

SQL and NoSQL Script Snippets

I have chosen to create the document of screenshots to demonstrate scripted examples instead of uploading it to a public repository because some scripts:

- are part of my final MSc project
- have been marked but as part of a module assessment, but I want to avoid plagiarism from students who are yet to complete the assessment in my cohort.
- may form part of a repeated syllabus to avoid plagiarism.

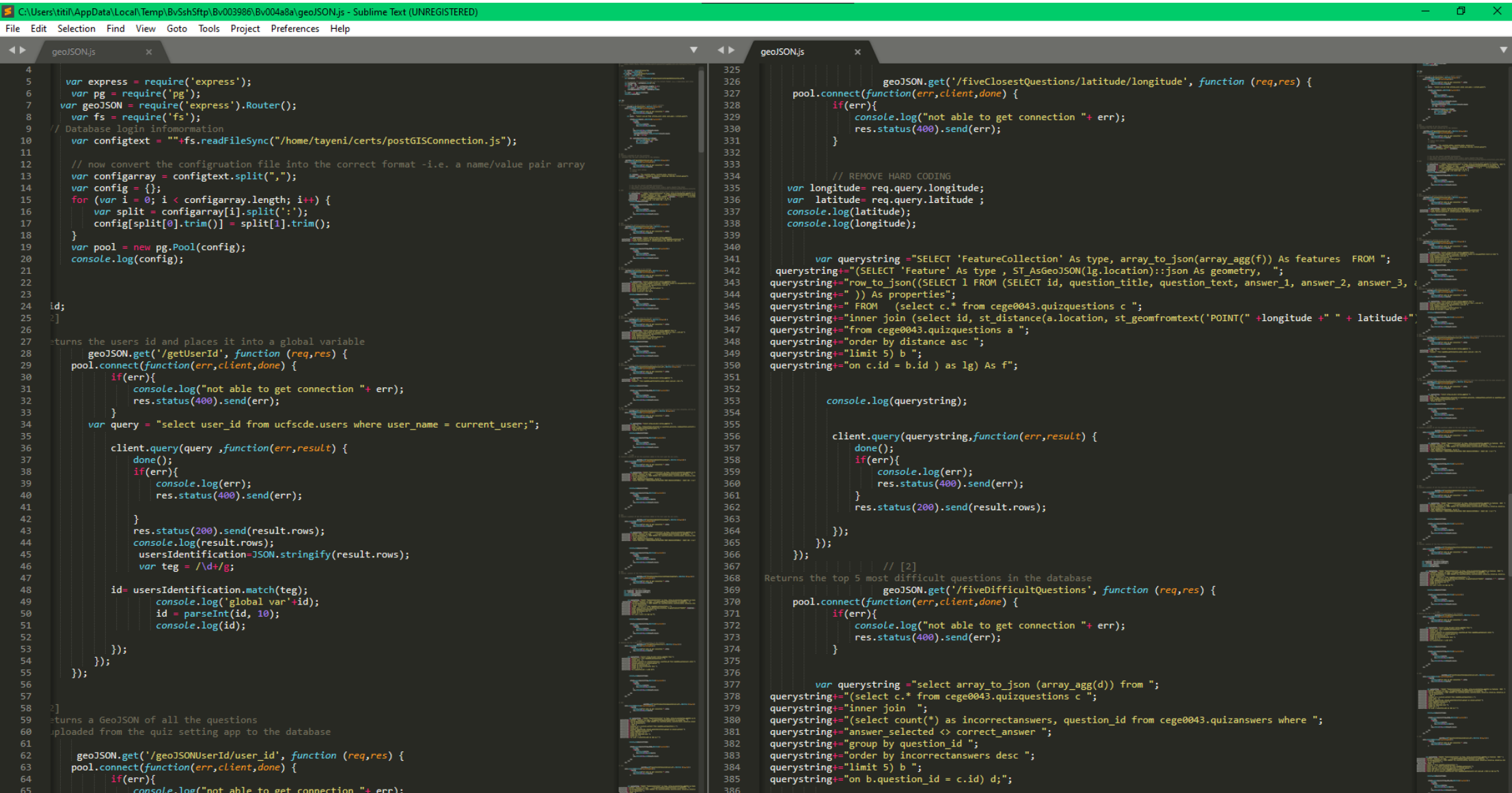
If required, I can send the complete code or create a private online repository for viewing.

Contents

SQL and JavaScript: Developing API to support a Mobile and Web Geospatial Application	2
SQL: Asset Management Database	3
SQL Queries	4
FME, NoSQL and JavaScript: Creating Spatial Operators (Area) in MongoDB	5

SQL and JavaScript: Developing API to support a Mobile and Web Geospatial Application

Developed a Restful API using node.js using the HTTP methods GET and POST to input data and retrieve data from a PostGIS database (Figure 1). To support a location-based web application that displayed the data using the JavaScript library, Leaflet.

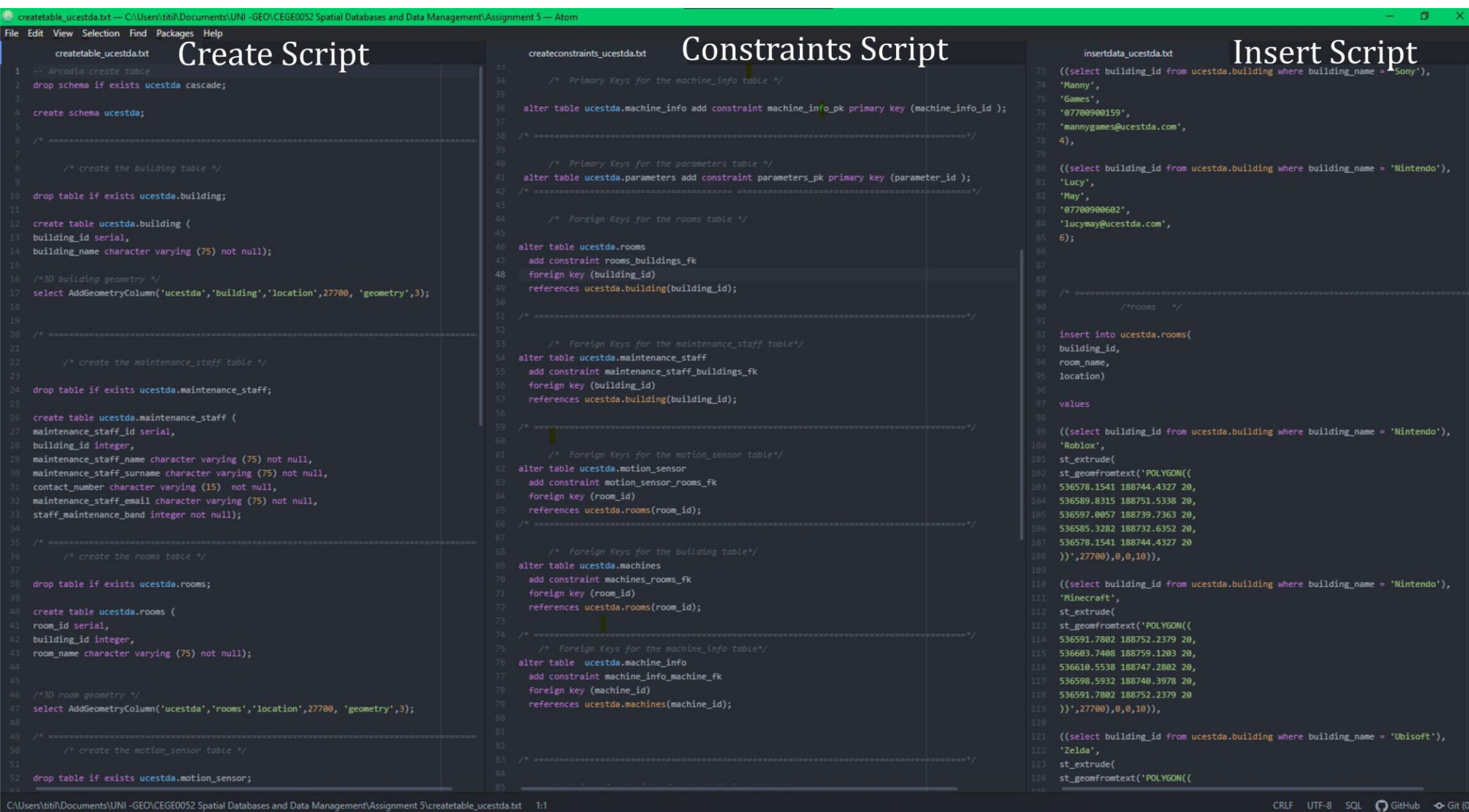


```
4  var express = require('express');
5  var pg = require('pg');
6  var geoJSON = require('express').Router();
7  var fs = require('fs');
8  // Database login information
9  var configtext = ""+fs.readFileSync("/home/tayeni/certs/postGISConnection.js");
10
11  // now convert the configuration file into the correct format -i.e. a name/value pair array
12  var configarray = configtext.split(",");
13  var config = {};
14  for (var i = 0; i < configarray.length; i++) {
15    var split = configarray[i].split(':');
16    config[split[0].trim()] = split[1].trim();
17  }
18  var pool = new pg.Pool(config);
19  console.log(config);
20
21  id;
22  2]
23
24  returns the users id and places it into a global variable
25  geoJSON.get('/getUserId', function (req,res) {
26    pool.connect(function(err,client,done) {
27      if(err){
28        console.log("not able to get connection "+ err);
29        res.status(400).send(err);
30      }
31      var query = "select user_id from ucfsdcde.users where user_name = current_user;";
32
33      client.query(query,function(err,result) {
34        done();
35        if(err){
36          console.log(err);
37          res.status(400).send(err);
38        }
39        res.status(200).send(result.rows);
40        console.log(result.rows);
41        usersIdentification=JSON.stringify(result.rows);
42        var teg = /\d+/g;
43
44        id= usersIdentification.match(teg);
45        console.log('global var'+id);
46        id = parseInt(id, 10);
47        console.log(id);
48      });
49    });
50  });
51
52  returns a GeoJSON of all the questions
53  uploaded from the quiz setting app to the database
54  geoJSON.get('/geoJSONUserId/user_id', function (req,res) {
55    pool.connect(function(err,client,done) {
56      if(err){
57        console.log("not able to get connection "+ err);
58      }
59
60      var querystring = "SELECT 'FeatureCollection' As type, array_to_json(array_agg(f)) As features FROM ";
61      querystring+="(SELECT 'Feature' As type , ST_AsGeoJSON(lg.location)::json As geometry, ";
62      querystring+="row_to_json((SELECT l FROM (SELECT id, question_title, question_text, answer_1, answer_2, answer_3, ";
63      querystring+=" )) As properties";
64      querystring+=" FROM (select c.* from cege0043.quizquestions c ";
65      querystring+="inner join (select id, st_distance(a.location, st_geomfromtext('POINT(" +longitude + " " + latitude+ ";";
66      querystring+="from cege0043.quizquestions a ";
67      querystring+="order by distance asc ";
68      querystring+="limit 5) b ";
69      querystring+="on c.id = b.id ) as lg) As f";
70
71      console.log(querystring);
72
73      client.query(querystring,function(err,result) {
74        done();
75        if(err){
76          console.log(err);
77          res.status(400).send(err);
78        }
79        res.status(200).send(result.rows);
80      });
81    });
82  });
83
84  Returns the top 5 most difficult questions in the database
85  geoJSON.get('/fiveDifficultQuestions', function (req,res) {
86    pool.connect(function(err,client,done) {
87      if(err){
88        console.log("not able to get connection "+ err);
89        res.status(400).send(err);
90      }
91
92      var querystring = "select array_to_json (array_agg(d)) from ";
93      querystring+="(select c.* from cege0043.quizquestions c ";
94      querystring+="inner join ";
95      querystring+="(select count(*) as incorrectanswers, question_id from cege0043.quizanswers where ";
96      querystring+="answer selected <> correct_answer ";
97      querystring+="group by question_id ";
98      querystring+="order by incorrectanswers desc ";
99      querystring+="limit 5) b ";
100     querystring+="on b.question_id = c.id) d";
```

Figure 1 Code snippet displaying the retrieval of data as GeoJSONs. The data was retrieved from a PostGIS database.

SQL: Asset Management Database

Primary and Foreign Keys | 2D and 3D Geometries | Constraints



The image shows a SQL editor window with three tabs: 'createtable_ucestda.txt', 'createconstraints_ucestda.txt', and 'insertdata_ucestda.txt'. The editor is displaying the contents of these files, which are SQL scripts for creating a database schema, adding constraints, and inserting data.

Create Script

```
1 -- Arcadio create table
2 drop schema if exists ucestda cascade;
3
4 create schema ucestda;
5
6 /* =====
7
8 /* create the building table */
9
10 drop table if exists ucestda.building;
11
12 create table ucestda.building (
13 building_id serial,
14 building_name character varying (75) not null;
15
16 /*3D building geometry */
17 select AddGeometryColumn('ucestda','building','location',27700, 'geometry',3);
18
19 /* =====
20
21 /* create the maintenance_staff table */
22
23 drop table if exists ucestda.maintenance_staff;
24
25 create table ucestda.maintenance_staff (
26 maintenance_staff_id serial,
27 building_id integer,
28 maintenance_staff_name character varying (75) not null,
29 maintenance_staff_surname character varying (75) not null,
30 contact_number character varying (15) not null,
31 maintenance_staff_email character varying (75) not null,
32 staff_maintenance_band integer not null;
33
34 /* =====
35
36 /* create the rooms table */
37
38 drop table if exists ucestda.rooms;
39
40 create table ucestda.rooms (
41 room_id serial,
42 building_id integer,
43 room_name character varying (75) not null;
44
45 /*3D room geometry */
46 select AddGeometryColumn('ucestda','rooms','location',27700, 'geometry',3);
47
48 /* =====
49
50 /* create the motion_sensor table */
51
52 drop table if exists ucestda.motion_sensor;
```

Constraints Script

```
33
34 /* Primary Keys for the machine_info table */
35
36 alter table ucestda.machine_info add constraint machine_info_pk primary key (machine_info_id);
37
38 /* =====
39
40 /* Primary Keys for the parameters table */
41 alter table ucestda.parameters add constraint parameters_pk primary key (parameter_id);
42 /* =====
43
44 /* Foreign Keys for the rooms table */
45
46 alter table ucestda.rooms
47 add constraint rooms_buildings_fk
48 foreign key (building_id)
49 references ucestda.building(building_id);
50
51 /* =====
52
53 /* Foreign Keys for the maintenance_staff table*/
54 alter table ucestda.maintenance_staff
55 add constraint maintenance_staff_buildings_fk
56 foreign key (building_id)
57 references ucestda.building(building_id);
58
59 /* =====
60
61 /* Foreign Keys for the motion_sensor table*/
62 alter table ucestda.motion_sensor
63 add constraint motion_sensor_rooms_fk
64 foreign key (room_id)
65 references ucestda.rooms(room_id);
66 /* =====
67
68 /* Foreign Keys for the building table*/
69 alter table ucestda.machines
70 add constraint machines_rooms_fk
71 foreign key (room_id)
72 references ucestda.rooms(room_id);
73
74 /* =====
75
76 /* Foreign Keys for the machine_info table*/
77 alter table ucestda.machine_info
78 add constraint machine_info_machine_fk
79 foreign key (machine_id)
80 references ucestda.machines(machine_id);
81
82 /* =====
83
84
85
```

Insert Script

```
73 ((select building_id from ucestda.building where building_name = 'Sony'),
74 'Manny',
75 'Games',
76 '07700900159',
77 'mannygames@ucestda.com',
78 4),
79
80 ((select building_id from ucestda.building where building_name = 'Nintendo'),
81 'Lucy',
82 'May',
83 '07700900602',
84 'lucymay@ucestda.com',
85 6);
86
87
88
89 /* =====
90 /*rooms */
91
92 insert into ucestda.rooms(
93 building_id,
94 room_name,
95 location)
96 values
97
98
99 ((select building_id from ucestda.building where building_name = 'Nintendo'),
100 'Roblox',
101 st_extrude(
102 st_geomfromtext('POLYGON((
103 536578.1541 188744.4327 20,
104 536589.8315 188751.5338 20,
105 536597.0057 188739.7363 20,
106 536585.3282 188732.6352 20,
107 536578.1541 188744.4327 20
108 ))',27700),0,0,10)),
109
110 ((select building_id from ucestda.building where building_name = 'Nintendo'),
111 'Minecraft',
112 st_extrude(
113 st_geomfromtext('POLYGON((
114 536591.7802 188752.2379 20,
115 536603.7408 188759.1203 20,
116 536610.5538 188747.2802 20,
117 536598.5932 188740.3978 20,
118 536591.7802 188752.2379 20
119 ))',27700),0,0,10)),
120
121 ((select building_id from ucestda.building where building_name = 'Ubisoft'),
122 'Zelda',
123 st_extrude(
124 st_geomfromtext('POLYGON((
```

Figure 2 Demonstration of creating a schema and inserting data.

SQL Queries

Nested Queries| Common Table Expressions | Joins | Spatial Joins

```
49 #####
50 "Which rooms and machines are and are not covered by
51 motion_sensor ? The null values are machines that are not covered
52 by motion_sensor."
53 ucestda.parameters ;ucestda.motion_sensor;ucestda.machines;ucestda.rooms
54 ##### 1 rooms motion_sensor
55 with
56 -- find buffer value for motion sensor
57 buffer_radius as (select parameter_value as buffer
58 from ucestda.parameters where parameter_name = 'buffer'),
59 -- find the machines which intersect this motion sensors radius
60 motion_sensor_covered as(
61 select b.motion_sensor_id, a.machine_id,a.machine_name,c.room_name
62 from ucestda.motion_sensor b
63 LEFT JOIN ucestda.machines a
64 on st_intersects((st_buffer(b.location,(select buffer from buffer_radius) )), a.location))
65 LEFT JOIN ucestda.rooms c
66 on st_contains(st_envelope(c.location),(a.location))),
67 -- returns the machine name , room it is in and id
68 all_info as (select y.machine_name,x.room_name,
69 y.machine_id
70 from ucestda.machines y
71 left join ucestda.rooms x
72 on x.room_id = y.room_id)
73 -- compares the results of the two temporary table
74 -- returning null if the machine is not in the motion sensor area
75
76 select m.*,i.motion_sensor_id
77 from all_info m
78 left join motion_sensor_covered i
79 on m.machine_id = i.machine_id order by machine_id ;
80
81
82
83 # 3D ROOM AND 2D MACHINE
84 select a.*,c.room_name
85 from ucestda.machines a LEFT JOIN ucestda.rooms c
86 on st_contains(st_envelope(c.location),(a.location));
87
88
89 #####
90 "SQL for most expensive costing machines that need to maintenance"
91
92 select purchase_cost,last_maintenance_date from ucestda.machine_info order
93 by last_maintenance_date,purchase_cost ;
94
95 select purchase_cost,last_maintenance_date from ucestda.machine_info
96 where last_maintenance_date <
97 PART('day', current_date::timestamp -
98 last_maintenance_date::timestamp)
99
100 "Machines most in need of maintenance"
101 (SELECT (current_date -last_maintenance_date) as days_since_last_maintenanceed,
102 last_maintenance_date,
103 machine_id,
104 purchase_cost
105 from ucestda.machine_info where
106 (current_date -last_maintenance_date) >365 order by last_maintenance_date)
107
108
109 WITH
110 needs_maintenance as
111 (SELECT (current_date -last_maintenance_date) as days_since_last_maintenanceed,
```

```
1 " Write a query that joins the window information to the criticality information to see the
2 criticality of each window."
3
4
5 select a.*,b.*,c.*
6 from ucfsdc.rooms a inner join ucfsdc.windows b inner join ucfsdc.window_condition c
7 on a.room_id = b.room_id
8 on b.window_id = c.window_id;
9 select a.*,b.*,c.*
10 from ucfsdc.rooms a inner join ucfsdc.windows b
11 on a.room_id = b.room_id
12 inner join ucfsdc.window_condition c
13 on b.window_id = c.window_id;
14
15
16
17 "Write a query to calculate the cost of a total refurbishment of the classrooms in each building, using
18 information from the parameters table."
19
20 With
21 classroom_cost as (select * from ucfsdc.parameters
22 where parameter_type = 'cost'and
23 parameter_name = 'rooms' and parameter_subname='total refurbishment'),
24
25 classrooms as (select * from ucfsdc.rooms where room_use = 'classroom'),
26
27 classroom_cost_calculation as (select a.*, b.parameter_value, b.parameter_units,
28 st_area(a.location)* b.parameter_value as refurb_cost from classrooms a, classroom_cost b)
29
30 select c.*, d.*
31 from ucfsdc.buildings c INNER JOIN classroom_cost_calculation d
32 on c.building_id = d.building_id;
33
34
35
36 "Write a query to join the sensors to the university. Use st_intersects, and a LEFT JOIN with the sensors table
37 as table a first and then the university as table b. Select all columns from both tables. Do not use WITH"
38
39 select a.*, b.*
40 from ucfsdc.temperature_sensors a LEFT JOIN
41 ucfsdc.university b
42 on st_intersects(a.location,b.location);
43
44
45
46 "Write a query to link the latest condition report for each room with the details of that room. You will need
47 to use ORDER BY and DISTINCT ON to get the latest condition report for each room. Use a WITH statement to
48 get the latest condition report for each room, with the temporary table called latest_condition_report.
49 Then use a LEFT join with the rooms table first with alias a and the latest_condition_report table second
50 with alias b."
51
52 WITH
53 latest_condition_report as (select distinct on (room_id) room_id, ceiling_condition, wall_condition,
54 windows_condition, doors_condition,
55 furniture_equipment_condition, air_conditioning_condition, sockets_condition,
56 lighting_and_switches_condition,
57 report_date, user_id from ucfsdc.room_condition order by room_id ,report_date desc)
58
59 select a.*,b.*
60 from ucfsdc.rooms a left join
61 latest_condition_report b
62 on a.room_id = b.room_id;
```

Figure 3 The left script queries the schema in Figure 2, and the right script is part of a Spatial Databases module quiz.

FME, NoSQL and JavaScript: Creating Spatial Operators (Polygon Area) in MongoDB

This page demonstrates how to use **JavaScript** to create a **spatial operator** in **MongoDB**. First, Ordnance Survey Building footprint **shapefiles** downloaded from **Digimap** are read into **FME**, an area attribute was added for later data validation then uploaded to a **MongoDB Collection** (Figure 5). Finally, the function is tested in a **JavaScript IDE** (Figure 5), then the **aggregation operator** is used to test the function in MongoDB (Figure 6), and the solution can be compared to the stored attribute.

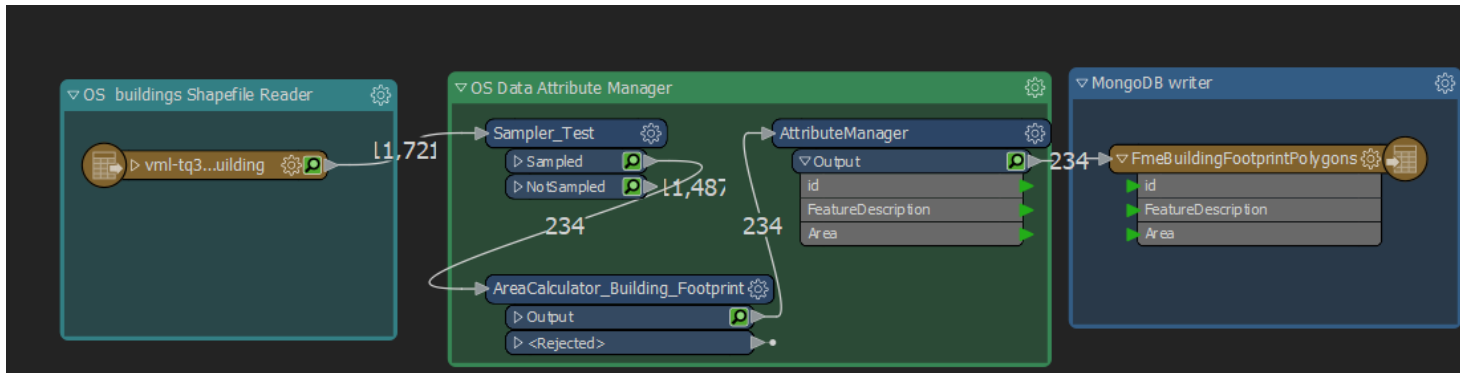


Figure 4 Simple tester FME workbench reading in and Ordnance Survey polygons and writing it into a MongoDB collection.

```
44 function polygonArea(geojson){
45   var coordArray = geojson.geometry.coordinates[0][0];
46   var numPoints = coordArray.length;
47   var X = []; var Y = [];
48   for (let i = 0; i < coordArray.length; i++) {
49     console.log(i, coordArray[i],
50       X.push(coordArray[i][0]),
51       Y.push(coordArray[i][1])
52     );
53   }
54   var area = 0; var j; j = numPoints-1;
55   for (var k = 0; k < numPoints; k++)
56     { area += (X[j]*X[k]) * (Y[j]-Y[k]);
57       j = k;
58     }
59   return area/2;
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
```

Figure 5 Javascript code to calculate the area of a regular polygon using coordinates.

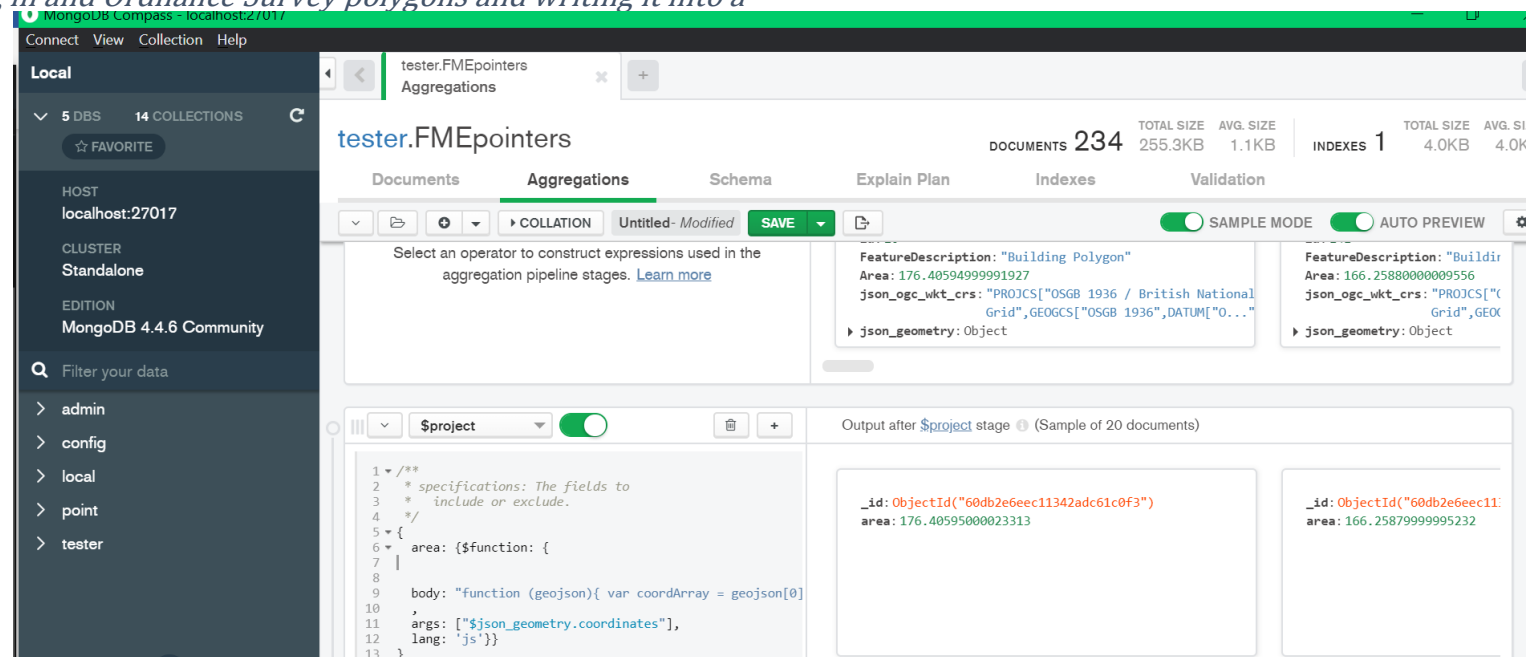


Figure 6 Results of querying the area of the building footprints in MongoDB using the Javascript function in a query. The attribute "Area" is used to validate the solution.