# Integration of Relational and NoSQL Databases

Jaroslav Pokorný(✉)

Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic
pokorny@ksi.mff.cuni.cz

**Abstract.** The analysis of relational and NoSQL databases leads to the conclusion that these data processing systems are to some extent complementary. In current Big Data applications, especially where extensive analyses are needed, it turns out that it is non-trivial to design an infrastructure involving data and software of both types. In terms of performance, it may be beneficial to use a polyglot persistence or multi-model approach or even to transform the SQL database schema into NoSQL and to perform data migration between the relational and NoSQL database. The aim of the paper is to show these possibilities and some new methods of designing such integrated database architectures.

**Keywords:** Relational database · NoSQL database · Big Data
Big Analytics · Database integration

## 1 Introduction

Recently, most large enterprises seem to be taking actually care about minimizing application maintenance of existing production systems. This causes "bad" database schemas to be used and "database decay" generally occurs. The authors of [14] build the assertion on discussions with nearly twenty database administrators (DBA) at three very large enterprises. The databases vary depending on business conditions, usually once a quarter or more. The environment leads to the often disappearing central DBA's roles and a more decentralized approach with more DBA groups maintaining databases in the enterprise. NoSQL databases, a database alternative for storage and processing so-called Big Data today, contribute to this state significantly.

The DBMS history always reflected requirements concerning new types of data to be stored in a database way. Several database models such as Object-Oriented (OO), Object-Relational (OR), XML, or RDF have been introduced since the relational data model was introduced. OO and OR DBMSs responded to object-oriented approaches to software engineering from the 1990s. However, these tools have never been competitive on the market. Reasons might be in the lack of their theoretical foundations and the limited performance in practice. The XML databases suffer from similar problems. Their goal is to promote the distribution of XML documents, but the use of native XML databases is rather limited. Major vendors of relational DBMS (RDBMS) such as Oracle, Microsoft SQL Server, and MySQL, include XML support in their products,


N. T. Nguyen et al. (Eds.): ACIIDS 2018, LNAI 10752, pp. 35–45, 2018.
https://doi.org/10.1007/978-3-319-75420-8_4

but native XML databases are not much involved in the database market. The initial enthusiasm for XML databases was based on Web application architectures and service orientations that use XML as a means to standardize data exchange format. However, this is now already possible with document-oriented NoSQL databases (see, popular JSON format), though not in such powerful languages as the XQuery in XML environment. However, the XML format has been added to the relational environment and is now the basic data type in SQL databases.

The situation in the database world today is affected by Big Data. Their *V*'s characteristics are *V*olume, *V*elocity, and *V*ariety. The author of [12] lists even 11 such *V*'s. They fundamentally affect the storage and processing infrastructure of Big Data. Effective use of systems involving the processing of large volumes of data requires, in many application scenarios, adequate tools for storing and processing such data at a low level and analytical tools at higher levels. From the user's point of view, the most important aspect of processing large volumes of data on a computer is their analysis, as it is now called *Big Analytics*. Unfortunately, large data collections include data in different formats, such as relational tables, XML data, text data, multimedia data, or RDF triples, which may cause problems in processing data mining algorithms. Also, the growing data volume in a repository or the number of users of this repository requires a reliable solution of scaling in these dynamic environments, and more advanced means of delivering high performance than traditional database architectures offer. Moreover, traditional RDBMSs lack the dynamic data model necessary to tackle high velocity data coming in from machine-oriented systems or time series applications, as well as cases needing to manage social media data.

It is obvious that Big Analytics is also performed over a large amount of transaction data by extending the methods commonly used in Data Warehouses (DW). But DW technology has always been focused on structured data compared to the much richer variability of data types, as it is today for Big Data. Analytical processing of large data volumes therefore requires not only new database architectures but also new methods for data analysis.

To store and process Big Data today, we can choose:

- traditional DBMS (hereinafter referred to as databases, DB) - relational (SQL), OO, OR,
- traditional parallel database systems ("shared-nothing"),
- distributed file systems (e.g., HDFS),
- NoSQL databases,
- new architectures (e.g. NewSQL database).

In practice, ITC and business professionals need to determine whether NoSQL technologies are better suited than RDBMS for a particular system. The choice of technology is critical for applications that can be both transactional and analytical. They typically require different software and hardware architectures. The aim of the paper is to discuss the relation between SQL databases and NoSQL databases, modelling databases in the SQL and NoSQL polyglot world, mainly towards Big Analytics. An attention is devoted to problems of integration of such heterogeneous platforms in

one architecture. In Sect. 2, we briefly describe the Big Analytics concept, i.e. the properties, processing and analysis of large volumes of data. In Sect. 3, we briefly review the NoSQL database technologies, especially their data models, architectures, and some their representatives. In Sect. 4, we show the duality between SQL databases and NoSQL databases and its reflection in various integrated database architectures. Section 5 contains conclusions and challenges for the database community.

## 2 Analytical Processing of Big Data

Big Analytics is used to transform information into knowledge through a combination of existing and new approaches. Related technologies include:

- data management (considering uncertainty, real-time query processing, information extraction, explicit time dimension management),
- new programming models,
- statistical methods, data mining (DM), and machine learning (ML),
- component architectures of data storage and processing systems, visualization of information.

As usual, two types of processing are distinguished:

- real-time processing (*data-in-motion*),
- batch processing of data obtained from different sources into one database (*data-at-rest*).

Batch analysis can then be:

- *small* (Small Analytics), i.e. OLAP over DW,
- *big* (Big Analytics), i.e. both DM and ML.

The problems that arise in this context are based on the fact that the requirements for Big Data are often more dynamic than the classic data processing in DWs. This concerns all 3 *V*'s mentioned in Sect. 1. The NoSQL database is an alternative. Another issue is how to analyse Big Data coming from relational DBs.

A volume is not only a problem for data storage but also influences Big Analytics. With the increase in data complexity, its analysis is also more complex. We need to scale both the infrastructure and the standard data processing techniques for Big Data. Speed can also be a problem because the value of the analysis (and often of the data) decreases over time. If multiple data stream passes are required, data must be entered into DW where further analysis can be performed. Data can thus be stored and processed in a relatively traditional way or using cheap systems such as distributed NoSQL DB.

Big Data is often mentioned only in relation to business intelligence (BI). However, not only BI developers but, generally, data scientists are analysing large data collections. The challenge for computer professionals or data scientists is to provide people with tools that can efficiently perform complex analytics, taking into account the particular nature of processing large volumes of data. It is important to emphasize that

Big Analytics does not only include analysis and modelling phases. Often, distorted context as well as data heterogeneity and interpretation of results are taken into account. All these aspects affect scalable strategies and algorithms, so more efficient pre-processing steps (filtering and integration) and advanced parallel computing environments are needed. Data variability is now part of Big Data storage design and analytical system design. But performance is still a first order requirement.

In addition to these rather classic issues of mining large volumes of data, other interesting issues have emerged in recent years, such as recognizing named entities. The analysis of views and opinions (such as positive, negative, neutral) and their mining (sentiment analysis) are actual as topics using information retrieval methods and Web data analysis. A specific problem is the search for and characterization of discrepancies based on views and opinions. Comparison of graph patterns is commonly used in social network analysis where graphs, for example, include a billion users and hundreds of millions of links. In any case, the main problems of the current DM techniques used for Big Data come from their lack of scalability and parallelization.

## 3 NoSQL Databases

Large-scale data collections are often used for the storage and processing in NoSQL databases. NoSQL means "not only SQL", which makes this database category very diverse and not very clearly specified. NoSQL databases, starting in the late 1990s, provide easier scalability and performance compared to traditional RDBMS. We briefly describe their properties and classification (Sect. 3.1), followed by a discussion of their usability (Sect. 3.2). A more detailed discussion of NoSQL and, more generally, Big Data issues can be found, e.g., in [4, 8, 10, 12].

### 3.1 Categories of NoSQL Databases

What is the main classical approach to databases - a (logical) data model - is described in NoSQL databases rather intuitively, without any formal basis. NoSQL terminology is also very diverse, and the difference between a conceptual and database view is mostly blurred.

The most well-known NoSQL databases can be classified according to the used data model as:

- *key-value stores*, such as Redis[1],
- *column store*, e.g. CASSANDRA[2],
- *document stores*, such as MongoDB[3].

Key-value stores contain a set of pairs (key, value). The key uniquely identifies the opaque value. The choice of the key is, unlike relational DB, solved pragmatically.

[1] https://redis.io/.

[2] http://cassandra.apache.org/.

[3] https://www.mongodb.com/.

The goal is only quick access to data. The value can even be a list of pairs (name, value) (e.g. in Redis). Data access operations, typically `get` and `put`, only work through the key. NULL values are not required, because these databases do not use the schema. Although it is a very efficient and scalable approach to implementation, the disadvantage of a too simple data model can be substantial for such databases.

NoSQL stores can contain a set of couples (name, value) in a *column family* in a row addressed by a key. A column family in different rows can contain different columns. Then we are talking about a column-oriented NoSQL database. There is also another level of structure called, e.g., *supercolumn* in Cassandra. The supercolumn contains nested (sub)columns. Access to data using `get` and `put` is enhanced by using column names.

Most general data models belong to document-oriented NoSQL DBs. They are the same as key-value repositories, but each key is coupled with any complex data structure that resembles a semi-structured document. The JSON format is usually used to present these data structures. JSON is a typed data model that supports basic data types and objects - non-ordered sets of couples (name, value), and the value can be structured (array). JSON is similar to XML, but it is smaller, faster and easier for parsing. For example, CouchDB[4] uses the JSON format whereas MongoDB stores data in BSON (binary coded serialization of JSON documents). It is possible to query the data in a document in other ways than using a key (e.g. through indexing). Moreover, selection and projection operations on the query results can be performed.

There are also other approaches. DB-Engines Ranking server[5], e.g., considers also *search engines* as NoSQL databases, e.g., Elasticsearch[6]. They are data management systems dedicated to the search for data content. They are not typically classical document systems. They typically offer a support for complex search expressions, full text search, ranking and grouping of search results, geospatial search, and distributed search for high scalability. More generally, NoSQL databases include also graph databases [11], and others, e.g. XML and RDF ones.

The first three NoSQL categories are basically of the key-value type. They differ mainly in the possibilities of aggregating couples (key, value) and accessing these values. For our considerations, we consider only them.

### 3.2 Usability of NoSQL Databases

There is much debate about the role of NoSQL databases in providing information services. NoSQL camp claims that this technology is the future of databases. On the other hand, the RDBMS camp argues that the NoSQL databases have a big disadvantage of failing to provide correct data integrity. In any case, NoSQL technologies are designed with Big Data needs in mind.

NoSQL are often a part of data intensive cloud applications (mainly Web applications). Examples of such applications include Web entertainment applications,

[4] http://couchdb.apache.org/.

[5] https://db-engines.com/en/ranking.

[6] https://www.elastic.co/products/elasticsearch.

high-traffic Web site services, media delivery in a streamlined fashion, or typical data found in social networking applications.

NoSQL systems are more suitable for interactive data services environments. Schema enforcing and row-level locking as in relational DBs may over-complicate these applications. The absence of some ACID properties even allows significant acceleration and decentralization of NoSQL databases.

On the other hand, one of the most famous problems with NoSQL repositories is the lack of semantics caused by their underling feature – they are schema-less. The lack of metadata prevents the database system from knowing which data is stored and how it is interconnected.

NoSQL databases usually have little means of ad-hoc querying and analysis. Even a simple query requires significant programming experience, and generally used BI tools do not provide connectivity to NoSQL. NoSQL databases can also not be recommended for applications requiring enterprise level functionality (ACID properties, security, and other relational technology features). NoSQL should not be the only choice in the cloud.

Experience with the NoSQL database shows that they can be used

- even on "small" dates,
- for applications not requiring transactional semantics, such as directories, blogs, or content management systems or for analysing high-volume, real-time data (such as Web site click-streams). In the mobile data processing environment, transactions are even more technically impossible in a larger range.

Among the good properties of NoSQL databases we can find:

- massive performance in `write` operations,
- quick search in a key-value way,
- they do have no portion causing a total network failure when an error occurs,
- enable rapid prototyping and development,
- allow scalability without user intervention,
- have easy maintenance.

On the other hand, a user may find unusual and often inappropriate phenomena in NoSQL approaches:

- have different behaviour in different applications,
- no language query standards are available,
- migration from one system to another is complicated,
- join operation is missing,
- some of them are more mature than others, but each of them is trying to solve similar problems,
- checking referential integrity "over" database partition segments is missing. As the performance is crucial, an integrity control or the implementation of complex operations is limited in a distributed environment.

**Table 1.** Comparison of relational and NoSQL DBMSs

| | Property | RDBMS | NoSQL |
|---|---|---|---|
| 1 | Data model | Relational | Domain-oriented |
| 2 | | Data is strongly typed | Data is potentially dynamically typed |
| 3 | | Data of dependent tables points to its parents (via foreign keys) | Parent's data points to children data |
| 4 | | Associated entities have an identi-ty (primary key) | Environment determines identity |
| 5 | | Not compositional | Compositional |
| 6 | | Referential integrity based on values | Weak referential integrity based on computation or only "over" partition segments |
| 7 | Integrity | Responsibility at the DB level | Responsibility moved to the application level |
| 8 | DB schema | Expressed in SQL | Typically do not require a fixed DB schema, i.e. they have a more flexible data model |
| 9 | Detection of problems in the DB schema | At the DB level | At the application level and data access procedures |
| 10 | DB modelling management | Begins from accessible data | Patterns for data access and updates |
| 11 | Querying | SQL | Simple API, if SQL, then only its very limited version; REST, client libraries |
| 12 | | Complex queries + ad hoc queries | Inappropriate for ad hoc queries and complex queries |
| 13 | | Join operation | Join emulation at application level |
| 14 | Data storage | Centralized or distributed | Horizontally scaled, replications |
| 15 | Data processing | Synchronous (ACID) updates over more rows | Asynchronous (BASE) updates within single values |
| 16 | | Environment coordinates changes (transactions) | Entities responsible to react to changes (eventual consistency) |
| 17 | | Strong consistency and also consistency tuneable by application | Eventual consistency and also consistency tuneable in application |
| 18 | | Query optimizer – responsibility by DBMS | Developer/pattern – responsibility by application |

Table 1 shows a comparison of NoSQL and SQL DB in more details.

In the database world NoSQL DBs occupy a significant place. In the DB-Engines Ranking, 339 various DB-Engines were tracked in December 2017. MongoDB, Redis, and Cassandra occupied positions 5, 8, and 9, respectively, in this rating.

## 4 SQL and NoSQL: Towards Integrated Architectures

In the work [9], the authors argue that the NoSQL databases are rather complementary to traditional transactional DBMSs. Should not they be called "co-relational"? Maybe more natural would be to say coSQL instead of NoSQL. In Table 1, according to [9], complementary differences are given by properties 2, 3, 4, 5, 6, 16, 17, and 19. This complementarity negatively influences integration possibilities of these datastores both at the data model and data processing level.

Particularly, normalization allows single object data in a relational database to be spread over multiple relations. For example, customer data is in one table, data about the banks where his/her account are is in the second table. The interconnection is realized via foreign keys. In NoSQL database, this can be done in such a way that each bank "row" can contain data and account numbers for each customer. The basic feature of NoSQL is that they are denormalized, that is, they store copies of an object instead of the object. This, of course, leads to worse data update options.

In ICT history, different DBMSs were designed to solve different problems, considering still new and new data types. In addition to centralized RDBMSs, specialized servers, universal servers, relational DW, etc. appeared in the past. These tools were based on a fixed database schema and an associated query language (mostly SQL). OR SQL and its other extensions supported this strategy for a long time.

Concerning an integration of distributed data from different databases, two approaches based on a database schema management were at disposal:

- top-down – starting with a global schema to design schemas for data in sites,
- bottom-up – through middleware, i.e. to use schema mapping for schemas in sites into a middleware (e.g., OLE DB, JDBC) and then use a query transformation. Data is loosely integrated and managed by multiple servers.

We remind that the former concerns rather homogenous databases models, while the letter supports heterogeneous database models and consequently DBMSs.

In context of RDBMSs and NoSQL databases, it is not possible to use simply traditional approaches to data integration. The reason is the complementarity of these database types. Moreover, the problem of analysts is that the lack of data schemas (semantics) prevents them from understanding their structure and thus generating serious analyses. Now, the tendency is to create multilevel modelling approaches involving both relational and NoSQL architectures including their integration [1]. Several approaches are under a development:

*Polyglot persistence*. We approach particular data stores with their original data access methods [13]. The truth is that polyglot persistence is a method for data modelling problems, not a solution to them. Developers need to customize data models for an application and often need more than one, but they should not have to adopt different DBMS to get them. "Polyglot" means "able to speak many languages", not integration. As an integration architecture, polyglot persistence is its weakest form.

*Multi-model approach*. Maybe, it presents a more user-friendly solution of heterogeneous database integration. Multi-model represents an intersection of multiple models

in one product. For example, OrientDB[7] is a multi-model DBMS including geospatial, graph, fulltext and key-valued data models. OO concepts are used for user domain modelling in OrientDB. Similarly, ArangoDB[8] is designed as a native multi-model database, supporting key-value, document and graph models. MarkLogic[9] enables to store and search JSON and XML documents and RDF triples.

*NoSQL relationally*. The multi-model solution [5] considers source document and column-oriented DB integrated through a middleware into a virtual SQL database.

*Multilevel modelling*. Despite of the fact that database schemas are mostly not used in the NoSQL world, some variations on multilevel modelling approaches exist. In relation to solution of an alternative for data processing with relational and NoSQL data in one infrastructure, common design methods for such DBs are based on the modification of the traditional 3-level ANSI/SPARC approach [7]. The approach involves not only heterogeneous data sources but also the development of a database schema in the overall infrastructure, i.e., its variability. A strong motivation for this approach is the fact that when designing a database for Big Analytics, we must consider DM/ML patterns, clustering of some attributes, etc., to ensure adequate system performance. However, the conceptual design assumes the correctness of the current knowledge of the application domain. The following examples document activities in this area:

- *Special abstract model*. A DB design methodology for NoSQL systems based on NoAM (NoSQL Abstract Model), a novel abstract data model for NoSQL databases, is presented in [2]. The associated design methodology starts with an UML class diagram, a designer identifies so called aggregates ("chunks" of related data) and maps the aggregates into NoAM blocks. These blocks are simply transformed into constructs of a particular NoSQL data model.
- *NoSQL-on-RDBMS*. A coexistence of RDBMS and a NoSQL DB includes, e.g., storing and querying JSON data in a RDBMS (see, ARGO/SQL [3]).
- *Ontology integration*. A more advanced integrating architecture including several NoSQL databases is proposed in [6]. The databases are described by several ontologies and a generated global ontology. Global SPARQL queries are transformed into query languages of sources.

*Schema and data conversion*. In practice, there are other options, such as the schema conversion model, in which the schema from the SQL database is converted to the NoSQL database schema [15]. Then, even a double-sided data migration between a RDBMS and a NoSQL DB can be performed.

## 5 Conclusions

Key issues for building Big Data processing infrastructure are in decisions concerning NoSQL databases. They include in particular

[7] http://orientdb.com/orientdb/.

[8] https://www.arangodb.com/.

[9] http://www.marklogic.com/.

- choosing the right (correct) product,
- designing a suitable database architecture for a given application class.

However, the role of a person is also significant especially in Big Analytics. Currently, the DM process is driven by an analyst or data scientist. Depending on the application scenario, the person determines a portion of the data from which, e.g., useful patterns can be extracted. A better solution would, however, be to have an automated DM process in place to get approximate synthetic information about both structure and content of large amounts of data. This is still a big problem for Big Data analysts.

Current challenges for database research include:

- Modelling polyglot and multi-model databases including relational and NoSQL in one infrastructure.
- Improving the quality and scalability of DM methods. Interpreting a query - especially in the schema absence - and received answers, may be non-trivial.
- Transforming content into a structured format for later analysis, because many data today is not natively in a structured format. At the same time, with a filtering we can reduce the volume of data.
- Develop a meaningful and usable formalisms for modelling NoSQL databases and a sufficiently general user-friendly query language.

**Acknowledgments.** This work was supported by the Charles University project Q48.

## References

1. Abelló, A.: Big data design. In: Proceedings of 11th International Workshop on Data Warehousing and OLAP, DOLAP 2015, pp. 35–38. ACM (2015)
2. Bugiotti, F., Cabibbo, L., Atzeni, P., Torlone, R.: Database design for NoSQL systems. In: Yu, E., Dobbie, G., Jarke, M., Purao, S. (eds.) ER 2014. LNCS, vol. 8824, pp. 223–231. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12206-9_18
3. Chasseur, C., Li, Y., Patel, J.M.: Enabling JSON document stores in relational systems. In: 16th International Workshop on the Web and Databases (WebDB 2013), pp. 1–6 (2013)
4. Corbellini, A., Mateos, C., Zunino, A., Godoy, D., Schiaffino, S.: Persisting big-data: the NoSQL landscape. Inf. Syst. **63**, 1–23 (2017)
5. Curé, O., Hecht, R., Le Duc, C., Lamolle, M.: Data integration over NoSQL stores using access path based mappings. In: Hameurlain, A., Liddle, S.W., Schewe, K.-D., Zhou, X. (eds.) DEXA 2011. LNCS, vol. 6860, pp. 481–495. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23088-2_36
6. Curé, O., Lamole, M., Duc, C.L.: Ontology Based Data Integration over Document and Column Family Oriented NOSQL, CoRR, arXiv:1307.2603 (2013)
7. Herrero, V., Abelló, A., Romero, O.: NOSQL design for analytical workloads: variability matters. In: Comyn-Wattiau, I., Tanaka, K., Song, I.-Y., Yamamoto, S., Saeki, M. (eds.) ER 2016. LNCS, vol. 9974, pp. 50–64. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46397-1_4
8. Marz, N., Warren, J.: Big Data: Principles and Best Practices of Scalable Realtime Data Systems, 1st edn. Manning Publications, New York (2015)

9. Meijer, E., Bierman, G.M.: A co-relational model of data for large shared data banks. Commun. ACM **54**(4), 49–58 (2011)
10. Pokorný, J.: NoSQL databases: a step to databases scalability in Web environment. Int. J. Web Inf. Syst. **9**(1), 69–82 (2013)
11. Pokorný, J.: Graph databases: their power and limitations. In: Saeed, K., Homenda, W. (eds.) CISIM 2015. LNCS, vol. 9339, pp. 58–69. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24369-6_5
12. Pokorný, J.: Big data storage and management: challenges and opportunities. In: Proceedings of 12th IFIP WG 5.11 International Symposium on Environmental Software Systems, IFIP AICT 507. Springer, Heidelberg (2018, to appear)
13. Sadalage, P.J., Fowler, M.: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Pearson Education Inc., London (2013)
14. Stonebraker, M., Deng, D., Brodie, M.L.: Database decay and how to avoid it. In: Proceedings of 2016 IEEE International Conference on Big Data, pp. 7–16. IEEE Explore (2016)
15. Zhao, G., Lin, Q., Li, L., Li, Z.: Schema conversion model of SQL database to NoSQL. In: Proceedings of the 9th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, pp. 355–362. IEEE (2014)