

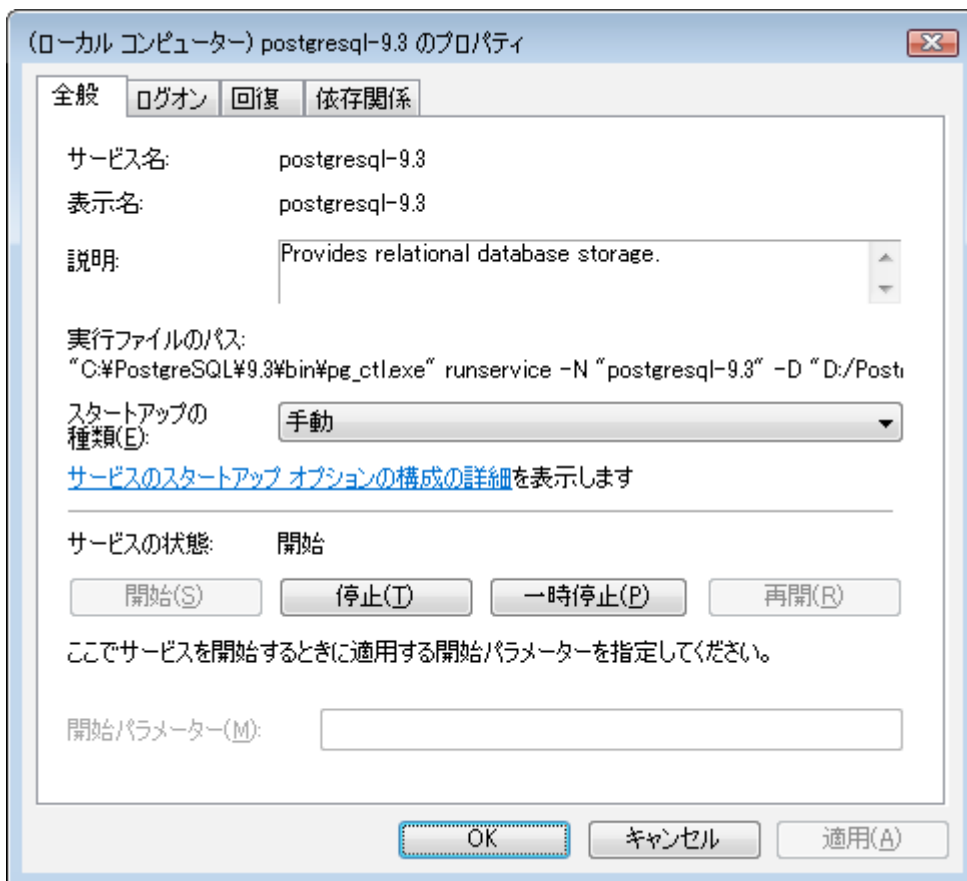


## (2) 作業環境の確認


※ 以下の pgAdminIII の操作画面は、表示（メニューの言語やフォント）をデフォルトから変更しています。皆様の PC での画面に適宜置き換えて理解していただき、不明な点があればご質問下さい。

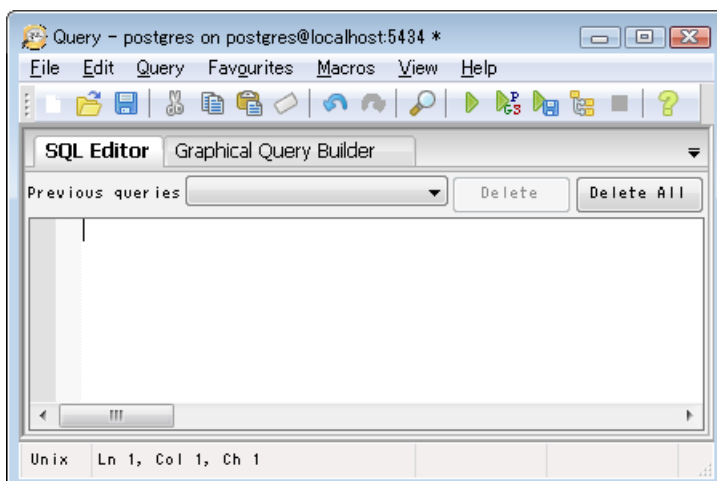
- PostgreSQL 9.3 と PostGIS 2.1 がインストールされている
- PostgreSQL のサーバが起動している

↓ Windows 7 なら例えば、コントロールパネル → 管理ツール → コンピュータの管理 → サービス → postgresql-9.3 のプロパティを開き、サービスの状態が「開始」になっているのを確認します。



- pgAdminIII や psql など、SQL の実行と結果確認をできるツールがある

↓ 例えば pgAdminIII なら「クエリツール」を使います（ツールバーの  をクリック)



□ PostGIS ラスタに必要な環境変数が設定されている

PostGIS 2.1 をインストールした時に自動的に設定されるはずですが、念のため確認します。

・システム環境変数 GDAL\_DATA

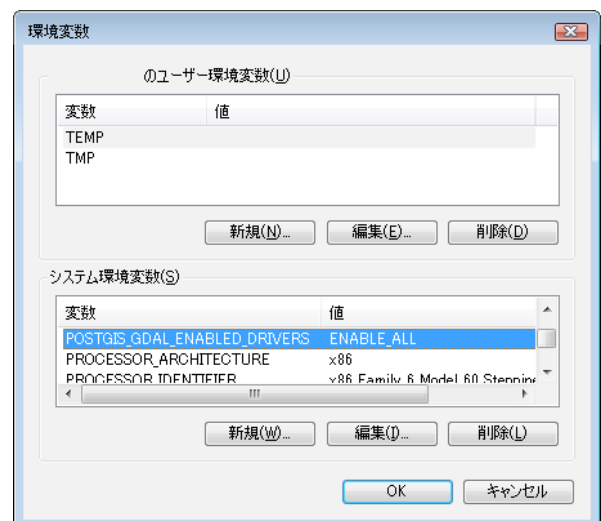
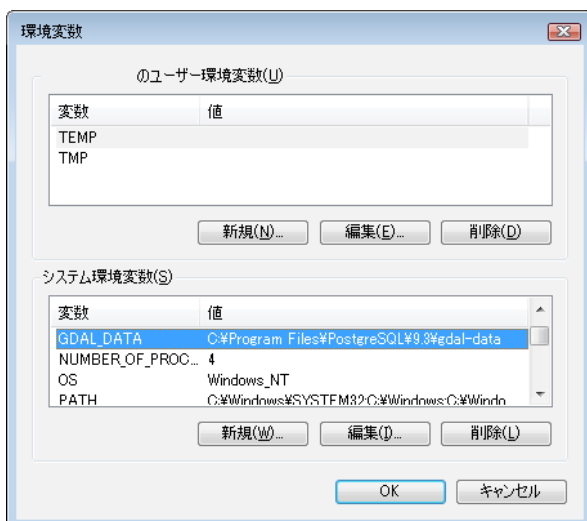
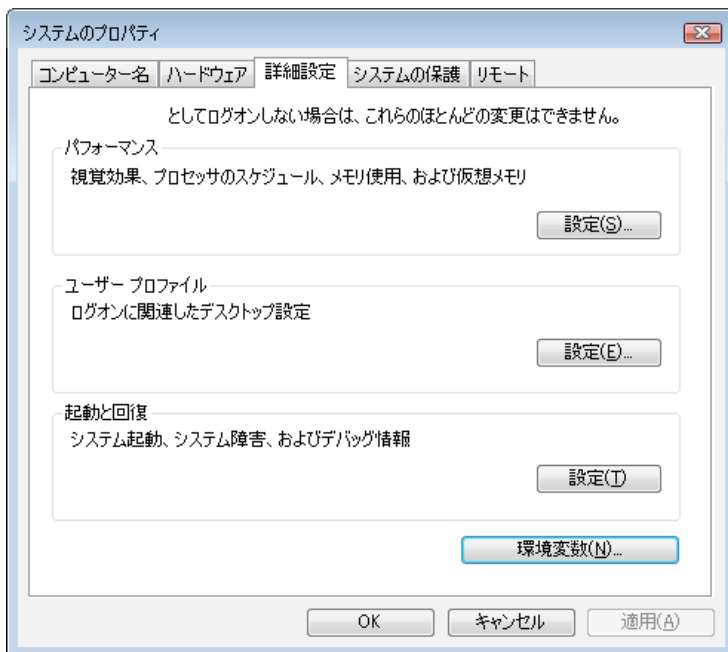
値：GDAL ライブラリのあるパス。

Windows 7 の場合、デフォルトで C:\Program Files\PostgreSQL\9.3\gdal-data

・システム環境変数 POSTGIS\_GDAL\_ENABLED\_DRIVERS

値：ENABLE\_ALL (GDAL で使える全ファイル形式が有効になります。本資料では PNG のみ使います)

↓ 環境変数の確認の例。Windows 7 なら、コントロールパネル → システム → システムの詳細設定 → 詳細設定タブ → 環境変数 → システム環境変数で確認します。



□ PostgreSQL のユーザ名・パスワードを分かっている

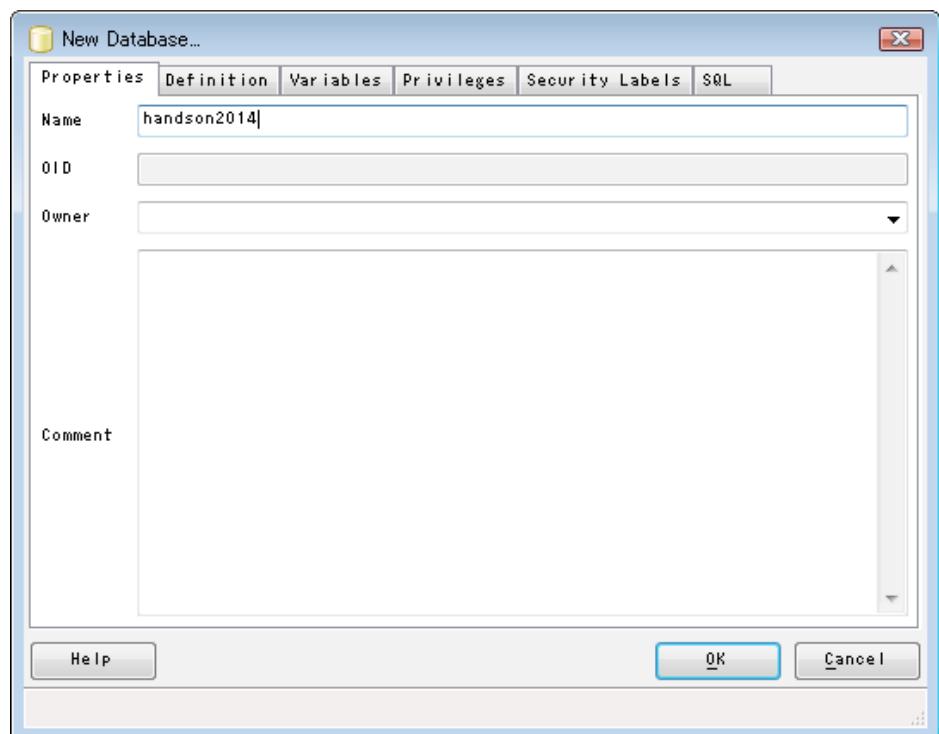
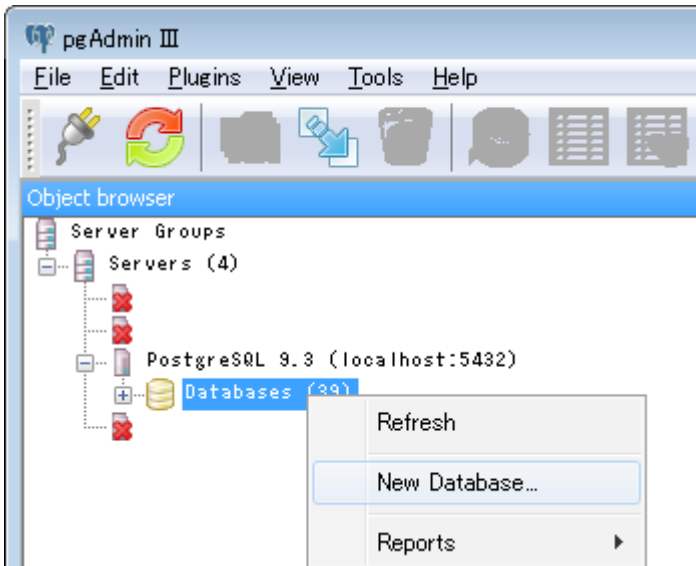
・今日の作業に使う PostgreSQL のユーザ名・パスワードを確認します。

・特に事情がなければ PostgreSQL のインストール時に作ったユーザ (postgres) とパスワードで構いません。

### (3) 作業環境の準備

#### ■ 作業用データベースの作成

- ・既に作成済みの人は、スキップして構いません。
- ・下記は、pgAdminIII で作業用データベースを handson2014 という名で作る例です。オブジェクトブラウザで「データベース」を選択して右クリックし、データベース作成のメニューを選ぶとダイアログが出ます。
- ・設定用のタブがたくさんありますが、今回は、データベース名以外に特に入力不要です。



#### ■ 作業用データベースで PostGIS を使えるよう準備

- ・既に終わっている人はスキップして構いません。作業用データベースに接続して（pgAdminIII ならオブジェクトブラウザで当該データベースを選択して）下記の SQL を実行します。

---

```
CREATE EXTENSION postgis ;
```

---

- データベースに入れた PostGIS の詳細バージョンを確認するため、下記の SQL を実行します。

---

```
SELECT * FROM postgis_full_version() ;
```

---

↓ こんな風に長い文字列が返され、最後に RASTER とあるはずですが。

Output pane	
Data Output	
postgis_full_version text	
1	POSTGIS="2.1.3 r12547" GEOS="3.4.2-CAPI-1.8.2 r3924" PROJ="Rel. 4.8.0, 6 March 2012" GDAL="G'

- PostGIS ラスタが、PostgreSQL のデータ型としてきちんと定義されているか、下記の SQL で確認します。WHERE の次は typname でなく typname です。

---

```
SELECT * FROM pg_type WHERE typname LIKE 'rast%' ;
```

---

↓ こんな風に四つのデータ型が返るはずですが。これで確認は終了します。

Output pane								
Data Output								
	typname	typnamespace	typowner	typplen	typbyval	typtype	typcategory	typis
	name	oid	oid	smallint	boolean	"char"	"char"	boole
1	rastbandarg	2200	10	-1	f	c	C	f
2	raster	2200	10	-1	f	b	U	f
3	raster_columns	2200	10	-1	f	c	C	f
4	raster_overviews	2200	10	-1	f	c	C	f

## 2. 前半の作業内容の説明

- まず 3. で、PostgreSQL / PostGIS ラスタの動作確認をかねて色付き 1 ピクセル画像の作成・出力を行います。
- SQL で 1 ピクセルのラスタを作り、色を付け、PNG ファイルに変換し、HTML で拡大して可視化します。
- 一つの SQL を実行するだけの単純な作業ですが、PostGIS ラスタの作成方法、基本構造、出力例が分かります。
- この章の資料では、今日使う主な SQL の確認をかね、一つの SQL を一手順ごとに分解して説明しています。
- 次に 4 (1) ~ (3) で、今回の柱となる OpenStreetMap タイルの取り込みを行います。
- OpenStreetMap の予備知識がなくてもできるよう、(1) (2) (3) で一歩ずつ行う手順になっています。
- (1) でタイル 1 枚を準備し、PostGIS ラスタとして取り込みます。これは、OpenStreetMap に限らず外部データを PostGIS ラスタにインポートする基本になります。
- (2) で複数のタイルを取り込み、相対的な位置関係を入力します。これは、PostGIS ラスタのパラメータを SQL で設定する方法の基本になります。
- (3) で、OpenStreetMap タイルの URL から経緯度を計算し PostGIS ラスタのパラメータとして入力します。これでラスタが、単なる画像でなく「地理情報」として扱えるようになります。
- ここまでの作業を前半として、いったん休憩する予定ですが、進み具合によっては途中で休憩を入れます。





---

### ▼ SQL 3-4

---

-- 先ほど作った PostGIS ラスタを PNG に変換し、新しい列として追加します

```
WITH a (colorname, r, g, b) AS (
    VALUES ('red', 255, 0, 0), ('green', 0, 255, 0), ('blue', 0, 0, 255)
), b (rast) AS (
    SELECT ST_MakeEmptyRaster(1, 1, 0, 0, 1) AS rast
), c AS (
    SELECT a.colorname, ST_AddBand(b.rast, ARRAY[
        ROW(1, text '8BUI', a.r, NULL),
        ROW(2, text '8BUI', a.g, NULL),
        ROW(3, text '8BUI', a.b, NULL)] :: addbandarg[] ) AS rast
    FROM a, b
)
SELECT *, ST_AsPNG(rast) AS png
FROM c ;
```

	colorname text	rast raster	png bytea
1	red	010000030000000000000000F03F000000000000	¥211PNG¥015¥012¥032¥012¥000¥000¥000¥015IHDR¥000¥00
2	green	010000030000000000000000F03F000000000000	¥211PNG¥015¥012¥032¥012¥000¥000¥000¥015IHDR¥000¥00
3	blue	010000030000000000000000F03F000000000000	¥211PNG¥015¥012¥032¥012¥000¥000¥000¥015IHDR¥000¥00

---

### ▼ SQL 3-5

---

-- 上で作った PNG をそのまま出力できないので「画像埋め込み HTML」に変換します

```
WITH a (colorname, r, g, b) AS (
    VALUES ('red', 255, 0, 0), ('green', 0, 255, 0), ('blue', 0, 0, 255)
), b (rast) AS (
    SELECT ST_MakeEmptyRaster(1, 1, 0, 0, 1) AS rast
), c AS (
    SELECT a.colorname, ST_AddBand(b.rast, ARRAY[
        ROW(1, text '8BUI', a.r, NULL),
        ROW(2, text '8BUI', a.g, NULL),
        ROW(3, text '8BUI', a.b, NULL)] :: addbandarg[] ) AS rast
    FROM a, b
), d AS (
    SELECT *, ST_AsPNG(rast) AS png FROM c
)
SELECT ' ' ||
    colorname || '<br />'
    -- PNG を Base64 というテキスト形式にエンコードし HTML に埋め込みます
    -- これなら各行がテキストになるので、PostgreSQL の COPY コマンドで出力できます
    -- ただし Base64 中に改行コードがあると、COPY コマンドが \n という文字に変換してしまうので、
    -- replace 関数を使って Base64 中の改行コードを削除します
FROM d ;
-- 結果は次頁
```

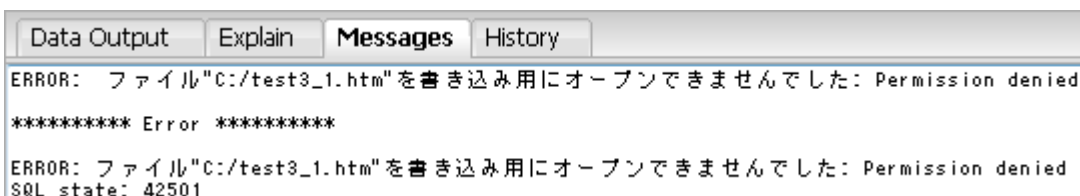
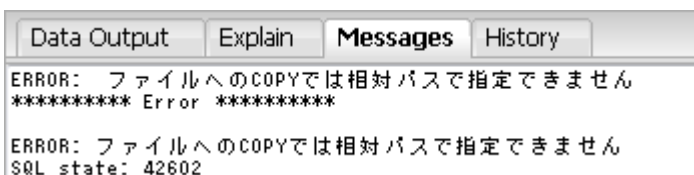


	?column? text	
1	'	
    colorname || '<br />'
  FROM d
)
TO 'C:/Program Files/PostgreSQL/9.3/data/part3_output_1.htm' ;
-- 出力ファイルは絶対パスで、PostgreSQL インストール時に設定したデータフォルダ内にします
```

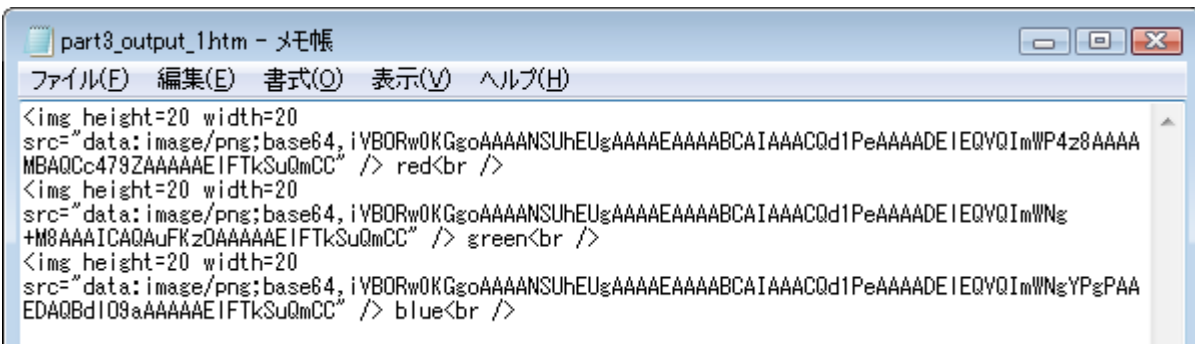
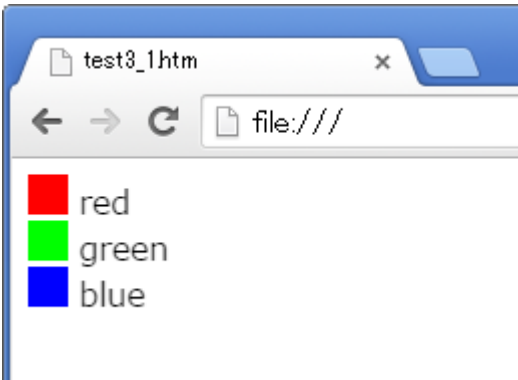
↓ 実行できたら、pgAdmin ではこのように表示されます



↓ 出力ファイルの場所指定が正しくないと、このようなエラーが出ます

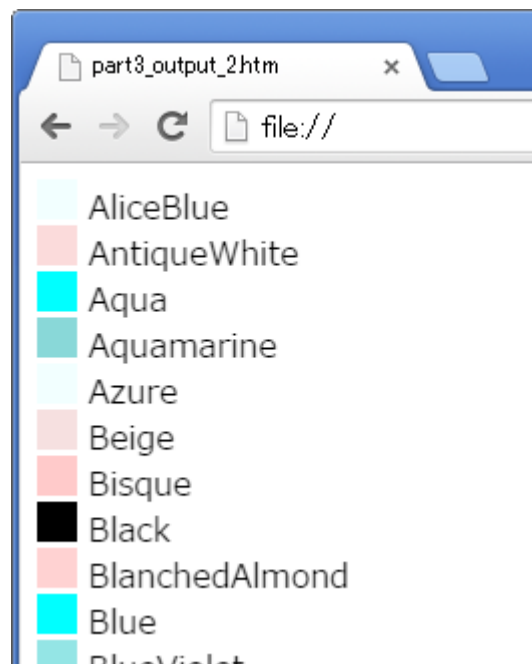
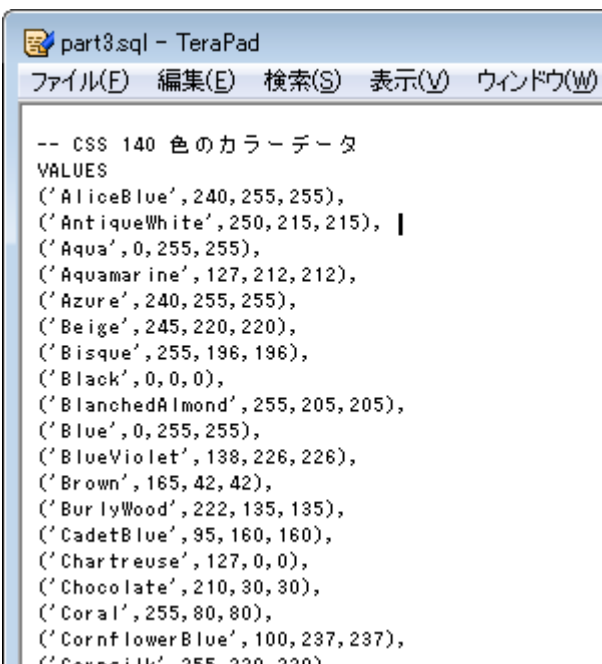


↓ 前頁の COPY コマンドで出力した HTML をブラウザで開くと、このようになります。色付きの四角が、1 ピクセル画像を少し大きく拡大したものです。また HTML をメモ帳などエディタで開くと、IMG タグの src に Base64 のテキストが書き込まれていると分かります。



- ・以上が、動作確認をかねた Hello World 的な PostGIS のテストです。
- ・SQL だけでラスタを作り、画像ファイルに変換して、ややトリッキーな方法ですが外部出力して確認しました。
- ・1 ピクセル画像に限らずどんなラスタでも、同じように SQL だけで外部出力して確認できます。

↓ 予備作業用に、140 色の RGB 値のデータを SQL で用意しました。(part3.sql の最後)  
 Web ページのスタイルシート (CSS) で定義されているものです。  
 これを本章の SQL の a ブロックに使うと、CSS 140 色の 1 ピクセルラスタや、色一覧の HTML 出力ができます。



## 4. OpenStreetMap タイル画像の取り込み（1）まずタイル 1 枚から

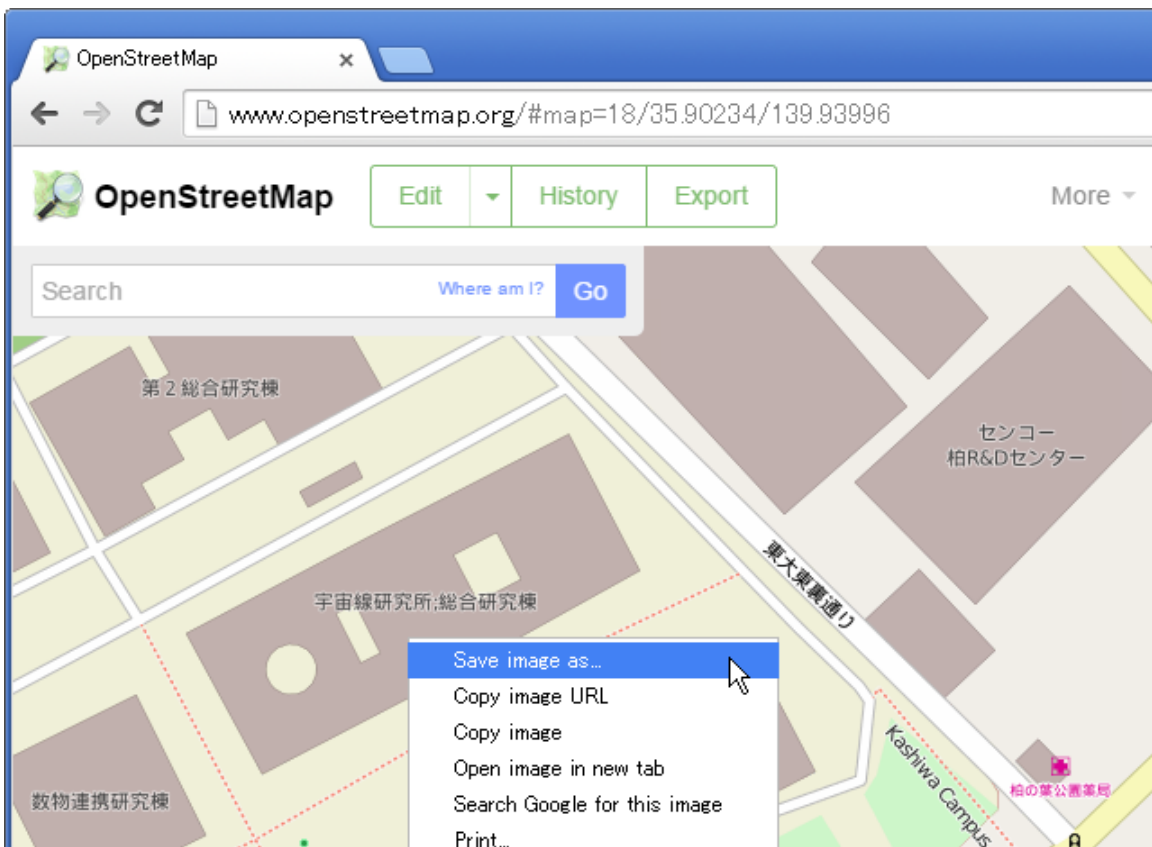
### ▼ WORK 4-1-1

- ・ブラウザで <http://www.openstreetmap.org/> を開き、適当な地点で右クリックして地図画像を保存します。
- ・普通はこの画像が、右クリックした地点のタイルになります。
- ・うまくタイルを保存できない方は、下記の URL を使って下さい。配布データの中にも入っています。

地図 … <http://www.openstreetmap.org/#map=18/35.902/139.939>

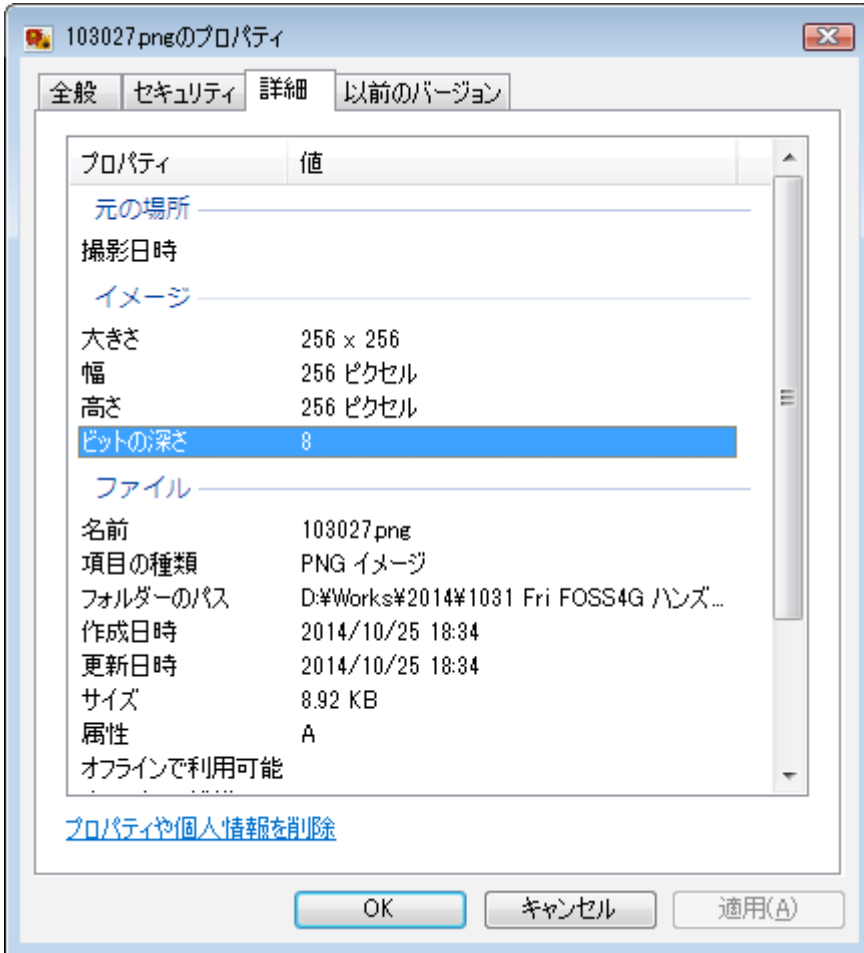
タイル … <http://tile.openstreetmap.org/18/232972/103027.png>

- ・ダウンロードしたら、適当な作業フォルダを作ってそこに入れます。

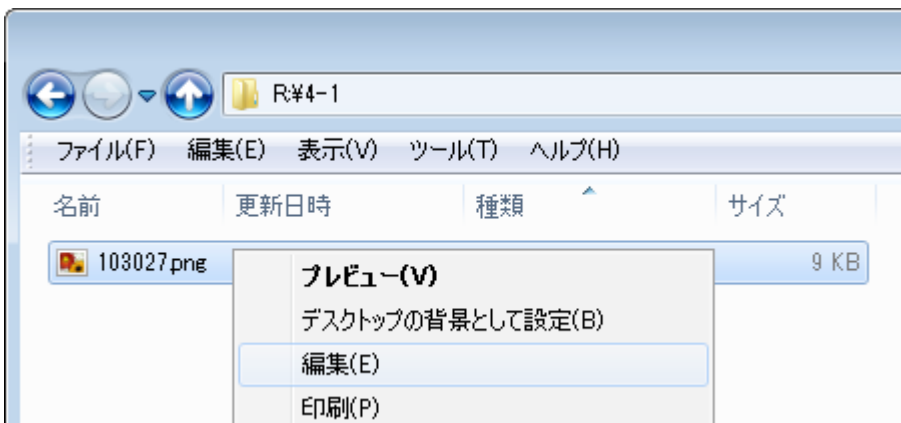


▼ WORK 4-1-2

- OpenStreetMap の標準地図のタイルは「インデックスカラー」の PNG ファイルです。
- 確認する例：Windows 7 のエクスプローラで作業フォルダを開く  
→ ファイルを右クリック → メニューの「プロパティ」を選択 → 詳細タブを表示
- 下のように「ビットの深さ」が 8、つまり 256 色しかありません。



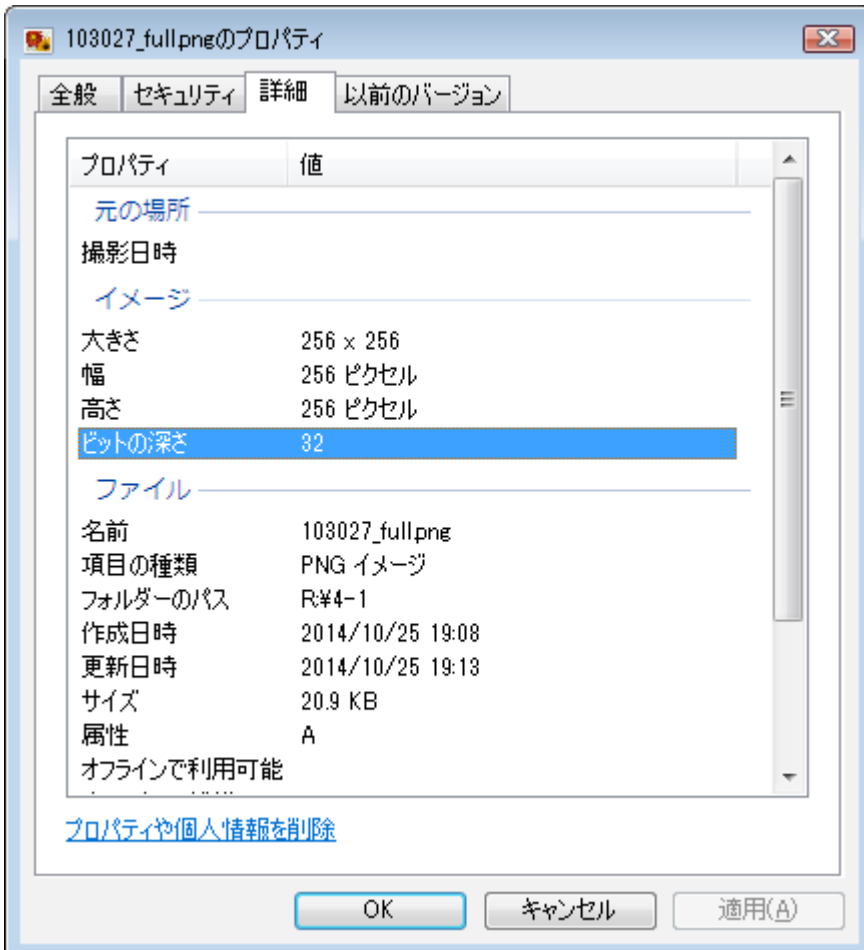
- このまま PostGIS に取り込むと、色の情報が失われます。(色に付けられた番号 = インデックスだけになる)
- フルカラーに変換すれば PostGIS に取り込めます。
- 変換といっても Windows 7 なら簡単で、ペイント (mspaint.exe) で開いて保存し直すだけです。
- Windows 7 のデフォルトの状態では、PNG ファイルを右クリック → 編集で、ペイントで開けます。



- ・上書き保存、または変換したことが分かるファイル名を付けて保存します。(以下の手順では後者)
- ・上手くいかない人は、配布データ中の 103027\_full.png を使って下さい。



- ・フルカラーのビットの深さは 24 または 32 です。(前者は RGB、後者は RGB + 透明度)
- ・ペイントで保存したファイルのプロパティを見ると ↓ 32 です。これで PostGIS に取り込む準備ができました。



▼ WORK 4-1-3

- フルカラーに変換した PNG ファイルと同じ場所に、PostGIS へのインポート用バッチファイルを作ります。
- 配布データにある raster2pgsql\_41.bat.txt がひな形です（下が内容）。拡張子を .bat にして使います。

```
@ECHO OFF
"C:\Program Files\PostgreSQL\9.3\bin\raster2pgsql.exe" *.png -F raster41 > tmp41.sql
"C:\Program Files\PostgreSQL\9.3\bin\psql.exe" -d handson2014 -U postgres -W -f tmp41.sql
PAUSE
```

- 上のひな形は、PostgreSQL / PostGIS のプログラムフォルダなど諸設定を、次のように仮定しています。

- プログラムフォルダ … C:\Program Files\PostgreSQL\9.3\bin (インストール時のデフォルト)
- データベース設定    host : localhost  
                          port : 5432 (いずれもインストール時のデフォルト)
- インポート先データベース名 : handson2014
- インポート先テーブル名 : raster41 (スキーマ設定はないので、デフォルトなら public 下になる)
- インポート用の一時 SQL ファイル名 : tmp41.sql

- 上で使っている raster2pgsql.exe と psql.exe の各パラメータの意味は、次のとおりです。

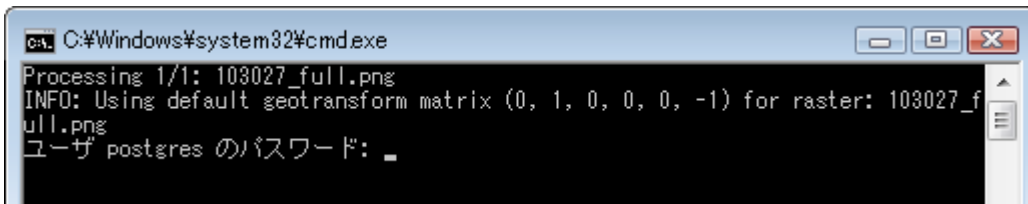
| コマンド         | パラメータ          | 意味                                                                           |
|--------------|----------------|------------------------------------------------------------------------------|
| raster2pgsql | *.png          | バッチファイルと同じフォルダにある全ての PNG をインポートする                                            |
|              | -F             | インポート先テーブルに filename 列を作り、ファイル名を入力する                                         |
|              | raster41       | インポート先テーブル名                                                                  |
|              | > tmp41.sql    | PNG から作成したインポートクエリを tmp41.sql に出力する<br>( raster2pgsql のパラメータでなく、コマンドラインの文法 ) |
| psql         | -d handson2014 | インポート先データベースを handson2014 に設定                                                |
|              | -U postgres    | インポートクエリを実行するユーザを postgres に設定                                               |
|              | -W             | インポート時にユーザのパスワードを入力させる                                                       |
|              | -f tmp41.sql   | tmp41.sql にある内容でクエリを実行する                                                     |

- raster2pgsql の \*.png と -F は、順序を入れ替えても実行できます。
- psql の各パラメータは、順序を入れ替えても実行できます。
- raster2pgsql と psql をパイプ | で連結し一行で実行する方法もあります。(一時ファイルが不要になる)

↓ バッチファイルができたなら、このようにタイル画像と同じ場所に置いて実行します。

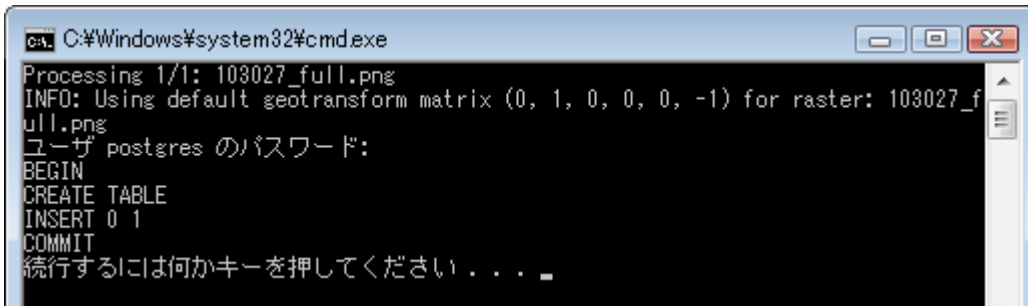


- ・インポートする PNG には位置情報がないので、デフォルトの geotransform matrix を使うとの INFO が出ます。
- ・raster2pgsql の処理が問題なく終わると psql に処理が移り、パスワードを要求して止まります。



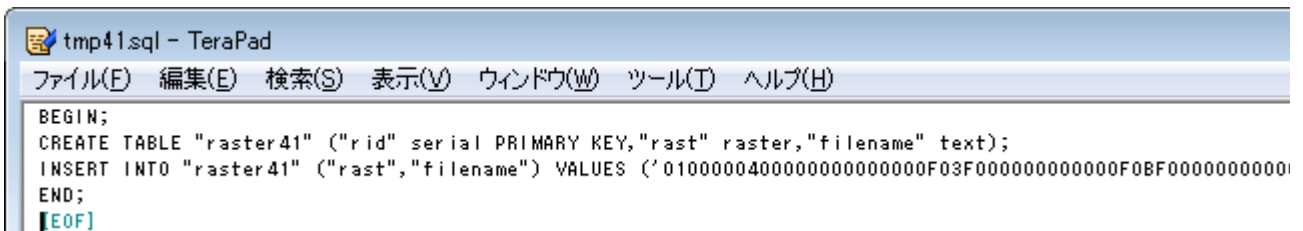
```
C:\Windows\system32\cmd.exe
Processing 1/1: 103027_full.png
INFO: Using default geotransform matrix (0, 1, 0, 0, 0, -1) for raster: 103027_full.png
ユーザ postgres のパスワード: _
```

- ・psql の処理が問題なく終わると ↓ このようになります。INSERT 0 1 が、ラスタを一行入力したという意味。



```
C:\Windows\system32\cmd.exe
Processing 1/1: 103027_full.png
INFO: Using default geotransform matrix (0, 1, 0, 0, 0, -1) for raster: 103027_full.png
ユーザ postgres のパスワード:
BEGIN
CREATE TABLE
INSERT 0 1
COMMIT
続行するには何かキーを押してください . . . _
```

- ・一時ファイル tmp41.sql の中身は下のようになっています。
- ・psql の処理で何かエラーが出た場合、この一時ファイルが残るので問題を検証できます。



```
tmp41.sql - TeraPad
ファイル(E) 編集(E) 検索(S) 表示(V) ウィンドウ(W) ツール(T) ヘルプ(H)
BEGIN;
CREATE TABLE "raster41" ("rid" serial PRIMARY KEY, "rast" raster, "filename" text);
INSERT INTO "raster41" ("rast", "filename") VALUES ('010000040000000000000000F03F000000000000F0BF0000000000',
END;
[EOF]
```

- ・以上が、外部ファイルを PostGIS ラスタにインポートする基本的なひな形です。
- ・raster2pgsql の他のパラメータについては、下記マニュアルを参照して下さい。

■ [http://www.finds.jp/docs/pgisman/2.2.0/using\\_raster\\_dataman.html#RT\\_Raster\\_Loader](http://www.finds.jp/docs/pgisman/2.2.0/using_raster_dataman.html#RT_Raster_Loader)

▼ WORK 4-1-4

・インポート結果の確認です。各 SQL は part41.sql にあります。まず単純にテーブルの各列を表示します。

※ テーブルの行数が多い場合や、行数が少なくてもラスタのサイズが大きい場合、このようにラスタの列を直接表示させると非常に時間がかかるので避けた方がいいです。

SELECT \* FROM raster41 ;

|   | rid<br>integer | rast<br>raster                                                  | filename<br>text |
|---|----------------|-----------------------------------------------------------------|------------------|
| 1 | 1              | 010000004000000000000000F03F000000000000F0BF0000000000000000000 | 103027_full.png  |

- ・次に ST\_MetaData 関数でラスタの基本的な情報を表示します。
- ・この関数は「複合型」という PostgreSQL 独自の型を返します。複数の列が一まとまりになっている型です。
- ・複合型の各列を展開するため (ST\_MetaData(rast)).\* という式を使います。

SELECT (ST\_MetaData(rast)).\* FROM raster41 ; -- 列数が多いので、結果を 2 行に分けて示します

|   | upper leftx<br>double precision | upper lefty<br>double precision | width<br>integer | height<br>integer | scalex<br>double precision | scaley<br>double precision |
|---|---------------------------------|---------------------------------|------------------|-------------------|----------------------------|----------------------------|
| 1 | 0                               | 0                               | 256              | 256               | 1                          | -1                         |

|  | skewx<br>double precision | skewy<br>double precision | srid<br>integer | numbands<br>integer |
|--|---------------------------|---------------------------|-----------------|---------------------|
|  | 0                         | 0                         | 0               | 4                   |

- ・ST\_MetaData が返すラスタの情報は次のとおりです。列名からもある程度分かります。

| 列名         | ラスタの情報                                              |
|------------|-----------------------------------------------------|
| upperleftx | 左上端 (upperleft) の X 座標                              |
| upperlefty | 左上端 (upperleft) の Y 座標                              |
| width      | 左右方向のピクセル数                                          |
| height     | 上下方向のピクセル数                                          |
| scalex     | 左右方向の 1 ピクセル当たりの座標上の長さ                              |
| scaley     | 上下方向の 1 ピクセル当たりの座標上の長さ (軸の方向を合わせるため、基本的に負数)         |
| skewx      | 左右方向に対する歪み (よく回転パラメータと呼ばれるもの。単位は scalex・scaley と同じ) |
| skewy      | 上下方向に対する歪み ( " )                                    |
| srid       | 測地投影系 ID                                            |
| numbands   | バンド数                                                |

マニュアル : ST\_MetaData 関数

■ [http://www.finds.jp/docs/pgsman/2.2.0/RT\\_ST\\_MetaData.html](http://www.finds.jp/docs/pgsman/2.2.0/RT_ST_MetaData.html)

- ・インポート時に位置情報がなかったため、width, height, numbands 以外がデフォルト値で設定されました。
- ・width と height はインポートした画像ファイルのピクセル数と同じです。(今回は 256 × 256)
- ・バンド数の 4 は、PNG ファイルの R・G・B・透明度という 4 種類の値が入っていることを示します。
- ・skewx と skewy は、GIS でよく「回転パラメータ」と言われる値です。正確には回転と少し違います。
- ・skewx と skewy の正確な意味は、英語ですが [http://en.wikipedia.org/wiki/World\\_file](http://en.wikipedia.org/wiki/World_file) が参考になります。



- ・前頁で4つのバンドがあると分かったので、各バンドの情報を ST\_BandMetaData 関数で表示します。
- ・今回の場合、実質的に意味がある列は pixeltype だけです。7 頁にまとめた「ピクセル値のタイプ」です。

```
SELECT band, (ST_BandMetaData(rast, band)).*
FROM raster41, generate_series(1, 4) AS band;
```

|          | <b>band</b>    | <b>pixeltype</b> | <b>nodatavalue</b>      | <b>isoutdb</b> | <b>path</b> |
|----------|----------------|------------------|-------------------------|----------------|-------------|
|          | <b>integer</b> | <b>text</b>      | <b>double precision</b> | <b>boolean</b> | <b>text</b> |
| <b>1</b> | 1              | 8BUI             |                         | f              |             |
| <b>2</b> | 2              | 8BUI             |                         | f              |             |
| <b>3</b> | 3              | 8BUI             |                         | f              |             |
| <b>4</b> | 4              | 8BUI             |                         | f              |             |

マニュアル：ST\_MetaData 関数

■ [http://www.finds.jp/docs/pgisman/2.2.0/RT\\_ST\\_BandMetaData.html](http://www.finds.jp/docs/pgisman/2.2.0/RT_ST_BandMetaData.html)

- ・確認の最後に、前章と同じ手順で「画像埋め込み HTML」に出力します。前章 SQL 3-6 の後半と同じです。

COPY (

```
-- SQL3-6 にあったブロック a ~ c が、ここでは不要（既にラスタがあるから）
```

```
WITH d AS (
```

```
    SELECT ST_AsPNG(rast) AS png FROM raster41
```

```
)
```

```
SELECT ''
```

```
-- 画像をそのまま埋め込むだけなので、IMG タグが SQL3-6 より簡単
```

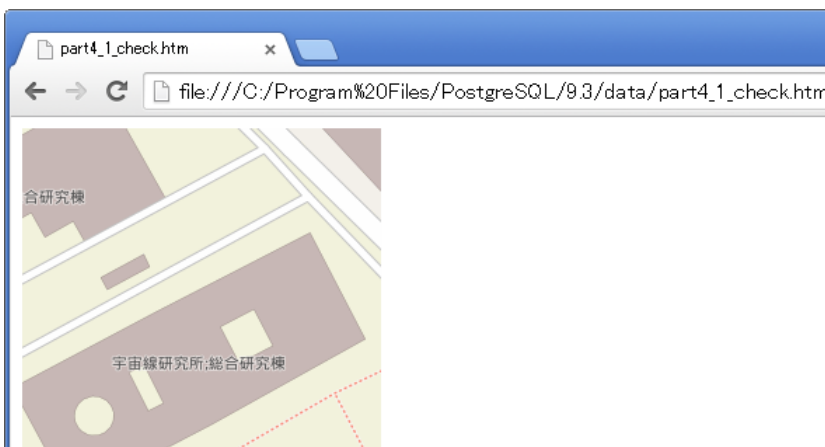
```
FROM d
```

)

```
TO 'C:/Program Files/PostgreSQL/9.3/data/part41_check.htm' ;
```

```
-- 出力ファイルは絶対パスで、PostgreSQL インストール時に設定したデータフォルダ内にします
```

↓ 出力が成功すると



## 4. OpenStreetMap タイル画像の取り込み（2）複数タイルをインポート

### ▼ WORK 4-2-1

- 先ほどの WORK 4-1-1 と同様に、適当な地点の OpenStreetMap タイルを、今度は複数ダウンロードします。
- 後でタイルの位置関係を入力するので、URL の数字を適当な形で（例えば \_ 区切りで）ファイル名に入れます。
- タイルをうまく保存できない方は、下記 4 つの URL を使って下さい。配布データにも入っています。

地図 … <http://www.openstreetmap.org/#map=18/35.902/139.939>

タイル … <http://tile.openstreetmap.org/18/232972/103027.png>

<http://tile.openstreetmap.org/18/232973/103027.png>

<http://tile.openstreetmap.org/18/232972/103028.png>

<http://tile.openstreetmap.org/18/232973/103028.png>

### ▼ WORK 4-2-2

- 先ほどの WORK 4-1-2 と同様に、タイル画像をフルカラーに変換します。
- うまくできない方は、配布データ（ファイル名に \_full が付いているもの）を使って下さい。
- このフルカラー画像だけ適当な作業フォルダに移します。（インポート時のファイル指定が \*.png で済みます）

### ▼ WORK 4-2-3

- 先ほどの WORK 4-1-3 と同様に、インポートする画像と同じフォルダにバッチファイルを作り実行します。
- 変更点は、基本的にインポート先テーブルだけです。ここでは一応、一時ファイル名も変えました。
- 下記の内容のバッチファイルが、配布データの raster2pgsql\_42.bat.txt です。拡張子を .bat にして使います。

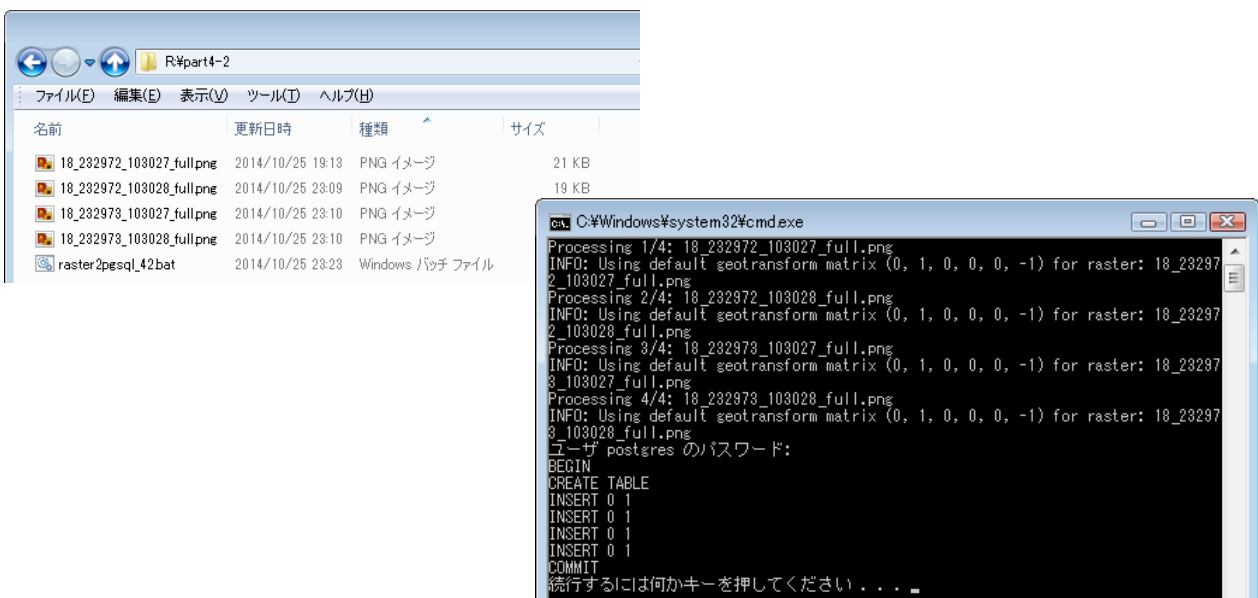
```
@ECHO OFF
```

```
"C:\Program Files\PostgreSQL\9.3\bin\raster2pgsql.exe" *.png -F raster42 > tmp42.sql
```

```
"C:\Program Files\PostgreSQL\9.3\bin\psql.exe" -d handson2014 -U postgres -W -f tmp42.sql
```

```
PAUSE
```

↓ こんな感じでバッチファイルを置き、実行します。今度は 4 枚の画像 → ラスタ 4 行に取り込まれました。



▼ WORK 4-2-4

- 先ほどと同様、ST\_MetaData 関数でラスタの属性を SQL で取得します。
- まだラスタに位置情報を設定していないので、すべて同じ位置にあるのが分かります。
- 下では最左列の upperleftx から、scaley までを示しています。

```
SELECT (ST_MetaData(rast)).* FROM raster42 ;
```

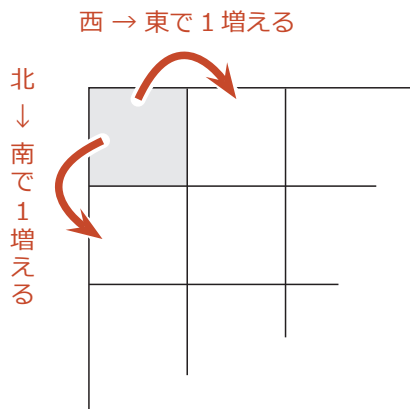
|          | <b>upper leftx</b><br><b>double precision</b> | <b>upper lefty</b><br><b>double precision</b> | <b>width</b><br><b>integer</b> | <b>height</b><br><b>integer</b> | <b>scalex</b><br><b>double precision</b> | <b>scaley</b><br><b>double precision</b> |
|----------|-----------------------------------------------|-----------------------------------------------|--------------------------------|---------------------------------|------------------------------------------|------------------------------------------|
| <b>1</b> | 0                                             | 0                                             | 256                            | 256                             | 1                                        | -1                                       |
| <b>2</b> | 0                                             | 0                                             | 256                            | 256                             | 1                                        | -1                                       |
| <b>3</b> | 0                                             | 0                                             | 256                            | 256                             | 1                                        | -1                                       |
| <b>4</b> | 0                                             | 0                                             | 256                            | 256                             | 1                                        | -1                                       |

▼ WORK 4-2-5

- ここから、複数タイルの位置情報入力になります。
- 実在の経緯度と OpenStreetMap タイルとの関係は少し複雑なので、まず相対的な位置でやってみます。
- タイル URL にあるスラッシュで区切られた三つの数字の意味は ↓ のとおりです。

<http://tile.openstreetmap.org/> / ズームレベル / 横方向のタイル番号 / 縦方向のタイル番号 .png

- OpenStreetMap のタイル番号の相対的な順序は、次のようになっています。



- タイルの URL のうち横方向・縦方向のタイル番号を、何らかの形式でファイル名に入力していれば、インポート先テーブルの filename 列にも文字列として入っています。
- この filename 列に正規表現の関数 substring を使うと、縦横のタイル番号を取り出せます。例えば下のよう。

-- SQL4-2-51

```
SELECT filename,
       substring(filename, '^(\d+_(\d+))') AS xtile, substring(filename, '_(\d+)_full') AS ytile
FROM raster42 ;
```

|          | <b>filename</b><br><b>text</b> | <b>xtile</b><br><b>text</b> | <b>ytile</b><br><b>text</b> |
|----------|--------------------------------|-----------------------------|-----------------------------|
| <b>1</b> | 18_232972_103027_full.png      | 232972                      | 103027                      |
| <b>2</b> | 18_232972_103028_full.png      | 232972                      | 103028                      |
| <b>3</b> | 18_232973_103027_full.png      | 232973                      | 103027                      |
| <b>4</b> | 18_232973_103028_full.png      | 232973                      | 103028                      |

- ・ラスタ各行における縦横のタイル番号を得られたら、SQL で、縦横それぞれの最小の番号を原点とする相対位置が  
出せます。次のクエリがその例です。

```
-- SQL4-2-52
```

```
WITH a AS (
    SELECT filename,
           substring(filename, '^\\d+_\\d+') :: int AS xtile,
           substring(filename, '\\d+_full') :: int AS ytile
    FROM raster42
), b AS (
    SELECT min(xtile) xmin, min(ytile) ymin FROM a
)
SELECT filename, xtile - xmin AS xloc, ytile - ymin AS yloc
FROM a, b ;
```

|   | filename<br>text          | xloc<br>integer | yloc<br>integer |
|---|---------------------------|-----------------|-----------------|
| 1 | 18_232972_103027_full.png | 0               | 0               |
| 2 | 18_232972_103028_full.png | 0               | 1               |
| 3 | 18_232973_103027_full.png | 1               | 0               |
| 4 | 18_232973_103028_full.png | 1               | 1               |

- ・横の相対位置 \* width \* scalex が、ラスタの左上端の X 座標 (upperleftx) になります。縦も同様です。
- ・先ほど ST\_MetaData 関数で見たとおり、どのラスタも width \* scalex = 256, height \* scaley = -256 です。
- ・従ってラスタの upperleftx ・ upperlefty を修正入力する SQL は、次のようになります。

```
-- SQL4-2-53
```

```
WITH a AS (
    SELECT filename,
           substring(filename, '^\\d+_\\d+') :: int AS xtile,
           substring(filename, '\\d+_full') :: int AS ytile
    FROM raster42
), b AS (
    SELECT min(xtile) AS xmin, min(ytile) AS ymin FROM a
), c AS (
    SELECT filename, (xtile - xmin) * 256 AS xloc, (ytile - ymin) * -256 AS yloc
    FROM a, b
    -- ここまでが、各ラスタの filename および修正入力する upperleftx , upperlefty の準備
)
UPDATE raster42 AS x
SET rast = ST_SetUpperLeft(rast, c.xloc, c.yloc)
FROM c WHERE x.filename = c.filename ;
-- ラスタテーブルと、上の c ブロックを filename で結合し、更新
```

↓ 実行した結果。

| Data Output                                                         | Explain | Messages | History |
|---------------------------------------------------------------------|---------|----------|---------|
| Query returned successfully: 4 rows affected, 40 ms execution time. |         |          |         |

↓ 再び ST\_MetaData で確認。横（東西）の軸方向は PostGIS とタイル番号で同じですが、縦（南北）は逆なので、縦タイル番号が大きいものほど upperlefty は小さくなります。

```
SELECT filename, (ST_MetaData(rast)).* FROM raster42 ;
```

|   | filename<br>text          | upper leftx<br>double prec | upper lefty<br>double prec | width<br>integer | height<br>integer | scalex<br>double p | scaley<br>double p |
|---|---------------------------|----------------------------|----------------------------|------------------|-------------------|--------------------|--------------------|
| 1 | 18_232972_103027_full.png | 0                          | 0                          | 256              | 256               | 1                  | -1                 |
| 2 | 18_232972_103028_full.png | 0                          | -256                       | 256              | 256               | 1                  | -1                 |
| 3 | 18_232973_103027_full.png | 256                        | 0                          | 256              | 256               | 1                  | -1                 |
| 4 | 18_232973_103028_full.png | 256                        | -256                       | 256              | 256               | 1                  | -1                 |

- ・前頁でラスタの upperleftx ・ upperleft を修正入力する際、ST\_SetUpperLeft 関数を使用しました。
- ・この関数は、文字通り左上端 UpperLeft の X ・ Y 座標を入力するものです。実際に使う時は、関数が返すラスタを修正前のラスタに差し替える、つまり **UPDATE 文として実行することが必要です**。
- ・この手続きは、ラスタへ何らかのパラメータや値を入力（修正）する際、常に必要になります。
- ・PostgreSQL では UPDATE に FROM や WITH を組み合わせられるので、修正入力する値の導出が多少複雑でも、前頁の SQL のように最後に UPDATE を付ける方法で、簡単にラスタを更新できます。

マニュアル：ST\_SetUpperLeft 関数

■ [http://www.finds.jp/docs/pgisman/2.2.0/RT\\_ST\\_SetUpperLeft.html](http://www.finds.jp/docs/pgisman/2.2.0/RT_ST_SetUpperLeft.html)

#### ▼ SQL 4-2-6

- ・入力した各タイルの相対位置が正しいかどうか、1 枚に貼り合わせて出力し、確認します。
- ・貼り合わせには ST\_Union 関数を使います。ジオメトリ用の ST\_Union とは別の、ラスタ用の関数です。
- ・ST\_Union の後は、先ほどと同じ「画像埋め込み HTML」への出力です。

```
COPY (
  WITH d AS (
    -- 複数タイルを 1 枚に統合し、続けて PNG に変換。ここ以外は前と基本的に同じ
    SELECT ST_AsPNG(ST_Union(rast)) AS png FROM raster42
  )
  SELECT ''
  FROM d
)
TO 'C:/Program Files/PostgreSQL/9.3/data/part42_check.htm' ;
-- 出力ファイルは絶対パスで、PostgreSQL インストール時に設定したデータフォルダ内にします
```

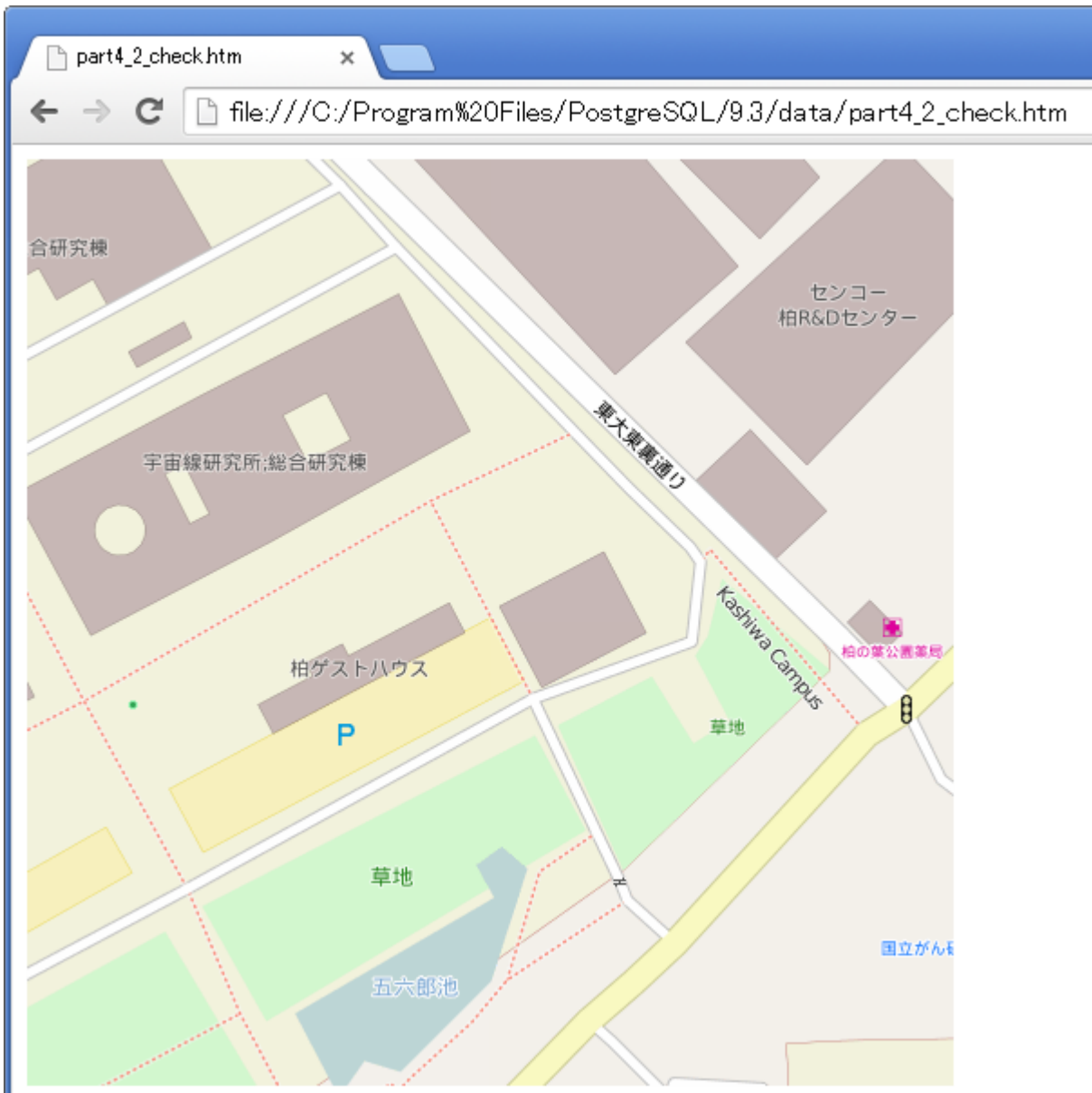
↓ 実行した結果。（出力された HTML は次頁）



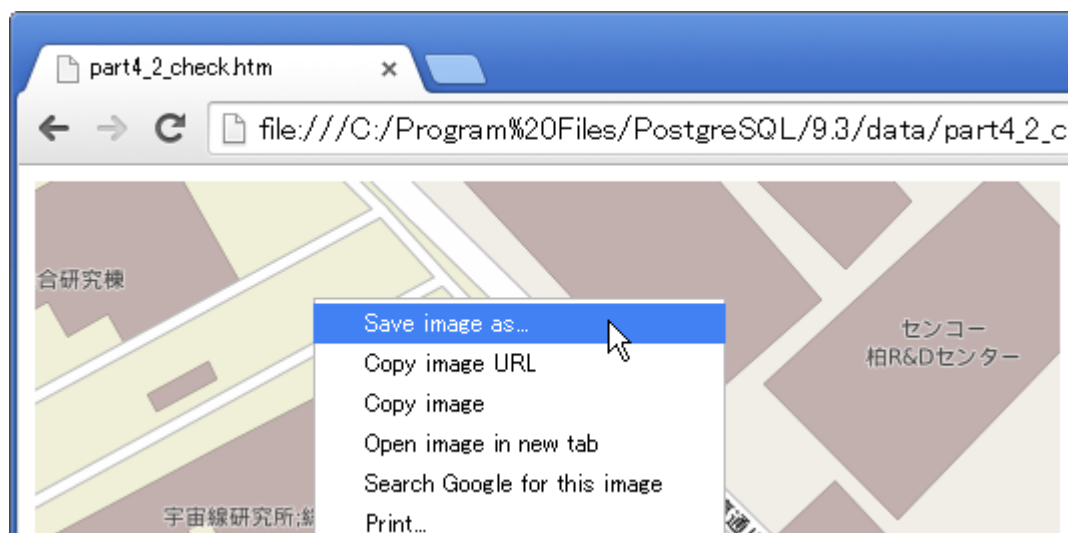
マニュアル：ST\_Union 関数（ラスタ用）

■ [http://www.finds.jp/docs/pgisman/2.2.0/RT\\_ST\\_Union.html](http://www.finds.jp/docs/pgisman/2.2.0/RT_ST_Union.html)

↓ このように複数タイルの貼り合わせができました。元のタイルサイズ（256 × 256）と比べて下さい。



↓ 画像埋め込み HTML から画像だけを抽出・保存するには、ブラウザの機能を使います。



## 4. OpenStreetMap タイル画像の取り込み (3) タイルに経緯度を設定

### ▼ WORK 4-3-1

- ・ここまで作成したラスタのうち、経緯度を設定する対象を決め、新規テーブルを作ってコピーします。
- ・新規テーブルには、タイルの URL (複数タイルの場合は左上端のタイルの URL) の列を含めます。
- ・また、縦・横それぞれのタイル数の列も作ります。
- ・以下、先ほど出力確認した複数タイルの貼り合わせ (ST\_Union) の結果を新規テーブルに使う例です。

```
CREATE TABLE raster43 AS
SELECT text 'http://tile.openstreetmap.org/18/232972/103027.png' AS upperleft_url,
       2 AS xtiles, -- 横方向のタイル数
       2 AS ytiles, -- 縦方向のタイル数
       ST_Union(rast) AS rast
FROM raster42 ;
```

- ・ここでは縦横のタイル数が 2 × 2 と単純なので、xtiles , ytiles に直接 2 を入力しました。
- ・タイル数が多い場合は、両端のタイル番号の差から計算してタイル数を求めます。

↓ 作成したテーブルの確認例です。列が多いので 2 行に分けて表示しました。

```
SELECT upperleft_url, xtiles, ytiles, (ST_Metadata(rast)).* FROM raster43 ;
```

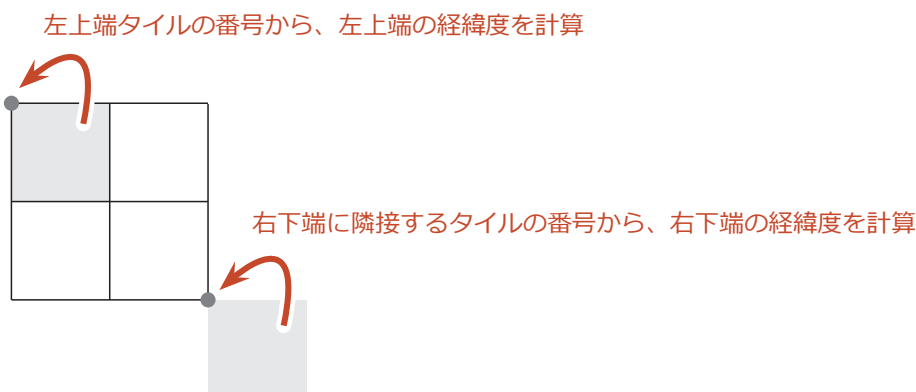
| Data Output |                                                    |                   |                   |                                 |                        |  |
|-------------|----------------------------------------------------|-------------------|-------------------|---------------------------------|------------------------|--|
|             | upper left_url<br>text                             | xtiles<br>integer | ytiles<br>integer | upper leftx<br>double precision | upper left<br>double p |  |
| 1           | http://tile.openstreetmap.org/18/232972/103027.png | 2                 | 2                 | 0                               |                        |  |

| upper lefty<br>double precision | width<br>integer | height<br>integer | scalex<br>double precision | scaley<br>double precision | skewx<br>double precision |  |
|---------------------------------|------------------|-------------------|----------------------------|----------------------------|---------------------------|--|
| 0                               | 512              | 512               | 1                          | -1                         | 0                         |  |

### ▼ SQL 4-3-2

- ・新規作成したテーブルの 3 列 : upperleft\_url と xtiles と ytiles から、ラスタ四隅の経緯度を出せます。
- ・その出し方の概略が、下図です。



- ・上の図で「タイルの番号から」とありますが、正確に言うとズームレベルも使います。
- ・次頁で、まず経緯度の導出だけ SQL でやってみます。それができたら、UPDATE 文でラスタに入力します。

```

-- とりあえずラスタの四隅の経緯度を見る
WITH a AS (
    -- URL に、正規表現関数を用いてズームレベル、横ラスタ番号、縦ラスタ番号を抽出し、数値に変換
    SELECT regexp_matches(upperleft_url, '/(\d+)/(\d+)/(\d+)\.png$') :: int[] AS reg,
           xtiles, ytiles
    FROM raster43
), b AS (
    -- 前ブロックで使った正規表現関数 regexp_matches の結果が配列なので、1 要素ずつ列に分ける
    -- さらにタイル番号 + 縦横のタイル数で右下に隣接するタイル番号を算出し、行として加える
    SELECT text 'upperleft' AS lab, reg[1] AS zoom, reg[2] AS xtile, reg[3] AS ytile
    FROM a
    UNION ALL SELECT 'lowerright', reg[1], reg[2] + xtiles, reg[3] + ytiles
    FROM a
), c AS (
    -- 複数回使う数値の準備 (1)
    SELECT *, 2 ^ zoom AS n FROM b
), d AS (
    -- タイル左上端の X 座標算出。同時に複数回使う数値の準備 (2)
    SELECT *, xtile / n * 360 - 180 AS lon, pi() * (1 - 2 * ytile / n) AS ytmp
    FROM c
)
SELECT lab, zoom, xtile, ytile, lon,
       degrees(atan((exp(ytmp) - exp(-ytmp)) / 2)) AS lat -- タイル左上端の Y 座標算出
FROM d ;

```

↓ 実行結果です。列 lab = upperleft の行が左上端、lowerright の行が右下端です。

|   | lab<br>text | zoom<br>integer | xtile<br>integer | ytile<br>integer | lon<br>double precision | lat<br>double precision |
|---|-------------|-----------------|------------------|------------------|-------------------------|-------------------------|
| 1 | upper left  | 18              | 232972           | 103027           | 139.938354492188        | 35.9035122565034        |
| 2 | lowerright  | 18              | 232974           | 103029           | 139.941101074219        | 35.901287478149         |

- ・上の SQL のブロック c 以降が、タイル番号とズームレベルから経緯度を求める計算部分です。
- ・計算式は下記の公式 Wiki にあります。その下の URL は筆者のブログで、同じ計算式を使った例です。

- [http://wiki.openstreetmap.org/wiki/Slippy\\_map\\_tilenames](http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames) (英語)
- <http://kenpg2.seesaa.net/article/407457559.html>

- ・これでラスタに位置情報を入力する準備ができました。
- ・ラスタに入力する左上端の経緯度は、上の upperleft の行の lon ・ lat の値をそのまま使います。
- ・ラスタに入力する scalex ・ scaley は、四隅の経度 (緯度) の差 ÷ 横 (縦) のピクセル数になります。
- ・以上を一回の SQL で行ないます。次の項がその内容です。

※ 経緯度の測地投影系 (SRID) は OpenStreetMap の仕様から 4326 (WGS 84) と分かっています。ここで入力してもいいですが、SQL の行数が増えてしまうので、別途行うことにします。



### ▼ SQL 4-3-3

- ・本章の最後の作業です。前頁で実行した SQL の結果をそのまま使ってラスタに位置情報を設定します。
- ・WITH と UPDATE でラスタを修正入力する方法は、20 頁の SQL4-2-53 と同じです。

```
WITH a AS (  
    SELECT regexp_matches(upperleft_url, '/(\d+)/(\d+)/(\d+)\.png$') :: int[] AS reg,  
           xtiles, ytiles  
    FROM raster43  
) , b AS (  
    SELECT text 'upperleft' AS lab, reg[1] AS zoom, reg[2] AS xtile, reg[3] AS ytile  
    FROM a  
    UNION ALL SELECT 'lowerright', reg[1], reg[2] + xtiles, reg[3] + ytiles  
    FROM a  
) , c AS (  
    SELECT *, 2 ^ zoom AS n FROM b  
) , d AS (  
    SELECT *, xtile / n * 360 - 180 AS lon, pi() * (1 - 2 * ytile / n) AS ytmp  
    FROM c  
    -- ここまで SQL4-3-2 と同じ  
) , e AS (  
    SELECT lab, zoom, xtile, ytile, lon,  
           degrees(atan((exp(ytmp) - exp(-ytmp)) / 2)) AS lat  
    FROM d  
    -- これが SQL4-3-2 で求めた結果  
) , f AS (  
    -- 左上端の経緯度と、ラスタの座標上の縦横サイズを準備。縦はマイナス  
    SELECT min(lon) AS upperleftx, max(lat) AS upperlefty,  
           (max(lon) - min(lon)) AS lon_width, (min(lat) - max(lat)) AS lat_height  
    FROM e  
)  
UPDATE raster43 AS x  
SET rast = ST_SetUpperLeft(  
    ST_SetScale(  
        x.rast, f.lon_width / ST_Width(x.rast), f.lat_height / ST_Height(x.rast)  
    ), f.upperleftx, f.upperlefty)  
FROM f ;
```

↓ 実行結果の確認のため、ラスタのメタデータを表示したところ。処理前（23 頁）と比較して下さい。

```
SELECT (ST_Metadata(rast)).* FROM raster43 ;
```

|          | <b>upper leftx</b><br><b>double precision</b> | <b>upper lefty</b><br><b>double precision</b> | <b>width</b><br><b>integer</b> | <b>height</b><br><b>integer</b> | <b>scalex</b><br><b>double precision</b> | <b>scaley</b><br><b>double precision</b> |
|----------|-----------------------------------------------|-----------------------------------------------|--------------------------------|---------------------------------|------------------------------------------|------------------------------------------|
| <b>1</b> | 139.938354492188                              | 35.9035122565034                              | 512                            | 512                             | 5.36441802978516e-006                    | -4.34527022345754e-006                   |

- ・これでラスタが、単なる画像でなく「地理情報」として使えるようになりました。
- ・次章でこのラスタを使い、簡単な SQL を試します。

## 5. 後半の作業内容の説明

- ・休憩の後、前半の進展具合と参加者の皆様からの意見を踏まえ、後半の内容を決めます。
- ・まず、前半でやり残した部分があればその続きを行います。
- ・希望があれば、前半の作業内容について追加的な説明や、質疑応答を行います。
- ・その後の予定として、前半に作った位置情報入りラスタに対し、いくつか SQL を試します。
- ・次に、ラスタの本格的な（画像埋め込み HTML という簡易なものでない）出力として、R を使う例を紹介します。

## 6. 位置情報を設定したラスタへの SQL 例

- ・使うラスタは 4. (3) で作ったもので、以下の説明ではテーブル名を raster43、列名 rast と仮定します。
- ・全ての SQL は配布データの part6.sql にあります。
- ・クエリの基本的な説明は SQL の冒頭にコメントとして記しています。

### ▼ SQL 6-1

-- 前章でやり残した SRID の設定を行います

```
UPDATE raster43 SET rast = ST_SetSrid(rast, 4326) ;
```

-- ラスタの SRID は ST\_Srid 関数で確認できます。↓ 実行前（左）と実行後（右）です。

```
SELECT ST_Srid(rast) FROM raster43 ;
```

|   | st_srid<br>integer |
|---|--------------------|
| 1 | 0                  |

|   | st_srid<br>integer |
|---|--------------------|
| 1 | 4326               |

### ▼ SQL 6-2

-- 最初に OpenStreetMap を開いた時の中心点（URL の経緯度）が、ラスタに含まれているかどうか調べます

-- 最初の URL を <http://www.openstreetmap.org/#map=18/35.902/139.939> を仮定します

```
SELECT ST_Intersects(rast, ST_SetSrid(ST_Point(139.939, 35.902), 4326)) FROM raster43 ;
```

|   | st_intersects<br>boolean |
|---|--------------------------|
| 1 | t                        |

|   | st_intersects<br>boolean |
|---|--------------------------|
| 1 | f                        |

↑ 左が含まれる場合（TRUE）、右が含まれない場合です。どちらになるかは保存したタイルの選び方によります。

- ・ある点がラスタに含まれるかどうかは、交差関係を調べる ST\_Intersects 関数で分かります。
- ・点だけでなくジオメトリ一般とラスタや、ラスタ同士の交差関係も、この ST\_Intersects で分かります。
- ・詳細は下記を参照して下さい。

マニュアル : ST\_Intersects 関数（ラスタ用）

- [http://www.finds.jp/docs/pgisman/2.2.0/RT\\_ST\\_Intersects.html](http://www.finds.jp/docs/pgisman/2.2.0/RT_ST_Intersects.html)

---

▼ SQL 6-3

---

-- 6-2 と似ていますが、ある点が、ラスタの中で縦横それぞれ何番目のピクセルに当たるかを返します  
-- 関数 ST\_WorldToRasterCoord を使います

```
SELECT ST_WorldToRasterCoord(rast, 139.939, 35.902) FROM raster43 ;  
SELECT (ST_WorldToRasterCoord(rast, 139.939, 35.902)).* FROM raster43 ;
```

|          | <b>st_worldtorastercoord<br/>record</b> |          | <b>columnx<br/>integer</b> | <b>rowy<br/>integer</b> |
|----------|-----------------------------------------|----------|----------------------------|-------------------------|
| <b>1</b> | (121, 349)                              | <b>1</b> | 121                        | 349                     |

↑ 左が一行目（複合型をそのまま表示）、右が複合型を列に展開したものです。

- ・注意点として、ある点がラスタ内に位置するか否かにかかわらず、ピクセル位置が計算されて返ります。
- ・例えばラスタの左側にある点の場合、columnx はマイナスになります。詳細は下記を参照して下さい。

マニュアル：ST\_WorldToRasterCoord 関数

- [http://www.finds.jp/docs/pgisman/2.2.0/RT\\_ST\\_WorldToRasterCoord.html](http://www.finds.jp/docs/pgisman/2.2.0/RT_ST_WorldToRasterCoord.html)

---

▼ SQL 6-4

---

-- ラスタの中心点と四隅の経緯度を取得します。基本は ST\_Envelope 関数で外郭をポリゴンにすることです  
-- ポリゴン化した後のクエリは、いろいろな書き方が可能です。下記は一例です

```
WITH a AS (  
    SELECT ST_Envelope(rast) geom FROM raster43  
) , b AS (  
    SELECT *, ST_Centroid(geom) cent FROM a  
)  
SELECT text 'center' AS lab, ST_X(cent) AS x, ST_Y(cent) AS y FROM b  
UNION ALL SELECT 'upperleft', ST_XMin(geom), ST_YMax(geom) FROM b  
UNION ALL SELECT 'upperright', ST_XMax(geom), ST_YMax(geom) FROM b  
UNION ALL SELECT 'lowerright', ST_XMax(geom), ST_YMin(geom) FROM b  
UNION ALL SELECT 'lowerleft', ST_XMin(geom), ST_YMin(geom) FROM b ;
```

|          | <b>lab<br/>text</b> | <b>x<br/>double precision</b> | <b>y<br/>double precision</b> |
|----------|---------------------|-------------------------------|-------------------------------|
| <b>1</b> | center              | 139.939727783203              | 35.9023998673262              |
| <b>2</b> | upper left          | 139.938354492188              | 35.9035122565034              |
| <b>3</b> | upperright          | 139.941101074219              | 35.9035122565034              |
| <b>4</b> | lowerright          | 139.941101074219              | 35.901287478149               |
| <b>5</b> | lower left          | 139.938354492188              | 35.901287478149               |

- ・ラスタに歪み (skewx , skewy) を設定していると、ST\_Envelope はラスタの外郭ではなくなります。
- ・その場合は ST\_Polygon 関数でラスタの外郭を得られますが、ST\_Envelope と違って、各ピクセルに値があるか否かが関係してきます。詳細は下記を参照して下さい。

マニュアル：ST\_Envelope 関数，ST\_Polygon 関数

- [http://www.finds.jp/docs/pgisman/2.2.0/RT\\_ST\\_Envelope.html](http://www.finds.jp/docs/pgisman/2.2.0/RT_ST_Envelope.html)
- [http://www.finds.jp/docs/pgisman/2.2.0/RT\\_ST\\_Polygon.html](http://www.finds.jp/docs/pgisman/2.2.0/RT_ST_Polygon.html)

---

▼ SQL 6-5

---

```
-- 経緯度を指定して、その点を含むラスタ内のピクセルでの値を得ます。関数 ST_Value を使います
-- 今回は OpenStreetMap タイルの RGB それぞれのバンドの値になります
-- 仮に標高メッシュ値をラスタ化していれば、経緯度から標高を得られます
WITH a AS (
    SELECT ST_SetSrid(ST_Point(139.939, 35.902), 4326) geom
)
SELECT band, ST_Value(rast, band, geom) FROM a, raster43, generate_series(1, 3) AS band;
```

|          | <b>band<br/>integer</b> | <b>st_value<br/>double precision</b> |
|----------|-------------------------|--------------------------------------|
| <b>1</b> | 1                       | 246                                  |
| <b>2</b> | 2                       | 238                                  |
| <b>3</b> | 3                       | 182                                  |

```
-- ST_Intersection 関数を使って下のようにもできますが、交差部分のジオメトリを作るため非常に遅いです
SELECT band, (ST_Intersection(ST_SetSrid(ST_Point(139.939, 35.902), 4326), rast, band)).val
FROM raster43, generate_series(1, 3) AS band ;
```

マニュアル : ST\_Value 関数 , ST\_Intersection 関数

- [http://www.finds.jp/docs/pgisman/2.2.0/RT\\_ST\\_Value.html](http://www.finds.jp/docs/pgisman/2.2.0/RT_ST_Value.html)
- [http://www.finds.jp/docs/pgisman/2.2.0/RT\\_ST\\_Intersection.html](http://www.finds.jp/docs/pgisman/2.2.0/RT_ST_Intersection.html)

---

▼ SQL 6-6

---

```
-- 任意の線と、ラスタの各ピクセルとの交差部分の値を得ます。関数 ST_Intersection を使います
-- 交差部分のジオメトリを作るため非常に遅いです (Core i5-4300M のノートパソコンで 8 秒弱)
-- また、ピクセル値が同じ部分はグループ化されてしまうため、線に沿った値にするには、別途工夫が必要です
WITH a AS (
    SELECT ST_SetSrid(
        ST_MakeLine(ST_Point(139.939, 35.902), ST_Point(140, 36)), 4326) AS geom
), b AS (
    SELECT (ST_Intersection(rast, 1, geom)).* FROM a, raster43
)
SELECT ST_AsText(geom), val FROM b ;
```

|           | <b>st_astext<br/>text</b>                                  | <b>val<br/>double precision</b> |
|-----------|------------------------------------------------------------|---------------------------------|
| <b>1</b>  | LINestring(139.939787134103 35.9032645761006, 139.93978983 | 181                             |
| <b>2</b>  | LINestring(139.939786791801 35.9032640261728, 139.93978713 | 196                             |
| <b>3</b>  | LINestring(139.939751972886 35.9032080875877, 139.93975460 | 237                             |
| <b>4</b>  | LINestring(139.939749268177 35.9032037423175, 139.93975197 | 193                             |
| <b>5</b>  | LINestring(139.939749240875 35.9032036984553, 139.93974926 | 225                             |
| <b>6</b>  | LINestring(139.93971410696 35.9031472538046, 139.93971681  | 221                             |
| <b>7</b>  | LINestring(139.939711402251 35.9031429085344, 139.93971168 | 237                             |
| <b>8</b>  | LINestring(139.939711689949 35.9031433707378, 139.93971410 | 197                             |
| <b>9</b>  | LINestring(139.939695173997 35.903116836913, 139.939695596 | 225                             |
| <b>10</b> | LINestring(139.939692469288 35.9031124916428, 139.93969517 | 192                             |
| <b>11</b> | LINestring(139.939690232277 35.9031088977564, 139.93969246 | 237                             |
| <b>12</b> | LINestring(139.939678945743 35.9030907652917, 139.93967950 | 209                             |

## ▼ SQL 6-7

-- 任意のポリゴンでラスタをクリップし（切り取り）、画像埋め込み HTML に出力する例  
-- ある点からバッファを作り、その円と、円を囲むバウンディングボックス（矩形）の両方でクリップします

```
COPY (  
    WITH a AS (  
        SELECT ST_SetSrid(ST_Point(139.939, 35.902), 4326) :: geography AS geog  
    ), b AS (  
        SELECT ST_Buffer(geog, 100) :: geometry AS geom -- 半径 100m のバッファを作成  
        FROM a  
    ), c AS (  
        SELECT ST_Clip(rast, geom) AS rast FROM b, raster43  
        UNION ALL SELECT ST_Clip(rast, ST_Envelope(geom)) FROM b, raster43  
    )  
    SELECT '<br />  
    FROM c  
) TO 'C:/Program Files/PostgreSQL/9.3/data/part67_output.htm' ;
```

↓ 出力結果。クリップするジオメトリの一部が、ラスタの範囲をはみ出していました。



## 7. R による PostGIS ラスタ読み込みと描画

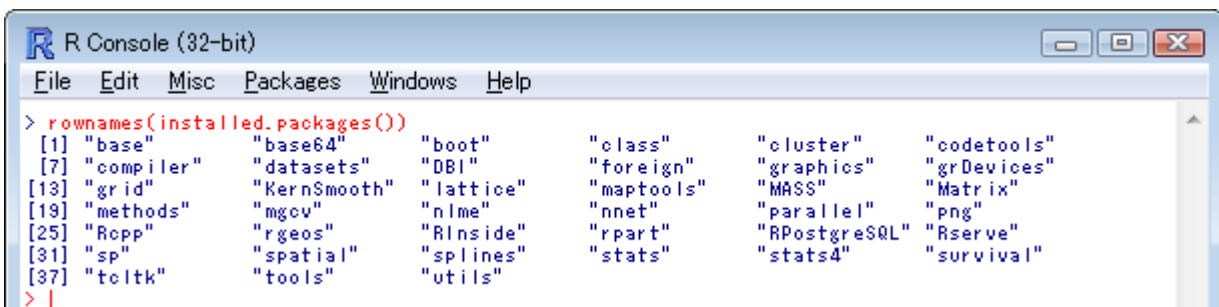
- ・ここまでラスタの出力に「画像埋め込み HTML」を使いましたが、ジオメトリとの重ね合わせなど地理情報として本格的に可視化するには厳しいです。(SVG を使うなど、何らかの方法は考えられますが)
- ・そこで本章では、統計処理言語 R を使い、ラスタとジオメトリを重ね合わせて描画する方法を紹介します。
- ・手順は、必要なパッケージインストール → ラスタ読み込みと描画 → ジオメトリ読み込み → 重ね合わせ、となります。すべての R のコードは part7.R にあります。

### ▼ WORK 7-1

- ・以下、R に付属のコンソールでの操作を示します。RStudio など別の環境でも、コマンドは同じです。
- ・今回の作業に必要なパッケージは、次の五つです。

|             |                                                        |
|-------------|--------------------------------------------------------|
| RPostgreSQL | … R から PostgreSQL に接続し、SQL を実行する                       |
| base64      | … R では PostgreSQL の bytea 型データを受け取れないので、Base64 形式で受け取る |
| png         | … Base64 形式から復元した PNG ファイルを読み込む                        |
| rgeos       | … PostGIS のジオメトリを WKT 形式で受け取り、R でのジオメトリに変換             |
| maptools    | … ジオメトリの処理やプロットに使う                                     |

- ・まず `rownames(installed.packages())` を打って、インストール済みパッケージを確認します。



```
R Console (32-bit)
File Edit Misc Packages Windows Help
> rownames(installed.packages())
 [1] "base"          "base64"      "boot"        "class"       "cluster"     "codetools"
 [7] "compiler"     "datasets"    "DBI"         "foreign"     "graphics"    "grDevices"
[13] "grid"         "KernSmooth" "lattice"     "maptools"    "MASS"        "Matrix"
[19] "methods"     "mgcv"        "nlme"        "nnet"        "parallel"    "png"
[25] "Rcpp"         "rgeos"       "Rinside"     "rpart"       "RPostgreSQL" "Rserve"
[31] "sp"           "spatial"     "splines"     "stats"       "stats4"      "survival"
[37] "tcltk"        "tools"       "utils"
> |
```

- ・未インストールのパッケージがあれば、次のインストールコマンドを入力します。パッケージ名をクォーテーションで囲みます (二重引用符でも可)。大・小文字を間違えるとパッケージが見つからないので注意。

```
install.packages(
  c('RPostgreSQL', 'base64', 'png', 'rgeos', 'maptools'))
```

### ▼ WORK 7-2

- ・RPostgreSQL パッケージを読み込み、PostgreSQL のデータベースに接続できるか確認します。以下のコードでは左辺への代入に `=` を使いますが (他のプログラミング言語と同様に)、R 独特の `<-` を使ってもいいです。

```
library(RPostgreSQL) # 先ほどのインストール時と違い、ロード時はクォーテーション無しです
con = dbConnect(PostgreSQL(),
  host='localhost', port=5432, dbname='handson2014', user="postgres",
  password='*****') # データベースへの接続を確立。パラメータは適宜環境に合わせて下さい
dbGetQuery(con, 'SELECT postgis_full_version()') # PostGIS のバージョン情報を得るテスト
```

- ・結果の例は次頁です。作業が終わったら `dbDisconnect(con)` を打って接続を閉じます。

```

R Console (32-bit)
File Edit Misc Packages Windows Help
> library(RPostgreSQL)# 先ほどのインストール時と違い、ロード時はクォーテーション無しです
Loading required package: DBI
> con = dbConnect(PostgreSQL(),
+ host='localhost', port=5432, dbname='handson2014', user="postgres", password='
> dbGetQuery(con, 'SELECT postgis_full_version()')# PostGISのバージョン情報を得るテスト
1 POSTGIS="2.1.3 r12547" GEOS="3.4.2-CAPI-1.8.2 r3924" PROJ="Rel. 4.8.0, 6 March 2012" GDAL="$
> |

```

▼ WORK 7-3

- ・現在のところ PostGIS ラスタを直接 R で読み込む方法はなく、また PostGIS ラスタから変換してデータベース内に作成した PNG などの画像ファイルも、直接 R で読み込めません。
- ・そこで手間がかかりますが、Base64 形式を経由して R で読み込みます。
- ・以下のコードでは、Base64 の一時ファイルパスを設定し、そこへラスタから変換した PNG を出力する SQL を実行して R へ読み込む処理をしています。
- ・使うラスタは、4. (3) で作成し位置情報を入力したもの (23 ~ 25 頁) です。ここではテーブル名 raster43、ラスタ列名 rast と仮定します。
- ・SQL 実行の際、先ほどは dbGetQuery を使いましたが、今回は値を受け取らないので dbSendQuery を使います。

```

library(base64)           # パッケージ読み込み
library(png)
b64 = 'C:/Program Files/PostgreSQL/9.3/data/part73_tmp.base64' # 一時ファイル、フルパスで
png = 'tmp.png'          # PNG の一時ファイル、こちらは相対パスでも可
sql = paste(sep='', "COPY (
      SELECT replace(encode(ST_AsPNG(rast), 'base64'), E'\n', '') FROM raster43
      ) TO '", b64, """) # SQL を、一時ファイルパスに合わせて動的に生成
dbSendQuery(con, sql)    # SQL 実行、これで Base64 ファイルが生成されたはず
decode(b64, png)         # Base64 → PNG に復元
img = readPNG(png)       # 復元した PNG を読み込む
class(img)               # 結果確認
str(img)                 # "

```

↓ PNG を読み込んだ変数が array クラスになり、str コマンドで縦横のピクセル数を確認できれば成功です。

```

R Console (32-bit)
File Edit Misc Packages Windows Help
> library(base64)# パッケージ読み込み
> library(png)
> b64 = 'D:/PostgreSQL/9.3/data/part73_tmp.base64'# 一時ファイル、フルパスで
> png = 'tmp.png'# PNGの一時ファイル、こちらは相対パスでも可
> sql = paste(sep='', "COPY (
+ SELECT replace(encode(ST_AsPNG(rast), 'base64'), E'\n', '') FROM raster43
+ ) TO '", b64, """)# SQLを、一時ファイルパスに合わせて動的に生成
> dbSendQuery(con, sql)# SQL実行、これでBase64ファイルが生成されたはず
<PostgreSQLResult: (5132, 0, 1)>
> decode(b64, png)# Base64 → PNG に復元
> img = readPNG(png)# 復元したPNGを読み込む
> class(img)# 結果確認
[1] "array"
> str(img)# "
num [1:512, 1:512, 1:4] 0.941 0.941 0.941 0.941 0.698 ...
> |

```

- ・PNG での受け取りが完了したら一時ファイルは削除して構いません。コマンド file.remove を使います。

```
file.remove(c(b64, png))
```

```
R Console (32-bit)
File Edit Misc Packages Windows Help
> file.remove(c(b64, png))
[1] TRUE TRUE
> |
```

#### ▼ WORK 7-4

- ・前項でラスタ自体を読み込んだのに続き、プロットするための位置情報を PostgreSQL から取得します。
- ・必要なのはラスタの四隅の座標です。27 頁の SQL 6-4 のように座標値を取得してもいいですが、ジオメトリ取り込みのテストとしてラスタの外郭（ポリゴン）を WKT 形式で取得し、そこから四隅座標を得ます。
- ・ここで rgeos と maptools パッケージを使います。

```
library(rgeos)      # パッケージ読み込み
library(maptools)

sql = 'SELECT ST_AsText(ST_Envelope(rast)) FROM raster43'
wkt = dbGetQuery(con, sql) # SQL 実行、これでラスタ外郭を WKT 形式で R に取り込めたはず
shp = readWKT(wkt) # WKT を R でのジオメトリ（ここではポリゴン）に変換
bbx = bbox(shp) # ポリゴンから四隅座標を取得
print(bbx) # 結果確認
```

↓ 最後にラスタの四隅座標が表示されれば成功です。

```
R Console (32-bit)
File Edit Misc Packages Windows Help
> library(rgeos)# パッケージ読み込み
rgeos version: 0.3-8, (SVN revision 460)
GEOS runtime version: 3.4.2-CAPI-1.8.2 r3921
Polygon checking: TRUE

> library(maptools)
Loading required package: sp
Checking rgeos availability: TRUE
> sql = 'SELECT ST_AsText(ST_Envelope(rast)) FROM raster43'
> wkt = dbGetQuery(con, sql)# SQL実行、これでラスタ外郭をWKT形式でRに取り込めたはず
> shp = readWKT(wkt)# WKTをRでのジオメトリ（ここではポリゴン）に変換
> bbx = bbox(shp)# ポリゴンから四隅座標を取得
> print(bbx)# 結果確認
      min      max
x 139.93835 139.94110
y  35.90129  35.90351
> |
```

- ・ここでは PostGIS の ST\_AsText 関数でジオメトリを WKT 形式のテキストに変換し、普通の文字列として R に読み込んでから rgeos パッケージの readWKT 関数でジオメトリに変換しました。
- ・PostGIS のジオメトリを R で直接読み込む方法としては rgdal パッケージの利用がありますが、動作環境に制約があるのと、テーブルとビュー以外は読み込めないため、今回は使いません。
- ・WKT 形式の詳細は、下記を参照して下さい。

マニュアル : OpenGIS WKB と WKT

- [http://www.finds.jp/docs/pgisman/2.2.0/using\\_postgis\\_dbmanagement.html#OpenGISWKBWKT](http://www.finds.jp/docs/pgisman/2.2.0/using_postgis_dbmanagement.html#OpenGISWKBWKT)

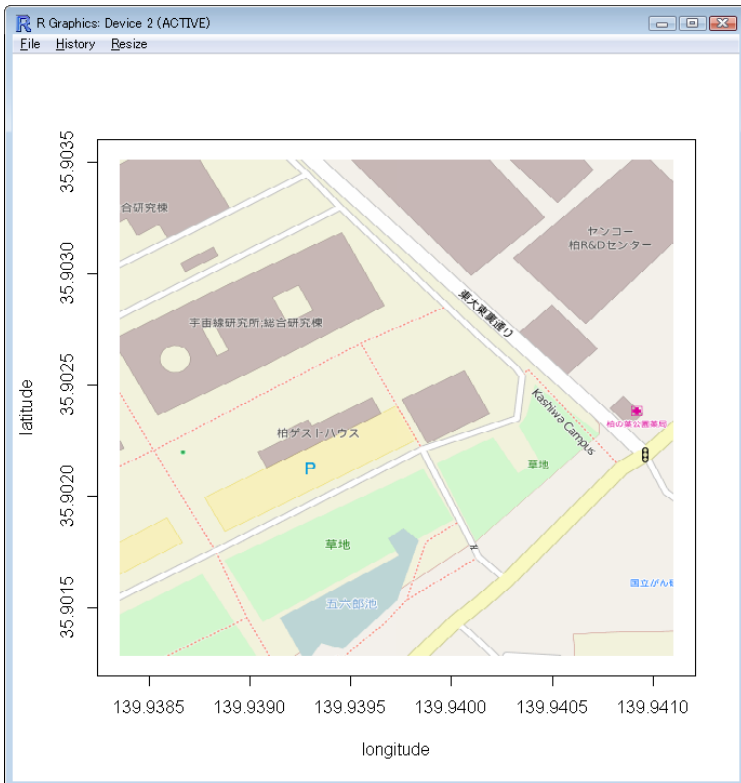


▼ WORK 7-5

- ・ここでラスタをプロットします。手順は、まず前項で取得したラスタの四隅座標からプロット用の枠を作り、次にラスタを貼り付けます。
- ・前項までで、必要なパッケージは全て読み込み済みです。

```
plot(type='n', x=bbx[1,], y=bbx[2,], xlab='longitude', ylab='latitude') # 枠だけプロット  
rasterImage(img, xleft=bbx[1,1], ybottom=bbx[2,1],  
            xright=bbx[1,2], ytop=bbx[2,2], interpolate=F) # interpolate=T は補間あり、少しぼやける
```

↓ こんな風に軸付きで描画されれば成功です。



↓ Rのグラフィックウィンドウのサイズを変えると、プロット領域も追尾して描画されます。

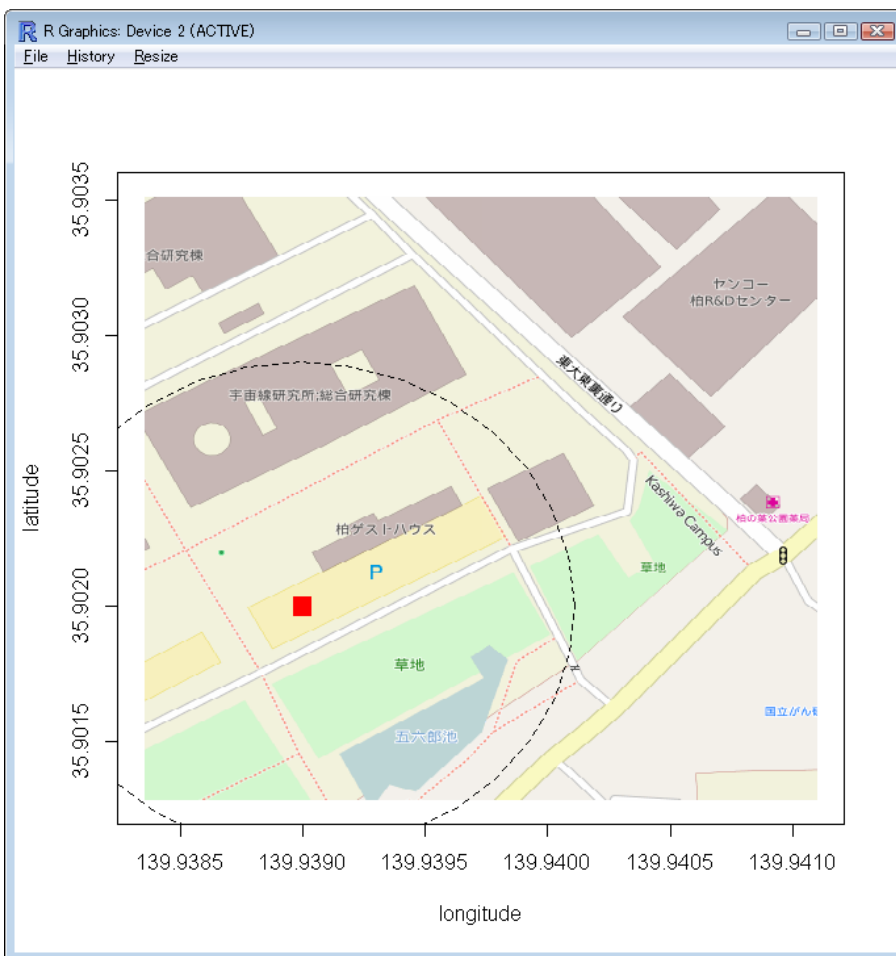


## ▼ WORK 7-6

- ・前項でプロットした軸は経緯度になっているので、同じ測地系のジオメトリはそのまま重ねられます。
- ・本来は、上下・左右方向が同じスケールになるよう plot 関数の asp オプションで縦軸と横軸の比を調整しますが、今回は時間の都合で省略します。
- ・重ねるジオメトリの例として、SQL 6-7 (29 頁) で作成したバッファとその中心点を試します。

```
sql = 'WITH a AS (  
      SELECT ST_SetSrid(ST_Point(139.939, 35.902), 4326) AS geom  
    )  
      SELECT ST_AsText(geom), ST_AsText(ST_Buffer(geom :: geography, 100) :: geometry)  
    FROM a '  
wkt = dbGetQuery(con, sql) # SQL 実行  
poi = readWKT(wkt[1,1])    # 一列目 (点) をジオメトリに変換  
buf = readWKT(wkt[1,2])    # 二列目 (バッファ) をジオメトリに変換  
plot(add=T, x=poi, cex=2, pch=15, col='red') # 重ね合わせは add=T を指定するだけ  
plot(add=T, x=buf, lty=2)
```

↓ こんな風になります。点線が、29 頁にバッファでクリップしたのと同じ場所になっています。



- ・上の場合、点座標を PostGIS から WKT 形式で取得しています。しかし R の側に点座標があれば、points 関数で直接、プロット枠に重ねられます。線データも同様です。
- ・このように背景地図を PostGIS ラスタで管理すれば、必要な時すぐに R のプロットに使える便利です。

---

## ▼ WORK 7-7

---

- ・最後に、29 頁で作った「クリップ済み」の円形のラスタを R でプロットしてみます。
- ・まとめとして、本章の最初から（パッケージインストールを除く）の手順を含めます。違うのは、読み込むラスタをテーブルにあるものでなく SQL で動的にクリップしたラスタにすること、そのクリップされた範囲を空プロット枠の四隅にすること、グラフィックウィンドウ全体の背景色を設定してクリップ範囲を明確にすること、です。

```
library(RPostgreSQL) # パッケージ読み込み
library(base64)
library(png)
library(rgeos)
library(maptools)
con = dbConnect(PostgreSQL(),
  host='localhost', port=5432, dbname='handson2014', user="postgres",
  password='*****') # データベースへの接続を確立。パラメータは適宜環境に合わせて下さい
b64 = 'C:/Program Files/PostgreSQL/9.3/data/part73_tmp.base64' # 一時ファイル、フルパスで
png = 'tmp.png' # PNG の一時ファイル、こちらは相対パスでも可
sql = paste(sep=' ', "COPY (
  WITH a AS (
    SELECT ST_SetSrid(ST_Point(139.939, 35.902), 4326) AS geom
  ), b AS (
    SELECT ST_Buffer(geom :: geography, 100) :: geometry AS geom
    FROM a
  )
  SELECT replace(encode(ST_AsPNG(ST_Clip(rast, geom)), 'base64'), E'\n', '')
  FROM b, raster43
) TO '", b64, "'") # SQL を、一時ファイルパスに合わせて動的に生成
dbSendQuery(con, sql) # SQL 実行、これで Base64 ファイルが生成されたはず
decode(b64, png) # Base64 → PNG に復元
img = readPNG(png) # 復元した PNG を読み込む
file.remove(c(b64, png)) # 一時ファイル削除
sql = 'WITH a AS (
  SELECT ST_SetSrid(ST_Point(139.939, 35.902), 4326) AS geom
)
SELECT ST_AsText(geom),
  ST_AsText(ST_Intersection(ST_Buffer(geom :: geography, 100) :: geometry,
    ST_Envelope(rast)))
FROM a, raster43'
wkt = dbGetQuery(con, sql) # SQL 実行
poi = readWKT(wkt[1,1]) # 一列目 (点) をジオメトリに変換
buf = readWKT(wkt[1,2]) # 二列目 (バッファ) をジオメトリに変換
bbx = bbox(buf) # ポリゴンから四隅座標を取得
par(bg='lightblue')
plot(type='n', x=bbx[1,], y=bbx[2,], xlab='longitude', ylab='latitude') # 枠だけプロット
rasterImage(img, xleft=bbx[1,1], ybottom=bbx[2,1],
  xright=bbx[1,2], ytop=bbx[2,2], interpolate=F) # 背景地図としてラスタ描画
plot(add=T, x=poi, cex=2, pch=15, col='red') # 重ね合わせは add=T を指定するだけ
plot(add=T, x=buf, lty=2)
dbDisconnect(con) # 最後にデータベースから切断
```

- ・結果は次頁にあります。

↓ 前頁の R スクリプトの結果です。ここまでお疲れ様でした。まだ時間がある場合用に、追加作業の資料とデータを用意しています。

