DOI: https://doi.org/10.30837/MMP.2023.229

STUDY OF THE METHODS AND MEANS OF DATA MIGRATION BETWEEN RELATIVE AND NON-RELATIVE DATABASES

Morozova A., Petrova R.

Nowadays, all information and information-analytical systems require the use of databases (DBs). Databases are an integral part of the information infrastructure of modern enterprises and organizations. These systems have to process, read, and record certain data sets that need to be organized, structured, and stored. The need to solve these problems has given rise to a number of new platforms and tools for large volumes of diverse and unstructured data. In this case, non-relational data warehouses appear in contrast to the well-known relational ones. NoSQL offers a certain concept in contrast to the SQL paradigm that has dominated for a long time. Therefore, the transition from a relational to a non-relational storage implies not only data migration, but also a revision of the concept of data processing and its model.

Introduction

Developers are increasingly choosing MongoDB for their projects. MongoDB is an open source document-oriented database management system (DBMS) that does not require a table schema description. MongoDB is not a replacement for relational databases, but rather an alternative, but for some projects where data can be presented in the form of documents, such as blog pages, it is rational to use it.

However, if existing projects written using relational SQL databases are migrated to MongoDB, the problem of optimizing the relational data structure arises.

Currently, there are several software solutions for converting SQL queries to JSON queries for the new DBMS. All of them use a simple query conversion with the preservation of the relational structure, which is in line with the MongoDB philosophy.

According to the theory of relational databases, they should be given a third normal form. The purpose of normalizing a relational database is to eliminate the flaws in its structure that lead to redundancy, which, in turn, potentially causes various anomalies and data integrity violations. Data can be stored in the third normal form of MongoDB, but this storage method does not provide all the advantages of MongoDB.

MongoDB operates with the concepts of database (db), collection, documents, and fields, which in relational DB terms are a database, table, and table element, respectively. A database can have zero or more collections. Collections consist of zero or more "documents". A document has one or more "fields", which, as you might guess, are similar to "columns". MongoDB supports nested arrays and collections.

The proposed solution will make it easier for developers to switch to a document-oriented database and migrate data and offer a new, MongoDB-optimized structure. In addition, there will be a choice between several options for a new data structure focused on the new database, as well as the possibility of corrections in the DB structure and fields.

The main direction of the work should be the regularities that arise in the process of transition from a relational to a non-relational data structure, as well as the optimization of this structure in accordance with the concepts of MongoDB.

Analysis of existing ways to represent DBs

The literature offers many definitions of the concept of "database", reflecting rather the subjective opinion of certain authors, but there is no single universally recognized formulation.

The author of the article believes that the following definition of databases is the most successful: "A database is a collection of data organized in accordance with certain rules and maintained in a computer's memory that characterizes the current state of a certain subject area and is used to meet the information needs of users" [1].

Currently, there is no single approach to data migrations, so a large number of software products use outdated versions of DBMSs that are limitedly or not supported by developers. Also, one of the reasons for the stupor on one version of the DBMS is that migrations lead to numerous risks in the form of data loss and incompatibility of the existing program code with the modernized DBMS [2]. All of the above factors discourage software product owners from migrations of this kind.

An unsuccessful migration can result in data degradation or loss of data altogether. This can happen even when the data in the source DBMS looks perfectly adequate and usable. In addition, any problems that existed in the data may be amplified after it is transferred to the new storage [3].

Typically, data migration strategies prevent the use of auxiliary software packages [4], which end up creating more problems than they solve. When planning and developing a work strategy, developers need to pay full attention to such migrations, without diminishing their importance and scope.

There are many types of databases that differ by different criteria. The main classifications are listed below.

Classification by data model: hierarchical, object and object-oriented, object-relational, relational, network, functional [5].

Classification by the level of distribution: centralized, or concentrated (a database fully supported on one computer), distributed (a database whose components are located in different nodes of a computer network in accordance with any criteria).

Distributed databases, in turn, are divided into: heterogeneous, homogeneous, fragmented, or sectioned, replicated [6].

A database management system is a set of software and linguistic tools for general or special purposes that manage the creation and use of databases [7].

Relational DBMS is a relational database management system. The relational data model is aimed at organizing data as a relationship [8].

Let's consider some of the most popular RDBMSs.

MySQL is a free relational database management system developed and maintained by Oracle Corporation. MySQL is a solution for small and medium-sized applications and is ranked second in the DBMS rating according to the results of research from the DB Engines website.

Microsoft SQL Server is a RDBMS developed by Microsoft. The main query language is Transact-SQL, created jointly by Microsoft and Sybase. Transact-SQL is an implementation of the ANSI/ISO standard for structured query language (SQL) with extensions. It is used to work with databases ranging in size from personal to large enterprise-scale databases; it competes with other DBMSs in this market segment. It is ranked third in the DB Engines rating [9].

PostgreSQL is the most advanced of the three DBMSs we have considered. It is freely distributed and complies with SQL standards to the maximum extent possible. PostgreSQL or Postgres tries to fully implement ANSI/ISO SQL standards simultaneously with the release of new versions [10].

PostgreSQL differs from other DBMSs by supporting the popular object-oriented and/or relational approach to databases. Thanks to powerful technologies, Postgre is very productive. PostgreSQL is easy to extend with procedures called stored procedures. These features simplify the use of constantly repeated operations.

Couchbase is a document-oriented database that is interesting for its relative simplicity, exceptional ease of setup and support, high query execution speed due to in-memory data storage, scalability, and automatic cluster recovery in case of machine crashes and other factors. It provides tools similar to Apache CouchDB for creating document-oriented databases in combination with Membase-like storages in the key-value format.

Thanks to support for the standard memcached protocol, the system remains compatible with a large number of legacy applications and can transparently replace a number of other NoSQL systems.

Couchbase documents are in JSON, a self-describing format capable of representing voluminous structures and relationships. Unlike a traditional DBMS, a schema on Couchbase Server is a logical construct fully defined in the program code and fixed in the structure of the stored documents. Because there is no explicit schema support, developers can add new objects and properties at any time by simply clicking on the new application code that stores the new JSON, without having to make the same changes to the schemas. This facilitates fast and easy program development.

Couchbase's architecture ensures that workloads are evenly distributed among cluster nodes, reducing bottlenecks and allowing users to fully utilize available hardware.

MongoDB is an open source document-oriented database management system (DBMS) that does not require a table schema description. It is written in C++ [11].

The DBMS architecture manages sets of JSON-like documents stored in binary form in BSON format. Saving and searching files in MongoDB is done by calling the GridFS protocol. Like other document-oriented DBMSs (CouchDB, etc.), MongoDB is not a relational DBMS.

MongoDB implements asynchronous replication in a master-slave configuration based on the transfer of the change log from the master node to the slaves. Automatic recovery is supported in the event of a master node failure. The servers running the mongod process must form a quorum for the new master to be automatically determined. Therefore, unless a special arbitrator process is used (a mongod process that only participates in quorum establishment but does not store any data), the number of running replicas must be odd.

To summarize. MongoDB contains databases that consist of collections. "Collections" consist of "documents". Each "document" contains "fields". "Collections" can be indexed, which improves the performance of selecting and sorting. And finally, retrieving data from MongoDB is reduced to retrieving a "cursor" that gives that data as needed. These terms, while close to their relational counterparts, are not completely identical. The main difference is that relational databases define "columns" only at the "table" level, while document-oriented databases define "fields" only at the "document" level. This means that any "document" within a "collection" can have its own unique set of "fields". In this sense, a collection is "dumber" than a table, while a document has much more information than a string.

So, according to the author of the article, MongoDB can be considered as a direct alternative to relational databases. MongoDB cannot be called a replacement for a RDBMS, but rather an alternative.

Purpose of the study

The aim of the study is to develop a subsystem for migrating from a relational to a document-oriented DBMS, as well as to study patterns and investigate dependencies between SQL and MongoDB queries that describe the same data but in different representations of the data model.

To solve this problem, you need to develop several relational data models with different structures and subject areas. For each model, create SQL queries in different DBMSs, because each DBMS has a different syntax. Next, for each relational data model, you need to create a hierarchical data model that describes the same fields and their attributes. In the process of model conversion, it is necessary to maintain data integrity and optimize its structure for the purposes of a document-oriented DBMS. Based on the hierarchical model (hierarchical data structure) of the document, create MongoDB queries that describe and create a document corresponding to this model.

After analyzing these parameters and identifying patterns of transitions between data models, an algorithm for translating queries and an algorithm for optimizing the document structure in accordance with MongoDB requirements were developed.

Description of the work results

A comparative analysis of the performance of MongoDB and MySql was conducted. The simplest database containing words and their lengths is used as a test database. The test results are shown in the following figures (the lower the execution time, the better).

Fig. 1 shows a comparison of the execution time of the insert operation in MongoDB and MySQL.

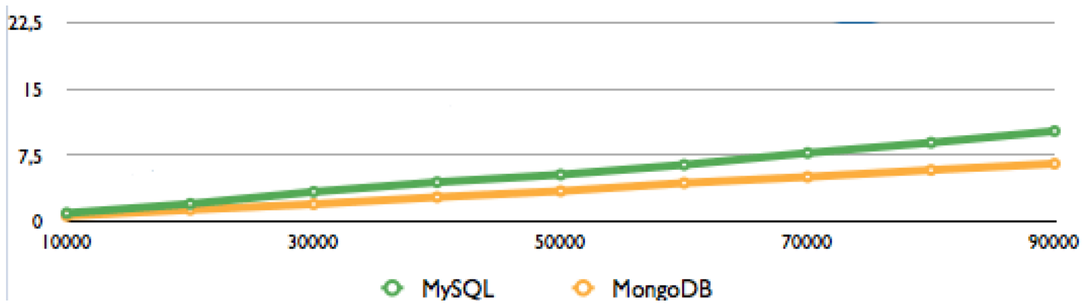


Fig. 1. Insertion execution time

Figs. 2–3 show diagrams of the dependence of query execution time on the sample size.

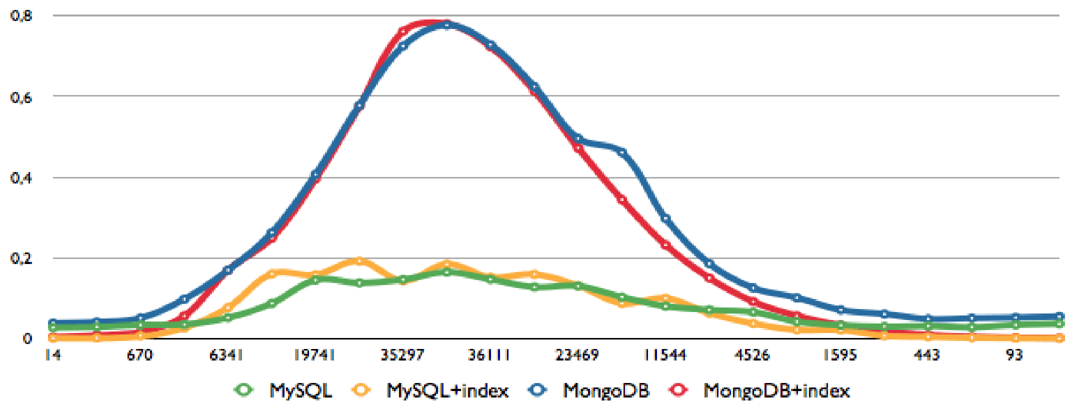


Fig. 2. Full sample of data by word length ranges with and without additional indexes

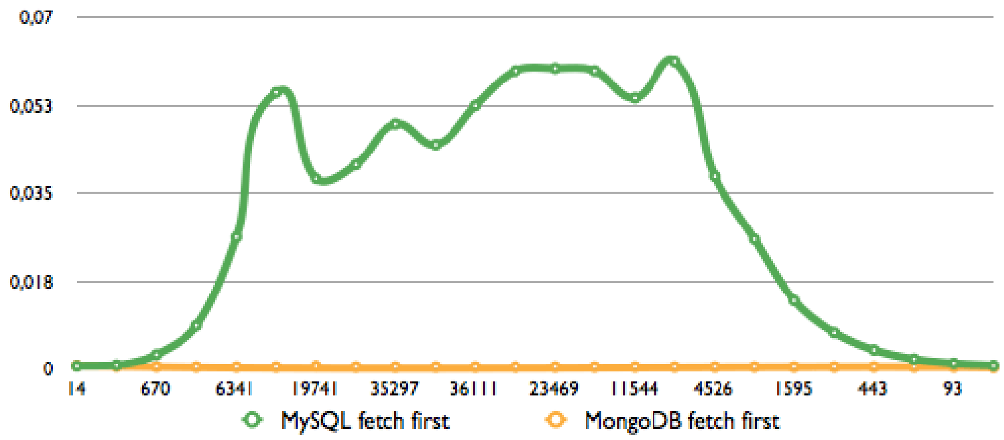
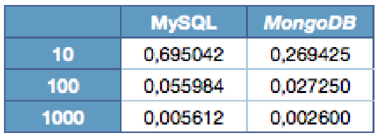


Fig. 3. Processing a request to select only the first result by word length range

Figs. 4–5 show the diagrams of the update and retrieval rates for each 10/100/1000 row.



	MySQL	*MongoDB*
10	0,695042	0,269425
100	0,055984	0,027250
1000	0,005612	0,002600

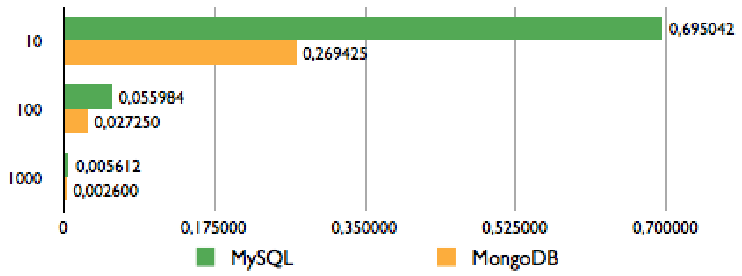


Fig. 4. Updating every 10/100/1000 records

	MySQL	MongoDB
10	0,804231	0,247743
100	0,082997	0,023055
1000	0,008276	0,002383

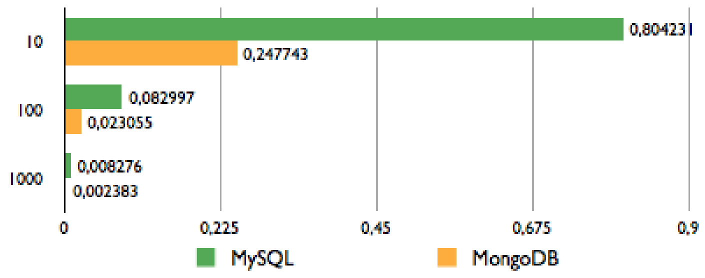


Fig. 5. Extracting every 10/100/1000 records

Development of a method for transition from a relational model to a document-oriented one

When it comes to data modeling, document-oriented databases are not as different from relational databases as other NoSQL solutions. The differences are not that significant, but that does not diminish their importance.

The first and most fundamental difference is that MongoDB does not have an analog of the JOIN construct. JOIN is not scalable. This means that if you start splitting data horizontally, you will still have to perform JOINs on the client (which is the application server). In the worst case, the lack of JOINs in MongoDB will often require an additional query.

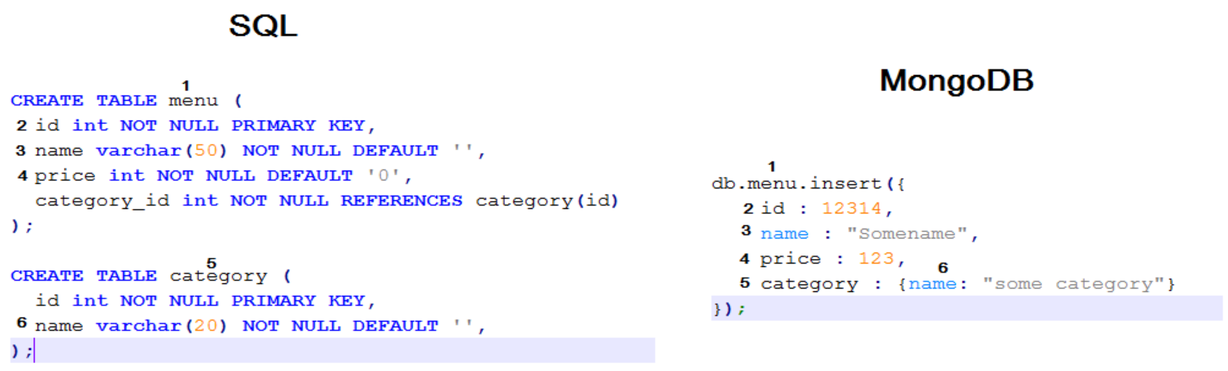
Arrays of values are much more convenient to use than many-to-many tables and provide a significant number of options for implementing multiple inheritance.

In addition to arrays, MongoDB also supports nested documents. They can be queried using dot notation:

MongoDB supports DBRefs. When a driver sees a DBRef, it can automatically retrieve the associated document. A DBRef contains the collection and _id of the document it refers to. This means that documents from one collection can refer to other documents from different collections.

In the process of moving from a relational data model with many tables, relationships, and keys to MongoDB documents, the question arises of the number of collections needed to reproduce all the data from the relational model without losing data. In other words, is it worth following the structure and creating a separate collection in MongoDB for what would be a table in a relational database (many-to-many tables are an important exception). Given that collections do not tie us to a specific schema, you can get by with a single collection that has documents of different structures.

Let's create a MongoDB collection based on the SQL queries from the previous section (Fig. 6).



```
                       1
CREATE TABLE menu (
2 id int NOT NULL PRIMARY KEY,
3 name varchar(50) NOT NULL DEFAULT '',
4 price int NOT NULL DEFAULT '0',
  category_id int NOT NULL REFERENCES category(id)
);
                         5
CREATE TABLE category (
  id int NOT NULL PRIMARY KEY,
6 name varchar(20) NOT NULL DEFAULT '',
);
```

```
     1
db.menu.insert({
  2 id : 12314,
  3 name : "Somename",
  4 price : 123,        6
  5 category : {name: "some category"}
});
```

Fig. 6. Creation requests

As you can see, MongoDB uses the names of the tables and fields from the SQL query (they are marked with numbers), and there is no foreign key in the query, it has turned into a field with a nested document.

Based on the analysis results and conclusions, let's describe the algorithm verbally.

Step 1. Find the "main" table (with the maximum number of foreign keys).

Step 2. Go through each field of the table.

Step 3. Check the field type.

Detailed algorithm of the method of transition from a relational data model to a document-oriented one:

Step 1. Read the original SQL query one by one (get a container of strings).

Step 2. Determine the number of tables (search for the line with the CREATE TABLE statements).

Step 3. For each row container for tables, determine the number of foreign keys (by REFERENCES). The container with the maximum number of foreign keys is passed to step 4. If two or more tables have the same number of foreign keys, then we pass the container with the larger number of rows (fields) to step 4.

Step 4. Get the field name, type, and additional attributes (foreign, primary, etc.) from each row. The type leads to a related data type from MongoDB with the default values set.

Step 5. Validate the fields.

Unlike the relational database model, the MongoDB schema is focused on the application and its interaction with the database. To optimize the model, many factors must be taken into account:

a) access to the field;

b) the frequency of access to a particular field;

c) response time for a particular field, etc.

So, shifting from a relational data model to a document-oriented one does not guarantee faster query processing. Sometimes it is advisable to leave the (model) database schema in its relational form to maintain integrity and logic.

In our opinion, there is no need to use only one collection:

a) when the structure becomes too complicated to understand (more than 3 levels of nesting are more difficult to perceive both by the programmer and the application);

b) when database queries become cumbersome (searching for massive objects in a nested collection within a collection will not speed up the application);

c) when the document exceeds the 16 MB limit on the size of the document (if you exceeded this limit, you probably didn't think about it when you created it).

All these factors directly or indirectly affect the performance of the application. The developed application takes these factors into account and does not greatly complicate the structure of the document, and if the structure becomes more complex or the number of fields in the table increases above the specified one, it partially preserves the relational structure.

The user is free to perform several actions to improve performance: indexing, sharding, and replication. Indexes in MongoDB work in the same way as indexes in relational databases: they speed up data selection and sorting. MongoDB supports auto-sharding. Sharding is an approach to scalability when separate parts of data are stored on different servers. A simple example: store data of users whose name begins with the letters A-M on one server, and H-Z on another.

Replication in MongoDB works in a similar way to replication in relational databases. Records are sent to one server, the master, which then synchronizes its state with other servers, the slaves. The user can allow or disallow reading from the managed servers, depending on whether their system allows reading inconsistent data. If the master server goes down, one of the slaves can take over the role of the main server.

The results of the tests show that MongoDB is much faster at inserting, updating, and deleting records from the database. The data may differ for different data types and structures, but it is safe to say that MongoDB does not lose to MySQL in terms of query execution speed and is a high-performance system that is great for projects with high database update dynamics.

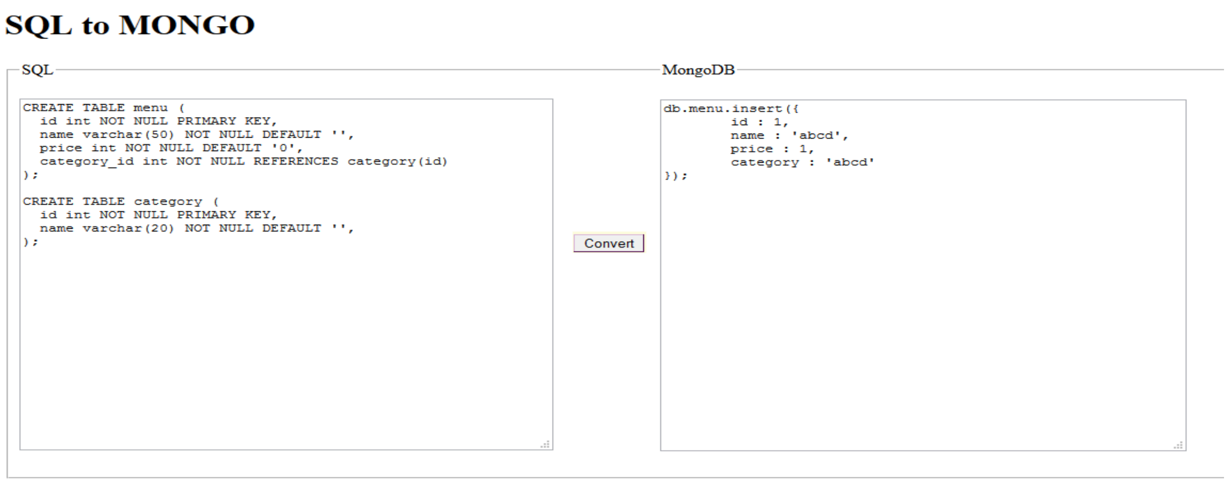
We created an application that implements the developed algorithm for transitioning from the relational to the document-oriented MongoDB data model.

This program generates MongoDB statements to create a document based on SQL queries entered by the user to create tables. The application accepts and correctly processes SQL queries from the DBMS: MySQL, MS SQL, PostgreSQL.

The test results are shown in Figs. 7 i 8. They allow us to talk about the exact operation of the algorithm. Based on the analysis of the test results, it is recommended to:

a) create a field for entering the number of tables;

b) create a separate text field for each SQL query to create a table.

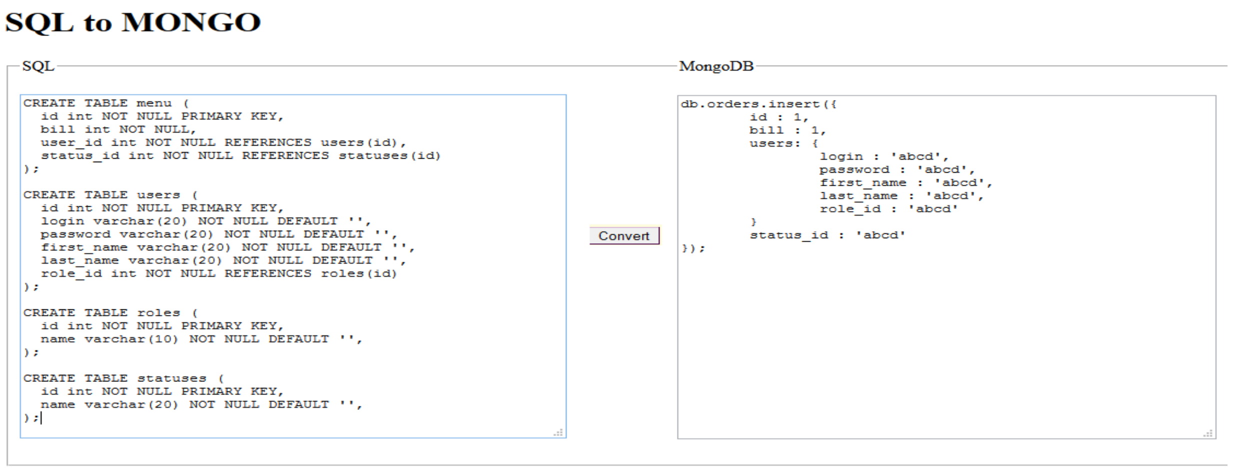


```
CREATE TABLE menu (
  id int NOT NULL PRIMARY KEY,
  name varchar(50) NOT NULL DEFAULT '',
  price int NOT NULL DEFAULT '0',
  category_id int NOT NULL REFERENCES category(id)
);

CREATE TABLE category (
  id int NOT NULL PRIMARY KEY,
  name varchar(20) NOT NULL DEFAULT '',
);
```

```
db.menu.insert({
        id : 1,
        name : 'abcd',
        price : 1,
        category : 'abcd'
});
```

Fig. 7. The result of the program with two tables



```
CREATE TABLE menu (
  id int NOT NULL PRIMARY KEY,
  bill int NOT NULL,
  user_id int NOT NULL REFERENCES users(id),
  status_id int NOT NULL REFERENCES statuses(id)
);

CREATE TABLE users (
  id int NOT NULL PRIMARY KEY,
  login varchar(20) NOT NULL DEFAULT '',
  password varchar(20) NOT NULL DEFAULT '',
  first_name varchar(20) NOT NULL DEFAULT '',
  last_name varchar(20) NOT NULL DEFAULT '',
  role_id int NOT NULL REFERENCES roles(id)
);

CREATE TABLE roles (
  id int NOT NULL PRIMARY KEY,
  name varchar(10) NOT NULL DEFAULT '',
);

CREATE TABLE statuses (
  id int NOT NULL PRIMARY KEY,
  name varchar(20) NOT NULL DEFAULT '',
);
```

```
db.orders.insert({
        id : 1,
        bill : 1,
        users: {
                login : 'abcd',
                password : 'abcd',
                first_name : 'abcd',
                last_name : 'abcd',
                role_id : 'abcd'
        }
        status_id : 'abcd'
});
```

Fig. 8. The result of the program with four tables

Conclusions

The emergence of the new generation of non-relational DBMSs was driven by the need to create parallel distributed systems for highly scalable Internet applications such as search engines, blogs, etc. Document-oriented DBMSs are used to store hierarchical data structures and are used in content management systems, publishing, document retrieval, etc. Examples of DBMSs of this type: CouchDB, Couchbase, MarkLogic, MongoDB, eXist, Berkeley DB XML. Document-oriented DBMSs are based on document repositories that have a tree structure. The tree structure starts with the root node and can contain several internal and leaf nodes. Leaf nodes contain data that is entered into indexes during the process of adding a document, which makes it possible to find the location (path) of the data being searched even with a rather complex structure. The search API finds documents and parts of documents on request. Unlike key-value storages, a query selection for a document repository can contain parts of a large number of documents without loading them into RAM.

The main problem with using document-oriented databases is the complexity of designing a data model. When it comes to data modeling, document-oriented databases are not as different from relational databases as other NoSQL solutions, but there are several significant differences.

When switching from a relational data model to a document-oriented one, the wrong approach is to keep the structure with tables and relationships. MongoDB provides opportunities to build complex documents using nested collections and arrays, and if it is impossible to abandon the relational model, then it allows you to stick to it.

Increasingly, developers are choosing MongoDB for their projects, and the problem they face is a data model that differs from the relational one.

We analyzed the patterns between SQL and MongoDB queries describing the same data, compiled query mappings, and analyzed the problems that arise in the process of model transition. An algorithm for transitioning from a relational model to a document-oriented model based on SQL queries has also been developed.

References

1. Kohalovskyi, M. (2002), Entsyklopediia tekhnolohii baz danykh, *Finansy ta statystyka,* 800 p., ISBN 5-279-02276-4
2. Bisbal, J. Lawless, D., (1999), Legacy systems: issues and directions, *IEEE Software,* Vol. (16), Issue 5, P. 103–111.
3. Tom, L., Prakash, N., (2011), Migrating to the Cloud: Oracle Client/Server Modernization, *Syngress Publishing*, 400 p.
4. William U., Philip N. (2010), Information Systems Transformation: Architecture- Driven Modernization Case Studies, Morgan Kaufmann Publishers, 456 p.
5. Fiaili, K., (2003), SQL: Instruktsiia z vyvchennia movy, *Peachpit Press,* 456 p.
6. Fiaili, K., (2003), SQL: Kerivnytstvo profesionala, *Peachpit Press,* 512 p.
7. DSTU 3330-96, Informatsiini tekhnolohii. Systema standartiv z baz danykh. Etalonna model keruvannia danymy (HOST 34.321-96)
8. Dovidkovyi posibnyk z MySQL, at: http://www.mysql.ru/docs/man/
9. Ofitsiinyi sait PostgreSQL, at: http://www.postgresql.org/
10. The little MongoDB book, at: http://openmymind.net/The-Little-MongoDB-Book/
11. Uolters, R., (2008), SQL Server 2008: pryskorenyi kurs dlia profesionaliv, "Viliams", 768 p.