Annex: IMPLEMENTATION OF METRIC S

This annex collects the (PostgreSQL) SQL scripts that implement the computation of metric *S*. The solution is general, so the scripts are-usable for carrying out specific case studies. The Spatial DataBase (briefly *SDB*) is called `Violation`.

The SDB's tables:

```
CREATE TABLE GeoArea(
  id       serial  PRIMARY KEY,
  geom     geometry(MultiPolygon, SRS_code)
);
```

`SRS_code` is the number of a metric Spatial Reference System (for instance, in the case of Italy: 32633).

```
CREATE TABLE ContourLines (
  id          serial PRIMARY KEY,
  elevation   numeric,
  geom        geometry(MultiLineString, SRS_code)
);

CREATE INDEX ContourLines_geom_gist
  ON public.ContourLines
  USING gist (geom);

CREATE TABLE Rivers (
  id            integer,
  name          character varying(35),
  geom          geometry(MultiLineString, SRS_code),
  river_buffer  geometry(Polygon, SRS_code)
);

CREATE TABLE Buildings(
  id          serial PRIMARY KEY,
  geom        geometry(Point, SRS_code),
  status      bool,
  elevation   numeric,
  S           numeric
);

CREATE INDEX Buildings_geom_gist
  ON public.Buildings
  USING gist (geom);
```

Column `status` holds `true` if the building intersects the strip of respect of some river, in other word if it is an IB.

Fig.A shows the screen of the PostgreSQL tables of the `Violation` SDB.

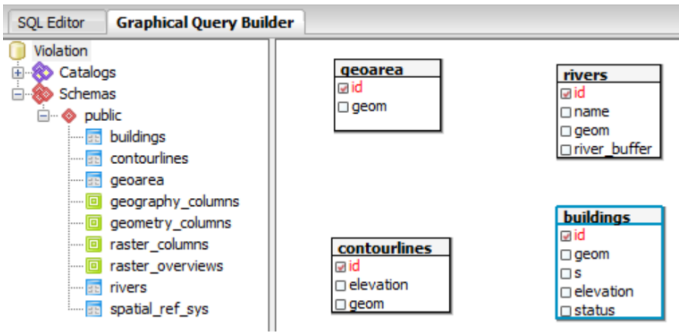

Fig. A The tables of the `Violation` SDB.

Initialization (to a dummy value) of columns `status`, `S` and `elevation`:

```
UPDATE Buildings
   SET status = false, S=-1, elevation=-1;
```

Creation of the rivers' buffer

```
UPDATE Rivers
   SET   river_buffer = ST_Buffer (geom, w)
   WHERE id BETWEEN 1 AND totalNumberOfRivers;
```

`w` is the width (in meters) of the strip of territory where it is forbidden to build (according to the Italian Law n.42 `w = 150`); while `totalNumberOfRivers` denotes the total number of the rivers that crosses the *GeoArea*.

Consistent with the DB-centric architecture adopted in the paper, the implementation of metric *S* is carried out in terms of spatial SQL queries. The invocation of several spatial SQL functions greatly simplified the implementation of metric *S*. In details, the following functions were used: `ST_Buffer()`, `ST_Area()`, `ST_Centroid()`, `ST_ClosestPoint()`, `ST_Distance()`, `ST_DWithin()`, `ST_Intersects()`, `ST_Intersection()`. In the implementation, we made large use of the `WITH` clause. `WITH` provides a way to write auxiliary statements for use in a larger query. These statements, which are often referred to as *Common Table Expressions* (CTEs), can be thought of as defining temporary tables that exist just for one query.

The queries

Creation of a working view aimed at simplifying the code of queries.

Q1

```
CREATE VIEW workingView (
      river_id, river_name, river_geom, river_buffer,
      building_id, building_geom, status,
      P, distance, S, elevation) AS
SELECT
      r.river_id, r.name, r.geom, r.river_buffer,
      b.id, b.geom,
      ST_Area(ST_Intersection(river_buffer, b.geom)) /
            ST_Area(b.geom) AS P,
      ST_Intersects (river_buffer, b.geom) AS status,
      ST_Distance(r.geom, b.geom) AS distance,
      S, elevation
FROM  Rivers AS r, Buildings AS b
WHERE ST_Intersects(river_buffer, b.geom) = true
```

In Q1 `P` implements Eq.1 (of the paper), while `distance` denotes the Euclidian distance between the boundary of the generic IB and the geometry that models the river bed. The tuples selected by `workingView` concern the IBs because of the condition `ST_Intersects(river_buffer, b.geom) = true`.

STEP 1: census of the IBs

Queries Q2A returns the total number of IBs, while query Q2B returns their list and the (WGS 84 long-lat) coordinates of their centroid.

Q2A (Number of IBs)

```
SELECT   COUNT(DISTINCT(building_id))
FROM     workingView
```

Q2B (IBs listing)

```
SELECT DISTINCT building_id,
       ST_AsText(ST_Transform(ST_Centroid(building_geom), 4326))
FROM   workingView
```

In Q2A and Q2B the clause `DISTINCT` is used to display only once the IBs that have a non-empty intersection with more than one river buffer. This circumstance takes place when two distinct rivers join in one point.

Query Q3 updates column `status` (of table `Buildings)` for each IB.

Q3
```
UPDATE Buildings AS b
   SET   status = w.status
   FROM  workingView AS w
   WHERE b.id = w.building_id;
```

STEP 2: ranking of the IBs

For the generic building (b_i), the parameter Δh (in Eq.2 of the paper) is computed as follows:

a) estimation of the elevation of b_i (let us denote it as elev_b_i)
b) identification of the point (let denote it as Q_j) such that:
   – Q_j belongs to the river whose buffer has no intersection with the geometry of b_i;
   – Q_j is at the minimum distance from b_i;
c) estimation of the elevation of Q_j (let as denote it as elev_Q_j);
d) computation of Δh = (elev_b_i - elev_Q_j).

To semplify the formulation of the remaining SQL queries, we added the columns: `partial_S`, `cp_cr` (the Closest Point, to building b_i, on the Closest River) and `e_cp_cr` (the elevation of point `cp_cr`) to the table `Buildings`. In these columns they will be copied, in order: MAX(P/d) (see Eq.2), the coordinates of point Q_j and its elevation.

```
ALTER TABLE Buildings
   ADD COLUMN partial_S numeric,
   ADD COLUMN cp_cr geometry(Point, SRS_code),
   ADD COLUMN e_cp_cr numeric;
```

Initialization of columns `cp_cr` and `e_cp_cr`:

```
UPDATE Buildings
   SET   cp_cr = ST_GeometryFromText('Point(0.0 0.0)', SRS_code)
   WHERE status;

UPDATE Buildings
   SET   e_cp_cr = -1
   WHERE status;
```

Computation of `partial_S`

Copy of the value MAX(P/d) in the column `partial_S` of the tuples referring to IBs.

Q4
```
WITH CTE1 AS (
  SELECT building_id, distance AS d, P
  FROM   workingView)

UPDATE Buildings AS new
SET partial_S =
  (SELECT DISTINCT
      CASE
         WHEN d BETWEEN 0 AND 0.999 THEN MAX(P)
         WHEN d BETWEEN 1 AND 150   THEN MAX(P/d)
      END AS partial_S
   FROM CTE1 AS cte1
   WHERE cte1.building_id = new.building_id
   GROUP BY building_id);
```

Q4 sets the value of `partial_S` to P if d<1m, otherwise it is set to max(P/d) (see Eq.2).

Estimation of the `elevation` *of the generic building* b_i

Q5 updates the column `elevation` (of `Buildings`).

Q5
```
WITH CTE2 AS (
 SELECT b.id AS building_id, c.id AS CL_id,
        ST_Distance(b.geom, ST_ClosestPoint(c.geom, b.geom)) AS distance,
        c.elevation
 FROM   Buildings  AS b, ContourLines AS c
 WHERE  status
)

UPDATE Buildings AS b
SET elevation =
  (SELECT MIN(elevation) AS elevation
   FROM CTE2 AS a
   WHERE distance =
     (SELECT MIN(x.distance) AS distance
      FROM CTE2 AS x
      WHERE    a.building_id = x.building_id  AND
               b.building_id = a.building_id  AND b.status
      GROUP BY x.building_id
      ORDER BY x.building_id ASC)
   GROUP BY a.building_id
   ORDER BY a.building_id ASC
);
```

Note

To speed-up the execution of query Q5, it is sufficient to replace in it table `ContourLines` with table `CL_insideStrip`, this latter computed as shown below. The spatial operation `ST_Intersection(ST_buffer(geom, L1), geom)` returns, for each contour line in `ContourLines`, the geometry of the portion of contour line that falls inside a strip of terrain of a given width (`L1` meters, here); strip centered around the geometry that models the river bed.

```
CREATE TABLE CL_insideStrip (
  id          serial PRIMARY KEY,
  geom        geometry(MultiLineString, SRS_code),
  elevation   numeric
);

INSERT INTO CL_insideStrip (geom, elevation)
  SELECT   ST_Intersection(ST_buffer(geom, L1), geom), elevation
  FROM     ContourLines;
```

Q6 copies (only for the IBs) the coordinates of point Q_j in column `cp_cr`. The filter `ST_DWithin(river_geom, building_geom, L2)` reduces the number of rivers to be taken into account to those whose (minimum) Euclidean distance from the generic building does not exceed `L2` meters.

Q6
```
WITH CTE3 AS (
 SELECT
       building_id, river_name,
       ST_ClosestPoint(river_geom, building_geom) AS cp_river_IB,
       ST_Distance(building_geom, ST_ClosestPoint(river_geom, building_geom))
                                                AS distance
 FROM  workingView
```

```
 WHERE ST_DWithin(river_geom, building_geom, L2)
)

UPDATE Buildings AS s
SET cp_cr =
  (SELECT cp_river_IB
   FROM   CTE3 AS a
   WHERE  distance =
      (SELECT MIN(distance)
       FROM   CTE3 AS b
       WHERE  s.building_id = a.building_id AND a.building_id=b.building_id
       GROUP BY b. building_id)
   GROUP BY a.building_id, river_name, cp_river_IB, distance
);
```

Q7 updates column `e_cp_cr` (of table `Buildings`)

Q7

```
WITH CTE4 AS (
  SELECT  s.id, ST_Distance(c.geom, s.cp_cr) AS distance,  c.elevation
  FROM    Buildings AS s, ContourLines AS c
  WHERE   ST_DWithin(c.geom, s.cp_cr, L2) AND status
)

UPDATE Buildings AS s1
SET e_cp_cr =
  (SELECT DISTINCT elevation
   FROM CTE4 AS a
   WHERE s1.id = id AND a.distance =
    (SELECT MIN(distance)
     FROM CTE4 AS b
     WHERE a.id=b.id AND s1.id = b.id
     GROUP BY b.id)
  )
WHERE s1.status;
```

Computation of S and update of the corresponding column in `Buildings`.

Q8

```
WITH CTE5 AS (
 SELECT id, elevation, e_cp_cr,
  CASE
     WHEN (b.elevation - b.e_cp_cr) < 0 THEN
                  (b.partial_S  * (1 -(b.elevation - b.e_cp_cr )))
     ELSE (b.partial_S / (1 + (b.elevation - b.e_cp_cr)))
  END AS S
 FROM  Buildings AS b
 WHERE b.status
)

UPDATE Buildings AS b1
SET S =
  (SELECT  S
   FROM   CTE5 AS a
   WHERE  b1.id = a.id);
```