Introducing the PostGIS Add-ons: An easy way to add functionality to PostGIS

Pierre Racine
Research assistant
GeoElucubrations
The PostGIS Add-ons

- Contributing to PostGIS core is hard
  - Complex code with lots of history
  - Hard to compile under Windows

- A single SQL file of PL/pgSQL only functions
  - Self documented
  - Companion tests and uninstall scripts

- Provide and easy way for advanced PostGIS users to contribute with new PL/pgSQL functions (via GitHub)
  - Good practice to write new functions as prototypes in PL/pgSQL before integrating them in PostGIS core
  - Make them available to other users to test and comment
  - Give time to determine the best function signature
Simple Functions

- **ST_DeleteBand**(rast, band)
- **ST_RandomPoints**(geom, nb, seed)
- **ST_AddUniqueID**(schemaname, tablename, colname, replace, indexit)

Useful to other functions

- **ST_ColumnExists**(schemaname, tablename, colname)
- **ST_HasBasicIndex**(schemaname, tablename, colname)
- **ST_ColumnIsUnique**(schemaname, tablename, colname)
More Complex Functions

Topology
- ST_GeoTableSummary()
- ST_DifferenceAgg()
- ST_SplitAgg()

Complex Geometries
- ST_TrimMulti()
- ST_NBiggestExteriorRings()
- ST_BufferedSmooth()
- ST_BufferedUnion()

Raster/Vector Analysis
- ST_CreateIndexRaster()
- ST_AreaWeightedSummaryStats()
- ST_ExtractToRaster()
- ST_GlobalRasterUnion()
- ST_SplitByGrid()
- ST_SummaryStatsAgg()

Others
- ST_Histogram()
ST_GeoTableSummary()

- Provides 9 types of summary about a geometry table
  1. Duplicate ids (S1 or IDDUP)
  2. Duplicate geometries (S2, GDUP or GEODUP)
  3. Overlapping geometries (S3 or OVL)
  4. Geometry types (S4, TYPES, GTYPES or GEOTYPES)
  5. Vertexes stats (min, max, mean) (S5 or VERTX)
  6. Vertexes histogram (S6 or VHISTO)
  7. Areas stats (min, max, mean) (S7, AREA or AREAS)
  8. Areas histogram (S8 or AHISTO)
  9. Small areas count (S9 or SACOUNT)

• ST_GeoTableSummary(
  schemaname, tablename, geomcolumnname, uidcolumnname,
  nbhistobins,
  list_of_summaries_to_do,
  list_of_summaries_to_skip,
  where_clause
)

- A uid column is created and indexed if necessary
- The geometry column is indexed if necessary

Still a lot of work to do:
- Add a gap summary
- Add a fixquery column
- Provide better search queries
- Support tables of linestrings
  - Intersections instead of overlaps
  - Length instead of areas
- Make it an aggregate?
ST_DifferenceAgg() and ST_SplitAgg()

Two methods to remove overlaps in a (multi)polygon table

ST_DifferenceAgg(geomA, geomB)

- The state function removes, using ST_Difference(), all geomB from geomA
- Except the first geomB identical to geomA
- The final function simply returns the clipped geometry

```sql
SELECT a.id, ST_DifferenceAgg(a.geom, b.geom) geom
FROM overlappingtable a, -- Join the table with itself
     overlappingtable b
WHERE a.id = b.id OR -- Make sure the polygon is passed to and returned by the function
  ((ST_Contains(a.geom, b.geom) OR -- Select all the containing, contained and overlapping polygons
     ST_Contains(b.geom, a.geom) OR
     ST_Overlaps(a.geom, b.geom)) AND
  (ST_Area(a.geom) < ST_Area(b.geom) OR -- Make sure bigger polygons are removed from smaller ones
    (ST_Area(a.geom) = ST_Area(b.geom) AND -- If areas equals, arbitrarily rem. one from the other in a det. order
      a.id < b.id)))
GROUP BY a.id
HAVING ST_Area(ST_DifferenceAgg(a.geom, b.geom)) > 0 AND -- Do not select polygons completely erased poly
  NOT ST_IsEmpty(ST_DifferenceAgg(a.geom, b.geom));
```
ST_DifferenceAgg() and ST_SplitAgg()

ST_SplitAgg(geomA, geomB, tolerance)
- The state function split (using ST_Difference()) geomA with all geomB
- Sliver smaller than tolerance are not removed from the results
- The final function returns an array of the splitted geometries
- Arrays have to be unnested and Duplicates polygons have to
- be cleaned with DISTINCT

```sql
SELECT DISTINCT ON (geom) a.id,
    unnest(ST_SplitAgg(a.geom, b.geom, 0.00001)) geom
FROM overlappingtable a, -- join the table with itself
    overlappingtable b
WHERE ST_Equals(a.geom, b.geom) OR -- select the polygon itself
    ST_Contains(a.geom, b.geom) OR -- and overlapping ones
    ST_Contains(b.geom, a.geom) OR
    ST_Overlaps(a.geom, b.geom)
GROUP BY a.id
ORDER BY geom, max(ST_Area(a.geom)) DESC; -- select the id of the biggest
```
ST_CreateIndexRaster()

• Create a raster having a specified index ordering

ST_CreateIndexRaster

rast, a reference raster (ST_MakeEmptyRaster)
startvalue, the start value
pixeltype, the pixel type of the pixels
incwithx, incwithy, left to right or right to left, top to bottom or bottom to top
rowsfirst, vertically of horizontally
rowscanorder, row scan or prime-row scan
colinc, rowinc increment in x and y

<table>
<thead>
<tr>
<th>Default</th>
<th>incwithx = false</th>
<th>rowscanorder = false</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 3 6</td>
<td>1 4 7</td>
<td>3 2 1</td>
</tr>
<tr>
<td>1 4 7</td>
<td>7 4 1</td>
<td></td>
</tr>
<tr>
<td>2 5 8</td>
<td>8 5 2</td>
<td></td>
</tr>
<tr>
<td>3 6 9</td>
<td>9 6 3</td>
<td></td>
</tr>
<tr>
<td>rowinc = 10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example:

- Start value = 1
- Inc with x = false
- Rows first = false
- Row scan order = false
ST_AreaWeightedSummaryStats() • Simplify the writing of aggregates values after an intersection
  - ST_AreaWeightedSummaryStats(geom, val)
  or
  - ST_AreaWeightedSummaryStats(geomval)

WITH inter AS (
  SELECT gt.id, ST_Intersection(rt.rast, gt.geom) gv
  FROM raster_table rt,
       geometry_table gt
  WHERE ST_Intersects(rt.rast, gt.geom)
), aws AS (
  SELECT id, ST_AreaWeightedSummaryStats(gv) aws
  FROM inter
  GROUP BY id
)
SELECT id, (aws).geom, (aws).totalarea,
       (aws).weightedmean
FROM aws;
ST_ExtractToRaster()

- Iterate over every pixels of a raster an extract a metric from a vector coverage
- Bit slow but provides much flexibility over the type of metric computed

```
SELECT ST_ExtractToRaster(
    ST_AddBand(
        ST_MakeEmptyRaster(rast), '32BF'::text, -9999, -9999),
        'public', 'forestcover', 'geom',
        'height',
        'AREA_WEIGHTED_MEAN_OF_VALUES'
    ) rast
FROM ref_raster_tiled_coverage;
```

- COUNT_OF_VALUES_AT_PIXEL_CENTROID
- MEAN_OF_VALUES_AT_PIXEL_CENTROID
- COUNT_OF_POLYGONS
- COUNT_OF_LINESTRINGS
- COUNT_OF_POINTS
- COUNT_OF_GEOMETRIES
- VALUE_OF_BIGGEST
- VALUE_OF_MERGED_BIGGEST
- VALUE_OF_MERGED_SMALLEST
- MIN_AREA
- SUM_OF AREAS
- SUM_OF_LENGTHS
- PROPORTION_OF_COVERED_AREA
- AREA_WEIGHTED_MEAN_OF_VALUES
ST_GlobalRasterUnion()

- Iterate over every pixels of a raster an extracts a metric from a raster coverage
- Bit slow but provides much flexibility over the type of metric computed

```sql
SELECT ST_GlobalRasterUnion(  
    'source_raster_public',  
    'source_raster_table',  
    'rast',  
    'MEAN_OF_RASTER_VALUES_AT_PIXEL_CENTROID'  
) rast;
```

**Values computed from the centroids**

- COUNT_OF_RASTER_VALUES_AT_PIXEL_CENTROID
- FIRST_RASTER_VALUE_AT_PIXEL_CENTROID
- MIN_OF_RASTER_VALUES_AT_PIXEL_CENTROID
- MAX_OF_RASTER_VALUES_AT_PIXEL_CENTROID
- SUM_OF_RASTER_VALUES_AT_PIXEL_CENTROID
- MEAN_OF_RASTER_VALUES_AT_PIXEL_CENTROID
- STDDEVP_OF_RASTER_VALUES_AT_PIXEL_CENTROID
- RANGE_OF_RASTER_VALUES_AT_PIXEL_CENTROID

**Values computed from the pixel extents**

- AREA_WEIGHTED_SUM_OF_RASTER_VALUES
- SUM_OF_AREA_PROPORTIONAL_RASTER_VALUES
- AREA_WEIGHTED_MEAN_OF_RASTER_VALUES
ST_SplitByGrid()

- (ST_SplitByGrid(geom, 1000)).* returns
  - each polygon splitted by a global grid and
  - the unique identifier of the cell (bigint)

Polygons (originally splitted) have to be reunited afterward with ST_Union()
Functions Working With Complex Geometries

- **ST_TrTrimMulti**(geom, minarea)
  - Trim a multipolygon from small inner parts

- **ST_NBiggestExteriorRings**(geom, nbrings, comptype)
  - comptype = 'area': Returns the N biggest rings
  - comptype = 'nbpoints': Returns the N more complex rings

- **ST_BufferedSmooth**(geom, size)
  - Apply and remove a buffer in order to smooth sharp polygons concave angles and remove thin "inlets" (not removed by ST_TrTrimMulti) or holes

- **ST_BufferedUnion**(geom, size)
  - Apply a buffer before ST_Union() and remove it afterward
  - Make sure shared borders are unioned properly
  - Avoid many sliver interior rings after ST_Union()
  - Easier union of very complex vector coverage
ST_Histogram()

ST_Histogram(schemaname, tablename, colname, nbinterval DEF 10, whereclause)

<table>
<thead>
<tr>
<th>intervals</th>
<th>cnt</th>
<th>query text</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>0</td>
<td>SELECT * FROM public.a forestcover mtn7 WHERE height IS NULL;</td>
</tr>
<tr>
<td>[0 - 2.450000000000002]</td>
<td>2176</td>
<td>SELECT * FROM public.a forestcover mtn7 WHERE height &gt;= 0 AND height &lt; 2.450000000000002 ORDER BY height;</td>
</tr>
<tr>
<td>[2.450000000000002 - 4.900000000000004]</td>
<td>1114</td>
<td>SELECT * FROM public.a forestcover mtn7 WHERE height &gt;= 2.450000000000002 AND height &lt; 4.900000000000004 ORDER BY height;</td>
</tr>
<tr>
<td>[14.699999999999991 - 17.149999999999999]</td>
<td>0</td>
<td>SELECT * FROM public.a forestcover mtn7 WHERE height &gt;= 14.699999999999991 AND height &lt; 17.149999999999999 ORDER BY height;</td>
</tr>
<tr>
<td>[17.149999999999999 - 19.600000000000001]</td>
<td>90</td>
<td>SELECT * FROM public.a forestcover mtn7 WHERE height &gt;= 17.149999999999999 AND height &lt; 19.600000000000001 ORDER BY height;</td>
</tr>
<tr>
<td>[19.600000000000001 - 22.0500000000000001]</td>
<td>0</td>
<td>SELECT * FROM public.a forestcover mtn7 WHERE height &gt;= 19.600000000000001 AND height &lt; 22.0500000000000001 ORDER BY height;</td>
</tr>
<tr>
<td>[22.0500000000000001 - 24.5]</td>
<td>2</td>
<td>SELECT * FROM public.a forestcover mtn7 WHERE height &gt;= 22.0500000000000001 AND height &lt;= 24.5 ORDER BY height;</td>
</tr>
</tbody>
</table>
Thanks!

Pierre.Racine@sbf.ulaval.ca
https://github.com/pedrogit/postgisaddons