SHOULD EU LAND USE AND LAND COVER DATA BE MANAGED WITH A NOSQL DOCUMENT STORE?

J.T. NAVARRO-CARRIÓN[1], B. ZARAGOZÍ[1], A. RAMÓN-MORTE[1] & N. VALCÁRCEL-SANZ[2]
[1]Instituto Interuniversitario de Geografía, Universidad de Alicante, Spain.
[2]Geodesy and Cartography Department, National Geographic Institute of Spain.

ABSTRACT

Land cover (LC) is a scientific landscape classification based on physical properties of earth materials. This information is usually retrieved through remote sensing techniques (e.g. forest cover, urban, clay content, among others). In contrast, Land use (LU) is defined from an anthropocentric point of view. It describes how a specific area is used (e.g. it is usual to indicate whether a territory supports an intensive, extensive use or it is unused). Both geospatial layers are essential inputs in many socio-economic and environmental studies. The INSPIRE directive provides technical data specifications for harmonization and sharing of voluminous LU/LC datasets across all countries of the EU. The INSPIRE initiative proposes Object-Oriented Modelling as a data modelling methodology. However, the most used Geographic Information Systems (GIS) are built upon relational databases. This may jeopardize LU/LC data usability, since GIS practitioners will eventually face the object-relational impedance mismatch. In this paper, the authors introduce the SIOSE database (Spanish Land Cover and Land Use Information System), which was the first implementation of an object-oriented land cover and Land-use datamodel, in line with the recommendation of the INSPIRE Directive, separating both themes. SIOSE data can be downloaded as relational database files, where information describing each single LU/LC object is divided among several related tables, so database queries can be complex and time consuming. The authors show these technical complexities through a computational experience, comparing SQL and NoSQL databases for querying spatial data downloaded from SIOSE. Finally, the authors conclude that NoSQL geodatabases deserve to be further explored because they could scale for LU/LC data, both horizontally and vertically, better than relational geodatabases, improving usability and making the most of the EU harmonization efforts.

Keywords: Land Use, Land Cover, document store, SIOSE, geodatabase, PostgreSQL, jsonb

1 LAND USE AND LAND COVER DATABASES IN THE EU

Land is a limited resource and its mismanagement is one of the main drivers of global change, with significant effects on ecosystem functions, goods and services [1]. There are complex environmental problems such as the over- exploitation of natural resources, biodiversity loss or climate change that require a long-term management perspective of natural resources. Many studies agree that these problems can be aggravated by land-use changes, so it is mandatory to monitor and apply long-term management policies at different scales [2].

Land use (LU) and land cover (LC) information at national and regional level has been historically recorded in many EU Member States because of their environmental and territorial management's needs and requirements. In addition to Corine Land Cover (CLC) 1990 databases, many EU countries have been producing LC databases to manage and satisfy their requirements on environmental, agricultural, forest and land planning issues. As a consequence, there are several regional and national LU/LC inventories with very different data collection methods, scales, nomenclatures, Minimum Mapping Units, and different production and update intervals [2].

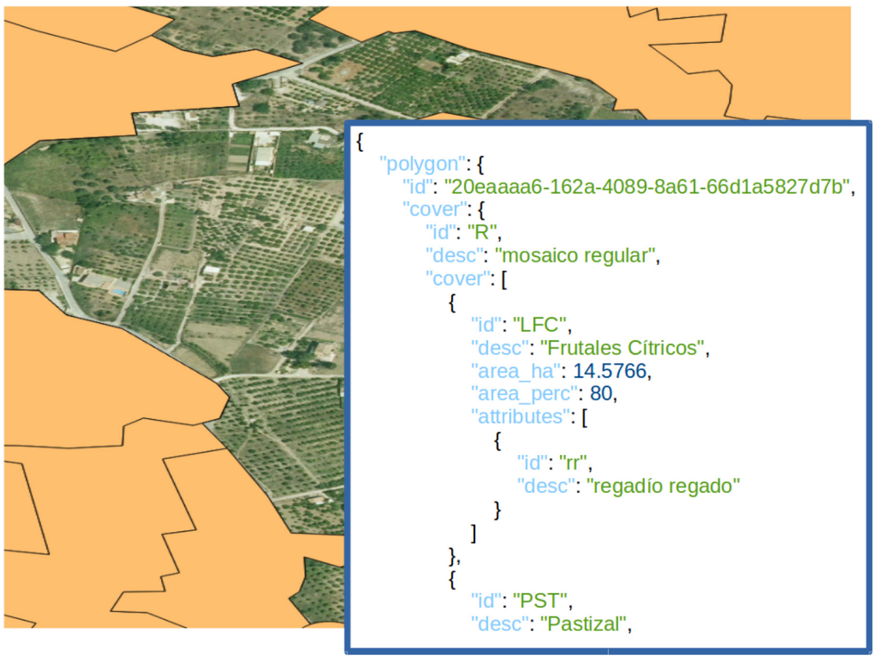
This paper is part of the Proceedings of the International Conference on Big Data (Big Data 2016)
www.witconferences.com

ISSN: 1755-7437 (paper format), ISSN: 1755-7445 (online), http://www.witpress.com/journals
DOI: 10.2495/DNE-V11-N3-438-446

The need for better harmonization between national and European data sets and the intention of avoiding redundant data production, has led many of these countries to use their national data to derive European scale data sets, such as CLC or LUCAS, following a ‘bottom-up’ approach [2]. Simultaneously, the information flow generated by these national developments needs to be integrated with other European ‘top-down’ land monitoring activities, such as Copernicus, which is the European Programme for the establishment of a European capacity for Earth Observation [3].

The EAGLE group was set up by the members of the Environmental Information and Observation Network (EIONET) on land cover in response to the growing need to discuss technical solutions for a better integration and harmonization of national mapping activities with European land monitoring initiatives. The objective of the working group is to elaborate a conceptual solution for land monitoring built on national data sources combined with pan-European information layers [3]. The EAGLE data model uses an Object-Oriented Data Modelling (OODM) approach, which takes into account existing standards or code lists, such as CLC, LUCAS, EUNIS as well as INSPIRE (2007/2/EC) data specifications and ISO standard 19144-2 (LCML-Land Cover Meta Language). Currently, the development of EAGLE concept and methodology is being funded by the European Environmental Agency under the framework of Copernicus program.

The National Geographic Institute of Spain (IGN), a member of the EAGLE group, created the Land Cover and Use Information System of Spain (SIOSE) as part of the National Land Monitoring Plan, which aims to achieve a multidisciplinary Spatial Data Infrastructure, periodically updated, for the Spanish national and regional administrations. The SIOSE database conforms with the INSPIRE data specifications and has been designed as an OODM, similar to the one proposed by the EAGLE group, ensuring backward compatibility and comparability with pre-existing databases like CLC90, CLC00, Murbandy/Moland, UN FAO LCCS, among others. However, in practice, the OODM is



```
{
  "polygon": {
    "id": "20eaaaa6-162a-4089-8a61-66d1a5827d7b",
    "cover": {
      "id": "R",
      "desc": "mosaico regular",
      "cover": [
        {
          "id": "LFC",
          "desc": "Frutales Cítricos",
          "area_ha": 14.5766,
          "area_perc": 80,
          "attributes": [
            {
              "id": "rr",
              "desc": "regadío regado"
            }
          ]
        },
        {
          "id": "PST",
          "desc": "Pastizal",
```

Figure 1: LU/LC example of object-oriented classification in JSON.

adapted and implemented into relational or object-relational database management systems with spatial capabilities, and database managers have to deal with incompatibilities at the conceptual level. This is a case of the object-relational impedance mismatch, and has been clearly identified in literature as a problem of data structure due to paradigm differences [4]. An example of LU/LC classification of a single parcel is shown in Fig. 1. The amount of semi-structured information to be stored for each LU/LC polygon adds some difficulties for managing the SIOSE LU/LC information through well-known GIS or relational geodatabases, so other technological alternatives might be explored. Nowadays, the SIOSE database is accessed via standard web mapping services, GIS file downloads and, in some cases, it is also distributed as serialized XML.

1.1 Research goals

The computational experience presented in this paper is considered a preliminary work. We basically conducted a peer-to-peer query performance test where LU/LC queries were run against the SIOSE relational model and compared with their translations run against a document-oriented derivative. Every query included a bounding box search clause and run iteratively using grids with varying cell size (Table 1). Both models were implemented in twin PostgreSQL/PostGIS instances. By constraining the experiment to a common DBMS we expected: (i) to get comparable response time and throughput figures; (ii) to obtain results not distorted by different implementations of spatial access methods; (iii) to get hints on how LU/LC hierarchy structure influences query performance; (iv) to ascertain query qualification categories for which the document-oriented approach should be considered.

2 EXPERIMENTAL SETUP

In order to initially answer the research question asked in the title of this paper, we chose to compare spatial query performance using different data type storage options for LU/LC observations within the same environment rather than measuring differences in performance among several database engines. PostgreSQL was selected to accomplish this task since it (i) provides an extensible type

Table 1: Database tables used for benchmark.

Type	Tables	#Rows	Total Size	External Size
Relational + Lookup	siose_values	10,435,032	3,160 MB	1,522 MB
	siose_polygons	2,477,593	6,456 MB	1,948 MB
	siose_coverages	116	48 kB	40 kB
	siose_attributes	26	40 kB	32 kB
	TOTAL	**12,912,625**	**9,616 MB**	**3,470 MB**
Document store	**docstore_jsonb**	**2,477,593**	**8,066 MB**	**2,615 MB**
Grids	spain_grid_10k	46,088	7,568 kB	40 kB
	spain_grid_25k	7,646	1,296 kB	40 kB
	spain_grid_50k	2,027	376 kB	40 kB
	spainvgrid_100k	568	136 kB	40 kB
	spain_grid_200k	171	72 kB	40 kB
	spain_grid_500k	42	48 kB	40 kB
	spain_grid_1m	15	48 kB	40 kB

system, (ii) implements the OGC's Simple Feature for SQL specification using the PostGIS extension – providing types, functions and access methods for geographic data –, (iii) allows for a formal representation of SIOSE's relational model, (iv) provides a binary JSON data type and operators to manage document-oriented models [5] and (v) generates query plans using a common relational query processor. Therefore, SQL spatial queries against relational and document-oriented models can be consistently compared to each other in terms of processing cost.

2.1 Data loading

As of this writing, there are two SIOSE datasets publicly available for download, corresponding to the compilation campaigns of 2005 and 2011. The SIOSE 2005 dataset was selected since, at the time of testing, it was the only one that covered the whole country. This dataset collects more than 10.4 million soil occupation observations for roughly 2.5 million polygon geometries. SIOSE data were obtained from the Spanish National Center for Geographical Information download site. Data for a particular year is organized as a series of ZIP archives. Each archive covers an administrative region or subregion and contains an ESRI Shapefile with polygon geometries and a Microsoft Jet MDB file with LU/LC observations. As to raw data preparation, a set of bash scripts (https://github.com/labgeo/siose2postgis) were written to automate the initial process of loading the SIOSE archives into a PostgreSQL/PostGIS database. This job was accomplished on a commodity computer running Ubuntu 14.04 and one PostgreSQL/PostGIS Docker container. The resulting database was dumped into a plain SQL script file using the common pg_dump utility. For the binary JSON model test, several scripts were prepared to transform the whole relational database into a set of JSON documents (Fig. 1). These JSON documents are created as a direct translation of the XML files prepared by the IGN for very particular purposes.

2.2 Use case

For this computational experiment, our intent was measuring workloads across the whole SIOSE dataset in a similar fashion to the use case of a web map monitoring session where, in order to reload its data, a reactive dashboard listens to range query events on a map view. This use case is in line with the concept of the map browsing macro benchmark scenario within the Jackpine spatial database benchmark methodology [6], which consists of a series of queries fetching geometries inside bounding boxes. Covering the study area in full was necessary so that the benchmark is comprehensive in terms of reflecting LU/LC regional variability. In order to meet this requirement, a tessellation function was written to build graticule macro scenarios overlapping Spain's mainland and islands at seven different scales or levels of detail, ranging from 1:10,000 to 1:1,000,000. Finally, a set of six bounding box search queries were prepared with three categories of qualifiers, namely polygon selection, aggregation and reclassifying conditions (Table 2). Grid cells played as arrays of predefined bounding boxes, each cell being visited three times per query plus one initial warming up iteration.

2.3 Test environment

The PostgreSQL query planner itself was used as the means of benchmarking. Query planners or optimisers have been formally defined as the component of the relational query processor which is in charge of mapping the set of logical operators in a DML statement syntax tree to an optimal or suboptimal graph of physical operators driving data flow [7, 8]. In the case of PostgreSQL, the internals of these diagrams, referred to as query plans, have been explained in great detail by source code

Table 2: Bounding box search benchmark queries.

LU/LC Condition	Query ID	Description
cover equals	coniferous	Select polygons with coniferous cover.
cover equals AND area greater than	large_coniferous	Select polygons with coniferous coverage greater than 1 Ha.
attribute equals OR attribute equals	reforested	Select polygons with forest coverage originating from plantation or agricultural abandonment.
cover equals AND parent(cover) equals	scattered_urb	Select polygons with scattered urbanisation coverage.
IF cover equals THEN sum(area)	area_coniferous	Sum all areas of coniferous plantations.
IF cover equals THEN reclass(area percent-age)	reclass	Reclassify all polygons into 4 density class categories based on conifer percentage (0%–25%, 25%–50%, 50%–75% and 75%–100%). Discard polygons with no coniferous cover.

reviewers [8] and are the primary resource for query performance monitoring. To sum up, a query plan is generated by issuing the EXPLAIN command, typically over a SELECT statement, and depicted as a binary tree of access paths. Each node in the tree represents an access method for scanning a relation (sequentially or by index) or an algorithm for joining a pair of relations. Nodes also have an associate processing cost. Queries with a low number of joins are optimised using an exhaustive search strategy, so the resulting query plans shall be considered optimal provided that statistical information involved with processing cost estimates is regularly fed to the system catalogue using the VACUUM and ANALYZE maintenance commands. As to this computational experiment, the query plans output by the EXPLAIN ANALYZE command were recorded for each query (6), grid cell (56,557) and iteration (4). We ultimately focused on actual execution times in order to assess query performance, although there are other metrics, such as the number of nodes, the number of rows processed or the accumulated processing cost carried over each node, which are also relevant for benchmarking and may well be used to consistently compare between query plans originated within each data model (relational and JSONB) and whose result set is equivalent.

2.4 Server instance replication and database deployment

Each PostgreSQL server instance was executed in isolation using containers running on Docker Engine 1.9. As highlighted in an analysis of the 'DevOps' approach to computational experiments replication, Boettiger [9] demonstrates how Docker container technology fulfills the requirements of software dependency resolution, precise documentation and portability by means of simple scripting, composition and light-weight image binaries sharing the host machine kernel. In order to ensure reproducibility of the computational environment used for the tasks described in this paper, we are providing a Dockerfile (https://github.com/labgeo/postgresql-9.5-postgis-2.1) which allows creating a PostgreSQL 9.5 image, including the PostGIS 2.1 extension for geospatial data storage.

A deployment utility (https://github.com/labgeo/deploy-db) was developed to automate launching database instances on the server. This bash script can be executed locally to run a PostgreSQL

Docker container, create a PostGIS database and subsequently restore the complete SIOSE dataset using the above mentioned SQL script file. Two database instances were generated so that we could effectively rely on isolated computational environments to compare query performance between the relational and the document-oriented models.

2.5 Schema setup

Several set-up scripts (https://github.com/labgeo/pg_siose_bench) were developed and documented. The LU/LC observations table stores adjacency lists to model the coverage hierarchy of each polygon, but the parent fields contain the full path of ancestor identifiers as comma-separated strings instead of the direct ancestor's identifier. The one to many relationship between a coverage and its attributes is also stored as a list of strings. During pre-processing, these lists were converted to one-dimensional arrays, thus rendering a 1NF compliant model whose data is accessible through Generalized Inverted Indexes (GIN) and array operators. Additional pre-processing scripts were used to rename identifiers, remove unneeded fields and build grid scenarios. As to indexing, only general indexes were built: BTree on scalar fields, GIN on array and binary JSON fields and Generalized Search Tree (GiST) on polygon geometry. No multi-column or functional indexes were created. This preserves flexibility since the choice of any particular index is always handed over to the PostgreSQL optimiser when processing WHERE clauses having multiple conditions. In summary, starting from two SIOSE raw geodatabase instances, the set-up process resulted in (i) a normalised geodatabase with a polygons table and an LU/LC observations table (relational model) and (ii) a second instance with a single polygons table where LU/LC observations for each polygon are stored as binary JSON values (document-oriented model). As part of the schema set-up, all LU/LC queries were defined as prepared statements and encapsulated, along with the benchmark log commands, within support functions.

3 BENCHMARK RESULTS

Firstly, the comprehensive series of scripts that were developed make themselves a first relevant result, since they guarantee the reproduction of the research by others in an automated fashion.

As to final test results, benchmarks were carried out on a Debian 8 server with kernel 3.16.0-4-amd64, Intel Xeon E5-2630 v2 CPU, 512 GiB SSD and 62 GiB RAM. Given a 4 iterations per grid cell, a total of 226,228 iterations per bounding box search query were run on each PostgreSQL server instance. The whole query stack took 3.58 hours to complete on the relational model. Completion time in the document-oriented model was 2.91 hours that is roughly a 19% faster. Benchmarking implied recording every single iteration query plan in a log table, which was in turn indexed to efficiently get measurements on response and throughput, namely averages by execution time and, as a byproduct, polygons per millisecond. Table 1 shows the database tables used in this computational experience, with their cardinality, total file size and external file size (kB or MB). Approximately, 'external size' corresponds the memory occupied by indexes, while 'total size' is equal to the sum of the data size and the 'external size'. There are three types of tables: (i) tables storing regular grids at 7 different scales where each record corresponds to a cell grid, (ii) tables containing the SIOSE relational database, optimised for PostgreSQL as explained in Subsection 2.5. There is one table (siose_polygons) storing the SIOSE 2005 geometries, a second table (siose_values) storing the LU/LC values associated with those polygons and two lookup tables containing LU/LC descriptions for the LU/LC codes in siose_values. (iii) Finally, docstore_jsonb stores the complete SIOSE database in one single table, very similar to siose_polygons, but with an additional binary JSON column storing the LU/LC model for each polygon (see Section 1). It can be seen that the relational database uses 5,2 times more rows than the document

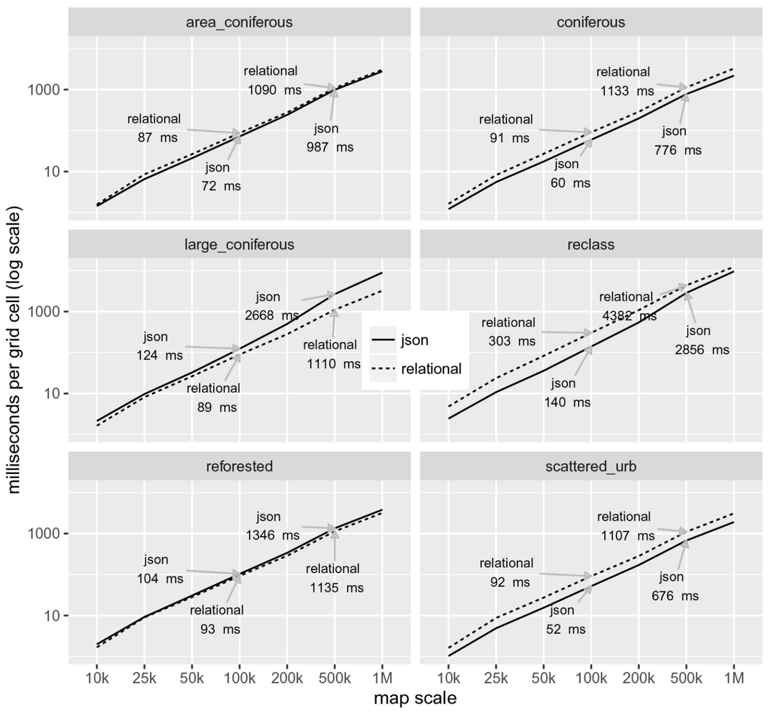


Figure 2: Binary JSON vs relational query response times.

store for the same information. It is also noteworthy that the relational model spends more disk space in both, tables and indexes.

Response time charts and throughput boxplots were composed using query plan execution times and number of polygons processed per millisecond, respectively. Charted values refer to average per query and grid. Comparative line charts in Fig. 2 show that queries by cover (query identifiers ‘coniferous’ and ‘scattered_urb’) render better response times on the document-oriented model. In fact, these queries scored maximum throughput and the greatest performance gains with regard to their siblings in the relational model as shown in Fig. 3. The reclassification query performs better on the document-oriented model, particularly at the 1:100,000 scale, where it runs twice as fast. However, response time gains tend to shrink at lower levels of detail. Differences in response and throughput are less significant in the aggregation (query id ‘area_coniferous’) and LU/LC attribute (query id ‘reforested’) queries. On the other hand, response times of the query by cover and LU/LC (query id ‘large_coniferous’) increase progressively at lower levels of detail on the document-store model, so much that overall performance is clearly better on the relational model.

4 DISCUSSION

Considering the exploratory nature of this experiment and the observed results, the answer to the research question is that there are common workflows for which a document-oriented model should be seriously taken into consideration. Massive polygon retrievals based on land coverage presence or absence seems an optimal use case. Reclassification operations may also benefit from a binary

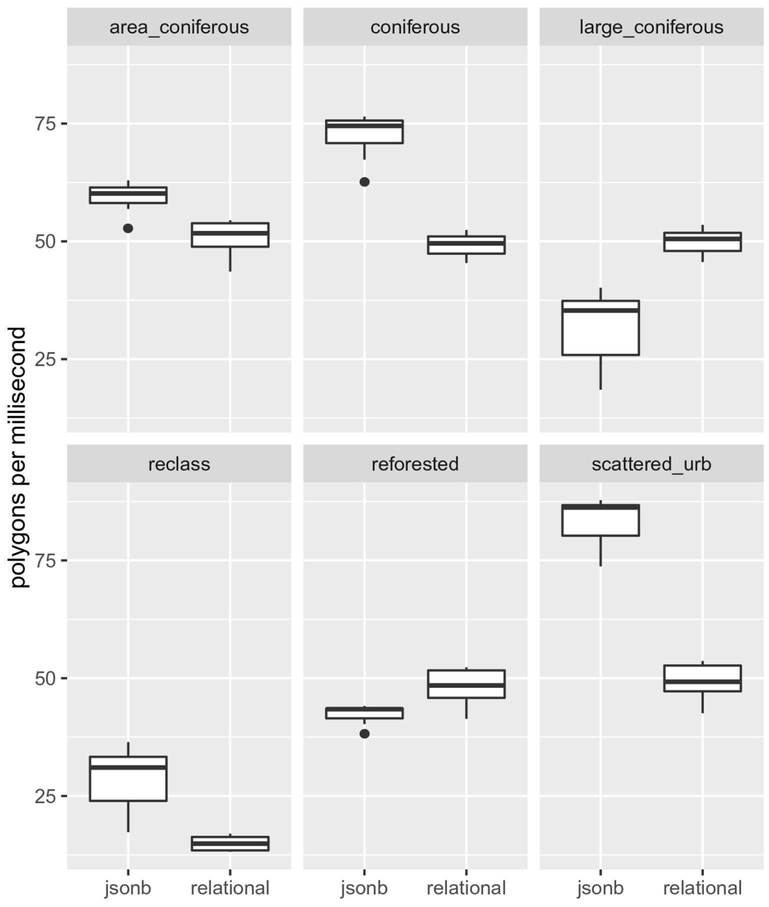


Figure 3: Binary JSON vs. relational query throughput

JSON implementation, although real applications such as those mentioned in Section 1 should be thoroughly tested (e.g. derive CLC following a 'bottom-up approach'). In contrast, inequality expressions on LU/LC numeric attribute values cannot make use of the GIN index on the binary JSON field. As seen in Section 3, this problem gets particularly exacerbated in the query by cover and area (query id 'large_coniferous'), which performs progressively slower at smaller scales (more polygons to process per grid cell) on the binary JSON model. The throughput of this query on the document-oriented model is remarkably poor considering its relative simplicity. Drastically improving performance on inequality expression evaluation requires an alternative approach for the document-oriented model, which most probably involves resorting to functional BTree indexes. Another issue that emerged during the devise of the experiment was the somewhat convoluted syntax of the JSON queries. This is a consequence of the deeply nested structure of the LU/LC JSON documents and the lack of native DOM operators. Different flattening strategies of the original JSON schema should be investigated in order to assess on tackling this issue and measure influence in performance. In the end, this computational experiment serves as a starting point to verify that, while the relational model is probably more reliable for OLTP, building LU/LC data marts upon binary JSON may boost Big Data applications not feasible otherwise. Finally, the computational experiment presented in this paper opens up research in scopes such as index optimisation and efficient search inside JSON documents.

REFERENCES

[1] Roces-Díaz, J.V., Díaz-Varela, E.R. & Álvarez-Álvarez, P., Analysis of spatial scales for ecosystem services: application of the lacunarity concept at landscape level in Galicia (NW Spain). *Ecological Indicators*, **36**, pp. 495–507, 2014. http://dx.doi.org/10.1016/j.ecolind.2013.09.010

[2] Manakos, I. & Braun, M., *Land Use and Land Cover Mapping in Europe*, Springer London, 18, p. 411, 2014.

[3] Arnold, S., Kosztra, B., Banko, G., Smith, G., Hazeu, G. & Bock, M., The eagle concept a vision of a future European land monitoring framework. *EARSeL Symposium Proceedings 2013, "Towards Horizon 2020"*, 2013.

[4] Ireland, C., Bowers, D., Newton, M. & Waugh, K., A classification of object-relational impedance mismatch. *Proceedings – 2009 1st International Conference on Advances in Databases, Knowledge, and Data Applications, DBKDA*, pp. 36–43, 2009. http://dx.doi.org/10.1109/dbkda.2009.11

[5] Bartunov, O. & Sigaev, T., Binary storage for nested data structures and application to hstore data type. PostgreSQL Conference Europe 2013 Tals, 2013. Available at: http://www.sai.msu.su/~megera/postgres/talks/hstore-dublin-2013.pdf.

[6] Ray, S., Simion, B. & Demke Brown, A., Jackpine: a benchmark to evaluate spatial database performance. *Proceedings International Conference on Data Engineering*, pp. 1139–1150, 2011. http://dx.doi.org/10.1109/icde.2011.5767929

[7] Hellerstein, J.M., Stonebraker, M. & Hamilton, J., Architecture of a database system. *Foundations and Trends in Databases*, **1**(2), pp. 141–259, 2007. http://dx.doi.org/10.1561/1900000002

[8] Wang, J., Li, J. & Butler, G., Implementing the PostgreSQL query optimizer within the OPT++ framework. *Proceedings Asia-Pacific Software Engineering Conference, APSEC*, pp. 262–272, 2003.

[9] Boettiger, C., An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, **49**(1), pp. 71–79, 2015. http://dx.doi.org/10.1145/2723872.2723882