Institute for Geoinformatics -Winter Semester 2014

Spatial Databases: Assignment 2

Lecturer: Malumbo Chipofya

Due Date: 10:00 am, Thursday, 4th December, 2014

Tasks

Slides 11 to 15 of the lecture given on 20th November 2014 contain an example design and SQL of a database for storing geometries. Answer the following questions with respect to the said database.

1. Write down the steps we must execute to consistently add a
    a. Point
    b. LineString
    c. LinearRing
2. What is a suitable notion of equality for LinearRings?
3. Assuming we live in a 2D grid world, with a coordinate system that has an origin at (0, 0) and in which all coordinates have non-negative integral values (whole numbers greater than or equal to 0). Write a function called make_world() to insert all points at grid intersections within the box ((0, 0, 0), (100, 100, 0)). make_world has no arguments. When inserting each point, set the "z" coordinate for that point to 0 – e.g. the point (1, 3) has to actually be inserted as (1, 3, 0). The points must be inserted one row at a time starting with the 0 row ending with the 100 – (0, 0), (1, 0), (2, 0), …, (100, 0), (0, 1), (1, 1), (2, 1), …, (100, 100). An illustration of the grid structure is shown below in terms of its coordinates.

(100,0)	(100,1)	(100,2)	.	.	.	(100,100)
.	.	.			.	.
.	.	.		.		.
.	.	.	.			.
(2,0)	(2,1)	(2,2)	.	.	.	(2,100)
(1,0)	(1,1)	(1,2)	.	.	.	(1,100)
(0,0)	(0,1)	(0,2)	.	.	.	(0,100)

   a. To execute your function use "SELECT make_world();"
   b. Give the formula required to determine the pid of a point from the grid given the coordinates of the point. You may use this formula to answer several of the next exercises. Test your formula with a few SELECT statements – calculate the pid manually and then check to see if the coordinates you get from "SELECT * FROM point WHERE pid = *yourComputedPID*;" are correct.

4. Complete the implementation of the function LineString_Verbose
   tip: use function regexp_split_to_array(string text, pattern text [, flags text]) to split the point list into an array list.
5. Insert two linestrings with attributes "no" and "non" respectively to the table LineString.
6. Use LineString_Verbose to specify their geometries: "no" = ((0,0), (0,2), (2,0), (2,2), (3,2), (3,0), (5,0), (5,2), (4,2), (4,1)) and "non" = ((0,0), (0,2), (2,0), (2,2), (3,2), (3,0), (5,0), (5,2), (4,2), (4,1), (6,0), (6,2), (8,0), (8,2))
7. We must accept, first, that inserting line string data into this database is a horrible task: Please elaborate or argue against this point of view!
8. A different approach: try this non-standard trick to simulate a simple sql table to which we will insert a linestring
   a. Create a view that retrieves a linestring by its attribute together with the pids of its points (in order)

b. Add a trigger to the view whose function has the same effect as the original functions LineString_Verbose.

Some hints for the SQL of the steps for this trick: Creating the view should be rather trivial. However, depending on your experience, the function get_vertices(INTEGER) may not be as trivial (I will quickly present my solution in class). To get all the vertices you must somehow iterate or recurse over the list of points belonging to the line string.

Step a.: CREATE VIEW LineStringView AS SELECT ls.lsid AS lsid, someattribute AS ls.someattribute, get_vertices(ls.lsid) AS vertices FROM LineString ls;

```
DROP FUNCTION IF EXISTS get_vertices(INTEGER) CASCADE;

CREATE OR REPLACE FUNCTION get_vertices(lsid INTEGER) AS TEXT
$$
DECLARE
currentV INTEGER; -- all local variables must declared here in pl/sql

-- By the way, comments start with a double minus as at the beginning of this line

BEGIN

        SELECT a random point2 from relation LineStringPointLists
                -- note that using the limit keyword at the end of a select statement allows you to do get a single
                row as the result of your query. If you do not use the ORDER BY expression the result of using
                LIMIT 1 is arbitrary – see the postgresql documentation.

        Iterate over all pairs in LineStringPointLists with LString = lsid and append each point to a list to be returned;

END;
$$ LANGUAGE plpgsql;
```

Step b.: To be presented in class

9. Insert the following linestrings using the new trick – i.e. using the INSERT INTO syntax

"none" = ((0,0), (0,2), (2,0), (2,2), (3,2), (3,0), (5,0), (5,2), (4,2), (4,1), (6,0), (6,2), (8,0), (8,2) (11,0), (9,0), (9,1), (11,1), (11,2), (9,2)~~, (9,1)~~)

"one" = ((2,2), (3,2), (3,0), (5,0), (5,2), (4,2), (4,1), (6,0), (6,2), (8,0), (8,2) (11,0), (9,0), (9,1), (11,1), (11,2), (9,2)~~, (9,1)~~)

"ne" = ((6,0), (6,2), (8,0), (8,2) (11,0), (9,0), (9,1), (11,1), (11,2), (9,2)~~, (9,1)~~)

NOTES:

Postgresql comes with a built in debugger. For those who do not know what that is and those interested in trying and learning please look it up here: http://en.wikipedia.org/wiki/Debugger

and here: http://www.pgadmin.org/docs/1.8/debugger.html

and here: http://www.postgresonline.com/journal/archives/214-Using-PgAdmin-PLPgSQL-Debugger.html

To install it edit the postgresql.conf file (located in C:\Program Files\PostgreSQL\9.3\data) by adding the following line at the end bottom of the file then saving it: shared_preload_libraries = '$libdir/plugins/plugin_debugger.dll'.

Then run "CREATE EXTENSION pldbgapi" and restart the server (in the Windows services panel). To debug a function find it in the database tree of pgadmin and right-click it. You should get an option for debugging.