OLAP and NoSQL: Happily Ever After

OLAP and NoSQL: Happily Ever After

Stefano Rizzi[1][0000−0002−4617−217X]

DISI - Univ. of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
stefano.rizzi@unibo.it

Abstract. NoSQL databases are preferred to relational ones for storing heterogeneous data with variable schema and structure. However, their schemaless nature adds complexity to analytical applications, in which a single OLAP analysis often involves large sets of data with different schemas. In this tutorial we describe the main approaches to enable OLAP on NoSQL data. We start from schema-on-read approaches, where data are left unchanged in their structure until they are accessed by the user, so they are put into multidimensional form at query time. Specifically, we show how this enables a form of approximated OLAP that embraces the inherent variety of schemaless data. Then we move to schema-on-write approaches, where a fixed multidimensional structure is forced onto data, which are loaded into a data warehouse to be then queried. In particular, we introduce multi-model data warehouses as a way to store data in multidimensional form and, at the same time, let each piece of data be natively represented through the most appropriate NoSQL model.

Keywords: NoSQL databases · OLAP · Multi-model databases

1 Introduction and Motivation

In recent years, NoSQL databases have been progressively eroding the predominance of relational databases [17]. A NoSQL database provides a mechanism for storage and retrieval of data that is modeled differently from the tabular relations used in relational databases; the particular suitability of a given type of NoSQL database (key-value, columnar, document-based, or graph-based) depends on the busines problem it must address. Among the potential benefits of NoSQL databases, we mention better performance scaling, no ACID transactions, and no need for a unique schema. Indeed, NoSQL databases adopt a *schemaless* representation for data: schema is a "soft" concept and the instances referring to the same concept can be stored using different local schemas. Hence, these databases are preferred to relational ones for storing heterogeneous data with variable schemas and structural forms, such as those located in data lakes. Typical schema variants within a collection consist in missing or additional fields, in different names or types for a field, and in different structures for instances [15].

The growing use of NoSQL databases has resulted in vast amounts of semi-structured data holding precious information, which could be profitably integrated into existing business intelligence (BI) systems [1]. On-Line Analytical

Processing (OLAP) is the querying paradigm normally used in the context of BI to analyze data stored in data warehouses, and it has been recognized to be an effective way for running analytics over big NoSQL data as well [12]. The OLAP paradigm entails dynamic analyses that read a huge quantity of data to compute a set of numbers that quantitatively describe a given business phenomenon. It assumes that data follow the *multidimensional model* [18], whose main concepts are *facts* (i.e., business phenomena such as sales), *dimensions* (coordinates used to analyze a fact, e.g., store, product, and date), *measures* (quantitative attributes that describe fact occurrences, e.g., sales revenue), and *hierarchies* (sequences of attributes that group dimension members at increasing levels of aggregation). OLAP comes in sessions, i.e., sequences of queries each obtained from the other by applying one OLAP operator (mainly, roll-up, drill-down, and slice-and-dice). Unfortunately, although the absence of a unique schema in NoSQL data grants flexibility to operational applications, it adds complexity to OLAP applications, in which a single analysis often involves large sets of data with different (and often conflicting) schemas.

In this tutorial we explore the most promising directions for enabling OLAP analyses on NoSQL data, distinguishing between the two approaches that can be followed (see Figure 1 for an intuition): schema-on-write and schema-on-read [11]. *Schema-on-write* approaches force a (fixed) multidimensional structure in data, load them into a data warehouse using an ETL (Extract, Transform, and Load) process, then let these data be queried by users via OLAP tools. We discuss these approaches in Section 2. *Schema-on-read* approaches leave data unchanged in their structure until they are accessed by the user. The multidimensional schema is not devised at design time and forced in a data warehouse, but decided by every single user at querying time; clearly, this requires OLAP queries to be rewritten over NoSQL data sources. These approaches are the subject of Section 3. Finally, in Section 4 we draw the conclusions.

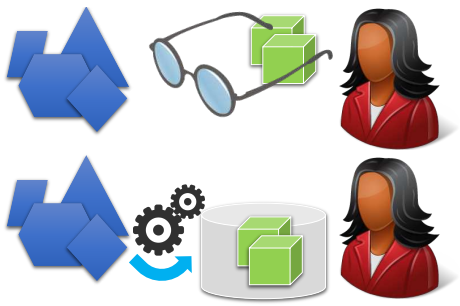

Fig. 1. In schema-on-read approaches (top), the user has a multidimensional view of data stored in their native (heterogeneous) form; in schema-on-write approaches (bottom), data are put into multidimensional form and stored

2 Schema-on-Read Approaches

In these approaches, source data are left unchanged in their own model and structure, to be directly queried in an OLAP fashion by the end-user without putting them in multidimensional form. Rather than being devised at design time, a multidimensional schema for accessing data is decided at querying time; while this requires OLAP queries to be rewritten over data sources on-the-fly and thus might give performance problems, it entails higher querying flexibility, simpler ETL, and lower effort for evolution. Schema-on-read approaches to enable OLAP on NoSQL data ground their roots into techniques for (i) schema discovery from XML/JSON documents, which deal with heterogeneity, quality, versioning, similarity, and comprehensiveness to produce unified schemas, schema matches, and skeleton schemas [3, 22, 24]; (ii) schema matching for XML/JSON documents using clustering or machine learning, in some cases considering a context [14, 5]; (iii) multidimensional design from XML/JSON/columnar data, possibly by detecting and chasing functional dependencies [13, 23].

In this tutorial we focus on two schema-on-read approaches, namely, Graph OLAP and Approximate OLAP; for other examples of schema-on-read approaches, see [2, 11, 19].

2.1 Graph OLAP

Given a graph-structured dataset, *Graph OLAP* [9] aims at returning a multidimensional view of it to enable efficient OLAP analyses. Source data are seen as a collection of *network snapshots*, each including some informational attributes (e.g., month and socialNetwork) and one graph (e.g., one where nodes are users and edges represent their interactions); both nodes and edges of this graph may be described by attributes (e.g., name is an attribute of user nodes, numberOfMessages is an attribute of collaboration edges). The multidimensional view of graph data provided by Graph OLAP relies on the two pillars of the multidimensional model, namely, dimensions and measures. Two types of dimensions are distinguished:

- *Informational dimensions* correspond to informational attributes and organize snapshots into groups based on different perspectives, where each group corresponds to a cube cell. Hierarchies can be defined on these dimensions, for instance, socialNetworks → all and month → year → all.
- *Topological dimensions* correspond to node/edge attributes and operate on individual network snapshots. Hierarchies can be defined on these dimensions too, e.g., user → nation → all.

As to measures, they are computed starting from numerical node/edge attributes by aggregating them in two different ways: (i) in *informational OLAP*, aggregation is done by grouping snapshots with identical values of informational dimensions; (ii) in *topological OLAP*, aggregation is done by grouping nodes with identical values of topological dimensions inside individual networks. An example is shown in Figure 2.

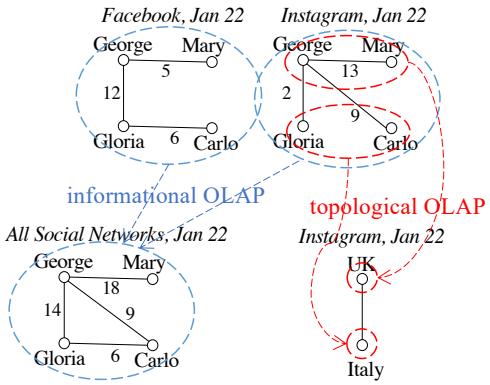

Fig. 2. Informational and topological OLAP in the Graph OLAP approach: in the first case, interactions are grouped on all social networks, in the second one, they are grouped by user's nation

2.2 Approximate OLAP

The basic idea of *Approximate OLAP* [16] is to enable multidimensional querying of document data with variable schemas, embracing data heterogeneity as an inherent source of information wealth in schemaless sources. Both inter-schema and intra-schema variety are considered; aimed at pursuing an inclusive approach to integration, OLAP querying is carried out on a "soft" schema where each source attribute is present to some extent. The approach encompasses four phases (see Figure 3 for an example):

1. *Schema extraction*, whose goal is to identify the set of distinct, tree-like local schemas that occur inside a collection of documents.
2. *Schema integration*, which relies on inter-schema mappings and schema integration techniques to determine a tree-like global schema that gives the user a single and comprehensive description of the contents of the collection.
3. *FD enrichment*. An OLAP-compliant multidimensional view of the document data is obtained from the global schema by building a *dependency graph*, i.e., a graph that represents functional dependencies between the document fields; these dependencies are either inferred from the structure of the schema or determined (in approximate form) by analyzing the documents.
4. *Querying*. Here, the user can formulate OLAP queries on the dependency graph and execute them on the documents. To this end, each query is translated to the query language of the underlying document-oriented DBMS and reformulated into multiple queries, one for each local schema in the collection; the results presented to the user are obtained by merging the results of the single local queries. To make users aware of the impact of schema variety, a set of indicators describing the quality and reliability of the query result are computed.

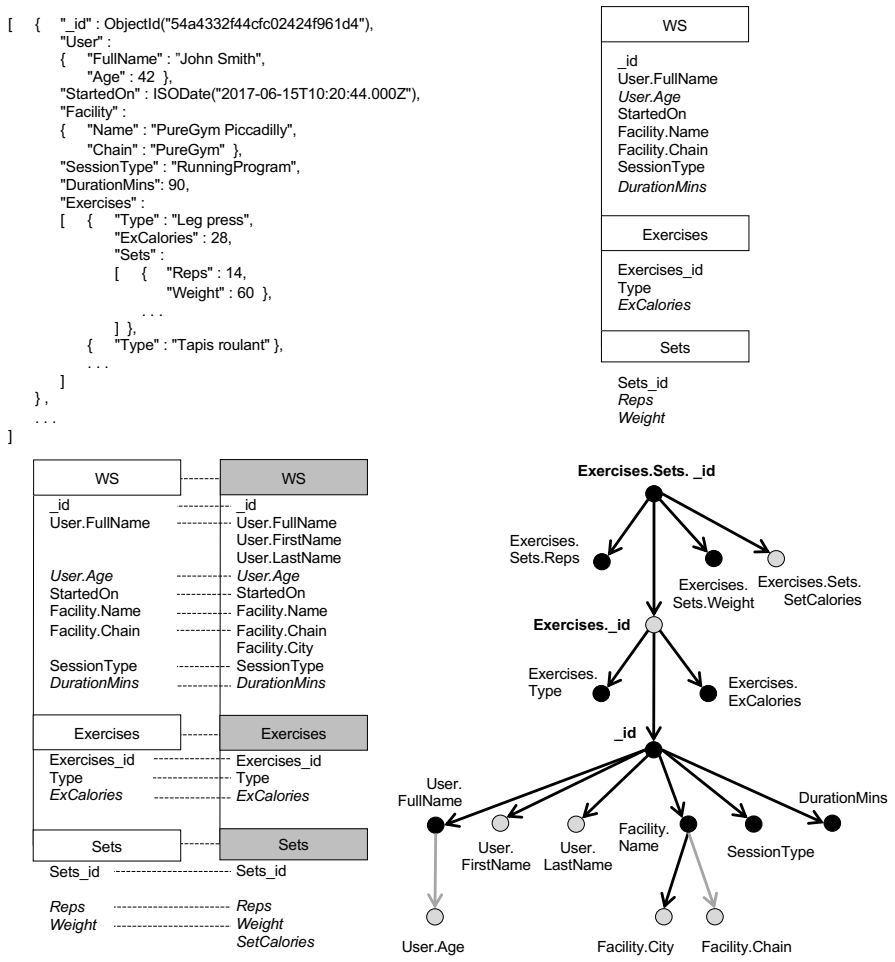

Fig. 3. Steps in approximate OLAP (adapted from [16]): a JSON document (top-left), its local schema (top-right, numerical fields in italica), its mappings with the global schema (bottom-left), and its dependency graph (bottom-right, grey arcs represent approximate functional dependencies discovered on documents)

3 Schema-on-Write Approaches

In these approaches, source data are moved into a data warehouse; this requires that they are put into multidimensional form to be then queried in an OLAP fashion by the user. The multidimensional schema is decided at design time and forced onto data at the time of writing them in the data warehouse, which entails better performances and simpler query formulation with no need for query rewriting. Schema-on-write approaches are based on the literature on (i) multidimensional design from NoSQL data [13, 23] and (ii) NoSQL data warehouses, that aim at storing warehoused data in document/columnar/graph form by following design guidelines.

In this tutorial we distinguish between *mono-model* approaches, in which multidimensional data are stored in the data warehouse according to a single model (e.g., document-based), and *multi-model* approaches, in which a multi-model DBMS is used to grant higher storage flexibility.

3.1 Mono-Model Approaches

Several examples of schema-on-write approaches targeting a single NoSQL model can be found in the literature.

We start with the document-based model, for which some papers have proposed and compared different solutions to multidimensional design. Specifically, four solutions are proposed in [10]: (i) a *denormalized flat schema* (where a fact is stored using a single collection of documents including all its measures and levels with no nesting); (ii) a *deco schema* (denormalized like the previous one, but the measures and the levels of each dimension are stored in separate subdocuments); (iii) a *shattered schema* (where each dimension is stored in a separate collection of documents and connected to the fact documents using a reference, see Figure 4 for an example); and (iv) a *hybrid schema* (like a shattered schema, but with all documents stored within a single collection). Based on experimental tests, it is argued that (i) the first two schemas require about 4 times the space required by the other two, which leads to significantly higher loading times; and (ii) denormalized flat schemas and shattered schemas tend to have better querying performances; however, there is not a single winner between these two since the execution times largely depend on the query features (mostly, on the number of joins they require). Similarly, two solutions are proposed in [8] and [28]: (i) a *simple schema* (where the fact and each dimension are stored in separate documents of the same collection, like in the hybrid schema mentioned above) and (ii) a *hierarchical schema* (like a simple schema, but using separate documents for each dimension hierarchy, much like the shattered schema mentioned above). The experimental comparison does not highlight significant differences in loading time and querying performance.

As to the graph-based model, in [26] two solutions are proposed. In the first one, the fact is stored in a graph node having measures as properties, and each level is stored in a node with its properties; the fact node points to the dimension nodes, which in turn point to the level nodes following the structure of the

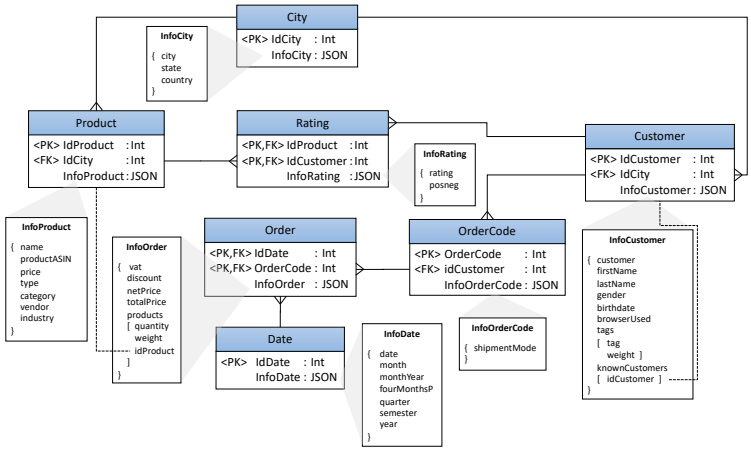

Fig. 4. A shattered schema

hierarchies (as also suggested in [7]). The second one is similar, except that the fact node points to a single node, which in turn points to each dimension node. A third solution is proposed in [27], where the fact node points to the dimension nodes, and each dimension node includes *all* the levels and properties of the corresponding hierarchy. Note that these three solutions are not experimentally compared in terms of efficiency.

Finally, as to the column-based model, different strategies to arrange attributes into column-families (CFs) are proposed in [25]: (i) a *sameCF* schema, where all attributes are put in the same CF; (ii) a *CNSSB schema*, where each dimension is stored in a different CF; and (iii) a *factDate schema*, where some of the most-frequently used dimensions (in their example, the date dimension) are grouped together with fact data. Based on experimental tests, the authors conclude that the sameCF schema provides better performance for high-dimensional queries (three or four dimensions), while the CNSSB and factDate schemas are preferable for low-dimensional queries (one or two dimensions). In the same direction, in [6] the authors propose an approach that clusters in the same CFs attributes that are frequently used together in the workload queries.

3.2 Multi-Model Approaches

A DBMS normally handles a specific data model (e.g., relational DBMSs, document-based DBMSs, etc.). When an application needs different types of data, the first possible solution is to integrate all data into a single DBMS; however, this means that some types of data cannot be stored and analyzed, and that querying performances may be unsatisfactory. The second solution is to use two or more DBMSs together (*polyglot persistence*); even in this case there are drawbacks, since technically managing more DBMSs is a challenge, the learning curve for developers is steep, performance optimization may be inadequate, and there is

a risk of data inconsistency. To overcome these issues, *multi-model databases* (MMDBMSs, e.g., PostgreSQL and ArangoDB) natively support different data models under a single query language to grant performance, scalability, and fault tolerance, so as to reduce maintenance and data integration issues, speed up development, and eliminate migration problems.

As argued in [4], a *multi-model data warehouse* (MMDW) can store data according to the multidimensional model and, at the same time, let each of its elements be natively represented through the most appropriate model; among the benefits, reducing the cost for ETL and ensuring better flexibility, extensibility, and evolvability thanks to the use of schemaless models. However, in a multi-model setting, several alternatives emerge for the logical representation of dimensions and facts, and some of them may be better than others from one or more points of view. Some preliminary tests show that:

- Different dimensions can use different models.
- From the points of view of querying performance, query formulation conciseness, data storage, and complexity of ETL, a multidimensional implementation via the relational model is generally better than a document-based one, which in turn is better than a graph-based one.
- From the point of view of flexibility, extensibility, and evolvability, schemaless models (namely, document- and graph-based) are preferable to the relational one.

Figure 5 shows an optimal multi-model schema for the same multidimensional data of Figure 4.

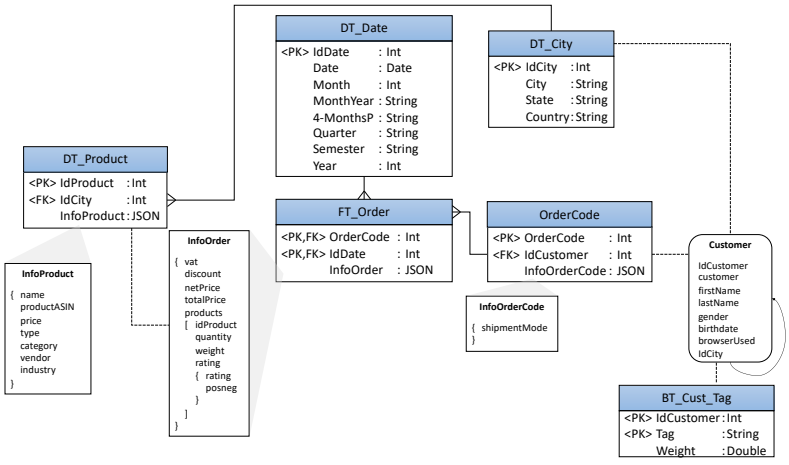

Fig. 5. A multi-model schema that mixes relational tables (e.g., DT_Date), documents (e.g., InfoProduct), and graphs (Customer)

4 Conclusion

Enabling OLAP queries over NoSQL data is getting more and more important today, but dealing with heterogeneity and schema variety intrinsic to NoSQL DBs is a challenge. In this tutorial we have discussed some directions for enabling OLAP on schemaless NoSQL data, using either schema-on-read or schema-on-write approaches. Though several solutions have been proposed in the literature, their level of maturity is not comparable yet to the one reached by relational implementations. Among the relevant issues to be further investigated, we mention the following:

- increase the efficiency of the querying phase in schema-on-read approaches by paving the way to a more sophisticated optimization of query;
- develop techniques for online repairing of approximate functional dependencies present in schemaless data, so that the user can get correct analysis results without modifying the original data;
- extend the existing conceptual models to cope with schemaless data [21, 20]);
- understand how to select and use materialized views in MMDWs, and what ad-hoc indexing strategies to adopt for them.

References

1. Abelló, A., Darmont, J., Etcheverry, L., Golfarelli, M., Mazón, J., Naumann, F., Pedersen, T.B., Rizzi, S., Trujillo, J., Vassiliadis, P., Vossen, G.: Fusion cubes: Towards self-service business intelligence. IJDWM **9**(2), 66–88 (2013)
2. Beheshti, S., Benatallah, B., Nezhad, H.R.M., Allahbakhsh, M.: A framework and a language for on-line analytical processing on graphs. In: Proc. WISE. pp. 213–227 (2012)
3. Bex, G.J., Gelade, W., Neven, F., Vansummeren, S.: Learning deterministic regular expressions for the inference of schemas from XML data. ACM TWEB **4**(4), 14 (2010)
4. Bimonte, S., Gallinucci, E., Marcel, P., Rizzi, S.: Data variety, come as you are in multi-model data warehouses. Inf. Syst. **104** (2022)
5. Bohannon, P., Elnahrawy, E., Fan, W., Flaster, M.: Putting context into schema matching. In: Proc. VLDB. pp. 307–318 (2006)
6. Boussahoua, M., Boussaid, O., Bentayeb, F.: Logical schema for data warehouse on column-oriented NoSQL databases. In: Proc. DEXA. pp. 247–256. Lyon, France (2017)
7. Castelltort, A., Laurent, A.: NoSQL graph-based OLAP analysis. In: Proc. KDIR. pp. 217–224. Rome, Italy (2014)
8. Challal, Z., Bala, W., Mokeddem, H., Boukhalfa, K., Boussaid, O., Benkhelifa, E.: Document-oriented versus column-oriented data storage for social graph data warehouse. In: Proc. SNAMS. pp. 242–247. Granada, Spain (2019)
9. Chen, C., Yan, X., Zhu, F., Han, J., Yu, P.S.: Graph OLAP: a multi-dimensional framework for graph data analysis. Knowl. Inf. Syst. **21**(1), 41–63 (2009)
10. Chevalier, M., Malki, M.E., Kopliku, A., Teste, O., Tournier, R.: Document-oriented models for data warehouses - NoSQL document-oriented for data warehouses. In: Proc. ICEIS. pp. 142–149. Rome, Italy (2016)

11. Chouder, M.L., Rizzi, S., Chalal, R.: EXODuS: Exploratory OLAP over document stores. Inf. Syst. **79**, 44–57 (2019)
12. Cuzzocrea, A., Bellatreche, L., Song, I.Y.: Data warehousing and OLAP over big data: current challenges and future research directions. In: Proc. DOLAP. pp. 67–70 (2013)
13. Dehdouh, K.: Building OLAP cubes from columnar NoSQL data warehouses. In: Proc. MEDI. pp. 166–179. Almería, Spain (2016)
14. Dhamankar, R., Lee, Y., Doan, A., Halevy, A., Domingos, P.: iMAP: discovering complex semantic matches between database schemas. In: Proc. ICMD. pp. 383–394 (2004)
15. Gallinucci, E., Golfarelli, M., Rizzi, S.: Schema profiling of document-oriented databases. Inf. Syst. **75**, 13–25 (2018)
16. Gallinucci, E., Golfarelli, M., Rizzi, S.: Approximate OLAP of document-oriented databases: A variety-aware approach. Inf. Syst. **85**, 114–130 (2019)
17. Gartner Research: Market share: Database management systems, worldwide, 2019. https://www.gartner.com/en/documents/3984279 (April 2020)
18. Golfarelli, M., Rizzi, S.: Data Warehouse Design: Modern Principles and Methodologies. McGraw-Hill, Inc., New York, NY, USA (2009)
19. Gómez, L.I., Kuijpers, B., Vaisman, A.A.: Online analytical processsing on graph data. Intell. Data Anal. **24**(3), 515–541 (2020)
20. Holubová, I., Contos, P., Svoboda, M.: Multi-model data modeling and representation: State of the art and research challenges. In: Proc. IDEAS. pp. 242–251. Montreal, QC, Canada (2021)
21. Holubová, I., Svoboda, M., Lu, J.: Unified management of multi-model data - (vision paper). In: Proc. ER. pp. 439–447. Salvador, Brazil (2019)
22. Izquierdo, J.L.C., Cabot, J.: Discovering implicit schemas in JSON data. In: Proc. ICWE. pp. 68–83 (2013)
23. Ouaret, Z., Chalal, R., Boussaid, O.: An overview of XML warehouse design approaches and techniques. IJICoT **2**(2/3), 140–170 (2013)
24. Ruiz, D.S., Morales, S.F., Molina, J.G.: Inferring versioned schemas from NoSQL databases and its applications. In: Proc. ER. pp. 467–480 (2015)
25. Scabora, L.C., Brito, J.J., Ciferri, R.R., de Aguiar Ciferri, C.D.: Physical data warehouse design on NoSQL databases - OLAP query processing over HBase. In: Proc. ICEIS. pp. 111–118. Rome, Italy (2016)
26. Sellami, A., Nabli, A., Gargouri, F.: Transformation of data warehouse schema to NoSQL graph data base. In: Proc. ISDA. pp. 410–420. Vellore, India (2018)
27. Sellami, A., Nabli, A., Gargouri, F.: Graph NoSQL data warehouse creation. In: Proc. iiWAS. pp. 34–38. Chiang Mai, Thailand (2020)
28. Yangui, R., Nabli, A., Gargouri, F.: Automatic transformation of data warehouse schema to NoSQL data base: Comparative study. In: Proc. KES. pp. 255–264. York, UK (2016)