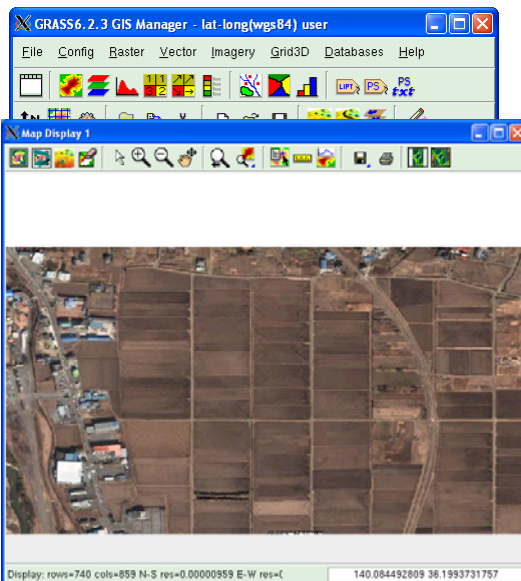


7 Extract Attribute Value at given POINT.

The way how to convert the information loaded as a raster layer to the attribute of a vector layer and save into a table of PostgreSQL is described.

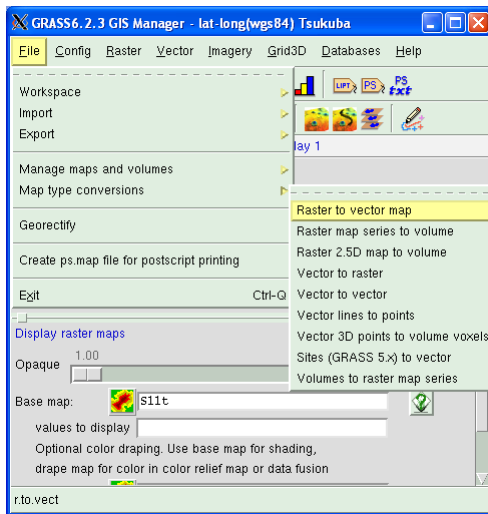
- 7.1 Transform Raster layer to Vector layer (POLYGON) using GRASS.
- 7.2 Export the transformed Vector layer (POLYGON) to PostgreSQL and POINT data to which the attributes values will be allocated.
- 7.3 Add column to POINT data to store the attributes values.
- 7.4 Store the attributes values into the added column using PostGIS function.

- 7.1 Transform Raster layer to Vector layer (POLYGON) using GRASS.
(It is necessary once to transform the raster layer to a vector layer (POLYGON)).



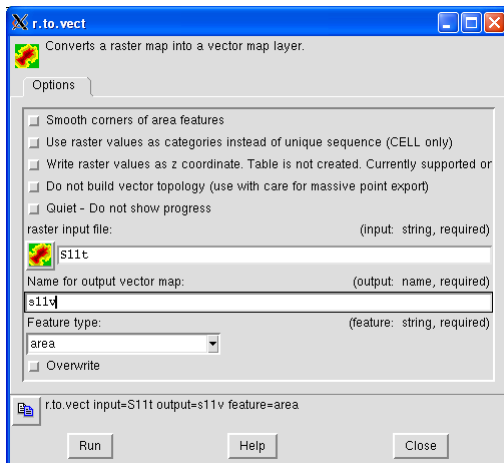
Open GRASS with
LOCATION=latlon(wgs84),
MAPSET=Tsukuba.

Display the raster "s11t".



Convert the raster map to a vector map.

“File”, “Map type conversion’ and “Raster to Vector map”.



Input Raster map:

S11t (example)

Name of Output Vector map:

s11v (arbitrary)

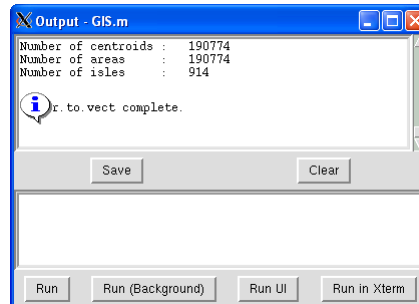
Feature type

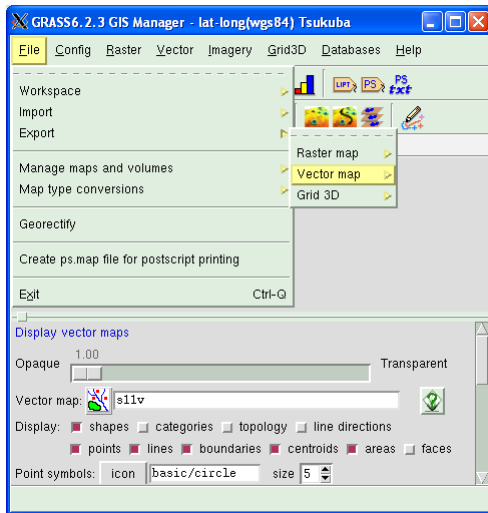
area

(Select area because POLYGON layer is needed for 7.1)

Click “Run”.

As every grids of the input raster are converted to squares (POLYGON) this takes much time. Be patient until the message “r.to.vect complete” in “output-GIS.m”.

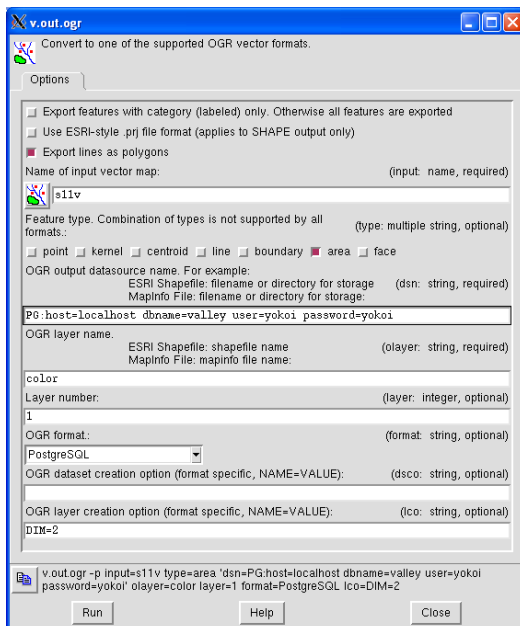




Export the converted vector map to PostgreSQL.

“File”, “Export” and “Vector map”.

7.2 Export the transformed Vector layer (POLYGON) to PostgreSQL and POINT data to which attributes values will be allocated.



Check ‘export lines as polygons’.

Name of input vector map

s11v

Select feature type

area

OGR output data source

PG:host=localhost
dbname=valley user=yokoi
password=yokoi

OGR layer name

color (arbitrary)

Layer Number

1

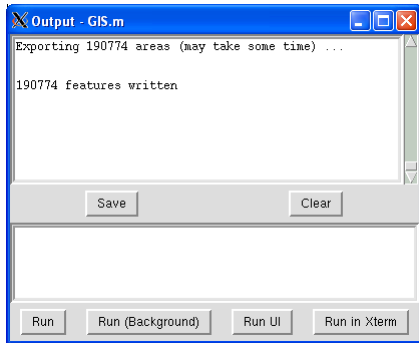
Select OGR format

PostgreSQL

OGR layer creation option

DIM=2

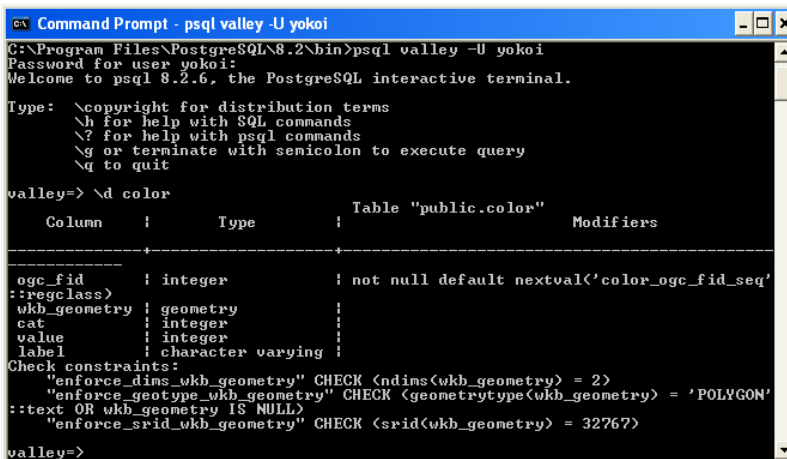
This takes much time.



Be patient until the message “...features written” is shown.

Exit from GRASS.

Check the exported data using PostgreSQL.



Open “Command Prompt” of PostgreSQL.
 psql valley -U yokoi
 Then, type passwd.
 Check the table “color”
 \d color

Geometry is stored in
 “wkb_geometry” column. The
 converted data are stored in “value”
 column. SRID is correctly set 32767.

```

Command Prompt - psql valley -U yokoi
valley=> select cat,value from color order by cat limit 10;
cat | value
-----+-----
 1 | 8526
 2 | 9550
 3 | 9549
 4 | 8493
 5 | 7435
 6 | 9516
 7 | 10606
 8 | 8459
 9 | 9549
10 | 8492
(10 rows)
valley=>

```

Browse first ten rows of the table "color" to check the contents.
 select cat,value from color order by cat limit 10;

```

Command Prompt - psql valley -U yokoi
valley=> alter table only color add constraint color_pkey primary key(ogc_fid);
NOTICE: ALTER TABLE / ADD PRIMARY KEY will create implicit index "color_pkey" f
or table "color"
ALTER TABLE
valley=>

```

The table "color" does not have the primary key setting that is necessary to be loaded on QGIS.

```
alter table only color add constraint color_pkey primary key(ogc_fid);
```

where "alter table" is a PostgreSQL command, "only color" denotes that this command is applied only to the table "color", "add constraint color_pkey primary key(ogc_fid)" means that the column "ogc_fid" is selected for the primary key of this table and named "color_pkey".

```

Command Prompt - psql valley -U yokoi
valley=> \d color
          Column          |          Type          |          Table "public.color"          |          Modifiers
-----+-----+-----+-----
 ogc_fid                  | integer                | not null default nextval('color_ogc_fid_seq' |
::regclass)
 wkb_geometry             | geometry                | |
 cat                      | integer                | |
 value                   | integer                | |
 label                   | character varying     | |
Indexes:
 "color_pkey" PRIMARY KEY, btree (ogc_fid)
Check constraints:
 "enforce_dims_wkb_geometry" CHECK (ndims(wkb_geometry) = 2)
 "enforce_geotype_wkb_geometry" CHECK (geometrytype(wkb_geometry) = 'POLYGON'
::text OR wkb_geometry IS NULL)
 "enforce_srid_wkb_geometry" CHECK (srid(wkb_geometry) = 32767)
valley=>

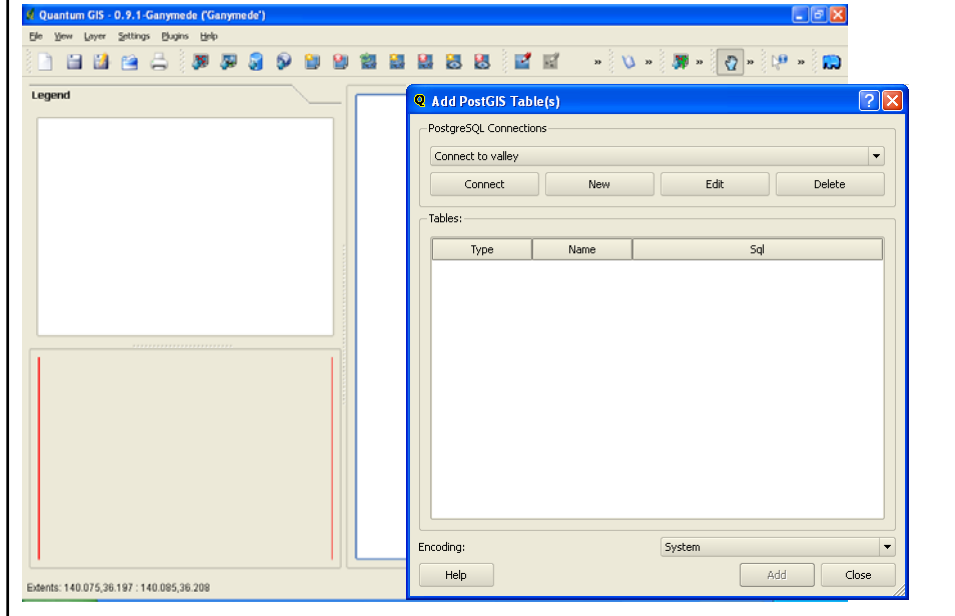
```

Check the table "color" using the command
 \d

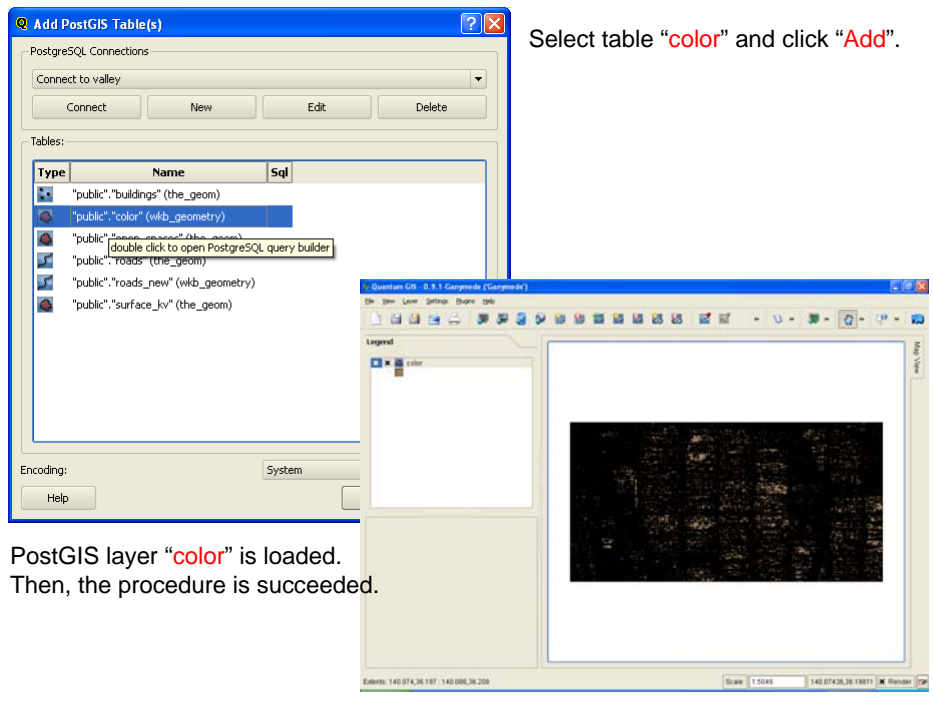
Notice to
 Indexes;
 "Color_pkey" PRIMARY KEY, btree (ogc_fid)

The primary key was set.

Open QGIS.
Click “Add PostGIS layer” button. Then connect to “valley” database.



Select table “color” and click “Add”.



PostGIS layer “color” is loaded.
Then, the procedure is succeeded.

Make a POINT data layer (if you have target POINT data layer already, this procedure can be skipped)

Add new table "obspoints" in the database "valley". This procedure has been described in "2_creating_vector_layer.ppt".

Copy the sql batch file "D:/batch_sql/mkpoint.sql" to "C:/TEMP/mkvalley.sql".

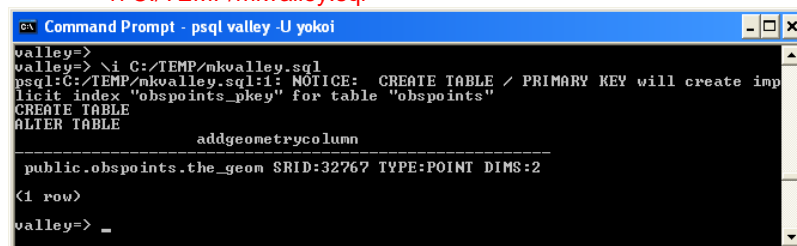
Edit "C:/TEMP/mkvalley.sql" using WordPad as shown below.

```
CREATE TABLE obspoints (id1 integer NOT NULL, CONSTRAINT
obspoints_pkey PRIMARY KEY (id1)) WITHOUT OIDS;
ALTER TABLE obspoints OWNER TO yokoi;
select AddGeometryColumn('obspoints', 'the_geom', 32767, 'POINT', 2);
```

where the changed parts are shown blue, SRID "32767" is set as same as that of the table "color".

Execute the sql batch file "C:/TEMP/mkvalley.sql" using "\i" command.

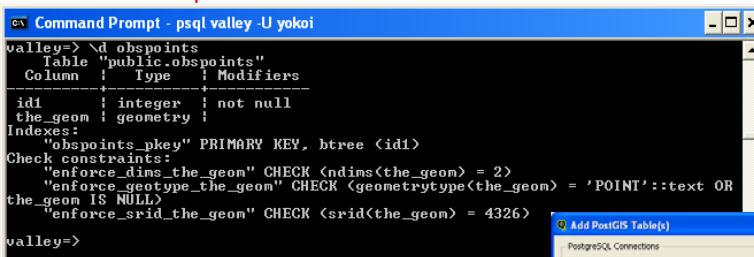
```
\i C:/TEMP/mkvalley.sql
```



```
Command Prompt - psql valley -U yokoi
valley=>
valley=> \i C:/TEMP/mkvalley.sql
psql:C:/TEMP/mkvalley.sql:1: NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "obspoints_pkey" for table "obspoints"
CREATE TABLE
ALTER TABLE
      addgeometrycolumn
-----
public.obspoints.the_geom SRID:32767 TYPE:POINT DIMS:2
<1 row>
valley=> _
```

Check the newly created table "obspoints".

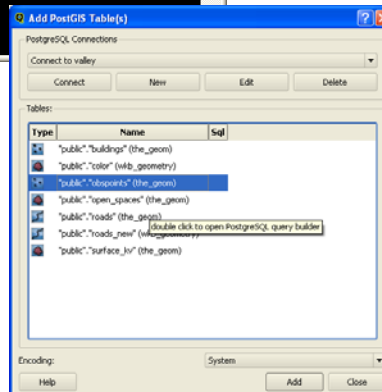
```
\d obspoints
```



```
Command Prompt - psql valley -U yokoi
valley=> \d obspoints
Table "public.obspoints"
Column | Type          | Modifiers
-----+-----+-----
id1    | integer      | not null
the_geom | geometry     |
Indexes:
    "obspoints_pkey" PRIMARY KEY, btree (id1)
Check constraints:
    "enforce_dims_the_geom" CHECK (ndims(the_geom) = 2)
    "enforce_geotype_the_geom" CHECK (geometrytype(the_geom) = 'POINT'::text OR the_geom IS NULL)
    "enforce_srid_the_geom" CHECK (srid(the_geom) = 4326)
valley=>
```

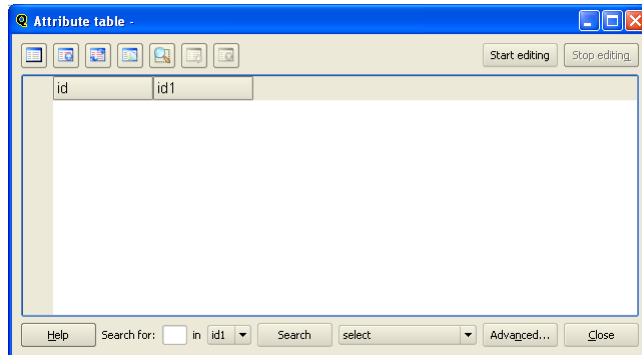
Back to QGIS.

Connect to the table "obspoints".

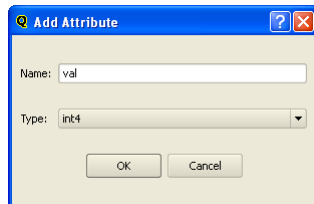


7.3 Add column to POINT data to store the attributes values.

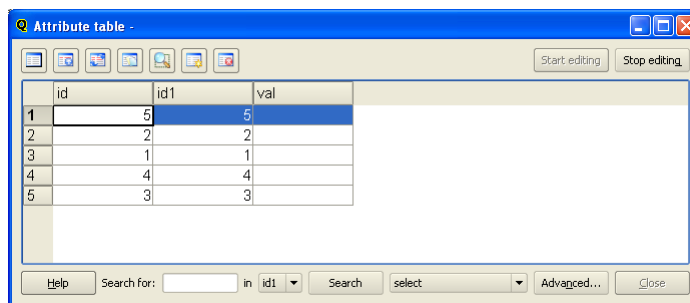
Open "Attribute Table" by using "Open Table" button.



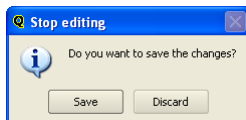
Click on "Start Editing" button and "New Column" button.



Set Name="vale", Type="Int4".

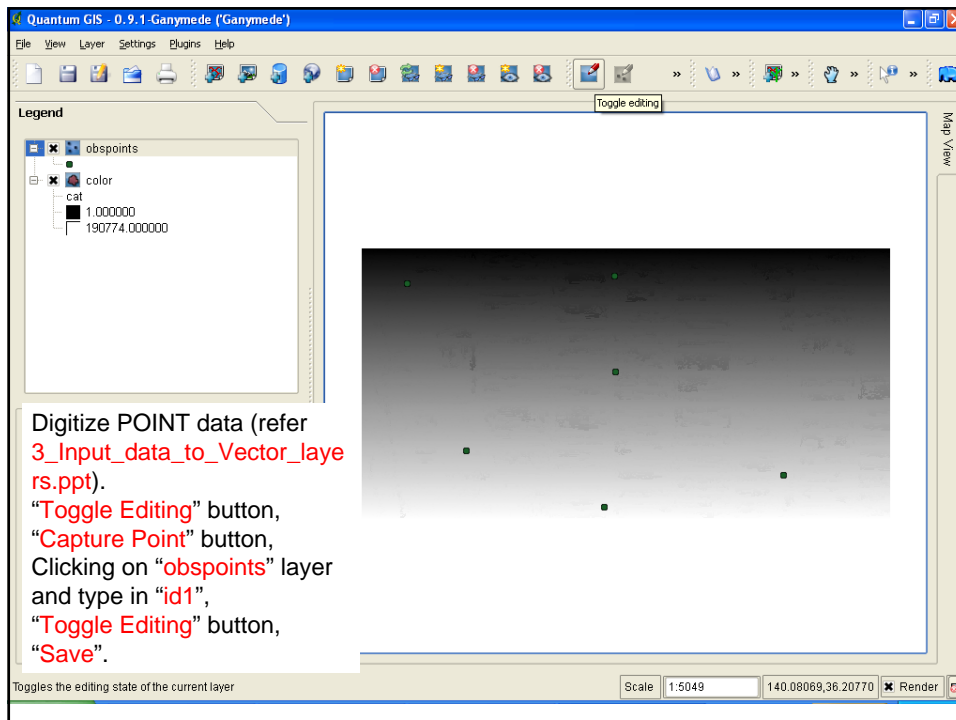


New column "value" is added. Click on "Stop Editing" button.



Click on "Save" button.

Then, click on "close" of "Attribute Table".



7.4 Store the attributes values into the added column using PostGIS function.

In "Command Prompt" of PostgreSQL.

Store data of "value" of the table "color" into the column "value" of the table "obspoints" at the locations of "obspoints".

`update obspoints set val=value from color where color.wkb_geometry && obspoints.the_geom;`

```

C:\> Command Prompt - psql valley -U yokoi
valley=>
valley=> update obspoints set val=value from color where color.wkb_geometry && obspoints.the_geom;
UPDATE 6
valley=>
  
```

This means:

Update the table "obspoints" setting the column "val" equal to the column "value" from the table "color" under the condition that is the value of "wkb_geometry" column of the table "color" coincides with that of "the_geom" column of the table "obspoints",

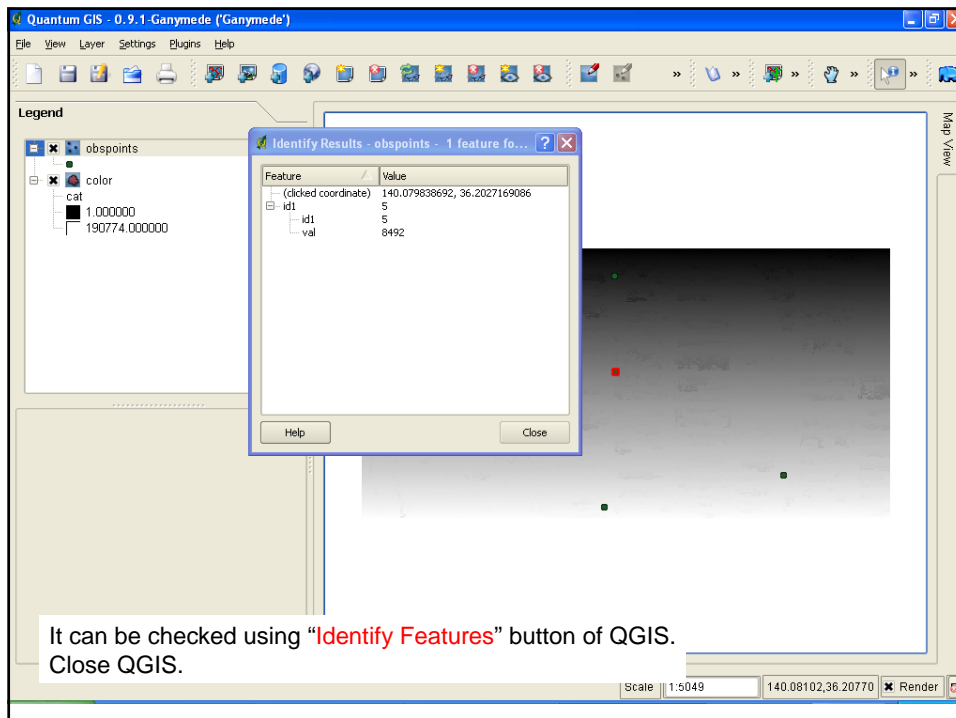
where "update" is a PostgreSQL command, && is the operator of PostGIS that reply TRUE where two geometries share the same point or area.

Check the added information.

`select id1, val from obspoints order by id1;`

```
Command Prompt - psql valley -U yokoi
valley=> select id1, val from obspoints order by id1;
 id1 | val
-----+-----
  1  | 9514
  2  | 13776
  3  | 8491
  4  | 8525
  5  | 8492
  6  | 12753
(6 rows)
valley=> _
```

Exit from the database “valley” using “\q”.
Close “Command Prompt” of PostgreSQL.



It can be checked using “Identify Features” button of QGIS.
Close QGIS.

