Aicha Aggoune
Mohamed Sofiane Namoune

METADATA-DRIVEN DATA MIGRATION FROM OBJECT-RELATIONAL DATABASE TO NOSQL DOCUMENT-ORIENTED DATABASE

Abstract *Object-relational databases (ORDB) are powerful tools for managing complex data, but they suffer from problems of scalability and managing large-scale data. Therefore, the importance of the migration of ORDB to NoSQL derives from the fact that the large volume of data can be handled in the best way with high scalability and availability. This paper reports a metadata-driven approach for the migration of ORDB to a document-oriented NoSQL database. Our data-migration approach involves three major stages: a pre-processing stage (to extract data and a schema's components), a processing stage (to provide data transformation), and a post-processing stage (to store migrated data as BSON documents). This approach maintains the benefits of Oracle ORDB in NoSQL MongoDB by supporting integrity constraint checking. To validate our approach, we developed the OR2DOD (object relational to document-oriented database) system, and the experimental results confirm the effectiveness of our proposal.*

Keywords NoSQL document-oriented database, object-relational database, data migration, mapping rules, metadata

Citation Computer Science 23(4) 2022: 495–519

1. Introduction

Many contemporary companies include complex data sets that are stored in object-relational databases [13, 28]. These databases were modeled by a hybrid data model that combines the advantages of the relational data model and the modeling primitives of the object-oriented paradigm to properly represent complex data, support rich data types, and address the object-relational impedance mismatch [31]. The hybrid data model is the object-relational model that allows for the natural representation of complex data (e.g., geospatial and multimedia data) using various concepts such as objects, identifiers, inheritances, encapsulation, and polymorphism [12]. In object-relational databases, the data is stored in different tables that are related to each other using references and collection data types [9]. Access to this data is easy by using what is known as the SQL:1999 structured query language and may be difficult when using the PL/SQL procedural language [14]. The ORDBMS object-relational database management system provides great support for the object-oriented programming language, and it is expected to become increasingly important for handling a large volume of complex data. ORDBMS supports transaction-oriented applications that provide data consistency within multiple partitions of a node [24]. However, the tremendously increasing volume of multiple data types (multimedia, embedded data, etc.) makes ORDBMS very complex, and its performance degrades rapidly [27]. Also, the data heterogeneity and rigid schema introduce a storage problem when the data to be inserted is not adequate [20].

In 1998, Carlo Strozzi coined the term NoSQL (not only SQL) to describe a new approach for managing a large amount of heterogeneous data [33].

NoSQL databases provide a good solution for supporting a large scale of different data types, enabling high horizontal scalability and availability that cannot be easily accomplished in ORDB [10]. In contrast to the object-relational database, which is based on a strict schema with integrity constraints, the NoSQL schema is characterized by a schema-less where data can be stored without any defined database schema [21]. A NoSQL database can also be modeled by different data models; therefore, different kinds of NoSQL databases can be distinguished. They are classified into four categories: key-value, wide columnar, document, and graph databases [10].

To efficiently and flexibly manage the large existing volume of object-relational databases, it is necessary to transform and migrate them to NoSQL databases rather than creating NoSQL data from scratch. Several approaches and frameworks have focused on the migration of relational databases to NoSQL [22]. These solutions are not suitable for the migration of object-relational databases. So far, very little research has investigated this type of data migration.

The main contribution of this paper is to ensure the migration of object-relational databases to NoSQL (which are more complex than relational databases). In this study, we focused on the MongoDB document-oriented database. MongoDB is the most popular NoSQL database and includes JSON- and BSON-based documents for storing data with replication that leads to high scalability, availability, auto-sharding, and data querying [30].

Our data-migration approach is based on the use of metadata that contains a list of mapping rules between the object-relational and NoSQL models. This metadata can be used for many data-migration activities such as data reconciliation, data integration, data repairing, and data management. The proposed data migration adopts a mapping process that involves three stages: a pre-processing stage (to extract data and a schema's components), a processing stage (to provide data transformation), and a post-processing stage (to store the migrated data as BSON documents).

The rest of this paper is organized as follows. Section 2 reviews the related works of data migration. Section 3 introduces an overview of our data-migration approach, while Section 4 describes the metadata that is used in this approach. Section 5 presents our proposal in detail. Section 6 shows the experimental results in order to demonstrate the performance of our OR2DOD system. Section 7 ends with our conclusions and future directions.

2. Related work

Recently, several researchers in the data-migration domain have focused on the transformation from relational databases to NoSQL. Generally, the existing studies have been focused on either relational or object-oriented data migration.

In relational data migration, different approaches have been proposed such as conversion rules-based approaches, which aim at defining a set of conversion or translation rules that applying to a relational database [7, 25, 32]. Some works are based on the integration of a mid-model that is used between source data and target data [3, 18, 19]. Rocha et al. [29] developed a complete framework for the auto-migration of relational to NoSQL document-oriented databases. This framework provides a seamless NoSQL layer for the transformation of SQL queries to NoSQL ones; thus, the results of any NoSQL queries must be transformed to relational data, which will then be sent to the user.

In addition, several object-NoSQL data-mapping (ONDM) frameworks have been developed for the migration of an object-oriented database to NoSQL [27]. As an example, the Hibernate OGM framework (OMG) [16] supports the migration of an object-oriented database to three types of NoSQL stores: MongoDB document-oriented, Neo4j graph-oriented, and Infinispan key-value databases.

These works are all confined to migrate from the relational/object databases to NoSQL databases. Unfortunately, these proposals are not fitted to the migration of ORDB to NoSQL. Despite the many advantages of the object-relational model, it is not appropriate for high speeds nor huge quantities of data (also known as big data) [8, 24]. The migration of a large volume of ORDB to NoSQL is a promising way to data manage with high scalability and availability.

While the migration of relational databases to NoSQL is a well-studied field of research, there is only one solution that focuses on the migration of object-relational databases. In [15], the authors proposed an approach for model transformation from an object-relational database (ORDB) to a NoSQL document-oriented database. This approach involves the use of an intermediate layer between the ORDB and the document-oriented model. The goal of this layer is to generate a data model of an object-relational database by extracting all of the elements of its logical schema such as abstract data type, relationships, tables, etc. Thus, a set of transformation rules is defined and applied to the generated data model to produce a NoSQL document model as an output. This contribution is still a preliminary work, and the proposal lacks insightful analysis. A more verbose comparison of our work to this related work is shown in Table 1.

Table 1
Comparison between our work and related work

Our work	Related work
Migration from object-relational database of Oracle to document-oriented database of MongoDB	Migration from object-relational database of Oracle to document-oriented database of MongoDB
Development of OR2DOD system	Ongoing research
Data migration	Model migration
Metadata-driven approach	Rules-guided approach
Ensuring Constraint migration	Only migration of primary key
Ensuring migration of Index, nullable, check, and default constraints	Nothing
Experimental analysis	Demonstrative examples without implementation

Compared to the related work that is presented in [15], our work defines a complete solution that enables users to migrate their object-relational databases to NoSQL document-oriented databases.

3. Overview of metadata-driven data-migration approach

We propose a metadata-driven approach for the migration of an object-relational database to a NoSQL document-oriented database. Moving to the leading new technology (termed NoSQL) is a suitable solution for storing and handling this next-generation data.

The aim of our data-migration approach was twofold. First, we addressed the main drawback of object-relational database management systems, which have been unable to live up to expectations when a large volume of data must be stored and processed. Second, we maintained the benefits of both the object-relational and NoSQL technologies (such as scalability to handle huge amount of data and integrity constraints to ensure data quality). The authors have presented a preliminary proposal that pays less attention to the problem of data migration from an object-relational database (ORDB) to a NoSQL database [2]; thus, it does not study integrity constraint management from the data-migration perspective.

In this paper, we present a metadata-driven approach for data migration. We also present OR2DOD (object-relational to document-oriented database) – a system for validating our proposal.

The approach involves our process to define three major stages: the pre-processing stage (for data acquisition and extraction), the processing stage (to perform the data transformation), and the post-processing stage (to provide adjustments of the constraints and data importing). Figure 1 pictorially shows an overview of the proposed approach.

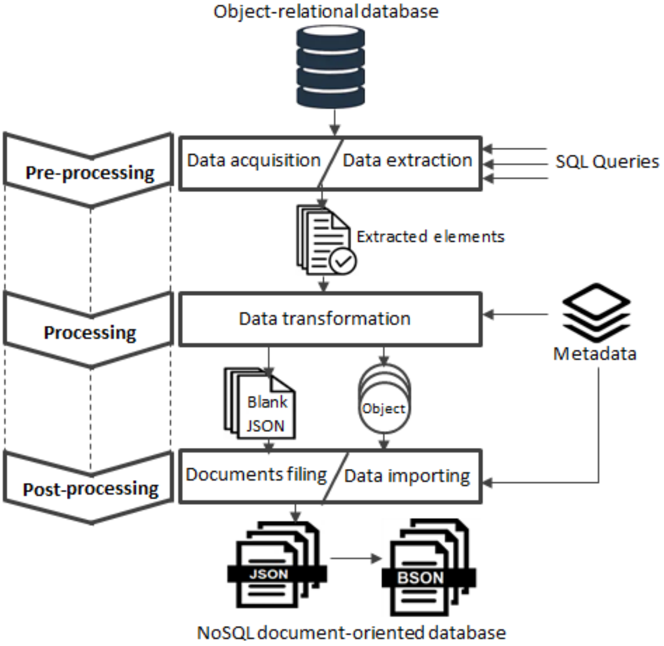


Figure 1. Overview of metadata-driven data-migration approach

The goal of data pre-processing is to acquire an original database that is modeled by an object-relational model and extract the data and schema components (table name, data type, column name, column value, constraints, etc.). This stage essentially relies on SQL queries in order to extract these elements, which will be stored in lists and used during the processing stage. This stage is the fundamental phase in the data-migration process. The processing stage is guided by the metadata of mappings between the source and target models; it aims to perform the migration by automatically creating blank JSON files from the parent and super tables. JSON (JavaScript object notation) is the principal format that is used to store data in a NoSQL document-oriented database [26, 32]. Due to the lack of schema in the target model (the NoSQL model), we make sure to ignore all empty columns (columns with a null value) while transforming the data so as not to transform any missing data.

The final stage aims at filling the objects in the appropriate blank JSON files and adding some constraints by using the metadata. After this, the generated JSON files can be imported in the BSON (binary JSON) format, which is directly managed by the MongoDB system.

Furthermore, this approach provides a mechanism for using integrity constraints as a target persistence backend in order to take advantage of the full benefits of the object-relational model. This combination makes it possible to handle a huge amount of data while retaining the advantages of both the object-relational and NoSQL document-oriented models.

4. Metadata description

As previously explained, our data-migration approach is guided by metadata that includes mapping rules that allow one to generate a NoSQL document-oriented database from an object-relational database. The motivation of the metadata is to minimize the migration costs for applications that access object-relational databases that are intended to be moved to NoSQL databases. To represent such mappings of our metadata, it is helpful to briefly describe the data model of both the object-relational and NoSQL document-oriented databases.

The object-relational model is an extended relational one for supporting the non-first normal form (NF2), which allows attributes to have complex types by using abstract data types (ADT). This intends to manage diverse data types by incorporating the object-oriented features into relational models, such as objects, methods, which are written in PL/SQL, encapsulation, inheritance, and polymorphism [31]. The object-relational model allows users to create their data-structured types by using ADT [9]. The latter is defined by four elements:

- ADT name;
- ADT type (object, REF reference to another ADT, and collection data types);
- list of attributes that are associated with domain that is built-in datatype or other ADT;
- list of methods signatures for representing behavior of ADT.

For example, the following SQL statement allows for the creation of an ADT named `Author_t` as an object type with three attributes (`Code, Name, Email`): `CREATE TYPE Author_t AS OBJECT (Code Integer, Name varchar2(40), email Varchar2(40))`.

ADT offers multi-valued attributes by using collection-data types such as varrays (arrays with variable sizes), nested tables, etc. [17]. These collection-data types differ from one ORDBMS to another. In our work, we focused on the best ORDBMS (the so-called Oracle). The object-relational table (also called the NF2 table) was created through the object ADT with a set of constraints. For example, we can create a table that is termed `Author` from the `Author_t` object type as follows: `CREATE TABLE Author OF Author_t (Primary key (Code))`.

In contrast to the relational model, a relationship between the NF2 tables can be achieved without using foreign keys [12]. Indeed, the relationship can be made according to the association between the object types on which the NF2 tables were created. There are two types of associations between the object types of NF2 tables: aggregation associations, and symmetric associations [11, 23]. The aggregation association (also called total nesting) is when the tables are nested within other tables as values in a column. In other words, the data type of this column is an object type (in the case of a one-to-one relationship) or collection-data types (in the cases of one-to-many or many-to-many relationships) like nested tables and varrays. The symmetric association (also called partial nesting) is when tables are created with a column whose data type is a reference or a collection of references to other tables.

The target data of the migration approach is modeled by the NoSQL document-oriented model. The document-oriented database stores data in the form of semi-structured documents that are fully schemaless (like the JSON and BSON formats) [10]. The document can be considered as a row or object in an NF2 table. Each document has a set of fields or attributes that are associated with ordinary values, complex values (such as references to documents), embedded documents, and a list of values or references. A set of documents that represent the same entity are organized in the so-called collection, which is the equivalent of an object-relational table [6]. The relationship in the document-oriented database can be modeled by using references or embedded documents [29]. We use one of the most popular NoSQL document-oriented database systems called MongoDB, which was developed by 10gen and an open-source community [6].

The list of the mapping rules that is included in the metadata is divided into three categories: component mapping, data mapping, and constraint mapping. Component mapping (CM) intends to convert the principal components of an object-relational schema into a NoSQL document model; it consists of four mapping rules:

CM1. The schema's name of the ORDB corresponds to the name of the NoSQL document-oriented database.

CM2. In the case of aggregation association, the name of the parent table corresponds to the name of the collection.

CM3. In the case of symmetric association, the names of related tables correspond to the names of collections.

CM4. The super table with its sub-tables that are produced by the inheritance between their object types is mapped to one collection (which takes the same name as the super table). This rule is motivated by the fact that the NoSQL schema is very flexible in storing data. So, the generated documents with minimum fields probably represent the content of the super table, and the documents with maximum fields represent the objects of the sub-tables.

Note that defining collections involves the creation of blank JSON files. In the second category of the mapping, we describe the data-mapping (DM) rules to transform the object-relational data into documents.

DM1. In the case of aggregation association, each row of parent tables corresponds to an empty document.

DM2. In the case of symmetric association, each row of related tables corresponds to an empty document.

DM3. Each row of both the super table with its sub-tables corresponds to an empty document.

DM4. If a column of a table has a null value, then it has nothing to do in the NoSQL document-oriented model.

DM5. Each atomic column of the mapped tables is represented in the appropriate document as a field that is associated with a column value.

DM6. Each column has a structured datatype (an object or row) and is mapped to a field (where its value is an empty embedded document).

DM7. Each column represents a nested table or a varray of values that is mapped to a list of empty embedded documents.

DM8. Each atomic column of a nested table (or of a structured type) is mapped to a field with a column value and added to the field's list of its mapped embedded documents.

DM9. Each column has a reference REF as a data type to a row of the second table (symmetric association) and is mapped to a field (where its value is a reference to a document of the second collection, which was previously generated).

DM10. Each column has a data type such as a nested table or varray of references to the second table and is mapped to a field (where its value is a list of references to documents of the second collection, which was previously generated).

The third category of mapping (termed constraint mapping – COM) is typically used during the post-processing stage. When documents are located in the appropriate JSON files, we translate the different constraints into a NoSQL document-oriented database.

COM1. The primary key of a table (simple primary or composite primary keys that contain multiple columns) corresponds to the creation of a unique index on one or multiple fields of the appropriate collection.

COM2. The unique keys of a table correspond to the creation of unique indexes.

COM3. Each index (or unique index) is mapped to the creation of an index (or unique index).

COM4. The sequence has auto-increment functionality that has a name (SN), initial value (IV), and increment value (step – SV). Each sequence is mapped to the creation of a function that takes SN and SV values as inputs and return an updated sequence number. The initial value (IV) is defined as a new collection called counters, which is used in the aforesaid function.

COM5. The check constraint allows one to specify a condition on each row in a table. This condition is mapped to the creation of function `CHK`, which is integrated into the new NoSql database that takes the benefits of the object-relational model. The created `CHK` function is very important for inserting and updating the documents of a new NoSQL database.

COM6. The default constraint indicates a default value for a column. This option is mapped to the creation of function `DFLT`, which contains all of the default values that are related to the corresponding fields of a new NoSQL database.

COM7. The `not null` constraint is very important for enforcing the completeness of a column value [1]. This means that we cannot insert or update a row without providing a value to this column. This constraint is also mapped to the creation of the `NUF` function, which defines a condition for each value of a specified field of a new NoSQL database.

5. Detailed approach

To illustrate the data-migration process, consider an example of the object-relational database that was used in our experimental study. This database specified a statistical data set on France's population distribution (FPD), which is available at the SQL.sh website (http://www.sql.sh). To clearly describe the FPD schema, we have adjusted all of the names of the schema components to English. The principal part of the object-relational model is shown in Figure 2.

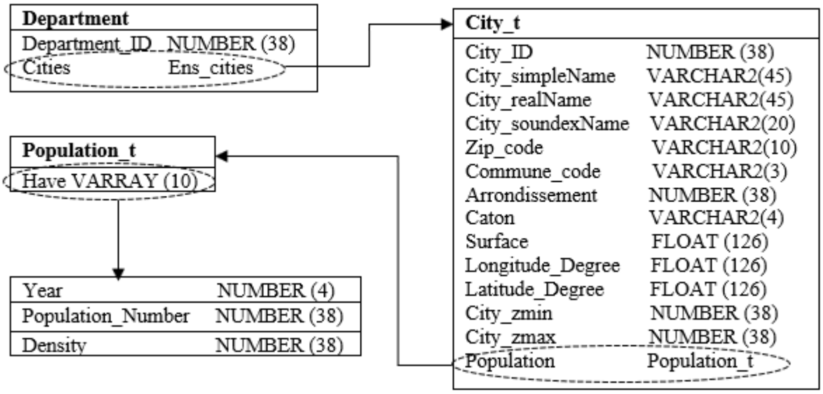
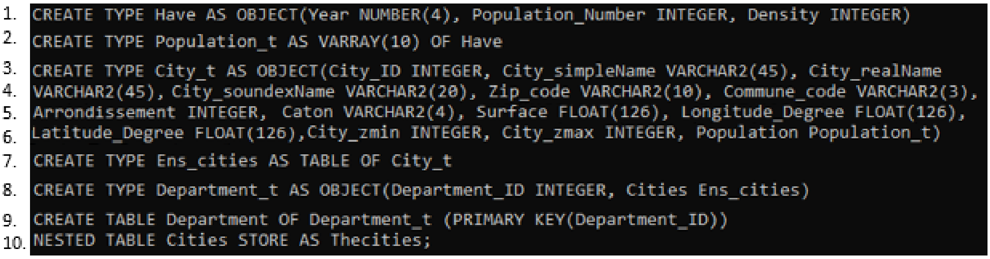


Figure 2. Object-relational model of FPD database

Figure 3 illustrates the SQL statements for creating the schema of FPD.

Our FPD schema is defined by the `Department` relationship, which describes the territorial division of France. The departments are further subdivided into cities, which in turn contain the important properties of each city and population (such as arrondissement, Caton, surface, etc.). The population of each city is defined by three properties: `Year` (defining the year of the population distribution), `Population_Number` (presenting the number of people), and `Density` (specifying the population density). We take an example of ten years of population data (between 1999 and 2019).



```
1.  CREATE TYPE Have AS OBJECT(Year NUMBER(4), Population_Number INTEGER, Density INTEGER)
2.  CREATE TYPE Population_t AS VARRAY(10) OF Have
3.  CREATE TYPE City_t AS OBJECT(City_ID INTEGER, City_simpleName VARCHAR2(45), City_realName
4.  VARCHAR2(45), City_soundexName VARCHAR2(20), Zip_code VARCHAR2(10), Commune_code VARCHAR2(3),
5.  Arrondissement INTEGER, Caton VARCHAR2(4), Surface FLOAT(126), Longitude_Degree FLOAT(126),
6.  Latitude_Degree FLOAT(126),City_zmin INTEGER, City_zmax INTEGER, Population Population_t)
7.  CREATE TYPE Ens_cities AS TABLE OF City_t
8.  CREATE TYPE Department_t AS OBJECT(Department_ID INTEGER, Cities Ens_cities)
9.  CREATE TABLE Department OF Department_t (PRIMARY KEY(Department_ID))
10. NESTED TABLE Cities STORE AS Thecities;
```

Figure 3. SQL statements for creating object-relational schema of FPD

The type of relationship between departments and cities is a one-to-many relationship with an aggregation association between their object ADT, because the Department table dominates the Cities table (each city must be included in its department). Therefore, we define an additional attribute called `Cities` in the object type of the Department table (termed `Department_t`). This attribute has a collection data type called `Ens_cities` (ensemble of cities) (Line 8), which is defined through the object ADT `City_T` (Line 7). `City_t` is defined by a set of the properties of a city (like surface, latitude, etc. – Lines 3–6). For each city, we also need to know information about its population for each of the years. We take the last ten years as an example. So, we define the population's information by a varray data type (termed `Population_t` of ten elements – Line 2). Each element of `Population_t` has the object ADT called `Have`, which is presented by the three aforementioned attributes (Line 1).

Finally, the `Department` table is created via the `Department_t` object type, with `Department_ID` as a primary key (Line 9). The nested table data of the `Cities` column will be stored in an appendix table named `Thecities`, which is accessed indirectly via the `Cities` complex column of the `Department` table (Line 10).

The data-migration process starts with a data investigation of the original database. We establish a set of SQL queries to automatically extract the components of the object-relational schema. Table 2 presents some SQL queries that were used in this stage.

Each type of extracted component will be stored in the lists of the string; for example, `Parent-list` includes all names of the Parent tables, `column_Nestedtable` contains the names of the columns of the extracted Nested table, etc. We continue with the example of the FPD database. The output of the pre-processing stage is introduced in Table 3.

Table 2

Some SQL queries of pre-processing phase

SQL query	Description
`SELECT Tname FROM Tab`	extracting the names of all of the existing tables

`SELECT DISTINCT Table_name`
`FROM ALL_NESTED_TABLE_COLS`
`WHERE owner=schema's name`
	extracting the names of all of the existing nested tables from the specific schema's name

`SELECT column_name, data_type`
`FROM SYS.ALL_TAB_COLUMNS`
`WHERE owner =schema's name AND`
`Table_name=selected table`
	extracting the column_name with its data type of the selected table defined in the table name from the specific schema that is presented by the owner property

Table 3

Output of pre-processing phase

Output list	Comment
`Parent-list=[Department]`	List of the parent tables. We have one table `Department`
`Nested-list=[Thecities]`	The nested table of the column `Cities` is represented by the appendix table `Thecities`
`Column_Parent (Department)= [Department_ID,Cities]`	Columns of the `Department` table
`Column_nestedtable(Thecities)= [City_id, City_simpleName, City_realName, City_soundexName, Zip_code, Commune_code, Arrondissement, Canton, Surface, Longitude_Degree, Latitude_Degree, City_zmin, City_zmax, Population]`	Columns of the nested table `Thecities`
`Column_array(Population)=(Have)`	Columns of the varray called Population
`Column_structured(Have)= (Year, Population_Number, Density)`	Columns of the structured type termed `Have`
`Constraint-parent(Department)= (primary key (Department_ID), Nestedtable (Thecities, cities))`	`Department` table has two properties: the primary key constraint and the nested table `Thecities` of the column `cities`
`Constraint-Nested(Thecities)= (primary key(City_ID))`	The primary key of the nested table `Lescities`

The atomic and referenced values of each column are also stored in lists. Thus, each complex value is represented by a set of column values of the corresponding data type. For example, the `Population` column of the `Thecities` nested table has a varray data type; namely, `Population_t`. So, each value of `Population` is defined by a list of the column values of the `Population_t` type.

The output of the pre-processing stage becomes the input for the processing stage, which provides the data transformation into the NoSQL document-oriented model. The result of the transformation process is a set of blank JSON files and a list of JSON objects.

Referring to our example, the output of the data-transformation stage is one blank JSON file (i.e., one collection), which takes the same name as the Departement parent table (applying the CM2 rule). Thus, a set of JSON objects is produced, which means the documents of the NoSQL document-oriented database. The following statements show the creation of the NoSQL database and the Departement collection, respectively:

- Use ⟨Schema's name⟩ // The name of the NoSQL database is the same name as the user schema of FPD.
- db.createCollection(⟨Parentlist.get(i)⟩), where i indicates the i^{th} name of the parent table.

An example of the creation of the JSON object from the Department table is as follows:

```
{” Department_ID”:”045”,
”Cities”:[
{”City_ID”:”16836”,”City_simpleName”:”selle-sur-le-bied”,
”Population”:[{”Year”:2017, ”Population_Number”:1012,
”Density” : 23,93}]},
{”City_ID”:”16837”,”City_simpleName”:”bouzy-la-fore”}, ”Population”:
[{”Year” : 2017, ”Population_Number” : 1221, ”Density” : 33}]}
] }
```

The documents nesting (equivalent to nesting tables in ORDB) is based on the Constraint-Parent and Constraint-Nested lists. From the Constraint-Parent Department, we have the property Nestedtable(Thecities, Cities), which means that all JSON objects of the nested table Thecities become the embedded documents in the Department collection. These embedded documents are defined by the field called Cities.

The post-processing stage is divided into three steps:

1. Filing the JSON objects in the appropriate JSON files.
2. Adding constraints. This final stage is based on the constraint mapping rules of metadata to transform the extracted constraints of the ORDB into functions of the NoSQL document-oriented database. In our example, we have two lists of constraints: Constraint-Parent(Department) and Constraint-Nested(Thecities). In the first list of constraints, the primary key of the Department table becomes the unique index that is created as follows: db.Department.ensureIndex("Department_ID":1, "unique":true). The same case applies in the second list of constraints, where we create the unique index of the embedded documents.

3. Importing the generated JSON files to the BSON that is directly managed by the MongoDB system.

The generated JSON files are well used not only in MongoDB system but also in any NoSQL document-oriented database systems such as CouchDB, DynamoDB, etc. Using MongoDB is for ensuring the matching between SQL and the Mongo language as well as importing the generated JSON files to the BSON that is directly managed by the MongoDB system.

6. Performance and experimental results

For the experimental study, we implemented the OR2DOD (object-relational to document-oriented database) system to validate our data-migration approach. We created two object-relational databases: the first was based on the data that was available at the SQL.sh website (http://www.sql.sh). This database describes information about the distribution of the population throughout France, which is presented in Section 5; it contains 36,700 records. The second database describes a running example of an e-commerce application that defines the relationships between providers and products in enterprises; at aims to present a many-to-many relationship, which is interpreted by the creation of a nested table of another related table in each table. The `E-commerce` database contains 45,000 records and is composed of three tables: Provider, Product, and Enterprise. These tables are related by many-to-many relationships where a provider can offer many different products to many enterprises; on the other hand, there are many providers that an enterprise works with. Figure 4 illustrates the object-relation model of the second database.

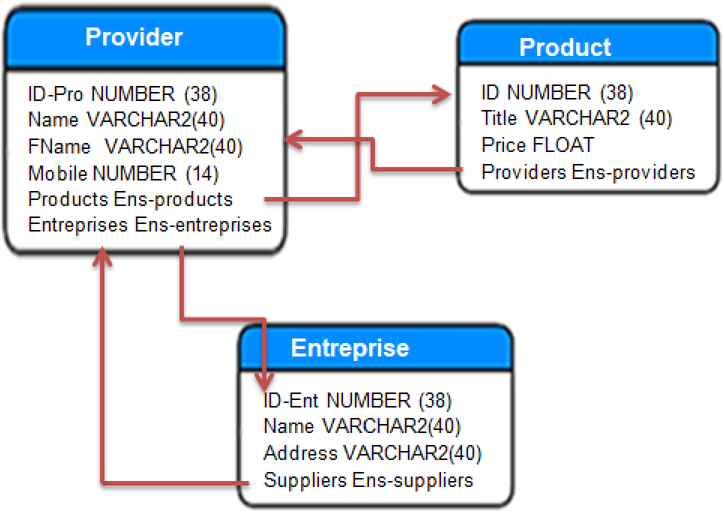


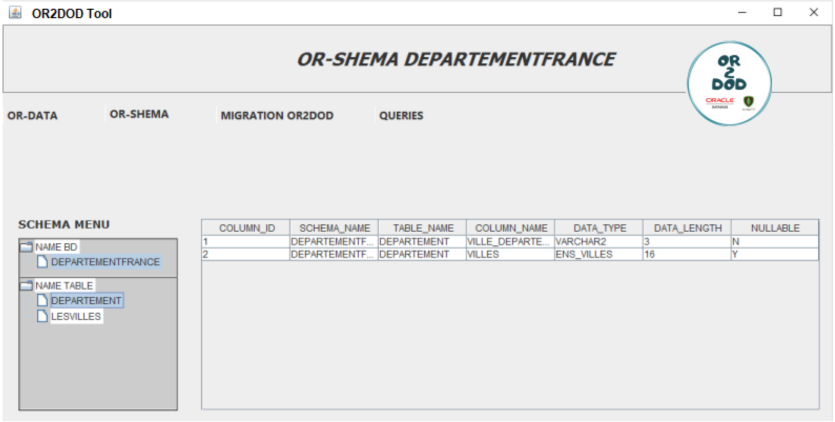
Figure 4. Object-relational model of E-commerce database

Both databases were implemented using Oracle (Release 11.2.0.2.0.), which provided the ability to view tables back in time and offered grid-computing functions. The output database of the data-migration framework was handled by using MongoDB (Version 3.2.22), which presented good performance in the replication and sharding of large volumes of BSON documents. The OR2DOD system was developed by using the Java language, and all of the experiments were performed on a computer that was equipped with an Intel Core i5 with 2.50 GHz, 4 GB RAM, and the 64-bit version of the Windows 10 operating system.

The OR2DOD tool not only offered data migration but also other functionalities (see Figure 5):

- `OR-DATA`: to display object-relational data via SQL queries.
- `OR-SCHEMA`: to display object-relational schema via standard SQL queries of the Oracle system.
- `Queries`: to ensure three kinds of data update queries (INSERT, DELETE, and UPDATE). This functionality was very important when we needed to increase/decrease/change the input and output data during the experimental study.

Figure 5 depicts the OR-SCHEMA of the FPD database as an example of one of the aforementioned OR2DOD functionalities. We kept all of the names of the departments of France in French, where 'Lesvilles' in the Schema menu meant the `Thecities` nested table. The column named 'Column_name' of the presented table returned all columns of the selected table (for example, Department) where the values Ville_departement:Integer and Villes:Ens_villes present respectively, `Departement_ID` and `Cities`, which is defined by a datatype called `Ens_cities`.

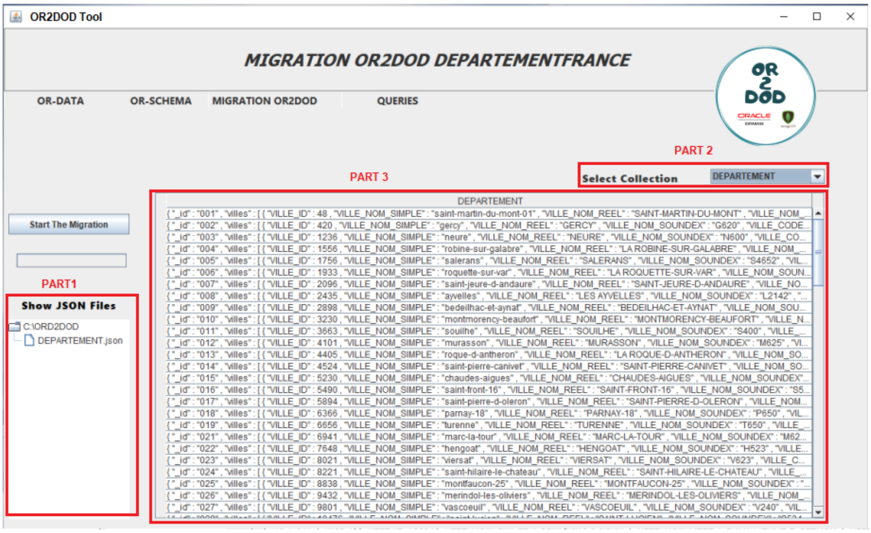


COLUMN_ID	SCHEMA_NAME	TABLE_NAME	COLUMN_NAME	DATA_TYPE	DATA_LENGTH	NULLABLE
1	DEPARTEMENTF...	DEPARTEMENT	VILLE_DEPARTE...	VARCHAR2	3	N
2	DEPARTEMENTF...	DEPARTEMENT	VILLES	ENS_VILLES	16	Y

Figure 5. Visualizing object-relational schema in OR2DOD tool

Figure 6 presents an example of data migration in the OR2DOD system.

In Figure 6, the OR2DOD system provided the transparent migration for users by pressing the "Start the migration" button of the selected object-relational database and viewing the data-migration progression with the result in two different ways. In the first mode, the user can click a JSON file's name to download it (see Part 1). The second mode is meant to select the generated collections that are given in Part 2 and show its contents in Part 3.



```
DEPARTEMENT
{ "_id" : "001" , "villes" : [ { "VILLE_ID" : 48 , "VILLE_NOM_SIMPLE" : "saint-martin-du-mont-01" , "VILLE_NOM_REEL" : "SAINT-MARTIN-DU-MONT" , "VILLE_NOM_...
{ "_id" : "002" , "villes" : [ { "VILLE_ID" : 420 , "VILLE_NOM_SIMPLE" : "gercy" , "VILLE_NOM_REEL" : "GERCY" , "VILLE_NOM_SOUNDEX" : "G620" , "VILLE_CODE...
{ "_id" : "003" , "villes" : [ { "VILLE_ID" : 1236 , "VILLE_NOM_SIMPLE" : "neure" , "VILLE_NOM_REEL" : "NEURE" , "VILLE_NOM_SOUNDEX" : "N600" , "VILLE_CO...
{ "_id" : "004" , "villes" : [ { "VILLE_ID" : 1556 , "VILLE_NOM_SIMPLE" : "robine-sur-galabre" , "VILLE_NOM_REEL" : "LA ROBINE-SUR-GALABRE" , "VILLE_NOM_...
{ "_id" : "005" , "villes" : [ { "VILLE_ID" : 1756 , "VILLE_NOM_SIMPLE" : "salerans" , "VILLE_NOM_REEL" : "SALERANS" , "VILLE_NOM_SOUNDEX" : "S4652" , "VIL...
{ "_id" : "006" , "villes" : [ { "VILLE_ID" : 1933 , "VILLE_NOM_SIMPLE" : "roquette-sur-var" , "VILLE_NOM_REEL" : "LA ROQUETTE-SUR-VAR" , "VILLE_NOM_SOUN...
{ "_id" : "007" , "villes" : [ { "VILLE_ID" : 2096 , "VILLE_NOM_SIMPLE" : "saint-jeure-d-andaure" , "VILLE_NOM_REEL" : "SAINT-JEURE-D-ANDAURE" , "VILLE_NO...
{ "_id" : "008" , "villes" : [ { "VILLE_ID" : 2435 , "VILLE_NOM_SIMPLE" : "ayvelles" , "VILLE_NOM_REEL" : "LES AYVELLES" , "VILLE_NOM_SOUNDEX" : "L2142" , "...
{ "_id" : "009" , "villes" : [ { "VILLE_ID" : 2898 , "VILLE_NOM_SIMPLE" : "bedeilhac-et-aynat" , "VILLE_NOM_REEL" : "BEDEILHAC-ET-AYNAT" , "VILLE_NOM_SOU...
{ "_id" : "010" , "villes" : [ { "VILLE_ID" : 3230 , "VILLE_NOM_SIMPLE" : "montmorency-beaufort" , "VILLE_NOM_REEL" : "MONTMORENCY-BEAUFORT" , "VILLE_N...
{ "_id" : "011" , "villes" : [ { "VILLE_ID" : 3663 , "VILLE_NOM_SIMPLE" : "souilhe" , "VILLE_NOM_REEL" : "SOUILHE" , "VILLE_NOM_SOUNDEX" : "S400" , "VILLE_...
{ "_id" : "012" , "villes" : [ { "VILLE_ID" : 4101 , "VILLE_NOM_SIMPLE" : "murasson" , "VILLE_NOM_REEL" : "MURASSON" , "VILLE_NOM_SOUNDEX" : "M625" , "VI...
{ "_id" : "013" , "villes" : [ { "VILLE_ID" : 4405 , "VILLE_NOM_SIMPLE" : "roque-d-antheron" , "VILLE_NOM_REEL" : "LA ROQUE-D-ANTHERON" , "VILLE_NOM_SO...
{ "_id" : "014" , "villes" : [ { "VILLE_ID" : 4524 , "VILLE_NOM_SIMPLE" : "saint-pierre-canivet" , "VILLE_NOM_REEL" : "SAINT-PIERRE-CANIVET" , "VILLE_NOM_SO...
{ "_id" : "015" , "villes" : [ { "VILLE_ID" : 5230 , "VILLE_NOM_SIMPLE" : "chaudes-aigues" , "VILLE_NOM_REEL" : "CHAUDES-AIGUES" , "VILLE_NOM_SOUNDEX"...
{ "_id" : "016" , "villes" : [ { "VILLE_ID" : 5490 , "VILLE_NOM_SIMPLE" : "saint-front-16" , "VILLE_NOM_REEL" : "SAINT-FRONT-16" , "VILLE_NOM_SOUNDEX" : "S5...
{ "_id" : "017" , "villes" : [ { "VILLE_ID" : 5894 , "VILLE_NOM_SIMPLE" : "saint-pierre-d-oleron" , "VILLE_NOM_REEL" : "SAINT-PIERRE-D-OLERON" , "VILLE_NOM...
{ "_id" : "018" , "villes" : [ { "VILLE_ID" : 6366 , "VILLE_NOM_SIMPLE" : "parnay-18" , "VILLE_NOM_REEL" : "PARNAY-18" , "VILLE_NOM_SOUNDEX" : "P650" , "VIL...
{ "_id" : "019" , "villes" : [ { "VILLE_ID" : 6656 , "VILLE_NOM_SIMPLE" : "turenne" , "VILLE_NOM_REEL" : "TURENNE" , "VILLE_NOM_SOUNDEX" : "T650" , "VILLE_...
{ "_id" : "021" , "villes" : [ { "VILLE_ID" : 6941 , "VILLE_NOM_SIMPLE" : "marc-la-tour" , "VILLE_NOM_REEL" : "MARC-LA-TOUR" , "VILLE_NOM_SOUNDEX" : "M62...
{ "_id" : "022" , "villes" : [ { "VILLE_ID" : 7648 , "VILLE_NOM_SIMPLE" : "hengoat" , "VILLE_NOM_REEL" : "HENGOAT" , "VILLE_NOM_SOUNDEX" : "H523" , "VILLE...
{ "_id" : "023" , "villes" : [ { "VILLE_ID" : 8021 , "VILLE_NOM_SIMPLE" : "viersat" , "VILLE_NOM_REEL" : "VIERSAT" , "VILLE_NOM_SOUNDEX" : "V623" , "VILLE_C...
{ "_id" : "024" , "villes" : [ { "VILLE_ID" : 8221 , "VILLE_NOM_SIMPLE" : "saint-hilaire-le-chateau" , "VILLE_NOM_REEL" : "SAINT-HILAIRE-LE-CHATEAU" , "VILLE_...
{ "_id" : "025" , "villes" : [ { "VILLE_ID" : 8838 , "VILLE_NOM_SIMPLE" : "montfaucon-25" , "VILLE_NOM_REEL" : "MONTFAUCON-25" , "VILLE_NOM_SOUNDEX" : "...
{ "_id" : "026" , "villes" : [ { "VILLE_ID" : 9432 , "VILLE_NOM_SIMPLE" : "merindol-les-oliviers" , "VILLE_NOM_REEL" : "MERINDOL-LES-OLIVIERS" , "VILLE_NOM_...
{ "_id" : "027" , "villes" : [ { "VILLE_ID" : 9801 , "VILLE_NOM_SIMPLE" : "vascoeuil" , "VILLE_NOM_REEL" : "VASCOEUIL" , "VILLE_NOM_SOUNDEX" : "V240" , "VIL...
```

Figure 6. Example of data migration in OR2DOD tool

The proposed approach was validated by the development of an end-to-end system called "OR2DOD", which required a performance evaluation. On this basis, we established two major evaluations: qualitative, and quantitative. In each type of evaluation, three different tests were performed.

6.1. Qualitative evaluation

The first performance study that we performed aims at assessing whether the proposed approach properly migrated the data without any data loss or data poorly transformed. In other words, we evaluated the quality of our metadata (which is the fundamental building block of the data-migration process). Therefore, we established three principal data tests in this evaluation: loss, corruption, and integrity constraints.

In the data-loss test, we checked whether the data from the input database migrated without any data loss. Our strategy relied on queries to compare the numbers of both the source data and the target data according to two dimensions: the rows, and the columns. In the row dimension, we verified the equality between the number of rows in each object-relational table and the number of documents of the corresponding collection. Thus, the same operation must be applied between the nested table and the embedded document of the corresponding document of a collection.

The column dimension aims to check each row and the number of its columns that do not have a null value. This concerns all of the columns of the parent tables, nested tables, structured columns, and columns with row- and varray-data types. Table 4 shows some SQL queries with their equivalent NoSQL ones in order to perform the first test of the qualitative evaluation.

Table 4
Some queries performed in qualitative evaluation

SQL query	NoSQL query	Description
`SELECT COUNT(Department_ID) FROM Department`	`db.Department.count().`	return number of departments
`SELECT Department_ID, COUNT(C.COLUMN_VALUE. City_ID) FROM Department D, TABLE(D.Cities) C GROUP BY Department_ID.`	`db.Department.aggregate ([{$group : {_id : "$Department_ID", num_cities:{$sum:1}}}]).`	return number of cities for each department
`SELECT column_name FROM SYS.ALL_TAB_COLUMNS, WHERE table_name= :{”$objectToArray":"$$ROOT" ’DEPARTMENT’ AND Nullable=’N’.`	`db.Department.aggregate ([{”$project": {”arrayofkeyvalue" }}}, {”$project": ”:{”keys": "$arrayofkeyvalue.k”}}])`	return columns with not-null constraint
`SELECT * FROM Department.`	`db.Department.find(). pretty().`	display all departements
`SELECT C FROM Department D, TABLE(D.Cities) C WHERE D.Department_ID=045`	`db.Department.find({ "Department_ID":"045"}, {"Cities":1}).pretty()`	return all cities of department that are identified by 045
`SELECT COUNT(P.Column_ value) FROM Deparement D, Table (D. cities) C, Table (C.population) P.`	`db. Department.distinct( ’cities.population’). length.`	return size of varray of population column from complex column cities

Several SQL queries were used during this qualitative evaluation. Due to a limitation of space, we directly present the results in Figure 7.

The plot that is presented in Figure 7 describes the different SQL and NoSQL queries that were performed on the two databases on the x-axis and the number of responses (NR) of each query on the y-axis. For example, the query named ‘Department’ aims to count the number of existing departments in the `Departement` table as well as in the `Departement` collection. Observing the results that were related to this assessment, our metadata-driven approach allowed for data migration without any data loss.

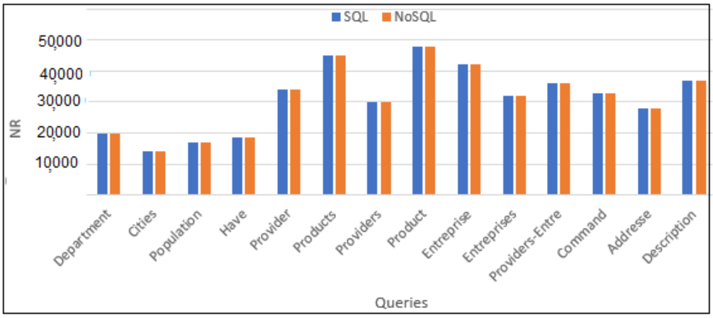


Figure 7. Results of test of data loss

Due to the huge amount of data and the heterogeneity of the data structure between the input and output of the migration process, it is recommended to verify the effectiveness of our proposal in terms of data-migration mismatches. Indeed, testing the data corruption is achieved by automatically comparing the values between the row of the ORDB and the document of the NoSQL document-oriented database.

We focused on a technique that was used to detect the mismatches between values of the input and their corresponding values in the output data. In fact, we proposed a similarity-based data-comparison algorithm to automatically determine that the input values had been correctly migrated. The proposed algorithm is essential for comparing the contents of two different databases (object-relational and document-oriented). The row of the object table has a value for each column. The extraction of the values of the columns and fields were respectively based on SQL and Mongo queries.

The proposed data-comparison algorithm is based on the use of two similarity measures: the Euclidean distance-based, and the cosine's. The Euclidean distance-based similarity **SimD** is most often used to compare two lists of numerical values, while the cosine's measure is applied for computing the resemblance between vectors of words for word-sense disambiguation [4, 7]. For example, we measured the Euclidean distance-based similarities between their values in ORDB and NoSQL in order to compare the population densities:

$$SimD(A, B) = 1/Dist(A, B) \tag{1}$$

where $Dist(A, B)$ is the Euclidean distance between A and B (which is given by the following formula):

$$Dist(A, B) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2}$$

In another example for comparing the names of the cities (`City_simpleName`), we applied the cosine's similarity between their values in ORDB and NoSQL. The cosine's similarity **Cosim** between the two vectors of Words X and Y is defined as follows:

$$Cosim(X,Y) = \frac{X \cdot Y}{\parallel X \parallel^2 \cdot \parallel Y \parallel^2} \tag{3}$$

$\parallel X \parallel$ means the magnitude of X. The perfect value of these similarities is 1, which indicates that the two compared elements are closely similar, while the bad value is 0 – this designates that the two elements are completely dissimilar.

Due to the flexibility of the NoSQL schema, we take only the values of those columns that are not null for each row of the object table; we then compare these with all of the values of the corresponding field in the NoSQL database. The following algorithm describes how to measure the similarity between the rows of ORDB and the documents of the generated NoSQL database.

Algorithm 1: Similarity-based data-comparison algorithm

```
  Input: T: Table of ORDB
         C: collection of NoSQL
1 begin
2     foreach row $Ri \in T$ do
3         foreach document $Di \in C$ do
4             foreach not null column $C_j \in Ri$ and Field $F_j \in Di$ do
5                 S[j]:=Compute similarity ($C_j$, $F_j$);
6             Avg[i]:=average(S[j]);
7         SimRow[i]:=Max (Avg[i]);
8 end
```

The similarity-based algorithm starts by computing and recording the similarities between the selected columns (columns without a not-null constraint) of the i^{th} row of Table T and the corresponding fields of the i^{th} document (Line 5). In Line 6, we calculate the average of similarities of these columns; this represents the similarity between row R_i and document D_i. The loop in Line 3 aims to compute the similarities between row R_i and the existing documents of collection C. In the final step, we select the most similar document to row R_i. Take the following object-relational row from the FPD database as an example:

```
Department (045, Cities (
City_t (16836, 'selle-sur-le-bied', 'SELLE-SUR-LE-BIED', L2426413,
45210, null, null, null, null, 53405, 25342, 480353, 97, 143,
Population (Population_t (Have(2017, 1012, 23, 93))),
City_t (16837, 'bouzy-la-fore', 'BOUZY-LA-FORE', B24163, 45460,
1215, null, null, null, null, '53168', '22243', '475105', 113, 142,
Population (Population_t(Have(2017, 1221, 33))))).
```

Due to the large amount of data, Table 5 exhibits an example of a comparison between one row and ten documents that are identified by `Department_ID`.

Table 5

Comparison between one row and ten documents

N	Department_ID	Average of similarities (Avg.)
1	001	0.320
2	020	0.178
3	030	0.430
4	040	0.310
5	045	1.000
6	050	0.634
7	066	0.213
8	070	0.193
9	073	0.314
10	114	0.291

The results of Table 5 assume that the data-migration process was successfully achieved without data corruption.

Moreover, the principal idea behind the metadata-driven data-migration approach is to retain the SQL integrity constraints. In this context, we carry out the insert and update operations on the generated NoSQL document-oriented database, and we evaluate the integrity-constraint checking. During this stage, we use the Queries menu of the OR2DOD tool to insert new documents and update others.

Table 6 shows the results of some operations of the data insertion and updating by displaying a dialog box. From these results, we assume that the data-migration process can achieve the integrity-constraint checking, thereby demonstrating the proper goal of our proposal.

Table 6

Some result of test of integrity constraints

N	Changing operation	Result
1	Insert same document identified by 60	data already exists
2	Insert same embedded document in document identified by 34	data already exists
3	Insert same list of population of city identified by 2 of document identified by 25	data already exists
4	Insert new department identified by 40.000	new data inserted
5	Insert new city in document identified by 40.000	new data inserted
6	Insert new city without `City_ID`	error of integrity constraint
7	Insert new department without `Department_ID`	error of integrity constraint
8	Update departement 50 by 001	error of integrity constraint
9	Update identifier of city by null	error of integrity constraint
10	Update name of existing city by new value	data updated

In sum, our data-migration tool achieves complex data migration and improves efficiency, thus effectively addressing two issues (that is, the complexity and data quality) of object-relational databases.

6.2. Quantitative evaluation

The quantitative evaluation aims at verifying the efficiency of our proposal in terms of the running time of the data migration. We conducted three fundamental tests: increasing the data source size, data structure complexity, and data sharding.

In the first test, we performed five experiments where we increased the size of the original database to 32, 64, 128, 256, and 512 MB. In fact, we use the Queries menu of the OR2DOD tool to insert new rows and update others.

Figure 8 displays the variations of the running time of the OR2DOD system under different data-set sizes.

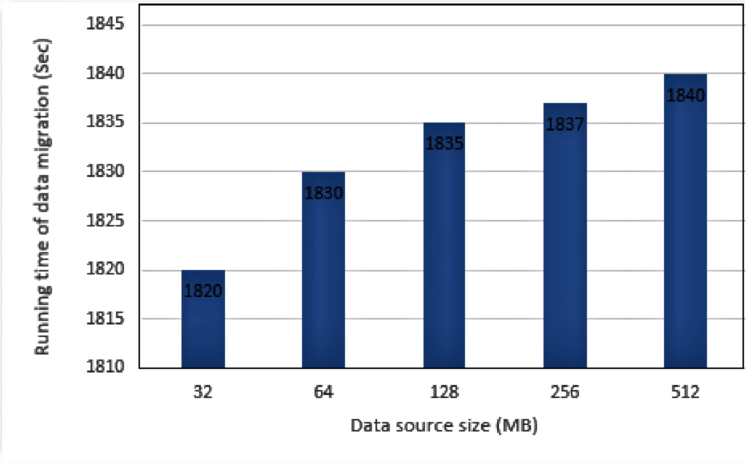


Figure 8. Results of quantitative evaluation according to increment in source size

Figure 8 shows that the execution time remained stable when the data source increased. Moreover, the time-cost gap among the five experiments was quite small when the data sizes increased (as was expected). Besides, the results indicated that the data-migration approach still had an obvious advantage when faced with the volume of the data sources.

The second quantitative evaluation aimed to assess the migration time in various complexities of the data sources. In this context, we used two different object-relational databases. The first one stored France's population distribution (FPD), and the second database (`E-commerce`) had a different structure. To achieve this test, we focused only on the complexity of the data structure without any other factors. Table 7 shows the overall performance of our data-migration approach.

Table 7

Result of quantitative evaluation according to data complexities

Criterion	FPD	E-commerce
Total number of records	36,700	45,000
Total number of nested tables	1	4
Total number of varrays	1	0
Total number of object columns	1	0
Migration running time (sec)	1832	1850

The result in Table 7 shows the time costs with varying data complexities. The migration of the FPD database costed less time than E-commerce, because the FPD structure was less complex than E-commerce was; so, it took less time in the data and schema migration. Besides, the E-commerce database contained more nested tables than FPD, and the time-cost gap between them was just 18 sec. Thus, the data-migration time was about 31 min in the scenario of migrating the E-commerce database with 4 nested tables, while that of FPD with 1 nested table (`thecities`), 1 varray (`Population_t`), and 1 object column (`Have`) was 30 min. This indicates that our data-migration approach has an obvious advantage when faced with complex data structures regardless of the number of records migrated per second.

Therefore, our data migration was still efficient in the scenario with the complex data type and obtained a good solution for migrating complex data of the object-relational database to the NoSQL document-based database.

In the third quantitative evaluation, we attempted to explore two important features of NoSQL databases, which are its data sharding and its replication to distribute data on multiple nodes. In this study, we provided two different modes: simple migration, and advanced migration.

In the simple migration, we took the same configuration of our OR2DOD system, while the advanced migration provided sharding with a replication of each generated collection from the document-oriented database. The sharding function partitioned the database in smaller and faster shards across different servers. We defined four servers: a configuration server (to configure the database engine to listen to Port 37017), sharding server (to ensure the auto-sharding of the generated collections [Port 37018]), and two servers in Ports 37019 and 37020 to receive the shards (partitions and chunks).

In the context of the complex data migration, the generated data sharding not only related to the quantity of the documents but also to the complexity of the data structure (documents with embedded documents, references, lists, etc.).

Hence, we performed two scenarios:

- applying simple and advanced migrations with varying data source sizes;
- applying simple and advanced migrations with varying data complexities.

We present these results in Figure 9.

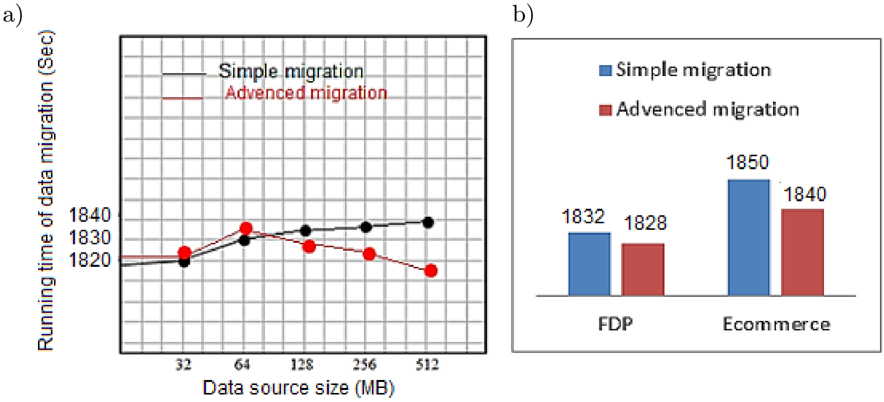


Figure 9. Results of quantitative evaluation according to two modes of data migration and under two different configurations: a) increasing data sizes; b) various data-structure complexities

As the results in Figure 9 show, advanced migration gave a better result for the migration of a large database. Furthermore, the advanced migration of the complex data set required less time for the large data size as compared to the simple migration.

We can also observe that advanced migration showed a slight increase in the running times of small databases (see Fig. 9a) as compared to simple migration; it also required less time when the increased sizes were greater than 64 MB. This explains that the data sharding is more efficient when the data source is large enough. In this study, migration with data sharding may depend greatly on the data volume and the application itself. The strength of our metadata-driven data-migration approach is that it easily transforms data with or without data sharding.

In general, our experimental results reflect the fact that our approach provides a good solution for managing the large object-relational database in the NoSQL model.

The OR2DOD tool aims to migrate an object-relational database that has already been created to a document NoSQL-oriented database while keeping all of the restrictions of the input data as well as the scalability and flexibility of the output data. So, the latter is considered to be a new NoSQL database that is created from an object-relational database has both of the features of the object-relational and document NoSQL data models.

7. Conclusion

This paper proposes an effective metadata-driven approach to migrate the large volumes of object-relational databases into NoSQL document-oriented databases that is easily scalable with high availability. The principal advantage of our approach is the ability to retain the integrity constraints of the original database in order to take the full benefits of both the object-relational and NoSQL models. We developed an OR2DOD system that validated our proposal, and we carried out two main evaluations (in each, three different tests were performed). A qualitative evaluation demonstrated the excellent performance of our data migration without any data loss or data corruption, while a quantitative evaluation proved that the various data-set sizes and data-structure complexities did not affect the running times of the data migration. The advanced data migration aims at offering the sharding of each generated collection of documents. This migration mode gives a better result when a large volume of data must be migrated.

Our results open future directions for data migration. We plan to extend our data-migration approach to other types of NoSQL databases (like a graph-oriented database, which is more suitable for describing relationships between components). Our ongoing work also aims at proposing an approach for NoSQL data mapping and incorporating it in our data-integration process, thus providing a second round of data migration. In this round, we will offer the ability to transform the migrated data from a document-oriented database to other NoSQL databases.

Finally, the proposed approach provides off-line migration, since object-relational databases offer a much more complex data model. Future studies should support this on-the-fly technique for improving the tool's functionalities.

References

[1] Aggoune A.: Intelligent data integration from heterogeneous relational databases containing incomplete and uncertain information, *Intelligent Data Analysis*, vol. 26(1), pp. 75–99, 2022.

[2] Aggoune A., Namoune M.S.: A method for transforming object-relational to document-oriented databases. In: *2020 2nd International Conference on Mathematics and Information Technology (ICMIT)*, pp. 154–158, IEEE, 2020.

[3] Alotaibi O., Pardede E.: Transformation of Schema from Relational Database (RDB) to NoSQL Databases, *Data*, vol. 4(4), 2019.

[4] Anderson M.J.: Distance-based tests for homogeneity of multivariate dispersions, *Biometrics*, vol. 62, pp. 245–253, 2006.

[5] Chen X., Liu Z., Sun M.: A unified model for word sense representation and disambiguation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1025–1035, 2014.

[6] Chodorow K.: *MongoDB: the definitive guide: powerful and scalable data storage*, O'Reilly Media, 2013.

[7] Chung W.C., Lin H.P., Chen S.C., Jiang M.F., Chung Y.C.: JackHare: a framework for SQL to NoSQL translation using MapReduce, *Automated Software Engineering*, vol. 21, pp. 489–508, 2014.

[8] Corbellini A., Mateos C., Zunino A., Godoy D., Schiaffino S.: Persisting big-data: The NoSQL landscape, *Information Systems*, vol. 63, pp. 1–23, 2017.

[9] Date C.J., Darwen H.: *Foundation for Object/Relational Databases: the third manifesto*, Addison Wesley Longman Publishing, 1998.

[10] Davoudian A., Chen L., Liu M.: A survey on NoSQL stores, *ACM Computing Surveys (CSUR)*, vol. 51, pp. 1–43, 2018.

[11] Delmal P.: *SQL2-SQL3: applications à Oracle*, De Boeck Supérieur, 2000.

[12] Devarakonda R.S.: Object-relational database systems the road ahead, *XRDS: Crossroads, The ACM Magazine for Students*, vol. 7, pp. 15–18, 2001.

[13] Eder J., Kanzian S.: Logical Design of Generalizations in Object-relational Databases. In: *ADBIS (Local Proceedings)*, 2004.

[14] Eisenberg A., Melton J.: SQL: 1999, formerly known as SQL3, *ACM Sigmod Record*, vol. 28, pp. 131–138, 1999.

[15] Fouad T., Mohamed B.: Model Transformation From Object Relational Database to NoSQL Document Database. In: *Proceedings of the 2nd International Conference on Networking, Information Systems & Security*, pp. 1–5, 2019.

[16] Hibernate: OMG. http://www.hibernate.org.

[17] Kreines D.C.: *Oracle SQL: the essential reference*, O'Reilly Media, 2000.

[18] Kuszera E.M., Peres L.M., Del Fabro M.D.: Exploring data structure alternatives in the RDB to NoSQL document store conversion process, *Information Systems*, vol. 105, 2022.

[19] Kuszera E.M., Peres L.M., Fabro M.D.D.: Toward RDB to NoSQL: transforming data with metamorfose framework. In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pp. 456–463, 2019.

[20] Laender A.H.F., Ribeiro-Neto B.A., Da Silva A.S., Teixeira J.S.: A brief survey of web data extraction tools, *ACM Sigmod Record*, vol. 31, pp. 84–93, 2002.

[21] Lee K.K.Y., Tang W.C., Choi K.S.: Alternatives to relational database: comparison of NoSQL and XML approaches for clinical data storage, *Computer Methods and Programs in Biomedicine*, vol. 110, pp. 99–109, 2013.

[22] Liao Y.T., Zhou J., Lu C.H., Chen S.C., Hsu C.H., Chen W., Jiang M.F., Chung Y.C.: Data adapter for querying and transformation between SQL and NoSQL database, *Future Generation Computer Systems*, vol. 65, pp. 111–121, 2016.

[23] Mansouri Y., Babar M.A.: The Impact of Distance on Performance and Scalability of Distributed Database Systems in Hybrid Clouds, *arXiv preprint arXiv:200715826*, 2020.

[24] Marcos E., Vela B., Cavero J.M.: A Methodological Approach for Object-Relational Database Design using UML, *Software and Systems Modeling*, vol. 2, pp. 59–72, 2003.

[25] Ouanouki R., April A., Abran A., Gomez A., Desharnais J.M.: Toward building RDB to HBase conversion rules, *Journal of Big Data*, vol. 4, pp. 1–21, 2017.

[26] Piech M., Marcjan R.: A new approach to storing dynamic data in relational databases using JSON, *Computer Science*, vol. 19, 2018.

[27] Reniers V., Van Landuyt D., Rafique A., Joosen W.: Object to NoSQL Database Mappers (ONDM): A systematic survey and comparison of frameworks, *Information Systems*, vol. 85, pp. 1–20, 2019.

[28] Roberts P.: Seamlessness as a desirable aspect of quality for MDE: the contribution of object-relational database stuctures. In: *2010 Seventh International Conference on the Quality of Information and Communications Technology*, pp. 253–258, IEEE, 2010.

[29] Rocha L., Vale F., Cirilo E., Barbosa D., Mourão F.: A framework for migrating relational datasets to NoSQL, *Procedia Computer Science*, vol. 51, pp. 2593–2602, 2015.

[30] Singh S.: Security Analysis of MongoDB, *International Journal of Digital Society (IJDS)*, vol. 10(4), pp. 1556–1561, 2019.

[31] Soutou C.: Modeling relationships in object-relational databases, *Data & Knowledge Engineering*, vol. 36, pp. 79–107, 2001.

[32] Stanescu L., Brezovan M., Burdescu D.D.: An algorithm for mapping the relational databases to MongoDB – a case study, *International Journal of Computer Science & Applications*, vol. 14, pp. 65–79, 2017.

[33] Strozzi C.: NoSQL: a non-SQL RDBMS. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page.

Affiliations

Aicha Aggoune
University of 8th of May, 1945, Computer Science Department, LabSTIC Laboratory, Guelma, Algeria, aggoune.aicha@univ-guelma.dz

Mohamed Sofiane Namoune
University of 8th of May, 1945, Computer Science Department, Guelma, Algeria, namoune.sofianemohamed@gmail.com

Received: 06.07.2021
Revised: 03.03.2022
Accepted: 07.07.2022