Science, Engineering & Education, 4, (1), 2019, 19-27

Mongo DB and why should we choose it

Evgenia Savova-Videnova*, Marko Uzunov

University of Chemical Technology and Metallurgy, Sofia, Bulgaria

Received 08 January 2019, Accepted 31 July 2019

ABSTRACT

In this paper we show how Mongo DB changes our ideas about databases, how world passes from relational database management system (RDBMS) to NoSQL databases and how quickly they take over our attention.

We describe some types of databases and compare them with document ones. Then we compare some document databases in order to show the advantages of Mongo DB. We show the usages of Mongo DB and reasons why it becomes most popular NoSQL database and why we shall choose it.

Keywords: Document-oriented NoSQL databases, Humongous, Mongo DB, NoSQL databases.

INTRODUCTION

Imagine a world where using a database is so simple that you soon forget you are even using it. Imagine a world where speed and scalability just work, and there is no need for complicated configuration or setup. Imagine being able to focus only on the task at hand, get things done, and then, just for a change, leave work on time. That might sound a bit fanciful, but Mongo DB promises to help you to accomplish all these things (and many more).

Mongo DB (derived from the word humongous) is a relatively new breed of database that has no concept of tables, schemas, SQL, or rows. It does not have transactions, ACID compliance, joins, foreign keys, or many of the other features that tend to cause headaches in the early hours of the morning. In short, Mongo DB is probably a very different database than what you are used to, especially if you have used a relational database management system (RDBMS) in the past. In fact, you might even be shaking your head in wonder at the lack of so-called "standard" features [1].

According to its creators, Mongo DB was designed to combine the best features of key-value stores and relational databases. Because of their simplicity, key-value stores are extremely fast and relatively easy to scale. Relational databases are more difficult to scale, at least horizontally, but have a rich data model and a powerful query language. Mongo DB is intended to be a compromise between these two designs, with useful aspects of both. The end goal is for it to be a database that scales easily, stores rich

**Correspondence to: Evgenia Savova-Videnova, University of Chemical Technology and Metallurgy, 8 Kliment Ohridski, 1756 Sofia, Bulgaria, E-mail: jeni79bg@gmail.com*

data structures, and provides sophisticated query mechanisms. In terms of use cases, Mongo DB is well-suited as a primary datastore for web applications, analytics and logging applications, and any application requiring a medium-grade cache. In addition, because it easily stores schema-less data, Mongo DB is also good for capturing data whose structure can not be known in advance. The preceding claims are bold [2].

MongoDB is in many ways like a power drill. Your ability to complete a task is framed largely by the components you choose to use (from drill bits of varying size to sander adapters). Mongo DB's strength lies in versatility, power, ease of use, and ability to handle jobs both large and small.

Although it is a much newer invention than the hammer, it is increasingly a tool builders reach for quite often.

Mongo is a JSON document database (though technically data is stored in a binary form of JSON known as BSON). A Mongo document can be likened to a relational table row without a schema, whose values can nest to an arbitrary depth. Mongo is an excellent choice for an ever-growing class of web projects with large-scale data storage requirements but very little budget to buy big-iron hardware. Thanks to its lack of structured schema, Mongo can grow and change along with your data model. If you are in a web startup with dreams of enormity or are already large with the need to scale servers horizontally, consider Mongo DB [3].

Mongo DB is a schema-free document database written in C++ and developed in an open-source project which is mainly driven by the company 10gen Inc that also offers professional services around Mongo DB. According to its developers the main goal of Mongo DB is to close the gap between the fast and highly scalable key-value-stores and feature-rich traditional RDBMSs relational database management systems [4].

Table 1. Database families.

	Examples	Data model	Scalability model	Use cases
Simple key-value stores	Memcached	Key-value, where the value is a binary blob.	Variable. Memcached can scale across nodes, converting all available RAM into a single, monolithic datastore.	Caching. Web ops.
Sophisticated key-value stores	HBase, Cassandra, Riak KV, Redis, CouchDB	Variable. Cassandra uses a key-value structure known as a column. HBase and Redis store binary blobs. CouchDB stores JSON documents.	Eventually consistent, Multimode distribution for high availability and easy failover.	High-throughput verticals (activity feeds, message queues). Caching. Web ops.
Relational databases	Oracle Database, IBM DB2, Microsoft SQL Server, MySQL, PostgreSQL	Tables.	Vertical scaling. Limited support for clustering and manual partitioning	System requiring transactions (banking, finance) or SQL. Normalized data model.

Mongo DB versus other databases

The number of available databases has exploded, and weighing one against another can be difficult. Fortunately, most of these databases fall under one of a few categories. In Table 1, and in the sections that follow, we describe simple and sophisticated key-value stores, relational databases, and document databases, and show how these compare with Mongo DB.

Simple key-value stores

Simple key-value stores do what their name implies: they index values based on a supplied key. A common use case is caching. For instance, suppose you needed to cache an HTML page rendered by your app. The key in this case might be the page's URL, and the value would be the rendered HTML itself. Note that as far as a key-value store is concerned, the value is an opaque byte array. There's no enforced schema, as you'd find in a relational database, nor is there any concept of data types. This naturally limits the operations permitted by key-value stores: you can insert a new value and then use its key either to retrieve that value or delete it. Systems with such simplicity are generally fast and scalable. The best-known simple key-value store is Memcached, which stores its data in memory only, so it trades durability for speed. It is also distributed; Memcached nodes running across multiple servers can act as a single datastore, eliminating the complexity of maintaining cache state across machines.

Compared with MongoDB, a simple key-value store like Memcached will often allow for faster reads and writes. But unlike MongoDB, these systems can rarely act as primary datastores. Simple key-value stores are best used as adjuncts, either as caching layers atop a more traditional database or as simple persistence layers for ephemeral services like job queues.

Sophisticated key-value stores

It is possible to refine the simple key-value model to handle complicated read/write schemes or to provide a richer data model. In these cases, you end up with what we shall term a sophisticated key-value store. One example is Amazon's Dynamo. The aim of Dynamo is to be a database robust enough to continue functioning in the face of network failures, datacenter outages, and similar disruptions. This requires that the system always be read from and written to, which essentially requires that data be automatically replicated across multiple nodes. If a node fails, a user of the system , perhaps in this case a customer with an Amazon shopping cart, will not experience any interruptions in service. Dynamo provides ways of resolving the inevitable conflicts that arise when a system allows the same data to be written to multiple nodes. At the same time, Dynamo is easily scaled. Because it is masterless, all nodes are equal, it is easy to understand the system as a whole, and nodes can be added easily. Although Dynamo is a proprietary system, the ideas used to build it have inspired many systems falling under the NoSQL umbrella, including Cassandra, HBase, and Riak KV.

By looking at who developed these sophisticated key-value stores, and how they have been used in practice, you can see where these systems shine. Let us take Cassandra, which implements many of Dynamo's scaling properties while providing a column oriented data model inspired by Google's BigTable. Cassandra is an open source version of a datastore built by Fac for its inbox search feature. The system scales horizontally to index more than 50 TB of inbox data, allowing for searches on inbox keywords and recipients. Data is indexed by user ID, where each record consists of an array of search terms for keyword searches and an array of recipient IDs for recipient searches. These sophisticated key-value stores were developed by major internet companies such as Amazon, Google, and Fac to manage cross-sections of systems with extraordinarily large amounts of data. In other words, sophisticated key-value stores manage a relatively self-contained domain that demands significant storage and availability. Because of their masterless architecture, these

systems scale easily with the addition of nodes. They opt for eventual consistency, which means that reads do not necessarily reflect the latest write. But what users get in exchange for weaker consistency is the ability to write in the face of any one node's failure.

This contrasts with Mongo DB, which provides strong consistency, a rich data model, and secondary indexes. The last two of these attributes go hand in hand; key-value stores can generally store any data structure in the value, but the database is unable to query them unless these values can be indexed. You can fetch them with the primary key, or perhaps scan across all of the keys, but the database is useless for querying these without secondary indexes.

Relational databases

Popular relational databases include MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database, IBM DB2, and so on; some are open-source and some are proprietary. Mongo DB and relational databases are both capable of representing a rich data model. Where relational databases use fixed-schema tables, Mongo DB has schema-free documents. Most relational databases support secondary indexes and aggregations.

Perhaps the biggest defining feature of relational databases from the user's perspective is the use of SQL as a query language. SQL is a powerful tool for working with data; it is not perfect for every job, but in some cases it is more expressive and easier to work with than Mongo DB's query language. Additionally, SQL is fairly portable between databases, though each implementation has its own quirks. One way to think about it is that SQL may be easier for a data scientist or full-time analyst who writes queries to explore data. Mongo DB's query language is aimed more at developers, who write a query once to embed it in their application. Both models have their strengths and weaknesses, and sometimes it comes down to personal preference. There are also many relational databases intended for analytics (or as a "data warehouse") rather than as an application database. Usually data is imported in bulk to these platforms and then queried by analysts to answer business-intelligence questions. This area is dominated by enterprise vendors with HP Vertica or Teradata Database, which both offer horizontally scalable SQL databases.

There is also growing interest in running SQL queries over data stored in Hadoop. Apache Hive is a widely used tool that translates a SQL query into a Map-Reduce job, which offers a scalable way of querying large data sets. These queries use the relational model, but are intended only for slower analytics queries, not for use inside an application [2].

Document databases

Documents are the main concept in document databases. The database stores and retrieves documents, which can be XML, JSON, BSON, and so on. These documents are self-describing, hierarchical tree data structures which can consist of maps, collections, and scalar values. The documents stored are similar to each other but do not have to be exactly the same. Document databases store documents in the value part of the key-value store; think about document databases as key-value stores where the value is examinable [5].

These databases store and manage volumes of collections of textual documents (e.g. emails, web pages, text file books), semi-structure, as well as no structure and de-normalized data; that would require extensive usage of null values as in RDBMS. Unlike key-value stores, the document databases support secondary indexes on sub-documents to allow fast searching. They allow horizontal scaling of the data over multiple servers called shards. Mongo DB [7], Couch DB [8], Couchbase [9], ReThink DB [10], and Cloudant [11] are some of the most popular document-oriented databases, as shown in the Table 2 in a categorical format along their different characteristics. Among these Mongo DB is the most popular one due to its efficiency, in

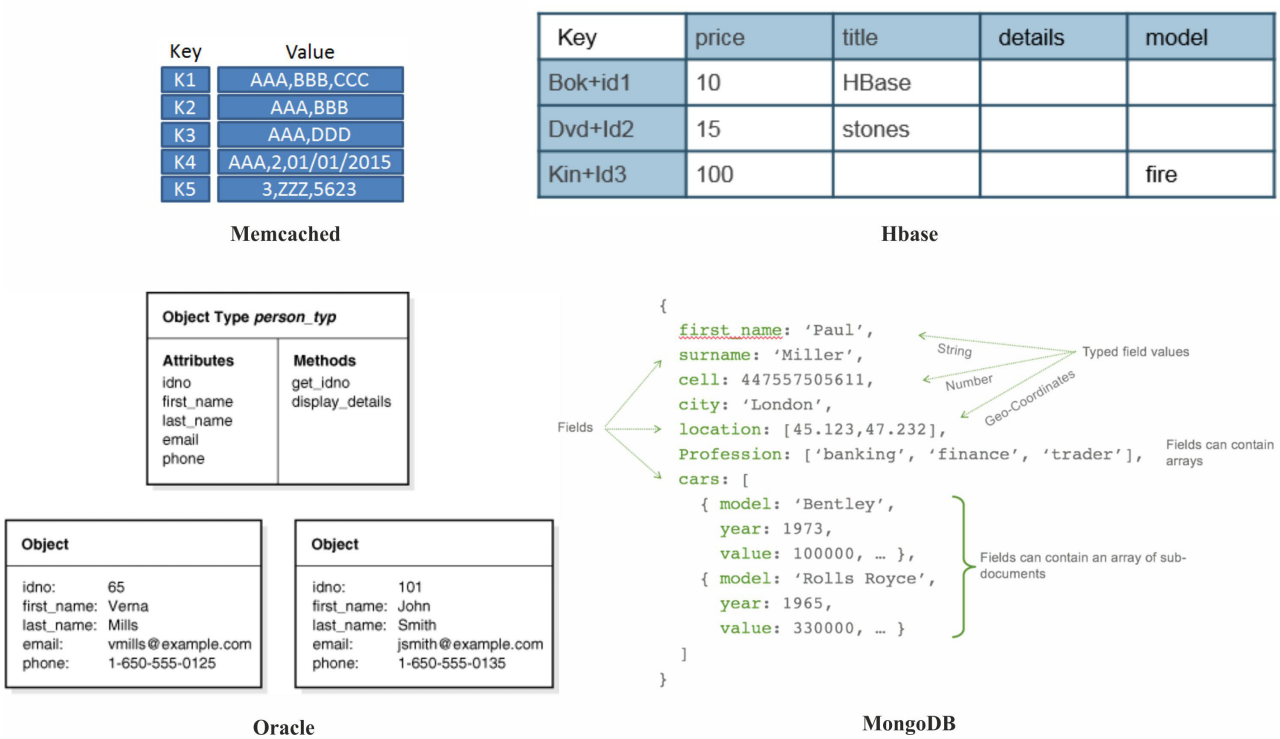

Fig. 1. Mongo DB versus other databases.

memory processing and complex data type features. The other databases such as Couchbase, ReThink DB, Cloudant and Couch DB do not offer in-memory processing features; Mongo DB is good for the dynamic queries, which the other document-oriented databases lack, such as Couch DB or Couchbase.

Besides this there are different object relational mapping middle wares available, to define out of the box multiple schemas depending upon the application requirements. The object nature of Mongo DB documents makes this mapping even more fluid and fast [6].

Databases and Collections

Mongo DB databases reside on a Mongo DB server that can host more than one of such databases which are independent and stored separately by the Mongo DB server. A database contains one or more collections consisting of documents. In order to control access to the database a set of security credentials may be defined for databases.

Collections inside databases are referred to by the Mongo DB manual as "named groupings of documents". As Mongo DB is schema-free the documents within a collection may be heterogeneous although the Mongo DB manual suggests to create "one database collection for each of your top level objects". Once the first document is inserted into a database, a collection is created automatically and the inserted document is added to this collection. Such an implicitly created collection gets configured with default parameters by Mongo DB, if individual values for options such as auto-indexing, preallocated disk space or size-limits are demanded, collections may also be created explicitly by the createCollection-command:

>>db. createCollection (<name >, {< configuration parameters >})

Documents

The abstraction and unit of data storable in Mongo DB is a document, a data structure comparable to an XML document, a Python diction-

Table 2. Document-oriented databases

Name	Data model	Scalability	Description	Who uses it
MongoDB	JSON-like hierarchical documents with or without schemas, object mapping, BSON	Sharding, replication and persistency	Most popular JSON document store, ACID, MapReduce, primary secondary indexing, eventual consistency, RESTful	Expedia, Bosh, MetLife, Facebook, comcast, sprinklr
CouchDB	Native JSON-document store; types: strings, numbers, dates, ordered lists and associative arrays	Multi-master replication	JavaScript as query language using MapReduce, and HTTP for an API, multi-version concurrency, MapReduce, ACID, eventual consistency	meebo, AirFi Sophos, BBC, npm CANAL+
Couchbase	Multi-model: key-value store, document-store; JSON documents	Sharding, master-master and master-slave replication	CouchDB based with Memcachedcompatible interface, eventual consistency, limited ACID, eventual consistency, RESTful HTTP API	Informatica, Joyent, intel, Wipro, Google, Simba
RethinkDB	JSON documents with dynamic schemas	Sharding, master-slave replication	Push real-time data; RethinkDB Query Language (ReQL); Hadoop-style MapReduce; primary&secondary indexes; Not ACID	Jive SW, Mediafly, Pristine Platzi, CMUNE, Wise.io
Cloudant	JSON based flexible documents	Sharding, master-master & master-slave replication	CouchDB based; primary and secondry indexes; MapReduce; Eventual Consistency; RESTful HTTP/JSON API	Samsung, IBM, Expedia, DHL, Microsoft, Pearson

ary, a Ruby hash or a JSON document. In fact, Mongo DB persists documents by a format called BSON which is very similar to JSON but in a binary representation for reasons of efficiency and because of additional datatypes compared to JSON. Nonetheless, "BSON maps readily to and from JSON and also to various data structures in many programming languages". Documents in Mongo DB are limited in size by 4 megabytes.

Datatypes for Document Fields

MongoDB provides the following datatypes for document fields:

- scalar types: **boolean**, **integer**, **double**
- character sequence types: **string** (for character sequences encoded in UTF-8), **regular expression**, **code** (JavaScript)
- **object** (for BSON-objects)
- **object id** is a data type for 12 byte long binary values used by MongoDB and all officially

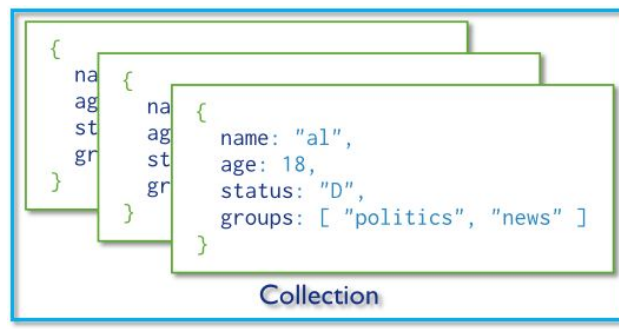

Fig. 2. MongoDB Collection.

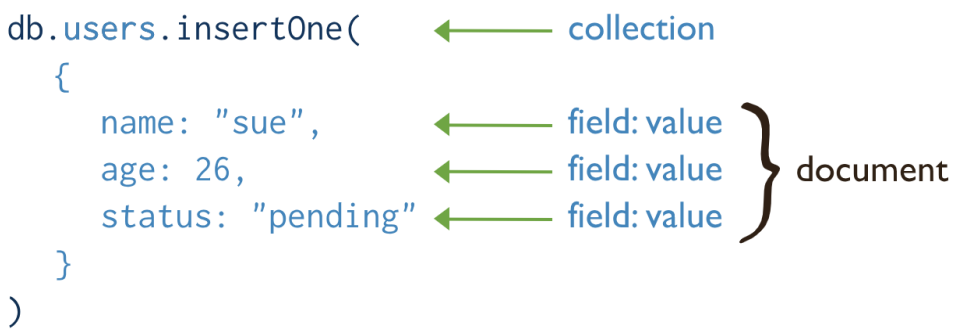

Fig. 3. Mongo DB Document structure.

supported

- programming language drivers for a field named id that uniquely idenfies documents within collections. The object
- id datatype is composed of the following components:
- timestamp in seconds since epoch (first 4 bytes)
- id of the machine assigning the object id value (next 3 bytes)
- id of the MongoDB process (next 2 bytes)
- counter (last 3 bytes)

For documents whose _id field has an object id value the timestamp of their creation can be Extracted by all officially supported programming drivers [4]

- null
- array
- date

Use cases of MongoDB

You are not going to choose a database solely on the basis of its features. You need to know that real businesses are using it successfully.

Web applications

MongoDB is well-suited as a primary datastore for web applications. Even a simple web application will require numerous data models for managing users, sessions, app-specific data, uploads, and permissions, to say nothing of the overarching domain. Just as this aligns well with the tabular approach provided by relational databases, so too it benefits from Mongo DB's collection and document model. And because documents can represent rich data structures, the number of collections needed will usually be less than the number of tables required to model the same data using a

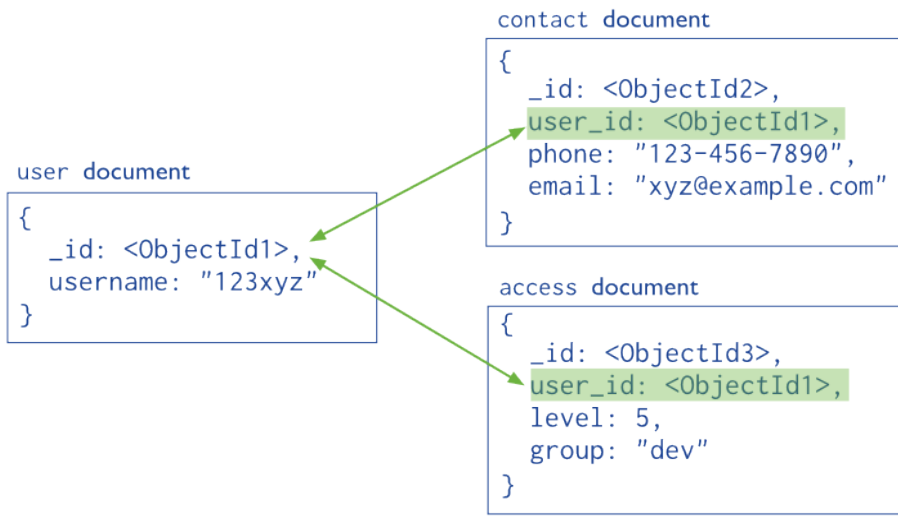

Fig. 4. Data model in Mongo DB.

fully normalized relational model.

In addition, dynamic queries and secondary indexes allow for the easy implementation of most queries familiar to SQL developers. Finally, as a web application grows, Mongo DB provides a clear path for scale.

Mongo DB can be a useful tool for powering a high-traffic website. This is the case with The Business Insider (TBI), which has used Mongo DB as its primary datastore since January 2008. TBI is a news site, although it gets substantial traffic, serving more than a million unique page views per day. What's interesting in this case is that in addition to handling the site's main content (posts, comments, users, and so on), Mongo DB processes and stores real-time analytics data. These analytics are used by TBI to generate dynamic heat maps indicating click-through rates for the various news stories.

Agile Development

Regardless of what you may think about the agile development movement, it is hard to deny the desirability of building an application quickly. A number of development teams, including those from Shutterfly and The New York Times, have chosen Mongo DB in part because they can develop applications much more quickly on it than on relational databases. One obvious reason for this is that Mongo DB has no fixed schema, so all the time spent committing, communicating, and applying schema changes is saved.

In addition, less time need be spent shoehorning the relational representation of data into an object-oriented data model or dealing with the vagaries and optimizing the SQL produced by object-relational mapping (ORM) technology [13]. Thus, MongoDB often complements projects with shorter development cycles and agile, mid-sized teams.

Analytics and logging

We alluded earlier to the idea that Mongo DB works well for analytics and logging, and the number of applications using Mongo DB for these is growing. Often, a well established company will begin its forays into the Mongo DB world with special apps dedicated to analytics. Some of these companies include GitHub, Disqus, Justin.tv, and Gilt Groupe, among others.

Mongo DB's relevance to analytics derives from its speed and from two key features: targeted atomic updates and capped collections. Atomic updates let clients efficiently increment counters and push values onto arrays. Capped collections are useful for logging because they store only the most recent documents. Storing logging data in a database, as compared with the filesystem, provides easier organization and greater query power. Now, instead of using grep or a custom log search utility, users can employ the MongoDB query language to examine log output.

Caching

Many web-applications use a layer of caching to help deliver content faster. A data model that allows for a more holistic representation of objects (it is easy to shove a document into Mongo DB without worrying much about the structure), combined with faster average query speeds, frequently allows Mongo DB to be run as a cache with richer query capabilities, or to do away with the caching layer all together. The Business Insider, for example, was able to dispense with Memcached, serving page requests directly from Mongo DB [2].

CONCLUSIONS

Mongo DB is an open source, document-based database management system. Designed for the data and scalability requirements of modern internet applications, Mongo DB features dynamic

stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time. The document model maps to the objects queries and secondary indexes, fast atomic updates and complex aggregations, and support for replication

with automatic failover and sharding for scaling horizontally [2].

Mongo DB in your application code, is making data easy to work with. Ad hoc queries, indexing, and real time aggregation provide powerful ways to access and analyze your data. NoSQL encompasses a wide variety of different database technologies that were developed in response to the demands presented in building modern applications. Mongo DB is a document NoSQL database with the scalability and flexibility that you want with the querying and indexing that you need [7].

The following are some additional reasons Mongo DB has become the most popular NoSQL database:

- **Document oriented:** Because Mongo DB is document oriented, the data is stored in the database in a format that is very close to what you will be dealing with in both server-side and client-side scripts. This eliminates the need to transfer data from rows to objects and back.
- **High performance:** Mongo DB is one of the highest-performing databases available.
- Especially in today's world, where many people interact with websites, having a back end that can support heavy traffic is important.
- **High availability:** Mongo DB's replication model makes it easy to maintain scalability while keeping high performance and scalability.
- **High scalability:** Mongo DB's structure makes it easy to scale horizontally by sharding the data across multiple servers.
- **No SQL injection:** Mongo DB is not susceptible to SQL injection (putting SQL statements in web forms or other input from the browser that compromises the DB security) because objects are stored as objects, not by using SQL strings [12].

REFERENCES

1. Eelco Plugge, Peter Membrey and Tim Hawkins, The Definitive Guide to MongoDB - The NoSQL Database for Cloud and Desktop Computing, 2010.
2. Kyle Banker, Peter Bakkum, Shaun Verch, Douglas Garrett and Tim Hawkins, MongoDB in action Second Edition, 2016.
3. Eric Redmond and Jim R. Wilson, Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement, 2012.
4. Christof Strauch, NoSQL Databases, 2018.
5. Pramod J. Sadalage and Martin Fowler, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, 2012.
6. Nadeem Qaisar Mehmood, Rosario Culmone and Leonardo Mostarda, Modeling temporal aspects of sensor data for MongoDB NoSQL database, Journal of Big Data 4:8 DOI 10.1186/s40537-017-0068-5, 2017.
7. MongoDB for GIANT ideas. https://www.mongodb.com. Accessed October 2019.
8. Apache CouchDB. http://couchdb.apache.org. Accessed October 2019.
9. Apache CouchBase. http://www.couchbase.com. Accessed October 2019.
10. Rethink DB. https://rethinkdb.com. Accessed October 2019.
11. IBM Cloudant. https://cloudant.com. Accessed October 2019.
12. Brad Dayley, Sams Teach Yourself NoSQL with MongoDB in 24 Hours, 2014
13. Object-relational mapping. https://en.wikipedia.org/wiki/Object-relational_mapping Accessed October 2019