Interfacing NoSQL with open-source GIS

Vasilios Kalogirou, Jan Boehm

Department of Civil, Environmental and Geomatic Engineering – University College London

January 12, 2017

Summary

This paper focused on bringing together two different types of GIS software: MongoDB, a spatially-enabled NoSQL Database Management System, and QGIS, a desktop GIS software. By assessing the functionality offered by the community-based plugins "MongoConnector" and "Load MongoDB Layers" the need to further enhance interfaces between the two systems was identified. With regard to this need, "Save Layer in MongoDB" was developed, a QGIS plugin which allows users to store vector data in MongoDB.

KEYWORDS: MongoDB, QGIS, open-source GIS, Python, vector data

1. Introduction

According to Steiniger and Hunter (2013, p.5) a spatially-enabled Database Management System (DBMS) is a DBMS which offers spatial data types in its data model, a query language, and spatial analysis operators. In addition, it may provide spatial indexing structures such as R-tree indexes. Spatially-enabled DBMSs are usually used for storing large geospatial datasets and for performing operations which need to be executed in as short time as possible (Steiniger and Hunter, 2012). Object-Relational DBMSs have been the leading spatially-enabled DBMSs.

1.1. NoSQL systems

Recent technological advances such as the growth of the World Wide Web have resulted in generating large sets of data such as links, social networks, and mapping data and prompted to the development of alternative data models (Sadalage and Fowler, 2013). The term NoSQL is an umbrella for the DBMSs that do not make use of the relational model and share the following key features (Näsholm, 2012): built for Big Data applications, distributed, not ACID compliant – eventually consistent, schemaless, open-source, replication supportive, sharding (horizontal scalability) supportive, not SQL supportive.

Various approaches have been suggested in regard to NoSQL systems' classification. Figure 1 presents Robinson's taxonomy (2015, p.196) based on the differences between the data models, while Table 1 classifies a number of NoSQL DBMSs according to their data model.

vasilios.kalogirou.15@ucl.ac.uk

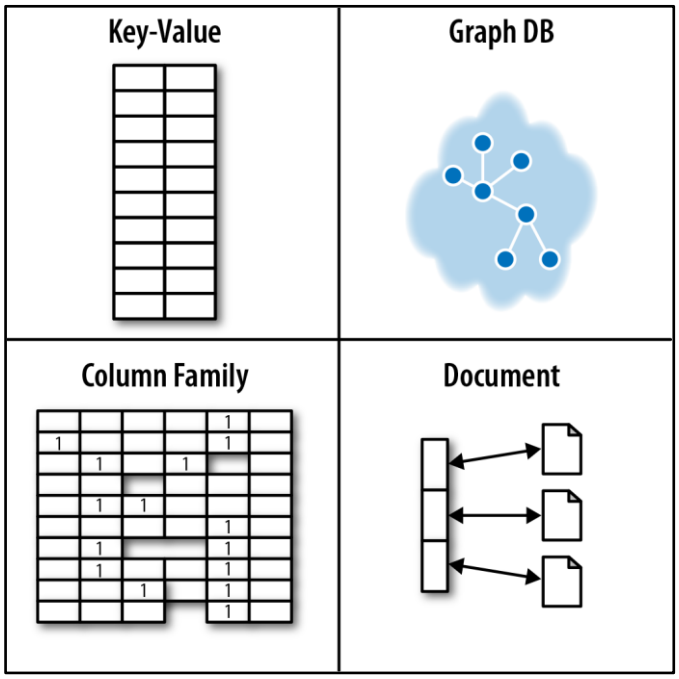


Figure 1 *Types of NoSQL data models [taken from:(Robinson, Webber and Eifrem, 2015)]*

Table 1 Classification of NoSQL systems by data model [taken from: (Sadalage and Fowler, 2013)]

Data Model	Example Databases
Key-Value	BerkeleyDB, LevelDB, Memcached, Project Voldemort, Redis, Riak
Document	CouchDB, MongoDB, OrientDB, RavenDB, Terrastore
Column-Family	Amazon SimpleDB, Cassandra, HBase, HyperTable
Graph	FlockDB, HyperGraphDB, Infinite Graph, Neo4j, OrientDB

1.2. MongoDB

MongoDB is a document DBMS which offers features such as horizontal scaling, secondary indexes, range queries, sorting, aggregations, file storage (GridFS), and geospatial indexes (Chodorow, 2013). A single instance of MongoDB can host multiple independent databases. Each database consists of collections which are composed of documents. A document is an ordered set of keys with associated values (Chodorow, 2013). MongoDB collections can have a dynamic schema, meaning that documents in a collection can vary in structure. MongoDB documents are stored in a binary encoding called BSON which extends the JSON representation to include additional types (MongoDB Inc., 2016a). Table 2 presents an SQL to MongoDB Mapping Chart for terminology and concepts.

Table 2 SQL vs MongoDB terminology and concepts [taken from: (MongoDB Inc., 2016d)]

SQL Terms/Concepts	MongoDB Terms/Concepts
database	database
table	collection
row	document or BSON document
column	field
index	index
table joins	embedded documents and linking
primary key (Specify any unique column or column combination as primary key)	primary key (In MongoDB, the primary key is automatically set to the _id field)
aggregation (e.g. group by)	aggregation pipeline

Spatial functionality in MongoDB is supported with the "2dsphere" and "2d" geospatial indexes. Geospatial data in MongoDB can be stored in two ways: either as GeoJSON objects on a spherical surface using a geographic 2D CRS or as legacy coordinate pairs using a projected CRS. The default CRS for GeoJSON objects is the WGS84 and the coordinate-axis order is [longitude, latitude]. The following types of GeoJSON objects are supported: "Point", "LineString", "Polygon" (can contain multiple rings), "MultiPoint", "MultiLineString", "MultiPolygon", "GeometryCollection" (MongoDB Inc, 2016). Legacy coordinate pairs should only be used to store point data. It is still possible to store an array of points as legacy coordinate pairs, however, the data will be used as an array of points and not as a line. This is important when applying an inclusion query to data of this type: the query will match a document if one of those points is within the polygon tested; it will not test if the line created by these points is within the polygon (Chodorow, 2013). The statements below are taken from the Mongo Manual 3.2. (MongoDB Inc., 2016b) and provide an example on how to store GeoJSON objects in MongoDB and how to create a "2dsphere" index on a collection.

```
>db.places.insert(
   {
      loc: {type: "Point", coordinates: [-73.97, 40.77]},
      name: "Central Park",
      category: "Parks"
   }
)

>db.places.createIndex({loc: "2dsphere"})
```

MongoDB supports three types of topology functions in the form of geospatial queries: queries for inclusion, intersection, and proximity. Within and intersection queries can be performed on data without a geospatial index, although having a geospatial index speeds up the query process. MongoDB does not support reprojection operations. Table 3 presents MongoDB's geospatial operators along with the geometry type each operator uses.

Table 3 MongoDB geospatial operators [taken from: (MongoDB Inc., 2016c)]

Query type	Geometry type	Notes
$near (GeoJSON point, 2dsphere index)	Spherical	
$near (legacy coordinates, 2d index)	Flat	
$nearSphere (GeoJSON point, 2dsphere index)	Spherical	
$nearSphere (legacy coordinates, 2d index)	Spherical	Use GeoJSON points instead.
$geoWithin : { $geometry: ... }	Spherical	
$geoWithin : { $box: ... }	Flat	
$geoWithin : { $polygon: ... }	Flat	
$geoWithin : { $center: ... }	Flat	
$geoWithin : { $centerSphere: ... }	Spherical	
$geoIntersects	Spherical	

1.3. Interfaces between spatially-enabled DBMSs and desktop GIS software

A significant limitation of spatially-enabled DBMSs is the absence of a Graphical User Interface for data visualisation. This limitation can be confronted by retrieving geospatial data stored in a DBMS and visualising them using a desktop GIS software. QGIS, a popular open-source desktop GIS software, supports retrieving data from a number of databases such as Oracle and PostgreSQL. It also offers enhanced capabilities for handling such data through community-developed plugins. More importantly, QGIS' DB Manager core plugin supports maintaining a connection between a PostgreSQL / Oracle / SQLite database with QGIS. The plugin enables users to visualise, query, edit and analyse data through the QGIS GUI, while data is maintained in the database. The fact that NoSQL databases have not been

around for long, and offer only a limited number of spatial capabilities means there have not been many efforts in interfacing spatially-enabled NoSQL systems with GIS software. The most significant interfaces are the community-developed plugins "MongoConnector" and "Load MongoDB Layers" which offer retrieving data stored in a MongoDB database and visualising them in QGIS.

2. Data

For the purposes of this paper, a number of GeoJSON files were downloaded from OSM (https://mapzen.com/data/metro-extracts/) presenting data in the city of London. Each of these files contained a GeoJSON feature collection object. The "restaurants.json" and "neighborhoods.json" files from MongoDB's online tutorial (https://docs.mongodb.com/v3.0/tutorial/geospatial-tutorial/) were also used. All files used are described in Table 4.

Table 4 Vector data sets used

File Name and extension	Type (geometry type)	CRS	Collection name in MongoDB
london_england_aeroways.geojson	Feature collection (linestring)	WGS84	aeroways
london_england_transport_points.geojson	Feature collection (point)	WGS84	transport_points
london_england_transport_areas.geojson	Feature collection (polygon)	WGS84	transport_areas
neighborhoods.json	Feature-like (polygon)	n/d*	neighborhoods
restaurants.json	Feature-like (point)	n/d*	restaurants

*n/d: not defined, there is no indication of the identity of the CRS inside the file.

3. Evaluation of MongoDB's spatial functionality and of existing interfaces between MongoDB and QGIS

3.1. Evaluation of MongoDB's spatial functionality

First, the GeoJSON feature collection files were imported in MongoDB using the "mongoimport" tool. The following commands were executed in the Windows command line:

```
>mongoimport --db mytestdb --collection transport_points <
~\london_england_transport_points.geojson

>mongoimport --db mytestdb --collection aeroways < ~\london_england_aeroways.geojson

>mongoimport --db mytestdb --collection transport_areas <
~\london_england_transport_areas.geojson
```

~ represents the directory where the file was stored. The result from executing the first command was the following:

```
connected to: localhost
mytestdb.transport_points  11.5 MB
imported 1 document
```

Consequently, the file was stored in MongoDB as one document in the "transport_points" collection. As stated in Section 1.2., geospatial data should be stored as separate documents and not as a single document in the form of a feature collection GeoJSON file. To achieve this, twenty features from the "london_england_transport_points.geojson" file were manually selected and formed a new JSON file named "london_england_transport_points_altered.json".

```
>mongoimport  --db mytestdb --collection transport_points2 <
~\london_england_transport_points_altered.json

connected to: localhost

imported 20 documents
```

A geospatial index on the geometry field was then created in the Mongo Shell.

```
> db.transport_points2.createIndex({geometry:"2dsphere"})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

The following statements evaluated MongoDB's topology operators.

```
> db.transport_points2.count({geometry: {$geoIntersects: {$geometry: {type:
"Point", coordinates:[-0.241885673740143, 51.606101317886576]}}}})

1
```

```
> db.transport_points2.count({geometry: {$geoWithin: {$geometry: {type: "Polygon",
coordinates: [[[-0.50, 51.10], [-0.10, 51.10], [-0.10, 51.70], [-0.50, 51.70], [-
0.50, 51.10]]]}}}})

7
```

```
> db.transport_points2.find({geometry: {$geoWithin: {$centerSphere: [[-0.20, 51.50],
10/6371]}}}).pretty()
{
        "_id" : ObjectId("57a4f3c0fa2886144433538f"),
        "type" : "Feature",
        "properties" : {
                "id" : 15,
                "osm_id" : 197446,
                "name" : null,
                "type" : "motorway_junction",
                "ref" : null
        },
        "geometry" : {
                "type" : "Point",
                "coordinates" : [
                        -0.229456903895425,
                        51.51495549694939
                ]
        }
}
{
        "_id" : ObjectId("57a4f3c0fa2886144433538e"),
        "type" : "Feature",
        "properties" : {
                "id" : 16,
                "osm_id" : 202077,
                "name" : "Staples Corner",
                "type" : "motorway_junction",
                "ref" : "1"
        },
        "geometry" : {
                "type" : "Point",
                "coordinates" : [
                        -0.229431003814597,
                        51.57310201654954
                ]
        }
}
```

```
db.transport_points2.find({geometry: {$nearSphere: {$geometry: {type: "Point",
coordinates: [-0.20, 51.50]}, $maxDistance:10000}}}).pretty()
{
        "_id" : ObjectId("57a4f3c0fa2886144433538f"),
        "type" : "Feature",
        "properties" : {
                "id" : 15,
                "osm_id" : 197446,
                "name" : null,
                "type" : "motorway_junction",
                "ref" : null
        },
        "geometry" : {
                "type" : "Point",
                "coordinates" : [
                        -0.229456903895425,
                        51.51495549694939
                ]
        }
}
{
        "_id" : ObjectId("57a4f3c0fa2886144433538e"),
        "type" : "Feature",
        "properties" : {
                "id" : 16,
                "osm_id" : 202077,
                "name" : "Staples Corner",
                "type" : "motorway_junction",
                "ref" : "1"
        },
        "geometry" : {
                "type" : "Point",
                "coordinates" : [
                        -0.229431003814597,
                        51.57310201654954
                ]
        }
}
```

Next, the "neighborhoods.json" and "restaurants.json" files were stored in the database. As shown in MongoDB's online tutorial (MongoDB Inc., 2016c), geospatial indexes and queries could be applied to the "neighborhoods" and "restaurants" collections.

3.2. Evaluation of the "MongoConnector" plugin

Figure 1 presents the plugin's interface.

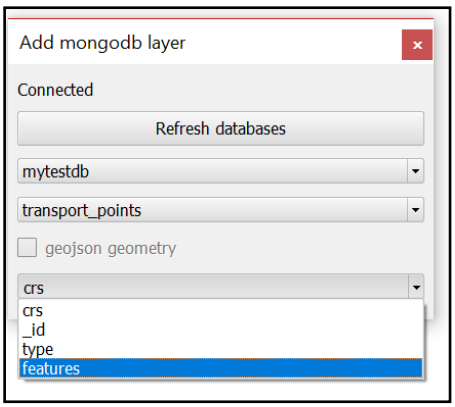


Figure 2 The "MongoConnector" plugin interface

The plugin successfully imported the “transport_points2” collection by creating a layer in the device’s memory which was imported in QGIS under the name “transport_points2-ebd4” as shown in Figure 3. The attribute table of the layer is shown in Figure 4. The layer’s labels were set according to the value of the “osm_id” attribute. However, the plugin failed to import the “transport_points”, “restaurants”, and “neighborhoods” collections.

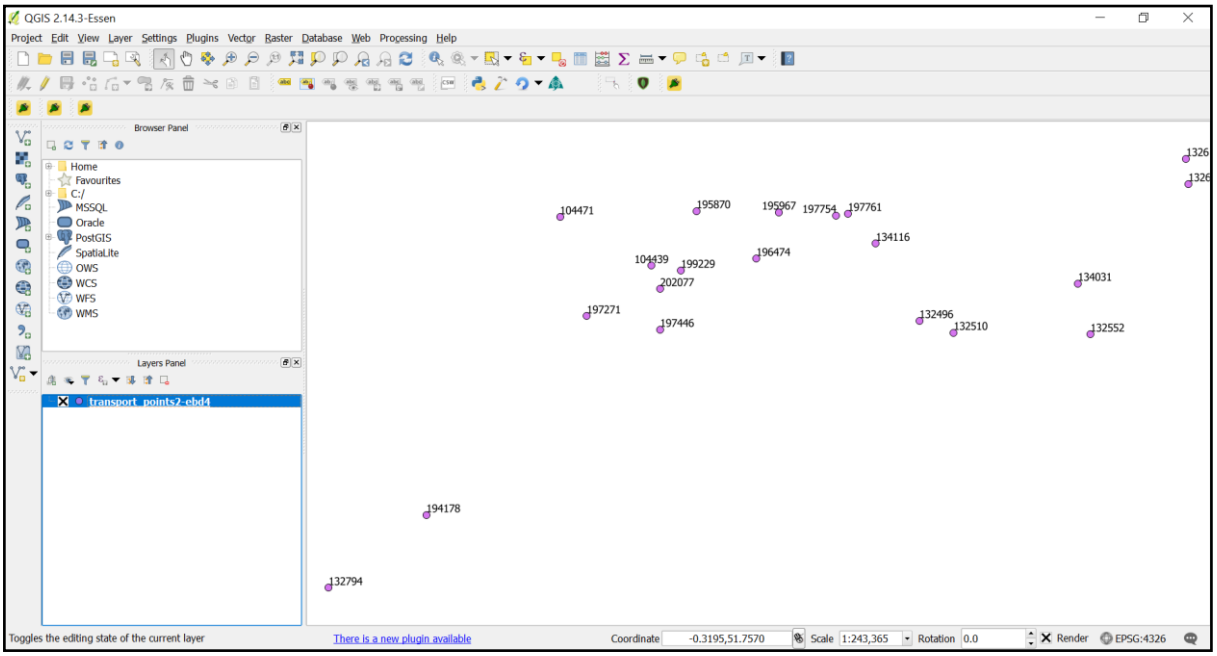
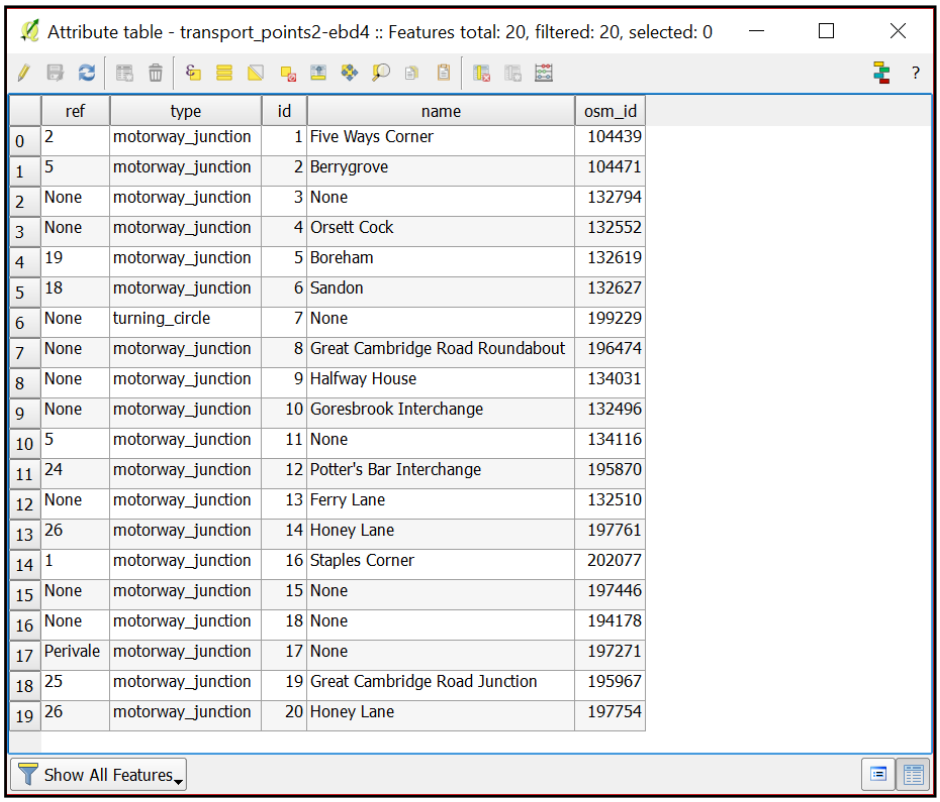


Figure 3 The “transport_points2-ebd4” layer created by “MongoConnector” representing the “transport_points2” collection



	ref	type	id	name	osm_id
0	2	motorway_junction	1	Five Ways Corner	104439
1	5	motorway_junction	2	Berrygrove	104471
2	None	motorway_junction	3	None	132794
3	None	motorway_junction	4	Orsett Cock	132552
4	19	motorway_junction	5	Boreham	132619
5	18	motorway_junction	6	Sandon	132627
6	None	turning_circle	7	None	199229
7	None	motorway_junction	8	Great Cambridge Road Roundabout	196474
8	None	motorway_junction	9	Halfway House	134031
9	None	motorway_junction	10	Goresbrook Interchange	132496
10	5	motorway_junction	11	None	134116
11	24	motorway_junction	12	Potter's Bar Interchange	195870
12	None	motorway_junction	13	Ferry Lane	132510
13	26	motorway_junction	14	Honey Lane	197761
14	1	motorway_junction	16	Staples Corner	202077
15	None	motorway_junction	15	None	197446
16	None	motorway_junction	18	None	194178
17	Perivale	motorway_junction	17	None	197271
18	25	motorway_junction	19	Great Cambridge Road Junction	195967
19	26	motorway_junction	20	Honey Lane	197754

Figure 4 The attribute table of the “transport_points2-ebd4” layer

3.3. Evaluation of the "Load MongoDB Layers" plugin

Figures 5 and 6 present the plugin's interface.

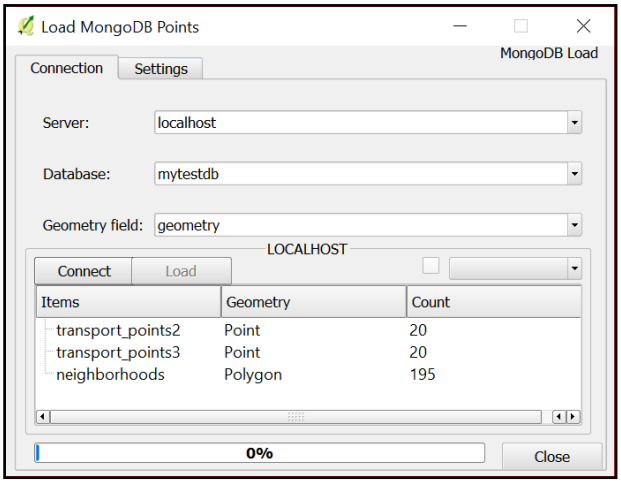


Figure 5 "Load MongoDB Layers" - "Connection" dialog

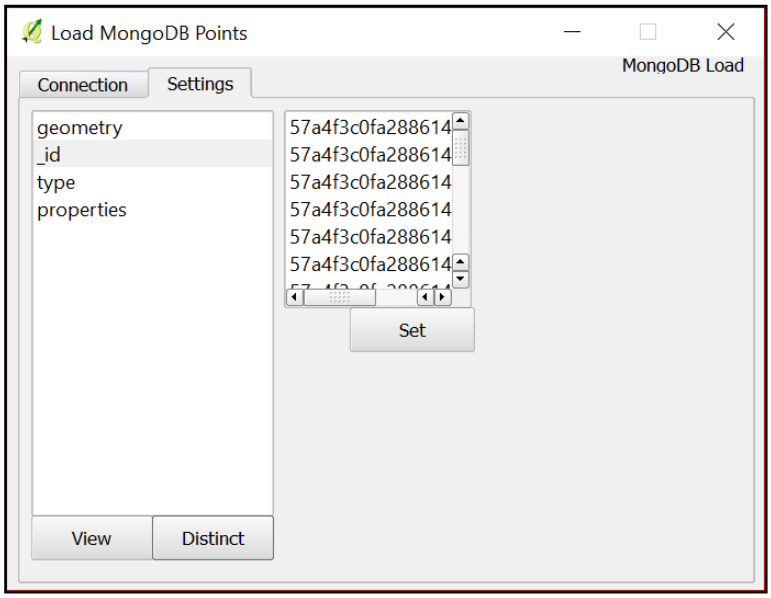


Figure 6 "Load MongoDB Layers" - "Settings" Dialog (used to select and import a collection's documents according to a field value)

The "transport_points2" collection was successfully imported in QGIS using the plugin under the name "transport_points2". Figure 7 shows the "transport_points2" layer in QGIS. The layer's labels were set according to the "id" attribute values. Before importing the collection, the additional capabilities of the plugin shown in Figure 6 were tested. However, the plugin failed to accomplish this task. The plugin also failed to import the "transport_points" collection.

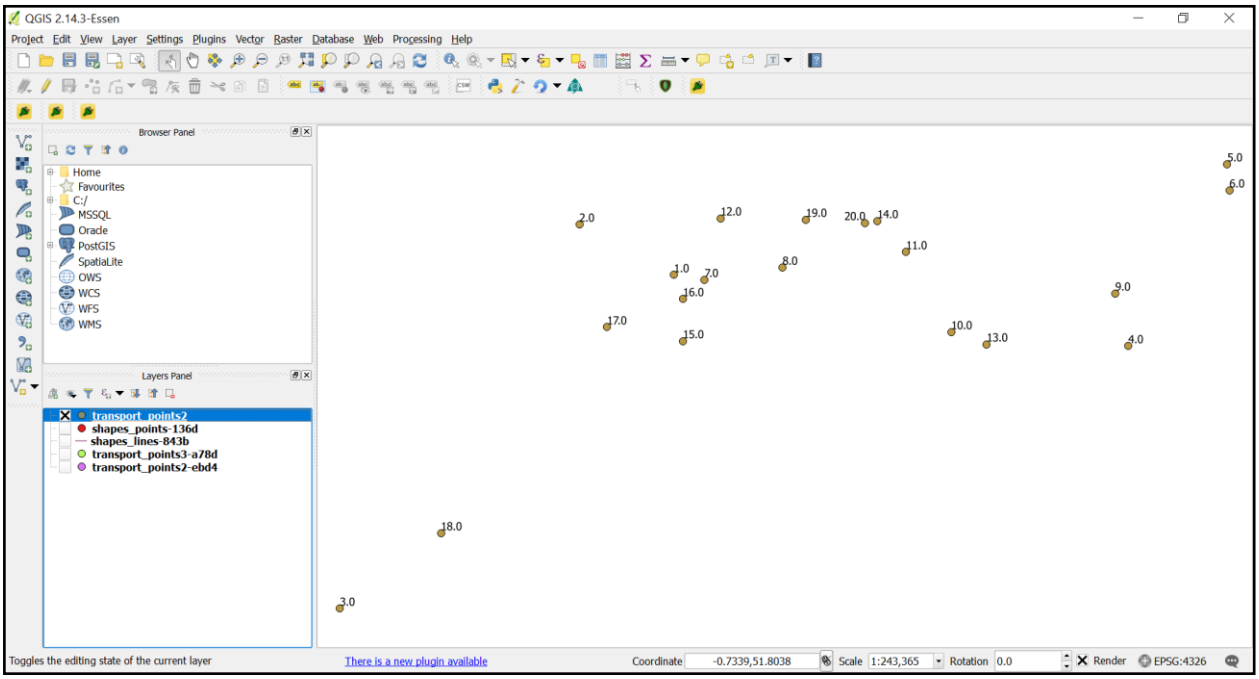
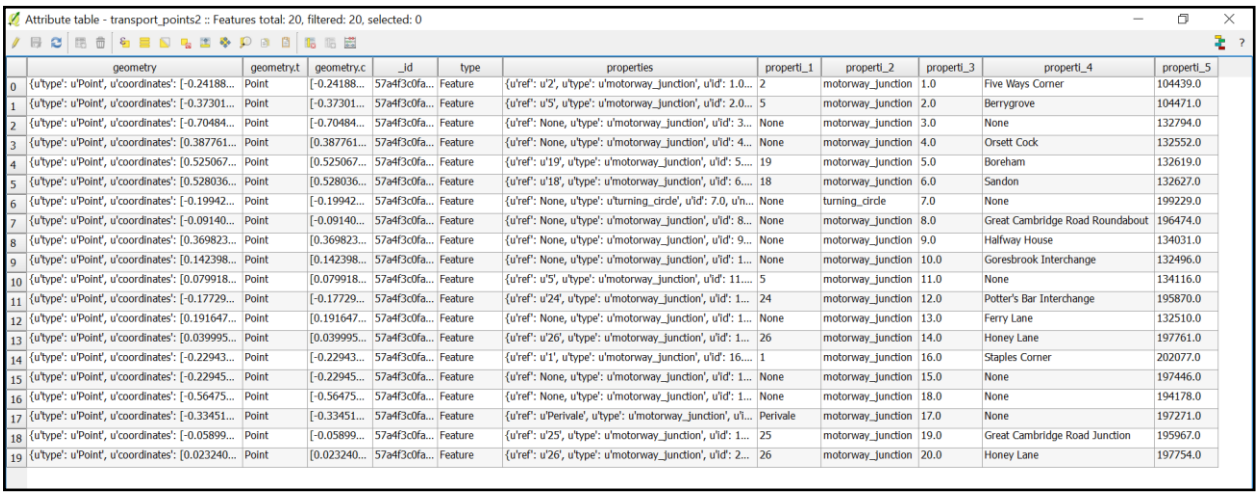


Figure 7 The "transport_points2" layer created by "Load MongoDB Layers" representing the "transport_points2" collection

As shown in figure 8, the plugin created an attribute for each field and added the properties' sub-fields as attributes under the names "properti1", "properti2", and so on, instead of assigning the exact names of the sub-fields.



	geometry	geometry.t	geometry.c	_id	type	properties	properti_1	properti_2	properti_3	properti_4	properti_5
0	{u'type': u'Point', u'coordinates': [-0.24188...	Point	[-0.24188...	57a4f3c0fa...	Feature	{u'ref': u'2', u'type': u'motorway_junction', u'id': 1.0...	2	motorway_junction	1.0	Five Ways Corner	104439.0
1	{u'type': u'Point', u'coordinates': [-0.37301...	Point	[-0.37301...	57a4f3c0fa...	Feature	{u'ref': u'5', u'type': u'motorway_junction', u'id': 2.0...	5	motorway_junction	2.0	Berrygrove	104471.0
2	{u'type': u'Point', u'coordinates': [-0.70484...	Point	[-0.70484...	57a4f3c0fa...	Feature	{u'ref': None, u'type': u'motorway_junction', u'id': 3...	None	motorway_junction	3.0	None	132794.0
3	{u'type': u'Point', u'coordinates': [0.387761...	Point	[0.387761...	57a4f3c0fa...	Feature	{u'ref': None, u'type': u'motorway_junction', u'id': 4...	None	motorway_junction	4.0	Orsett Cock	132552.0
4	{u'type': u'Point', u'coordinates': [0.525067...	Point	[0.525067...	57a4f3c0fa...	Feature	{u'ref': u'19', u'type': u'motorway_junction', u'id': 5....	19	motorway_junction	5.0	Boreham	132619.0
5	{u'type': u'Point', u'coordinates': [0.528036...	Point	[0.528036...	57a4f3c0fa...	Feature	{u'ref': u'18', u'type': u'motorway_junction', u'id': 6....	18	motorway_junction	6.0	Sandon	132627.0
6	{u'type': u'Point', u'coordinates': [-0.19942...	Point	[-0.19942...	57a4f3c0fa...	Feature	{u'ref': None, u'type': u'turning_circle', u'id': 7.0, u'n...	None	turning_circle	7.0	None	199229.0
7	{u'type': u'Point', u'coordinates': [-0.09140...	Point	[-0.09140...	57a4f3c0fa...	Feature	{u'ref': None, u'type': u'motorway_junction', u'id': 8...	None	motorway_junction	8.0	Great Cambridge Road Roundabout	196474.0
8	{u'type': u'Point', u'coordinates': [0.369823...	Point	[0.369823...	57a4f3c0fa...	Feature	{u'ref': None, u'type': u'motorway_junction', u'id': 9...	None	motorway_junction	9.0	Halfway House	134031.0
9	{u'type': u'Point', u'coordinates': [0.142398...	Point	[0.142398...	57a4f3c0fa...	Feature	{u'ref': None, u'type': u'motorway_junction', u'id': 1...	None	motorway_junction	10.0	Goresbrook Interchange	132496.0
10	{u'type': u'Point', u'coordinates': [0.079918...	Point	[0.079918...	57a4f3c0fa...	Feature	{u'ref': u'5', u'type': u'motorway_junction', u'id': 11....	5	motorway_junction	11.0	None	134116.0
11	{u'type': u'Point', u'coordinates': [-0.17729...	Point	[-0.17729...	57a4f3c0fa...	Feature	{u'ref': u'24', u'type': u'motorway_junction', u'id': 1...	24	motorway_junction	12.0	Potter's Bar Interchange	195870.0
12	{u'type': u'Point', u'coordinates': [0.191647...	Point	[0.191647...	57a4f3c0fa...	Feature	{u'ref': None, u'type': u'motorway_junction', u'id': 1...	None	motorway_junction	13.0	Ferry Lane	132510.0
13	{u'type': u'Point', u'coordinates': [0.039995...	Point	[0.039995...	57a4f3c0fa...	Feature	{u'ref': u'26', u'type': u'motorway_junction', u'id': 1...	26	motorway_junction	14.0	Honey Lane	197761.0
14	{u'type': u'Point', u'coordinates': [-0.22943...	Point	[-0.22943...	57a4f3c0fa...	Feature	{u'ref': u'1', u'type': u'motorway_junction', u'id': 16....	1	motorway_junction	16.0	Staples Corner	202077.0
15	{u'type': u'Point', u'coordinates': [-0.22945...	Point	[-0.22945...	57a4f3c0fa...	Feature	{u'ref': None, u'type': u'motorway_junction', u'id': 1...	None	motorway_junction	15.0	None	197446.0
16	{u'type': u'Point', u'coordinates': [-0.56475...	Point	[-0.56475...	57a4f3c0fa...	Feature	{u'ref': None, u'type': u'motorway_junction', u'id': 1...	None	motorway_junction	18.0	None	194178.0
17	{u'type': u'Point', u'coordinates': [-0.33451...	Point	[-0.33451...	57a4f3c0fa...	Feature	{u'ref': u'Perivale', u'type': u'motorway_junction', u'i...	Perivale	motorway_junction	17.0	None	197271.0
18	{u'type': u'Point', u'coordinates': [-0.05899...	Point	[-0.05899...	57a4f3c0fa...	Feature	{u'ref': u'25', u'type': u'motorway_junction', u'id': 1...	25	motorway_junction	19.0	Great Cambridge Road Junction	195967.0
19	{u'type': u'Point', u'coordinates': [0.023240...	Point	[0.023240...	57a4f3c0fa...	Feature	{u'ref': u'26', u'type': u'motorway_junction', u'id': 2...	26	motorway_junction	20.0	Honey Lane	197754.0

Figure 8 The attribute table of the "transport_points2" layer

The "neighborhoods" and "restaurants" collections were successfully imported as shown in Figure 9.

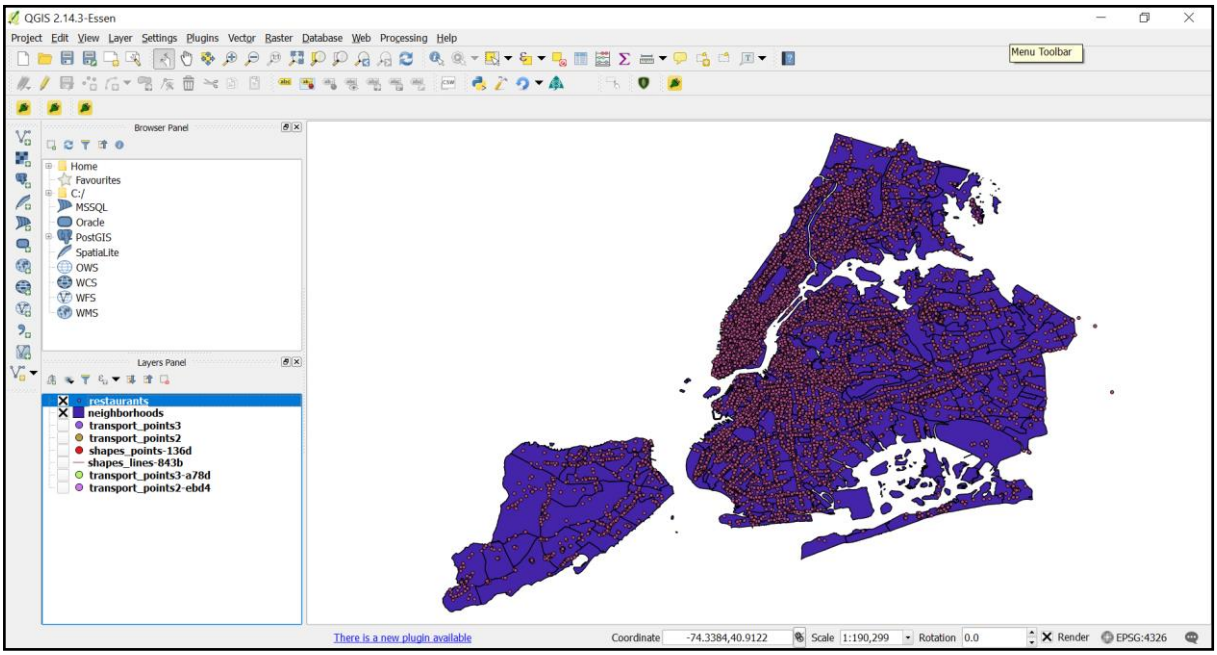


Figure 9 The "restaurants" and "neighborhoods" layers created by "Load MongoDB Layers" representing the "restaurants" and "neighborhoods" collections respectively

4. Development of the "Save Layer in MongoDB" plugin

In order to enhance the interfaces between MongoDB and QGIS, the "Save Layer in MongoDB" Python plugin was developed. The plugin allows users to store features from a vector layer in QGIS to a local MongoDB server as GeoJSON objects. Its interface, which is shown in Figures 2 and 5, was designed in Qt Creator. The plugin requires the Pymongo module to be installed and can be accessed from: https://github.com/VasiliosKalogirou/Save-layer-in-MongoDB.

4.1. Testing the plugin

To test the plugin's functionality, the "london_england_aeroways.geojson", "london_england_transport_points.geojson", and "london_england_transport_areas.geojson" files were imported in QGIS. In order for the plugin to connect to the database server, an instance of the server must be activated. If the server is offline the message shown in Figure 10 is generated by the plugin. The same message is shown if the user types the server name or port number incorrectly.

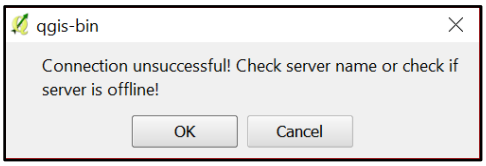


Figure 10 Connection error message generated by "Save Layer in MongoDB"

Using the plugin, all three layers were successfully stored in the "london" database in MongoDB as "aeroways", "transport_points", and "transport_areas" collections respectively. To test the plugin's functionality over storing a layer's selected features, a subset of 109 features were selected from the transport points layer as shown in Figure 11. Figure 12 presents the plugin interface before storing the selected features in the "transport_points_selected" collection. Figure 13 presents the message generated by the plugin.

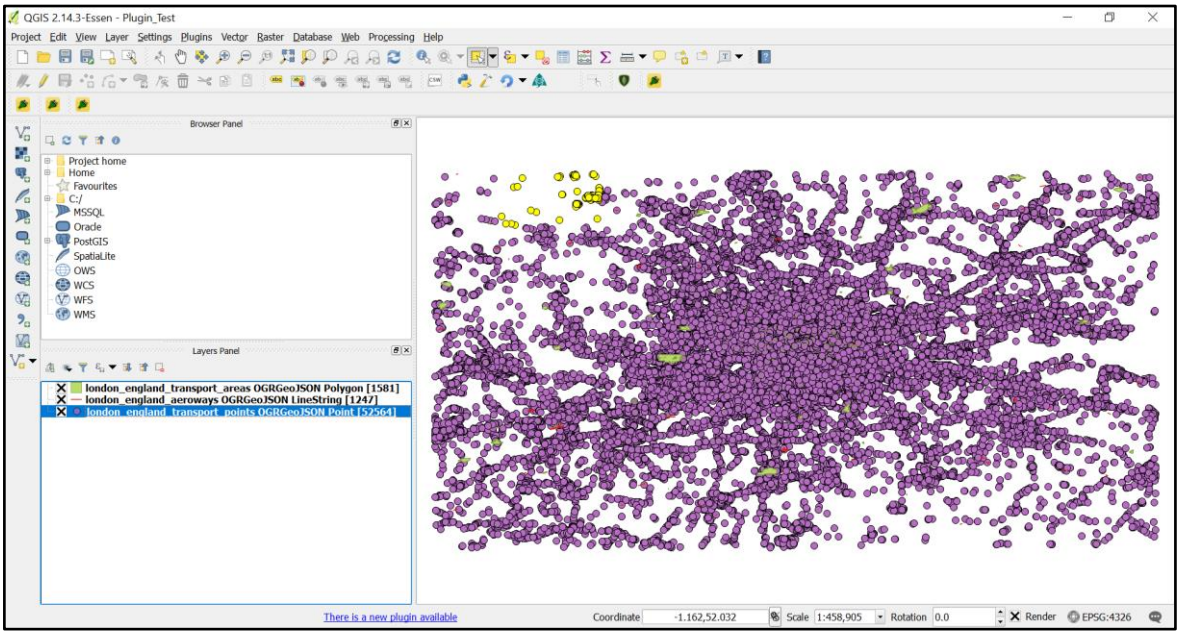


Figure 11 Transport points layer selected features

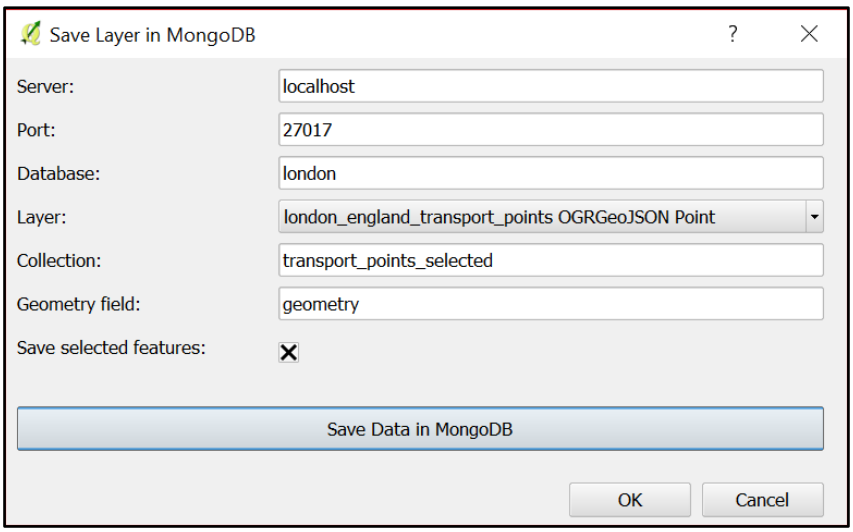


Figure 12 "Save Layer in MongoDB" interface - Storing selected features in the "transport_points_selected" collection

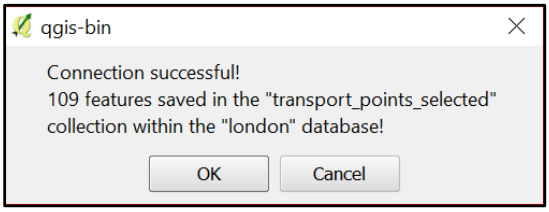


Figure 13 Message generated by "Save Layer in MongoDB" for successfully storing selected features in the "transport_points_selected" collection

To check the data stored in MongoDB, the contents of the "london" database were examined within the Mongo Shell as shown below.

```
> use london
switched to db london

> show collections
aeroways
transport_areas
transport_points
transport_points_selected

> db.aeroways.count()
1247

> db.transport_areas.count()
1581

> db.transport_points.count()
52564

> db.transport_points_selected.count()
109

> db.transport_points.findOne()
{
        "_id" : ObjectId("57c1becff345db1b28ab19c2"),
        "geometry" : {
                "type" : "Point",
                "coordinates" : [
                        -0.241885673740143,
                        51.606101317886576
                ]
        },
        "properties" : {
                "ref" : "2",
                "type" : "motorway_junction",
                "id" : 1,
                "name" : "Five Ways Corner",
                "osm_id" : 104439
        }
}
```

5. Conclusions and Further Work

This paper explored and evaluated MongoDB's spatial functionality and the interfaces between MongoDB and QGIS. Although MongoDB and NoSQL systems in general do not offer a large number of spatial capabilities, they can be used in geospatial applications where complex analysis functions are not required. In order to further enhance the interfaces between the two systems this paper introduces a plugin which enables users to store vector data from QGIS in MongoDB.

The plugin can be used as part of a collective effort by the community aiming at the development of an interface which will maintain a connection between MongoDB and QGIS, similar to the functionality offered by "DB Manager" for ORDBMSs. By achieving this, users will be able to visualise, query, manipulate, and analyse geospatial data using QGIS while the data is maintained in the database.

6. Biography

Vasilios is a graduate in Geographical Information Science from University College London. He also holds a bachelor's degree in Surveying Engineering and Geomatics from Cyprus University of Technology. His main interests are spatial databases and development of geospatial applications.

7. References

Chodorow, K., 2013. *Mongo DB: The Definitive Guide*. *Mongo DB: The Definitive Guide*.

MongoDB Inc, 2016. *Geospatial Indexes and Queries — MongoDB Manual 3.2*. [online] Available at: <https://docs.mongodb.com/manual/applications/geospatial-indexes/>.

MongoDB Inc., 2016a. *$type — MongoDB Manual 3.2*. [online] Available at: <https://docs.mongodb.com/manual/reference/operator/query/type/#document-querying-by-data-type>.

MongoDB Inc., 2016b. *2dsphere Indexes — MongoDB Manual 3.2*. [online] Available at: <https://docs.mongodb.com/manual/core/2dsphere/>.

MongoDB Inc., 2016c. *Find Restaurants with Geospatial Queries — MongoDB Manual 3.2*. [online] Available at: <https://docs.mongodb.com/manual/tutorial/geospatial-tutorial/>.

MongoDB Inc., 2016d. *SQL to MongoDB Mapping Chart — MongoDB Manual 3.2*. [online] Available at: <https://docs.mongodb.com/manual/reference/sql-comparison/>.

Näsholm, P., 2012. *Extracting Data from NoSQL Databases A Step towards Interactive Visual Analysis of NoSQL Data*. [online] Available at: <http://publications.lib.chalmers.se/records/fulltext/155048.pdf>.

Robinson, I.,, Webber, J., and Eifrem, E., 2015. *Graph databases : new opportunities for connected data*. Second edi ed. [online] *O'REILLY*. Sebastopol, CA : O'Reilly. Available at: <http://graphdatabases.com>.

Sadalage, P., and Fowler, M., 2013. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*.

Steiniger, S., and Hunter, A.J.S., 2012. The 2012 free and open source GIS software map - A guide to facilitate research, development, and adoption. *Computers, Environment and Urban Systems*, [online] 39, pp.136–150. Available at: <http://dx.doi.org/10.1016/j.compenvurbsys.2012.10.003>.