted)	not counted
Inherits tables	No
Inherited tables count	0
Has OIDs?	No
System table?	No
Comment	

The SQL pane shows the following SQL code:

```
-- Table: bc_voting_areas
-- DROP TABLE bc_voting_areas;
CREATE TABLE bc_voting_areas
(
gid integer NOT NULL DEFAULT nextval('bc_voting_areas_gid_seq'::regclass),
code character varying(3),
id character varying(8),
riding character varying(29),
```

外部テーブル作成

```
CREATE EXTENSION postgres_fdw;

CREATE SERVER udig FOREIGN DATA WRAPPER postgres_fdw
    OPTIONS (host 'www.refractions.net', port '5432',
            dbname 'demo-bc');

CREATE USER MAPPING FOR postgres SERVER udig
    OPTIONS (user '****', password '****');

IMPORT FOREIGN SCHEMA public LIMIT TO (
    bc_border,
    bc_hospitals,
    bc_municipality,
    bc_pubs,
    bc_voting_areas
) FROM SERVER udig INTO public;

-- Query returned successfully with no result in 3806 ms.
```

外部テーブルができたことの確認

The screenshot shows the pgAdmin III interface. The Object browser on the left displays the database structure, with the 'bc_voting_areas' foreign table highlighted under the 'public' schema. The Properties pane on the right shows the table's details, and the SQL pane at the bottom contains the SQL commands used to create the table.

Object browser:

- Server Groups
 - MacBook (1)
 - Servers (2)
 - PostgreSQL 9.5 Beta (localhost:5432)
 - Databases (2)
 - postgres
 - test_gis
 - Extensions (3)
 - Foreign Data Wrappers (1)
 - Schemas (2)
 - myviews
 - public
 - Aggregates (23)
 - Foreign Tables (5)
 - bc_border
 - bc_hospitals
 - bc_municipality
 - bc_pubs
 - bc_voting_areas
 - Functions (1111)
 - Tables (6)
 - Types (19)
 - Views (4)
 - PostgreSQL Portable 9.4 (127.0.0.1:5433)
 - Servers old (2)
 - uDig (1)
 - Virtual Machines (4)

Properties:

Property	Value
Name	bc_voting_areas
OID	32076
Owner	postgres
Server	udig
Columns	gid integer NOT NULL, code character varying(3), id character varying(8)
Options	schema_name 'public', table_name 'bc_voting_areas'
Comment	

SQL pane:

```
-- Foreign Table: public.bc_voting_areas
-- DROP FOREIGN TABLE public.bc_voting_areas;

CREATE FOREIGN TABLE public.bc_voting_areas
(gid integer NOT NULL,
code character varying(3),
id character varying(8),
riding character varying(29),
region character varying(29),
"number" character varying(4),
ndp integer,
lib integer,
gp integer,
upbc integer,
vtotal integer,
vreject integer,
vregist integer,
the_geom public.geometry )
SERVER udig
OPTIONS (schema_name 'public', table_name 'bc_voting_areas');
ALTER FOREIGN TABLE public.bc_voting_areas
OWNER TO postgres;
```

外部テーブルをマテビューとしてコピー

```
create materialized view myviews.bc_voting_areas as
select * from public.bc_voting_areas;
```

```
-- 7986 rows affected, 55490 ms execution time.
```

```
-- インターネット経由でデータ取得するので時間かかるが、この1回のみ
-- 他のテーブルも、必要なら同様に
```

↓ 左：外部テーブルから取得（5428 ms）, 右：マテビューから（16 ms）

Query - test_gis on postgres@localhost:5432 - [D:\AppsPortable\GitPortable\...]

SQL Editor | Graphical Query Builder

Previous queries: [] Delete Delete All

```
select * from bc_voting_areas limit 10; -- 外部テーブルを取得
```

Output pane

	gid integer	code character varying(3)	id character varying(8)	riding character varying(29)
1	1	VFR	VFR 115	Vancouver-Fairview
2	2	VTB	VTB 119	Victoria-Beacon Hill
3	3	RCC	RCC 89	Richmond Centre

OK. Unix Ln 2, Col 1, Ch 2 | 10 rows. 5428 ms

Query - test_gis on postgres@localhost:5432 - [D:\AppsPortable\GitPortable\...]

SQL Editor | Graphical Query Builder

Previous queries: [] Delete Delete All

```
select * from myviews.bc_voting_areas limit 10; -- マテビューを取得
```

Output pane

	gid integer	code character varying(3)	id character varying(8)	riding character varying(29)
1	1	VFR	VFR 115	Vancouver-Fairview
2	2	VTB	VTB 119	Victoria-Beacon Hill
3	3	RCC	RCC 89	Richmond Centre

OK. Unix Ln 2, Col 58, Ch 69 | 10 rows. 16 ms

PL/R（統計処理言語Rの組み込み）と PostGIS

例：RのGISパッケージの一つ「spatstat」に、PostGISデータを投入して分析するテンプレを作る

- 必要なもの
 - R <http://www.r-project.org/>
 - PL/R <http://www.joeconway.com/plr/>
 - ↑ まだ Win + Postgres 9.5 用がないので 9.4 を使用
- spatstat と若干のRのパッケージが必要だが、R上で簡単に入れられる
- 主な流れ
 - テンプレとなる自作ストアドを作成（言語にplrと指定）
 - ストアド内で、自DBのPostGISデータをRへ読み込む
 - 後は、普通のRスクリプトと同様にストアド内で処理を書く
- spatstatで「点分布 → カーネル密度推定」を実行、結果をファイル出力

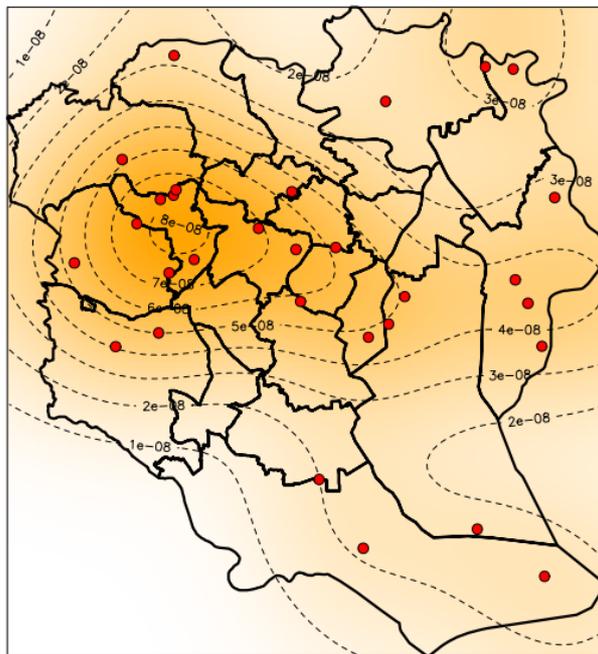
出来上がり後の実行例

```
select myfunc.poi_dens(  
    -- ジオメトリを集約し、メートル座標系に変換してストアドに渡す  
    st_astext(st_transform(st_setSrid(geom, 4326), 3095)),  
    30, 'R:/test.svg', jname)  
  
from (  
    select st_collect(geom) as geom,  
           text '東京23区' as jname  
    from borders_geom  
    where prop->>'name' like '%区%'  
) foo;
```

出来上がり後の実行例

```
select myfunc.po  
  
  -- ジオメトリ  
  st_astext(st  
  30, 'R:/test  
  
from (  
  select st_co  
  text '東  
  from borders  
  where prop->  
) foo;
```

東京23区



8e-08
6e-08
4e-08
2e-08

PostgreSQL 9.4 で PostGIS, PL/R を準備

```
create extension postgis;
create extension plr;

select postgis_full_version();
-----
POSTGIS="2.1.5 r13152" GEOS="3.4.2-CAPI-1.8.2 r3922" PROJ="Rel. 4

select plr_version();
-----
08.03.00.16

select r_version(); -- これが R のバージョン (準備OK)
-----
(platform,i386-w64-mingw32)
(arch,i386)
(os,mingw32)
(system,"i386, mingw32")
(status,"")
(major,3)
(minor,2.2)
(year,2015)
```

PL/Rで使うパッケージを、R本体上でインストール

```
install.packages(rgeos)  
# PostGISの地理データ（のテキスト出力）を、Rに読み込む  
# 具体的には readWKT() 関数を使用
```

```
install.packages(sp)  
# Rの地理データクラスの基本
```

```
install.packages(spatstat)  
# 空間的な点分布への分析等ができる
```

※ PostgreSQLサーバ起動中にインストールした場合、もしかしたらサーバ再起動が必要かも。

好きな場所にストアドを作り、R言語で書く

```
create or replace function myfunc.poi_dens
  (wkt text,
   num int,
   svg text,
   jname text)

# 数値，文字など一般的なデータ型は、そのまま引数で渡せる
# 引数名も付けられる

# wkt : PostGISデータを ST_AsText() で文字にして渡す
# num : 今回ポリゴン内にランダムに点を打つ。その数
# svg : 結果を SVG で出力する。そのパス
# jname : SVG の中に書き込む、処理対象のラベル

returns void language plr immutable as $$

library(rgeos)
library(sp)
library(spatstat) # インストールしたパッケージを読み込む
```

好きな場所にストアドを作り、R言語で書く (続)

(つづき)

```
g1 = readWKT(wkt) # PostGISデータ → Rの地理データクラス
bb = g1@bbox

p1 = spsample(g1, n=num, type='random') # ランダムな点
xy = p1@coords

pp = ppp(xy[,1], xy[,2], bb[1,], bb[2,], c('unit', 'unit'))
dense = density(pp)

# ppp ... spatstatパッケージで分析する際の一つの基本クラス
# point pattern dataset in the two-dimensional plane
# density() ... カーネル密度推定の関数
```

(つづく。後はプロットしてSVG出力するだけ)

好きな場所にストアドを作り、R言語で書く（続）

(つづき)

```
colors = colorRampPalette(c('white', 'orange')); # 色は適当
svg(file=svg)
plot(dense, main='', xlim=bb[1,], ylim=bb[2,], col=colors)
contour(add=T, dense, lty=2)
plot(add=T, g1, lwd=2)
points(p1, pch=21, bg='red')
mtext(cex=2, side=3, family='Japan1', text=jname)
graphics.off()
$R$;
```

このストアドの基本的な使い方：

```
select myfunc.poi_dens(
  ST_AsText(地理データ), 20, 'R:/test.svg', 'ラベル');
```

好きな場所にストアドを作り、R言語で書く（続）

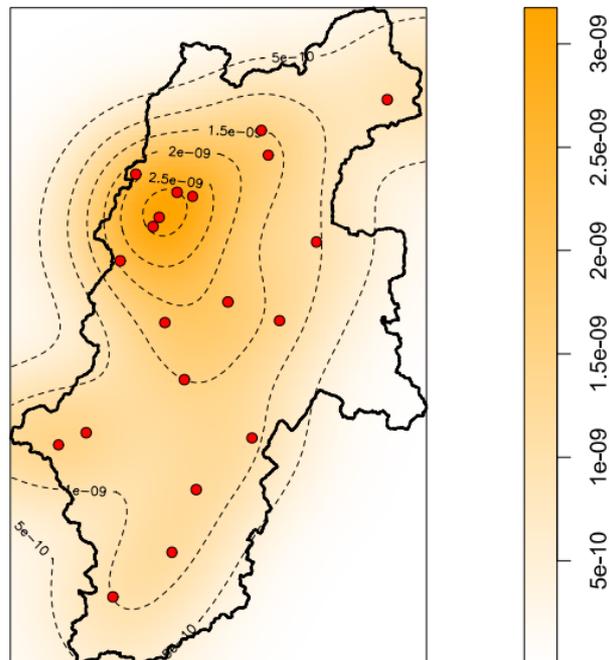
(つづき)

```
colors = colorRampPa
svg(file=svg)
plot(dense, main='',
contour(add=T, dense
plot(add=T, g1, lwd=
points(p1, pch=21, b
mtext(cex=2, side=3,
graphics.off()
$R$;
```

このストアドの基本

```
select myfunc.poi_de
ST_AsText(地理デ
```

長野県



まとめ

1. MySQL … ピンポイントで、重要な所に注力している感

- ・ kNN 検索、GeoJSON 対応
- ・ Workbench でのGISデータビュー（正直うらやましい）

2. PostGIS … 豊富な集約関数、グループ的なデータを作る/ばらす等、RDBMSならではのメリットは圧倒的

3. PostGIS 2.2 … 今回は割愛したけど、Temporal Spatial Analysis : 時空間分析用の関数が初めて追加されたり、さらに進化

4. PostGISの泣き所? … 簡単なビューがない / 機能・情報が多すぎてよく分からない / チュートリアル用データが少ない

今後オープンソースのGIS-DBが発展することを期待してます