Assignment-7: Raster in DBMS, GEO-1006

Ming-Chieh Hu, Carmem E. F. Aires, Benthe Kock

January 2025

Project Description

This report was made by:

1. **Ming-Chieh Hu**, student number: 6186416
2. **Carmem E. F. Aires**, student number: 4325893
3. **Benthe Kock**, student number: 6104819

Contents

1 Check if raster support is available in the database

1a Test extension functionality

We cannot access raster functions directly.
According to the post, postGIS excluded the raster types and functions from the main extension as of version 3.x; a separate `CREATE EXTENSION postgis_raster` is then necessary to get raster support.

```sql
-- Use this query to make the extension available:
create extension postgis_raster;
```

```
psql -U username -d dbname -f ./test_raster.sql
```

```
[~/Documents/TUD/GEO-1006/Assignments/07]$ psql -d geo-1006 -f ./test_raster.sql
 postgis_raster_lib_build_date | postgis_raster_lib_version
-------------------------------+----------------------------
                               | 3.5.0 0
(1 row)

CREATE TABLE
psql:test_raster.sql:40: NOTICE:  SRID value -1 converted to the officially unknown SRID value 0
INSERT 0 2
 rid
-----
   1
   2
(2 rows)

 st_width | st_height | st_srid | st_numbands
----------+-----------+---------+-------------
       10 |        20 |       0 |           0
        5 |         5 |       0 |           3
(2 rows)

DROP TABLE
```

1b Raster functions

The 'test_raster.sql' script uses some raster functions, like ST_Width() and ST_NumBands() to test if the raster module works correctly. What type of raster functions are these, and what are the other, major, function categories for raster available in PostGIS?

1b.1 ST_Width() and ST_NumBands()

These 2 functions are both type of **Raster Accessors** according to *postgis.net*. They work the same way as "getter" in other programming language, that is, to get attributes (metadata) from a raster object. Below are their function descriptions:

- `ST_Width()`: Returns the width of the raster in pixels.
- `ST_NumBands()`: Returns the number of bands in the raster object.

1b.2 Other major function categories

The full list of the categories according to *postgis.net* are:

- Raster Management
- Raster Constructors
- Raster Accessors
- Raster Band Accessors
- Raster Pixel Accessors and Setters
- Raster Editors
- Raster Band Editors
- Raster Band Statistics and Analytics
- Raster Inputs

- Raster Outputs
- Raster Processing: Map Algebra
- Built-in Map Algebra Callback Functions
- Raster Processing: DEM (Elevation)
- Raster Processing: Raster to Geometry
- Raster Operators
- Raster and Raster Band Spatial Relationships

In my personal perspective, they can also be divided into 6 major categories:

- Data Setters (constructor)
- Data Getters (accessor)
- Data Conversion Functions (converts object types)
- Data Processing Functions (calculations and processing)
- Data Comparison Functions (Raster Operators)
- Data I/O Functions (save file as a specific format)

2 Load raster data in the database

2a Use gdalinfo to retrieve the metadata

Download the sample images '93000_436000.tif' and '93000_436500.tif' from Brightspace to your local disk. The images are in the GeoTIFF format (Tagged Image File Format) with in some of the tags geo-reference information included. What is the spatial metadata of these image? Use 'gdalinfo' to determine this.

93000_436000.tif

```
gdalinfo ./93000_436000.tif
```

```
[~/Documents/TUD/GEO-1006/Assignments/07]$ gdalinfo ./93000_436000.tif
Driver: GTiff/GeoTIFF
Files: ./93000_436000.tif
Size is 10000, 5000
Coordinate System is:
ENGCRS["unnamed",
    EDATUM["Unknown engineering datum"],
    CS[Cartesian,2],
        AXIS["(E)",east,
            ORDER[1],
            LENGTHUNIT["unknown",1]],
        AXIS["(N)",north,
            ORDER[2],
            LENGTHUNIT["unknown",1]]]
Data axis to CRS axis mapping: 1,2
Origin = (92999.999999999796273,436500.000000000582077)
Pixel Size = (0.100000000000000,-0.100000000000000)
Metadata:
  TIFFTAG_SOFTWARE=Inpho GmbH
  TIFFTAG_XRESOLUTION=1
  TIFFTAG_YRESOLUTION=1
  TIFFTAG_RESOLUTIONUNIT=1 (unitless)
  AREA_OR_POINT=Area
Image Structure Metadata:
  INTERLEAVE=PIXEL
Corner Coordinates:
Upper Left  (   93000.000,  436500.000)
Lower Left  (   93000.000,  436000.000)
Upper Right (   94000.000,  436500.000)
Lower Right (   94000.000,  436000.000)
Center      (   93500.000,  436250.000)
Band 1 Block=512x512 Type=Byte, ColorInterp=Red
Band 2 Block=512x512 Type=Byte, ColorInterp=Green
Band 3 Block=512x512 Type=Byte, ColorInterp=Blue
```

93000_436500.tif

```
gdalinfo ./93000_436500.tif
```

```
[~/Documents/TUD/GEO-1006/Assignments/07]$ gdalinfo ./93000_436500.tif
Driver: GTiff/GeoTIFF
Files: ./93000_436500.tif
Size is 10000, 5000
Coordinate System is:
ENGCRS["unnamed",
    EDATUM["Unknown engineering datum"],
    CS[Cartesian,2],
        AXIS["(E)",east,
            ORDER[1],
            LENGTHUNIT["unknown",1]],
        AXIS["(N)",north,
            ORDER[2],
            LENGTHUNIT["unknown",1]]]
Data axis to CRS axis mapping: 1,2
Origin = (92999.999999999796273,437000.000000000582077)
Pixel Size = (0.100000000000000,-0.100000000000000)
Metadata:
  TIFFTAG_SOFTWARE=Inpho GmbH
  TIFFTAG_XRESOLUTION=1
  TIFFTAG_YRESOLUTION=1
  TIFFTAG_RESOLUTIONUNIT=1 (unitless)
  AREA_OR_POINT=Area
Image Structure Metadata:
  INTERLEAVE=PIXEL
Corner Coordinates:
Upper Left  (   93000.000,  437000.000)
Lower Left  (   93000.000,  436500.000)
Upper Right (   94000.000,  437000.000)
Lower Right (   94000.000,  436500.000)
Center      (   93500.000,  436750.000)
Band 1 Block=512x512 Type=Byte, ColorInterp=Red
Band 2 Block=512x512 Type=Byte, ColorInterp=Green
Band 3 Block=512x512 Type=Byte, ColorInterp=Blue
```

According to the displayed information, we're able to get a grasp on these 2 images:

- The size for both the images are 10000 by 5000 (width, height).
- They both have 3 bands (RGB)
- If we tile both images, `93000_436500.tif` will be in the north of `93000_436000.tif`

2b Completeness of CRS information

Is the spatial reference information of the images complete? (the coordinates in the GeoTIFF files refer to the Dutch national grid, Spatial Reference ID = 28992)

CRS Information from gdalinfo

```
Coordinate System is:
ENGCRS["unnamed",
    EDATUM["Unknown engineering datum"],
    CS[Cartesian,2],
        AXIS["(E)",east,
            ORDER[1],
            LENGTHUNIT["unknown",1]],
        AXIS["(N)",north,
            ORDER[2],
            LENGTHUNIT["unknown",1]]]
Data axis to CRS axis mapping: 1,2
Origin = (92999.999999999796273,437000.000000000582077)
```

The CRS metedata for these 2 images are not very clear, specifically:

Coordinate System

- `ENGCRS["unnamed"]`: Indicates that the dataset uses an undefined CRS or non-georeferenced system.
- `EDATUM["Unknown engineering datum"]`: The datum is not defined.

- `CS[Cartesian,2]`: Specifies a 2D Cartesian coordinate system with two axes (East and North).

Axes

- `AXIS["(E)",east]`: First axis is labeled East.
- `AXIS["(N)",north]`: Second axis is labeled North.
- Both axes have `LENGTHUNIT["unknown",1]`, meaning the unit of measurement for the coordinates is undefined. It could be meters, feet, or any other unit.

Data Axis to CRS Axis Mapping

- `1,2`: Confirms that the dataset's data axis aligns directly with the CRS axis (East → 1, North → 2).

2c Load the images in the PostGIS database

Using `raster2pgsql` to load raster data involves a 2-step procedure. In the first step `raster2pgsql` analyzes the input images and processes the options specified. This results in a file with SQL commands which have to be executed in a second step to actually load the raster data in the database. For this to work properly you have to run the `raster2pgsql` command from the operating system "command line".

Step 1

```
raster2pgsql -C -s 28992 -t 500x500 -I -Y ./*.tif rotterdam > ./load2.sql
```

Step 2

```
psql -U username -d dbname -f ./load2.sql
```

```
[~/Documents/TUD/GEO-1006/Assignments/07]$ psql -U mchu -d geo-1006 -f ./load2.sql
BEGIN
CREATE TABLE
COPY 50
COPY 50
COPY 50
COPY 50
COPY 50
COPY 50
COPY 50
COPY 50
CREATE INDEX
ANALYZE
psql:load2.sql:421: NOTICE:  Adding SRID constraint
psql:load2.sql:421: NOTICE:  Adding scale-X constraint
psql:load2.sql:421: NOTICE:  Adding scale-Y constraint
psql:load2.sql:421: NOTICE:  Adding blocksize-X constraint
psql:load2.sql:421: NOTICE:  Adding blocksize-Y constraint
psql:load2.sql:421: NOTICE:  Adding alignment constraint
psql:load2.sql:421: NOTICE:  Adding number of bands constraint
psql:load2.sql:421: NOTICE:  Adding pixel type constraint
psql:load2.sql:421: NOTICE:  Adding nodata value constraint
psql:load2.sql:421: NOTICE:  Adding out-of-database constraint
psql:load2.sql:421: NOTICE:  Adding maximum extent constraint
 addrasterconstraints
----------------------
 t
(1 row)

COMMIT
```

2d Comparing the file size

After loading the images check and report on the size of the data: how does the size of the 'raw' images compare to the size of the same data stored in the database? Can you explain the differences, if there are any?

File on the disk

```
ls -l
```

```
[~/Documents/TUD/GEO-1006/Assignments/07]$ ls -l
total 1845552
-rw-r--r--@ 1 mchu  staff  157288404 Jan  7 12:54 93000_436000.tif
-rw-r--r--@ 1 mchu  staff  157288404 Jan  7 12:54 93000_436500.tif
```

The file on the disk is approximately 314 MB (157 megabytes each).

File in the DB

```sql
select pg_size_pretty(pg_table_size('rotterdam'));
```

	pg_size_pretty
1	217 MB

The file in the database is approximately 217 MB, with 400 rows (because we use tile size of 500x500) in total.

Comparing differences

The main differences are in:

- Tiling Improves Compression Efficiency:
  - Splitting the raster into 500x500 tiles enhances compression by exploiting repetitive patterns within smaller, localized areas.
  - Uniform terrain areas particularly benefit from this approach.
- Efficient Storage of Numeric Data:
  - PostgreSQL uses a structured and compact format for raster storage.
  - Redundant parts of the TIFF file structure (e.g. headers) are optimized or removed during import.

2e Why is it advisable to specify tiling, and how is this related to indexing?

Tiling divides the data into chunks, which is good for data performance. This approach is particularly useful for large datasets. When loading the data, only the visible tiles are retrieved. Similarly, when querying, only the number of tiles required will be retrieved, depending on the region required. Indexing is required to retrieve the data. Each tile, when created, manages a number of x and y pixels that are stored in each BLOB field.

2f Why or why not it would be useful to include one or more 'overviews' (-l option) of the raster.

An overview is a low-resolution representation of the core tables of a raster. This allows quick map zoom-in and-out. Computations can also be performed on them in less time. However, their use depends on the application, as each pixel generalises a larger area, they will give less accurate results than the same calculation performed in their higher resolution parents.

3 Retrieve information from, query and manipulate raster data in the database

3a

```sql
select ST_Width(rast), ST_Height(rast), ST_SRID(rast), ST_NumBands(rast), ST_GeoReference(rast)
from rast_table rt;
```

	st_width	st_height	st_srid	st_numbands	st_georeference
1	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶92999.9999999998¶436500.0000000010¶
2	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93049.9999999998¶436500.0000000010¶
3	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93099.9999999998¶436500.0000000010¶
4	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93149.9999999998¶436500.0000000010¶
5	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93199.9999999998¶436500.0000000010¶
6	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93249.9999999998¶436500.0000000010¶
7	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93299.9999999998¶436500.0000000010¶
8	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93349.9999999998¶436500.0000000010¶
9	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93399.9999999998¶436500.0000000010¶
10	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93449.9999999998¶436500.0000000010¶
11	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93499.9999999998¶436500.0000000010¶
12	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93549.9999999998¶436500.0000000010¶
13	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93599.9999999998¶436500.0000000010¶
14	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93649.9999999998¶436500.0000000010¶
15	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93699.9999999998¶436500.0000000010¶
16	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93749.9999999998¶436500.0000000010¶
17	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93799.9999999998¶436500.0000000010¶
18	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93849.9999999998¶436500.0000000010¶
19	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93899.9999999998¶436500.0000000010¶
20	500	500	28,992	3	0.1000000000¶0.0000000000¶0.0000000000¶-0.1000000000¶93949.9999999998¶436500.0000000010¶

3b

```sql
select rid, ST_Value(rast, 1, ST_SetSRID(ST_Point(93780, 436208), 28992)) as band1, ST_Value(rast, 2, ST_SetSRID(ST_Point(93780, 436208), 28992)) as band2, ST_Value(rast, 3, ST_SetSRID(ST_Point(93780, 436208), 28992))as band3
from rast_table rt
where ST_Intersects(rast, ST_SetSRID(ST_Point(93780, 436208), 28992));
```

	rid	band1	band2	band3
1	116	130	124	105

3c

```sql
-- This doesn't save the value
select ST_SetValue(rast, 1, ST_SetSRID(ST_Point(93780, 436208), 28992), 101)
from rast_table rt
where ST_Intersects(rast, ST_SetSRID(ST_Point(93780, 436208), 28992));

-- This one will save the value
UPDATE rast_table
SET rast = ST_SetValue(
    ST_SetValue(
        ST_SetValue(rast, 1, ST_SetSRID(ST_Point(93780, 436208), 28992), 101),-- Band 1
        2, ST_SetSRID(ST_Point(93780, 436208), 28992), 102),                  -- Band 2
    3, ST_SetSRID(ST_Point(93780, 436208), 28992), 103)                       -- Band 3
WHERE ST_Intersects(rast, ST_SetSRID(ST_Point(93780, 436208), 28992));
```

	rid	band1	band2	band3
1	116	101	102	103

3d

```sql
CREATE TABLE clipped_raster_3d AS
select ST_Clip(rast, ST_MakeEnvelope(93750,436186, 93778,436211, 28992))
from rast_table rt
where ST_Intersects(rast, ST_MakeEnvelope(93750,436186, 93778,436211, 28992));
```

Figure 14: 3d - Clipped raster

3e

```sql
CREATE TABLE clipped_raster_3e AS
SELECT ST_Clip(rast, ST_MakeEnvelope(93550,436350, 93750,436600, 28992)) AS rast
FROM rotterdam rt
WHERE ST_Intersects(rast, ST_MakeEnvelope(93550,436350, 93750,436600, 28992));
```

Figure 16: 3e - extent view in QGIS

Figure 17: 3e - Clipped raster

3f

```sql
CREATE TABLE rescaled_raster AS
SELECT
    ST_Rescale(
        ST_Clip(rast, ST_MakeEnvelope(93550, 436350, 93750, 436600, 28992)),
        0.25,
        0.25
    ) AS rast
FROM rotterdam rt
WHERE ST_Intersects(rast, ST_MakeEnvelope(93550, 436350, 93750, 436600, 28992));
```

Figure 19: 3f - Rescaled raster

General

Name	clipped_raster_3e
Source	dbname='Assignment7' host=localhost port=5432 srid=0 table="public"."clipped_raster_3e" (rast)
Provider	postgresraster

Information from provider

Extent	93549.9999999997962732,436349.9000000006053597 : 93750.0999999998020940,436600.0000000005820766
Width	2001
Height	2501
Data type	Byte - Eight bit unsigned integer
Additional information	
Is Tiled	false
Overviews	
Pixel Size	0.1, -0.1
Primary Keys SQL	ctid
Temporal Column	
Where Clause SQL	

Figure 20: 3f - Properties before rescale

General

Name	rescaled_raster
Source	dbname='Assignment7' host=localhost port=5432 srid=0 table="public"."rescaled_raster" (rast)
Provider	postgresraster

Information from provider

Extent	93549.9999999997962732,436349.7500000005820766 : 93750.2499999997962732,436600.0000000005820766
Width	801
Height	1001
Data type	Byte - Eight bit unsigned integer
Additional information	
Is Tiled	false
Overviews	
Pixel Size	0.25, 0.25
Primary Keys SQL	ctid
Temporal Column	
Where Clause SQL	

Figure 21: 3f - Properties after rescale

4 Visualize raster data from the database in QGIS frontend

We have presented some of the results of Part III in QGIS and attached them to Part 3.

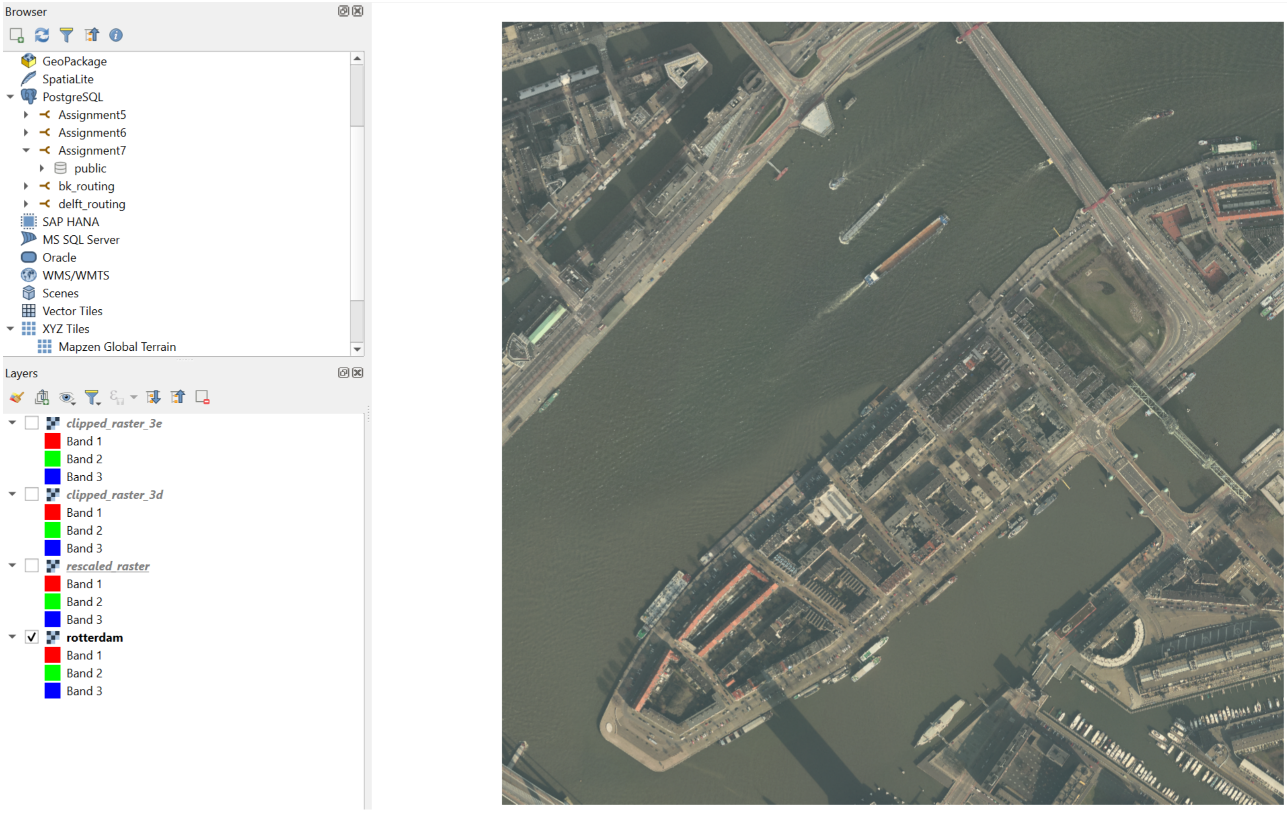

Figure 22: Overview of the raster data in QGIS