Received: April 29, 2024. Revised: May 24, 2024. 634

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Migrating Relational Databases to NoSQL-Oriented Documents Using Object-Oriented Concepts

Alae El Alami[1]* Youness Khourdifi[2] Zakariyaa Ait El Mouden[1] Mohammed Lahmer[1] Moulay Lahcen Hasnaoui[1]

[1]*Department of Computer Engineering, Higher School of Technology, University Moulay Ismail, Meknes, Morocco*
[2]*Sultan Moulay Slimane University, Laboratory of Materials Science, Mathematics and Environment, Polydisciplinary Faculty, Khouribga, Morocco*
* Corresponding author's Email: elalamialae@gmail.com

Abstract: This paper presents a migration from relational databases (RDB) to NoSQL databases, leveraging a data model that utilizes concepts such as objects, semantic enrichment, and metadata. It addresses the limitations of previous approaches and extends them to overcome issues encountered in the relational model, allowing for simpler and faster handling of large datasets. This approach eliminates problems related to joins, incorporates the concepts of references and embedded documents, and includes a comparative study between our approach and other methods for migrating from RDB to NoSQL. This comparison covers aspects such as structural changes, insertion rates, selection, and deletion. Automated migration uses document-oriented nested types to store related data in one database record. This simplifies the design phase and reduces queries and updates, speeding up user responses. This research provides a comprehensive summary of the achievement of migrating from a relational database to a document-oriented NoSQL database. The essence of this migration lies in its automatic nature, aimed at transforming the pre-existing relational model into an object-oriented model. This transition aims to align the database structure with MongoDB's document-oriented paradigm, ensuring optimal integration and efficient utilization of the technology's features. This research significantly distinguishes itself from previous work, primarily due to its ability to facilitate automatic database migration. It accomplishes this task by autonomously identifying object concepts, including those related to inheritance. This functionality is unique and represents a significant advancement, as no other migration method has thus far been capable of performing this automatic detection of object concepts, especially inheritance. The result of the migration provides several advantages in terms of reducing record count, particularly as we deal with massive data sets, and in terms of the time required for data deletion and insertion. A prototype shows how effective this automated approach is. Maintain that the migration process is automated.

Keywords: NoSQL databases, Key-value, Document store, Columnar DB, Graph DB, MongoDB, Software engineering, Enrichment semantic, RDB relational databases, Database migration.

1. Introduction

The daily generation of billions of bytes of data across various sectors in recent years has triggered a migration away from relational databases to other types of databases based on NoSQL solutions that complement the classic relational approach. NoSQL offers various storage formats, including document-based, graph, column-oriented, and key-value stores [1].

In comparison to the relational database model, NoSQL provides improved scalability and enhanced performance for handling structured, unstructured, and semi-structured data. It guarantees a flexible, object-oriented programming plat-form that is easy to use, offering a wide range of software architecture options due to its support for non-fixed physical schemas. NoSQL also supports distributed systems across multiple servers, ensuring data integrity through synchronisation [2].

The key difference in data integrity and consistency between RDBMS and NoSQL databases lies in their respective transaction models. RDBMS adheres to the ACID properties (atomicity, consistency, isolation, and durability) [3, 4], which define specific rules for data transactions. Atomicity ensures that a transaction is either completed in full or not at all; consistency ensures that each step of a transaction maintains a valid state; isolation guarantees independent execution of transactions, and durability ensures permanent data persistence, even in the event of system failures [5].

In contrast, NoSQL adheres to the CAP Theorem (consistency, availability, and partition tolerance) [6, 7], which allows only two of the three constraints to be achieved in practice. This led to the development of the BASE theorem (Basically Available, Soft state, and Eventual consistency) [8]. BASE accepts that database consistency may be in a flux state, with 'Basically Available' indicating data availability even if responses to requests contain errors or the data is temporarily inconsistent. 'Soft state' suggests that the system's status can change over time, and 'Eventual consistency' ensures that the system will eventually reach a consistent state.

NoSQL encompasses a set of database types that follow a non-relational approach, each characterized by a straight-forward concept for storing and handling data, such as key-value, column-oriented, document-oriented, and graph-oriented databases [9, 10].

A key-value database, which is a logical data storage system, is not constrained by a schema tailored to caching and is efficient at the data access level, offering improved performance for writing and reading during disk access [11].

A column-oriented database is a solution that provides flexible logic for data storage by creating different columns for each row, unlike the RDB, which fixes the number of columns without considering the number of records [12].

A document-oriented database is a representation of the key-value concept in a document form, facilitating hierarchical organization based on indexed fields. It is designed for the storage of general sessions, files, and web pages [13].

A graph-oriented database is a solution initially dedicated to social networks for handling complex variable data relationships. This solution doesn't optimize performance but rather addresses problems not solved by the RDB [14].

In this article, we discuss the transition from RDB to NoSQL in document-oriented mode using the MongoDB database. MongoDB does not require a predefined schema and manipulates objects with BSON (Binary JSON) in binary-encoded format, which extends the JSON (JavaScript Object Notation) model [15].

The article is organized into five parts. In the first part, we conduct a general study of different types of NoSQL databases. The second part presents previous work proposing migration solutions. The third part introduces a new way of representing RDB in a flattened format enriched by the object concept. In the fourth part, we propose our main migration solution.

Finally, we establish a comparative study between our approach and both the MySQL and mongoVue approaches.

2. Related works

There is an increasing interest in developing adaptable data migration frameworks capable of transferring data from relational databases to NoSQL data storage [16-19]. These papers provide merely a market overview of a novel category of transformation tools and offers decision support for selecting the appropriate tool. However, my approach goes beyond simply facilitating migration from relational to NoSQL MongoDB databases; it introduces a novel method for transforming a relational database into an object-relational database model. This approach involves semantic enrichment in a flattened model, enabling a seamless transition to the new document-oriented database architecture, which inherently incorporates the concept of embedded documents.

The success of the SQL-to-NoSQL data transfer carries significant implications. Extracting valuable insights from internal data, such as system resource utilisation and employee performance evaluations, is imperative for any company.

Moreover, NoSQL is the preferable choice over relational databases for cloud environments due to the distributed nature of cloud platforms. To minimise the amount of work required by cloud consumers when transferring data between multiple cloud platforms and to have control over concerns related to the compatibility and transferability of data, the current standard necessitates the use of unified API frameworks [20].

An approach to migrate from SQL to NoSQL is based on the JackHare Framework, associated with the JDBC driver, a SQL query compiler, and systematic methods using the MapReduce principle for processing unstructured data in HBase [21]. The developed framework is based on Hadoop and HBase to store RDB data in line with the logic of SQL queries and MapReduce methods. It presents a

conversion model that stores all RDB tables in a single table. As a result, queries involving numerous foreign keys may not perform well due to the high number of join operations [22]. Our approach to automatic migration from the relational model to the document-oriented model is significantly more comprehensive than traditional methods using MapReduce, for several reasons. Firstly, our method fully embraces the fundamental concepts of the document-oriented model, such as schema flexibility and the use of rich data structures like nested objects and arrays. Unlike transformation via MapReduce, which may primarily focus on data conversion without considering these essential characteristics of the document-oriented model, our approach ensures a complete and faithful transition from the relational model to the document-oriented model.

Another solution for migrating from SQL to NoSQL aims to reduce operational expenses and transition from the relational concept to a document-oriented one. The approach begins by creating physical data on two logical levels, with users generating the logical data model and configuring data for each document. This approach allows for monitoring over several days to obtain proof-of-concept for the new document-oriented database, with document relationships managed solely at the application layer [23]. Our method integrates a semantic enrichment process that goes beyond simple schema conversion. By applying semantic enrichment techniques, we're able to capture and leverage semantic relationships and implicit meanings present in relational data. This allows for a richer and more contextual representation of data in the document-oriented model, thereby enhancing the quality and relevance of the migration. Moreover, our approach ensures data consistency and integrity during migration, ensuring that specific constraints and business rules are adhered to in the resulting document-oriented model. This is crucial for ensuring the reliability and robustness of systems that depend on this data.

S. Sabrina discuss the use of NoSQL solutions to address issues related to the ever-increasing volume of data. Their solution provides an algorithm for transforming data from an RDB to NoSQL databases, Redis, Cassandra, MongoDB, and Neo4j, offering a semi-automatic migration approach [24].

C. Carlos describes a data model for creating a NoSQL database from a relational database. This involves highlighting similarities between the CDM conceptual data model and the LDM logical data model of the relational model, followed by transformation into a PDM physical data model for a NoSQL database. The process includes extracting dependencies between entities for different types of relationships. The same conceptual and logical models used for RDBMS modeling are applied to NoSQL modeling [25]. Our approach to automatic migration from the relational model to the document-oriented model is not only more comprehensive and faithful to the principles of the document-oriented model, but it also offers a richer and more nuanced portrayal of data while ensuring data consistency and integrity throughout the migration process.

Cabral, J. V. L., et al confronts the complexities inherent in NoSQL databases, where data storage lacks a fixed schema, leading to intricate query development due to schema dependencies and the need to revise queries following schema alterations. This method leverages conceptual data modeling and code generation to facilitate intricate data retrieval queries in a schema-agnostic fashion. It introduces a language grounded in classic ER algebra tailored for MongoDB, facilitating the establishment of mappings between entities, relationships, and document collections [26].

In our increasingly computer-driven world, crucial areas such as social communication, security, commerce, and education heavily depend on computer science. Institutions handle extensive data, commonly stored in relational databases for their user-friendly interfaces and support for intricate computations. However, the emergence of NoSQL databases, offering four distinct categories, prompts the need for migrating data from relational systems. This paper undertakes a thorough analysis of NoSQL categories using the WSM method, aiming to identify the most suitable category for transitioning data from relational systems, initiating the exploration of this migration process [27]. Conducting a multi-criteria analysis can be time-consuming and resource-intensive. It requires expertise in evaluating and weighting different criteria, which might introduce subjectivity. Additionally, this approach may not fully account for dynamic changes in database technologies or evolving project requirements over time. However, our migration approach offers efficiency and speed, as it can rapidly transform data from a relational database to a NoSQL database based on predefined rules and intelligent algorithms. there is no need for manual intervention and ensures consistency in the migration process.

This research by Khan. M, et al. delves into the pivotal role of data as a company's most prized asset, crucial for analysis, decision-making, and judgment, necessitating sophisticated cache and accessibility mechanisms. It explores the effectiveness of SQL and NoSQL database systems in scientific data production. SQL, or RDBMS, arranges data into

tables, whereas NoSQL provides scalability and an unstructured framework for handling large data applications, encompassing diverse types like wide column stores, documents, graph databases, and key-value pairs, distinct from SQL's standardized structure. Both systems, being open-source, have the capacity for horizontal scaling. The study compares SQL and NoSQL databases, evaluating data organization and performance through analyses of loading, response, and retrieval times to discern efficiency and efficacy [28].

All these migration approaches offer solutions for transitioning from a relational database to a document-oriented database. These approaches primarily focus on how to establish and execute migrations because Big Data imposes fewer constraints on data modeling. They do not concentrate on the physical structure of the target database or improvements in data storage efficiency.

MongoDB stands out for its ability to efficiently handle unstructured and semi-structured data, Thanks to its schema flexibility, horizontal scalability, and high read and write performance. Its document-oriented data model facilitates seamless integration with modern applications and provides increased agility in data development and management.

MongoDB's embedded documents enable the storage of complex data structures hierarchically and flexibly using nested JSON documents. This approach simplifies modeling relationships between data without requiring complex joins, as in relational databases. Grouping related data enhances the efficiency of read and write operations, leading to more effective horizontal scalability by diminishing join costs during data distribution across multiple nodes.

None of the other techniques or approaches have so far addressed the transition from the relational physical schema to a schema adaptable to the document-oriented model. All works have favored the relational transformation to MongoDB, whether manually or automatically, due to the inherent flexibility of MongoDB's schema. However, despite the establishment of the migration, the full exploitation of the potential offered by MongoDB's document-oriented paradigm remains largely underutilized, thus highlighting a major challenge in maximizing the benefits of this technological transition.

Our migration approach focuses on how relational data will be stored in our new ODDB (Object-Oriented Document Database). This distinct approach influences the number of queries and updates needed to successfully perform operations and meet user needs. Our approach also reduces the number of records and time required for insertion, deletion, and selection (see section 5.2). The automation of our approach significantly shortens the design phase and allows for faster responses to user needs. In a broader context, companies can fully harness the benefits of Big Data, such as agility and responsiveness, by employing a robust analytical data mining approach.

3. Development of the enrichment semantic via the relational database

3.1 Semantic enrichment

Semantic enrichment is an enhanced representation of a Relational Database (RDB) that expands upon the traditional relational concept by introducing a new object-oriented concept. Enhancing data, information, or texts involves supplementing them with additional content or context to improve their comprehension, significance, and usefulness. This process may encompass the inclusion of metadata, semantic connections, tags, or annotations, thereby enriching the structure and streamlining analysis, searchability, and interpretation for both machines and users. It achieves this through the use of metadata, a collection of processing elements, and various components that enable the retrieval of information about the database. This information includes details about the extraction procedure, utilization of result set metadata, and the information extraction process pertaining to the parameters used for querying objects. For a more comprehensive understanding of the implementation of semantic enrichment and the algorithms responsible for its creation, please consult the referenced article [29,30].

In the context of semantic enrichment, a database takes on a flattened structure that defines the diverse relationships between tables. It is defined as a collection of classes, denoted as C, where each class is represented as C := (cn, degree, cls, a, contributor) [31, 32].

Here's a breakdown of the components within a class definition:

- **Cn:** This represents the name of the class.
- **Degree:** It can be categorized as the first degree (referring to tables containing the Primary Key (PK)) or the second degree (referring to tables containing Foreign Key (FK) without PK).
- **Cls:** This encompasses aggregation, association, inheritance, and simple class (classes that do not belong to the other classifications).
- **Contributor:** This represents a list of classes.

Table 1. Graphical representation of the semantic enrichment

Cn	Degre	Classification	Attribut DatabaseMetaDatadmd = connection.getMetaData(); ResultSet tables= dmd.getTables(catalog,schemaPattern,tableNamePattern,types);							Contributor
			Tables	An	Type	Tag	l	N	D	
TABLE_NAME	Intern Traitment	parameterizable part integrating the concept of object modeling	Next()	COLUMN_NAME	DATA_TYPE	ResultSet clefs = dmd.getPrimaryKeys (catalog, schema, table) dmd.getExportedKeys (catalog, schema, table)	COLUMN_SIZE	IS_NULLABLE	COLUMN_DEF	Intern Traitment

- **A:** It refers to an attribute and is defined as a set of attributes, where each attribute is denoted as a := (an, t, tag, l, n, d).

Here's what each attribute component represents:

- **an:** Name of the attribute.
- **t:** Type of the attribute.
- **tag:** Indicates whether the attribute is a primary key (PK) or a foreign key (FK).
- **l:** Length of the attribute.
- **n:** Indicates whether the attribute can take a null parameter.
- **d:** Specifies the default value of the attribute.

Table1. provides a schematic representation of the development of semantic enrichment:

3.2 Modeling techniques for transforming a RDB towards NoSql (Mongodb)

In contrast to the RDB, which relies on a rigid physical schema, NoSQL offers a flexible physical schema that doesn't impose a predefined document structure. This flexibility enables the establishment of data relationships through references and embedded documents [33].

* References function similarly to foreign keys in the relational model. They are used to prevent data redundancy and represent relationships and sets using OBJECTID.

- Used in the case of the relationship (x, n) (x, n), where x can be 1 or 0, especially in the context of associations when connecting two tables during the key migration when transitioning from the conceptual model to the logical model.
- Used in the case of the relationship (x, n) (x, 1), including scenarios involving ternary relationships.
- Used in the case of reflexive relations when a table is related to itself.
- Used in the case of a one-to-one (1-1) relationship, applicable in both simple relationships and inheritance relationships.

* Embedded documents offer a technical representation for relationships within a document, enhancing data readability and facilitating atomic updates.

Syntax used for creation:

```
{
  "_id": { "$oid": "ObjectId" },
  "Attribute1": "value_1",
  "Attribute2": {
    "attr1": "val1",
    "attr2": "val2"
  }
}
```

This syntax is employed for:

- Representing composition relationships.
- Visualizing data within the context of other data, particularly in cases involving composite attributes that combine values from elementary attributes.

4. Migrating a relational database to NoSQL

When migrating from a relational database to a document-oriented database with no predetermined schema, storage takes the form of BSON documents, and data is organized into collections with shared indexes. Each collection corresponds to a table extracted from the semantic enrichment process, and this organization incorporates the concepts of references and embedded documents [34]. This integration of new concepts helps eliminate the need

```
public String[][] selectAllO(String tableName) {
    String req = "SELECT * FROM " + tableName;
    try {int type = ResultSet.TYPE_SCROLL_INSENSITIVE;
    int mode = ResultSet.CONCUR_UPDATABLE;
    Statement sql=db.createStatement(type,mode);
    ResultSet rs = sql.executeQuery(req);ResultSetMetaData rsm = rs.getMetaData();
    int columns = rsm.getColumnCount(); String data[][];rs.last();
    int rows = rs.getRow() + 1;
    data = new String[rows][columns];
    for (int i = 1; i <=columns; i++) { data[0][i-1] = rsm.getColumnName(i);}
    int row = 1;rs.beforeFirst();
    while (rs.next()) { for (int i=1; i<=columns; i++) {data[row][i-1] = rs.getString(i);}
    row++;}
    return data;
    }catch (Exception e) {  e.printStackTrace();return null;}
}
```

Figure. 1 The method responsible for selecting data from the relational database

for joins commonly used in the relational model, which are often not scalable.

The selection of data from the Relational Database (RDB) is based on the parameter C.Cn from the semantic enrichment process, which is passed as a parameter to the selection method, as depicted in Fig. 1 [35].

The function selectAllO is used to query a database and extract all the data stored in a specific table. The function establishes a connection to the specified MySQL database, constructs an SQL statement to retrieve all rows from the specified table, executes the SQL query, and iterates through the result set to display the values of each column. This method retrieves all the columns and rows of the relevant table, providing a comprehensive view of its contents without excluding any information.

The ResultSet object provides valuable information about the data, including table names, column names, and column properties. The following methods are used:

getColumnCount: This method returns the number of columns contained in the ResultSet.

Statement objects are employed to execute basic SQL queries and retrieve results through the ResultSet class. To create a Statement instance, the createStatement method is called on the Connection object obtained using one of the DataSource getConnection methods.

executeQuery: This method returns a ResultSet object.

To obtain metadata from the data source, the **getMetaData** method is called using the Connection object created earlier.

{"_id":{"$oid":"_id field that stores ObjectId"},
"RDB.Table1.Attribut_1)":"RDB.Table1.Attribut_1(value_1)",
"RDB.Table1.Attribut_2)":"RDB.Table1.Attribut_2(value_1)",...,"RDB.Table1.Attribut_n)":"RDB.Table1.Attribut_n(value_1)"}
{"_id":{}}
{"_id":{}}
{"_id":{}}...
{"_id":{}}
{"_id":{"$oid":"_id field that stores ObjectId"},
"RDB.Table1.Attribut_1)":"RDB.Table1.Attribut_1(value_n)",
"RDB.Table1.Attribut_2)":"RDB.Table1.Attribut_1(value_n)",...,"RDB.Table1.Attribut_n)":"RDB.Table1.Attribut_1(value_n)"}

The syntax for migrating from our approach of a RDB to NoSQL is generated through semantic enrichment, serving as an intermediary between the RDB and MongoDB. This process involves the integration of the object-oriented concept into the new NoSQL database. The syntax varies based on the classification assigned to the table in the semantic enrichment and its contribution. Two methods of creation are chosen: one involving a straightforward creation that incorporates the reference concept, and another that integrates the concept of embedded documents.

For the **creation of the NoSQL database using the referencing concept:**

{"_id":{"$oid":"_id field that stores ObjectId"},"Cn.Attribute.An(element_1)":"Cn.Attribute.An(value_1)",
"Cn.Attribute.An(element_2)":"Cn.Attribute.An(value_1)",...,"Cn.Attribute.An(element_n)":"Cn.Attribute.An(value_1)"}
{"_id":{}}
{"_id":{}}
{"_id":{}}...
{"_id":{}}

```
public String[] select(String tableName, String key, String value) {
    String req = "SELECT * FROM " + tableName + " WHERE " + key + " = '" + value + "'";
    try {Statement sql = db.createStatement();
    ResultSet rs = sql.executeQuery(req);ResultSetMetaData rsm = rs.getMetaData();
    int columns = rsm.getColumnCount(); String data[];
    data = new String[columns];
    if (rs.next()) {for (int i=1; i<=columns; i++) {data[i-1] = rs.getString(i);}
    return data;}else return null;
    }catch (Exception e) {e.printStackTrace();return null;  }
}
```

Figure. 2 The method responsible for handling the request dedicated to the specific selection

{"_id":{"$oid":"_id field that stores ObjectId"},
"Cn.Attribute.An(element_1)":"Cn.Attribute.An(value_n)",
"Cn.Attribute.An(element_2)":"Cn.Attribute.An(value_n)",...,"Cn.Attribute.An(element_n)":"Cn.Attribute.An(value_n)"}

Creation of the NoSQL database using the concept of embedded documents:

When incorporating the embedded document concept, we generate documents within another document.

To facilitate this creation, a specific selection request is necessary, tailored to the identifier of the table, which typically serves as a foreign key in the interacting table. For instance, this selection might be expressed as **Cn.Classification:=composition && Cn.Attribute.tag:=PK**.

The select function allows querying a database and extracting specific data that meets defined criteria. It enables the selection of particular columns from a table or computed data based on these columns, depending on conditions specified in the WHERE clause. This capability enables users to retrieve precise and relevant information by filtering data according to defined parameters, thus providing increased flexibility in retrieving and analyzing data stored in the database. This selection criteria will be included as a parameter in the request, as shown in Fig. 2. The creation process follows the syntax below.

{"_id":{"$oid":"_id field that stores ObjectId"},"Cn.Attribute.An(element_1)":"Cn.Attribute.An(value_1)",
"Cn.Attribute.An(elements_2)":{"element_2_1":{"Cn.Attribute.An(elements_2_1_1)":"Cn.Attribute.An(value_1)","Cn.Attribute.An(elements_2_1_2)":"Cn.Attribute.An(value_1)",...,"Cn.Attribute.An(elements_2_1_n)":"Cn.Attribute.An(value_1)"},...,
"element_2_n":{"Cn.Attribute.An(elements_2_n_1)":"Cn.Attribute.An(value_1)","Cn.Attribute.An(elements_2_n_2)":"Cn.Attribute.An(value_1)",...,"Cn.Attribute.An(elements_2_n_n)":"Cn.Attribute.An(value_1)"}},...,
"Cn.Attribute.An(element_n)":"Cn.Attribute.An(value_1)"}

{"_id":{}}
{"_id":{}}
{"_id":{}}...
{"_id":{}}
{"_id":{"$oid":"_id field that stores ObjectId"},
"Cn.Attribute.An(element_1)":"Cn.Attribute.An(value_n)",
"Cn.Attribute.An(elements_2)":{"element_2_1":{"Cn.Attribute.An(elements_2_1_1)":"Cn.Attribute.An(value_1)","Cn.Attribute.An(elements_2_1_2)":"Cn.Attribute.An(value_1)",...,"Cn.Attribute.An(elements_2_1_n)":"Cn.Attribute.An(value_1)"},...,
"element_2_n":{"Cn.Attribute.An(elements_2_n_1)":"Cn.Attribute.An(value_1)","Cn.Attribute.An(elements_2_n_2)":"Cn.Attribute.An(value_1)",...,"Cn.Attribute.An(elements_2_n_n)":"Cn.Attribute.An(value_1)"}},...,
"Cn.Attribute.An(element_n)":"Cn.Attribute.An(value_n)"}

For each class extracted from semantic enrichment, such as C.Classification:=(simple || association || inherBy || Inherts || agregation), a collection is created with a name defined in the semantic enrichment, denoted as C.Cn. This collection is populated with elements associated with the C.Attribute.An element, along with their corresponding data. This approach eliminates issues related to joins when integrating the reference concept.

In the case of compositions, where the classification is 'composition' and C.Contributor equals C1.Cn, a collection is created alongside it. This results in a collection containing elements composed of sets of collections that model the semantics expressed by 'component' and 'compound.' However, it should be noted that the contents are destroyed when the container is destroyed [36].

Below is the algorithm responsible for migrating an RDB to NoSQL:

Algorithm 1: Algorithm for the migration of a RDB to a NoSQL databases

1 **Begin**
2 **Create the semantic enrichment**

```
3 If (( C.Classification == ( simple || association ||
inherBy || inherts || aggregation )) &&
(( C.Contributor = C1.Cn ) &&
( C1.Classification != composition )) )
4 Create C.Cn collection with DBCollection
5 Execute the selectQuery( C.Cn )
6 Instantiate C.Cn with a BasicDBObject for each
row
7 For ( i = 1; i <= selectQuery.nbrElement; i++ )
8 For (j = 0; j <attribut.nbrElement; j++)
9  CnElement(i).put("data[0][j]","data[i][j]")
10 End for
11 DBCollection.insert(CnElement(i))
12 End for
13 Else if (( C.Classification == (simple ||
association || inherBy || inherts || aggregation )) &&
(( C.Contributor = C1.Cn ) && ( C1.Classification
== composition )) )
14 Create C.Cn collection with DBCollection
15 Execute the selectQuery(C.Cn)
16 Instantiate C.Cn with a BasicDBObject for each
row
17 For (i = 1; i <= selectQuery.nbrElement; i++)
18 For (j = 0; j < attribut.nbrElement; j++)
19 CnElement(i).put("data[0][j]","data[i][j]")
20 If C.Attribut.tag == pk
21 Instantiate C1.Cn with a BasicDBObject to store
related pieces with Embedded Documents for each
row
22 Execute the selectSpecifiqueQuery( C1.Cn,
data[0][j], data[i][j] )
23 For (a = 1; a <=
selectSpecifiqueQuery.nbrElement; a++ )
24 For ( b = 0; b < attribut.nbrElement; b++)
25 CnElement(a).put("data[0][b]","data[a][b]")
26 End for
27 DBObjectEmbedded.put(Cn.element(a))
28 End for
29 CnElement(i).put( "data[0][j]",
DBObjectEmbedded )
30 End if
31 End for
32 DBCollection.insert(CnElement(i))
33 End for
34 End
```

The migration algorithm relies on a complex process that begins with acquiring information from our flattened model, which has been semantically enriched. This initial phase is crucial as it allows capturing all the nuances and details necessary for efficient data migration.

Once this information is obtained, the algorithm proceeds to extract the various object principles present in the source relational database. These object principles represent the fundamental entities and relationships that structure the source database. By identifying and precisely extracting them, the algorithm can then manipulate them appropriately to integrate them into the target system coherently and completely.The first step of our migration process involves creating a flattened model enriched semantically by object concepts. The flattened model refers to a simplified and linear representation of data, used to facilitate their manipulation and transfer. Enriching this model semantically means giving it a deeper and more contextual meaning. This involves incorporating additional information about the meaning and relationships between the data, facilitating their interpretation by computer systems. This automatic process is based on previous work we have conducted and described in the references we have published. These references provide a theoretical and methodological basis for our migration approach, detailing the techniques used and demonstrating their effectiveness through thorough experiments and analyses.

The algorithm outlines the steps for migrating an RDB (Relational Database) to a NoSQL ODDB (Object Oriented Database) following the procedure for semantic enrichment. Based on the classification extracted from the meta-model's semantic enrichment, we determine whether to create an embedded document or establish a reference. This determination applies to all the tables within the RDB.

5. Comparative study: Evaluating our approach against MySQL and MongoVue

The comparison of approaches centers around two cornerstones of the open-source world: MySQL and MongoDB. These platforms have implemented automatic migration from RDBs to NoSQL oriented documents and offer a programming interface that facilitates self-monitoring [37][38].

A RDB is taken as an example in its logical form, and from it, we perform the extraction of semantic enrichment.

Logical Data Model of the Relational Database:

Dept (`dno`, `dname`)
Employ (`pno`, `salary`, `grade`)
Kids (`kno`, `kname`, `sexe`, `pno`)
Person (`pno`, `pname`, `bdate`, `adress`, `dno`, `pnosup`)
Proj (`prno`, `pname`, `description`)
Trainee (`pno`, `levell`, `typee`)
Works_on (`pno`, `prno`)

The semantic enrichment was obtained after the exploitation of metadata and a series of processing

Table 2. The obtained semantic enrichment extracts from MySQL

Cn	Degre	Classification	Attribut						Contributor
			An	Type	tag	l	N	D	
Person	1st	inherBy	Pno	Varchar	PK		N		Kids
									Works_on
									Trainee
									Employ
			Pname	Varchar			N		
			Bdate	Date			N		
			Adress	Varchar		255	N		
			Dno	Int	FK		N		Dept
			PnoSup	Varchar	FK		Y		Person
Trainee	2nd	Inherts	Pno	Varchar	FK		N		Person
			Level	Varchar			N		
			Type	Varchar			N		
Employ	2nd	Inherts	Pno	Varchar	FK		N		Person
			Salary	Int			Y		
			Grade	Varchar			N		
Works_on	2nd	Association	Prno	Int	FK		N		Proj
			Pno	Varchar	FK		N		Person
Dept	1st	Simple	Dno	Int	PK		N		Person
			Dname	Varchar			N		
Proj	1st	Simple	Prno	Int	PK		N		Works_on
			Prname	Varchar			N		
			Description	Varchar		255	Y		
Kids	1st	Aggregation	Kno	Int	PK		N		
			Kname	Varchar			N		
			Sex	Char			N		
			Pno	Varchar	FK		N		Person

steps to extract various object concepts from the RDB. MySQL was used as the RDBMSS.

This section provides an evaluation that describes the working methodologies of both MySQL/mongoVue and our approach. These methodologies start with the same relational database as a foundation and result in two distinct NoSQL databases, each with its unique characteristics.

5.1 Migration result for MySQL and MongoVue approach

The outcome of the RDB migration approach to NoSQL (using MySQL and mongoVue) involves capturing only the first three records and employing MongoDB as the OODBMS.

{"_id":{"$oid":"565068b23172f67743c24172"},"pno":"d543","salary":"9000","grade":"engineer"}
{"_id":{"$oid":"565068b23172f67743c24173"},"pno":"g234","salary":"12000","grade":"director"}
{"_id":{"$oid":"565068b23172f67743c24174"},"pno":"f552","salary":"7000","grade":"commercial"}
{"_id":{"$oid":"565068b23172f67743c24178"},"kno":"34","kname":"badr","sexe":"m","pno":"d543"}
{"_id":{"$oid":"565068b23172f67743c24179"},"kno":"23","kname":"sarah","sexe":"f","pno":"d543"}
{"_id":{"$oid":"565068b23172f67743c2417a"},"kno":"21","kname":"jeff","sexe":"m","pno":"g234"}
{"_id":{"$oid":"565068b23172f67743c24175"},"pno":"g234","pname":"azar","bdate":"1984-04-24","adress":"lotissement 34 rue des far appt 6","dno":"1","pnosup":null}
{"_id":{"$oid":"565068b23172f67743c24176"},"pno":"d543","pname":"alae","bdate":"1987-03-15","adress":"residence ibn sina imm d4 appt 3","dno":"1","pnosup":"g234"}
{"_id":{"$oid":"565068b23172f67743c24177"},"pno":"e234","pname":"fouad","bdate":"1987-01-03","adress":"rayhan imm 4 appt 5","dno":"2","pnosup":"d543"}

5.2 Migration result for our approach

The result of the migration from a RDB to NoSQL (our approach) involves capturing only the first three records, using MongoDB as the OODBMS.

{"_id":{"$oid":"56506960317209a6d85970fa"},"pno":"d543","salary":"9000","grade":"engineer","kids":{"kids":{"kno":"34","kname":"badr","sexe":"m","pno":"d543"},"kids1":{"kno":"23","kname":"sarah","sexe":"f","pno":"d543"}}}
{"_id":{"$oid":"56506960317209a6d85970fb"},"pno":"g234","salary":"12000","grade":"director","kids":{"kno":"21","kname":"jeff","sexe":"m","pno":"g234"}}
{"_id":{"$oid":"56506960317209a6d85970fc"},"pno":"f552","salary":"7000","grade":"commercial"}
{"_id":{"$oid":"56506960317209a6d85970fd"},"pno":"g234","pname":"azar","bdate":"1984-04-24","adress":"lotissement 34 rue des far appt 6","dno":"1","pnosup":null}
{"_id":{"$oid":"56506960317209a6d85970fe"},"pno":"d543","pname":"alae","bdate":"1987-03-15","adress":"residence ibn sina imm d4 appt 3","dno":"1","pnosup":"g234"}
{"_id":{"$oid":"56506960317209a6d85970ff"},"pno":"e234","pname":"fouad","bdate":"1987-01-03","adress":"rayhan imm 4 appt 5","dno":"2","pnosup":"d543"}

All tests are conducted on the same relational database, resulting in the insertion of 3 records, 9 records, and 12 records. We capture the results in milliseconds to illustrate the differences between the three migration approaches. It is worth noting that the blue line represents both the MySQL and MongoVue approaches, which yield the same migration results.

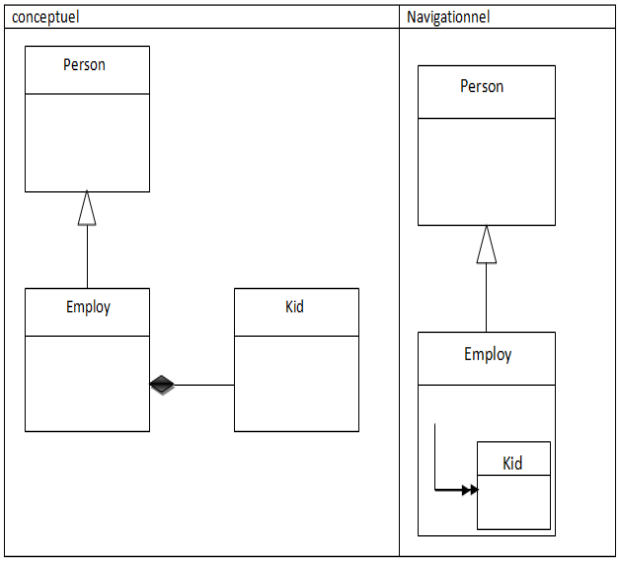


Figure. 3 The representation of the conceptual transformation to navigational

The concept addressed in this analysis is the concept of inheritance with composition. Fig.3 illustrates the conceptual transition to the navigational approach applied during our RDB migration towards NoSQL:

The real function (office) of three variables is an application of R 3 in value in R, we note Df domain of definition of f such as

f: R^3→R,(x1,x2,x3)→z=f(x1,x2,x3)

The function

f: R^3→R,(x1,x2,x3)→z=f(x1,x2,x3)

The tables x1, x2, and x3 are successive tables in the relational database, where x1 represents 'person', x2 represents 'employee', and x3 represents 'children'. Semantically, the relationship between a person and an employee is represented, with the employee inheriting from the 'person' table. Additionally, the relationship between an employee and a child is established, where the 'child' table depends on the 'employee' table because deleting the employee results in the deletion of the child. For instance, x1 represents the number of records in the 'person' table, x2 represents the number of records in the 'employee' table, and x3 represents the number of records in the 'child' table. These variables are defined by values x, y, and z, respectively, as follows:

x+y+z = nbrRecord.

After evaluating the two migration approaches in terms of data access, we observed that the execution time is better in our approach, which integrates the object concept into the NoSQL structure.

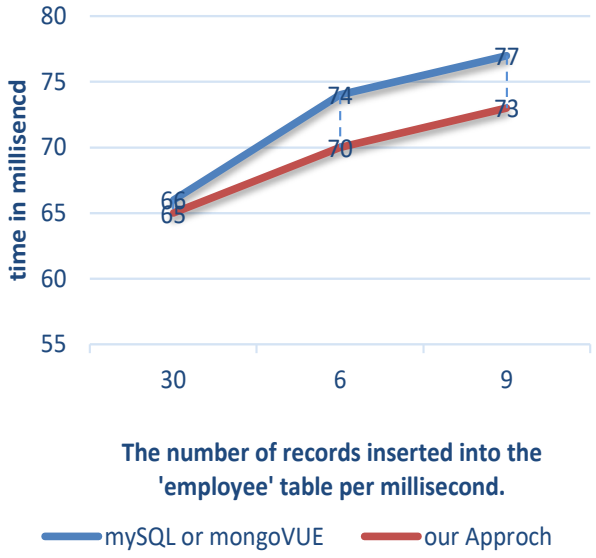


Figure. 4 Graph showing the speed of insertion between the three approaches

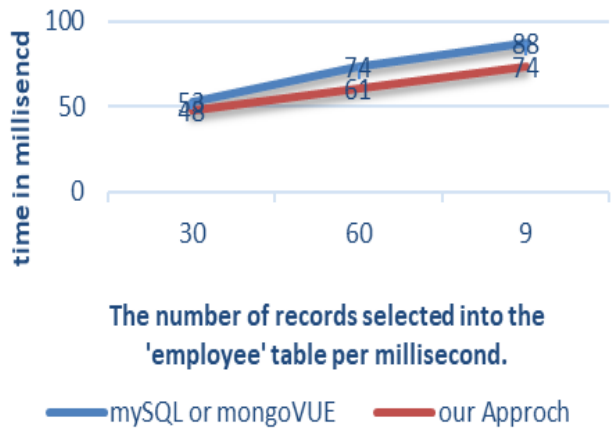


Figure. 5 Graph showing the speed of selection between the three approaches

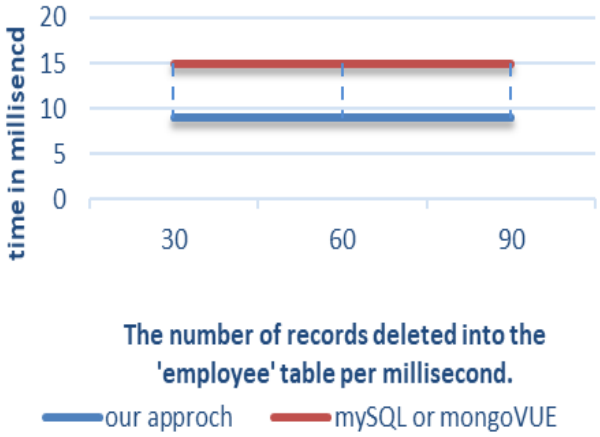
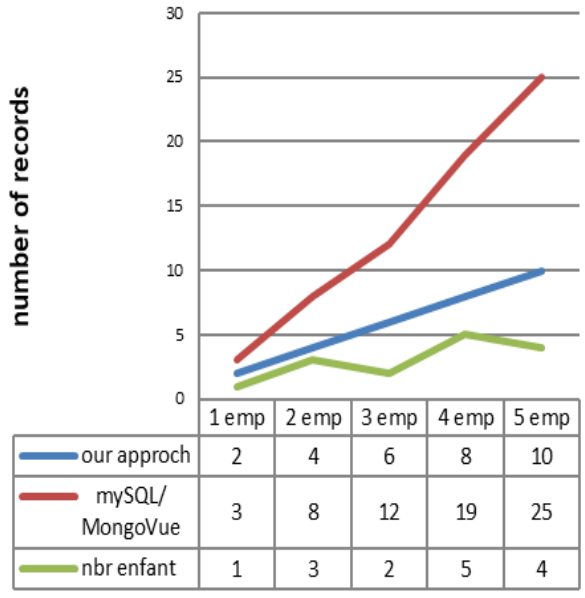


Figure. 6 Graph showing the speed of deletion between the three approaches



	1 emp	2 emp	3 emp	4 emp	5 emp
our approch	2	4	6	8	10
mySQL/ MongoVue	3	8	12	19	25
nbr enfant	1	3	2	5	4

Figure. 7 Graph showing the evolution of recording between the three approaches

The deletion time for a specific query, targeting the removal of an item from the 'employ' collection in the approach followed by MySQL and MongoVue, varies depending on the number of children for each employee. This is because the query involves removing both the employee and their children to maintain the semantics extracted from the RDB. In contrast, our migration approach stores data in a manner that preserves the semantics derived from semantic enrichment, resulting in a consistent deletion time.

In the previous tests, we assigned a child to each employee to examine the execution time. In the upcoming tests, we will add a 'Kid' item to track the evolution of records in both approaches as the number of children for each employee increases.

In our approach, the attributes of the composed class directly include the attributes of the component class. This allows easy access to the data of the component class from the composed class. The composed class will determine the number of records, as each instance of the composed class will contain a set of records corresponding to the attributes of the component class. The number of records is aEmploy+mPerson.

The approach used by MongoVue and MySQL stores a reference to the component class instead of copying its attributes. This makes accessing the data of the component class more complex and increases the number of records. The number of records is obtained through a sequence, such as S_0 = 0 and n ∈N^* if for all n integer we have: Sn =Sn-1 + 2 + wKids, such as S_(n) is the number of records we want to calculate.

6. Discussion

Previous research on migrating relational databases to document-oriented databases primarily focused on data transfer, neglecting the transformation of the physical relational schema into a document-oriented physical schema. These approaches preserve the relational structure by organizing data in tables composed of rows and columns. In contrast, the document-oriented model stores data in documents, each of which can contain multiple sub-documents. This approach involves moving data from one system to another without necessarily altering the underlying structure or logic of the data.

The rigidity of the relational structure ensures data integrity and facilitates complex queries involving joins. In contrast, our approach proposes a complete redesign of the data schema to adapt it to the structure and principles of the new document-

oriented system. This involves rethinking data structuring and access.

The migration focuses on transforming the physical relational schema into a document-oriented physical schema to efficiently model different object concepts. Then, the data is migrated to the new database according to the target physical schema.

6.1 Extraction and semantic enrichment of object concepts

The first step is to extract the various object concepts from the relational database. This involves semantic enrichment using semantic techniques to identify and extract the object concepts present in the relational database. This step may include analyzing table and column names, relationships between tables, referential integrity constraints, and developing specific algorithms to automate the extraction of object concepts and their organization into a flattened model. This model then serves as the basis for the transformation to the document-oriented schema (Table 1).

6.2 Advantages of schema transformation

The schema transformation approach offers several advantages over data transfer-based approaches:

Better Suitability to the Document-Oriented Model: The schema structure is optimized for the document-oriented model, allowing full exploitation of the features of this type of database, such as flexibility and scalability.

More Efficient Queries: The document-oriented schema is designed for more efficient queries, especially for complex queries involving relationships between object concepts.

Simplified Maintenance: A well-designed schema is easier to understand, maintain, and evolve over time.

Better Support for Heterogeneous Data: The document-oriented model offers better support for heterogeneous and complex data, making it more suitable for modern applications.

6.3 Performance criteria

We compared our approach for migrating relational databases to document-oriented databases with two solutions available on the market: MongoVue and MySQL. The results of this comparison show that our method significantly outperforms existing approaches in terms of performance and efficiency.

We conducted a series of tests to compare the performance of our migration approach with those of MongoVue and MySQL. The comparison criteria included selection time, insertion time, and record minimization in certain migration cases.

6.3.1. Selection time

Our Approach: The selection time is significantly reduced thanks to the optimized structure of the documents, allowing faster queries.

MongoVue and MySQL: The selection time is reasonable but less efficient compared to our approach, as it can be longer due to the joins necessary to access data distributed across multiple tables.

6.3.2. Insertion time

Our Approach: The insertion time is optimized thanks to the reduction of join operations and the efficiency of document organization.

MongoVue and MySQL: The insertion time is acceptable but higher than in our approach due to the need to respect referential integrity constraints and the rigid organization of data.

6.3.3. Minimization of records

Our Approach: We observed a significant reduction in the number of records needed to represent complex relationships, such as between an employee and their children, thanks to the use of nested documents.

MongoVue and MySQL: The traditional relational structure often results in a multiplication of records to maintain relationships between entities.

7. Conclusion

This article explores the transition from a relational database to a new document-oriented NoSQL database using MongoDB, guided by a semantic enrichment mechanism. This mechanism extracts various object concepts from the relational physical schema and transforms them into a semantic enrichment that flattens the relational database.

The primary challenge in data modeling lies in finding a balance between the requirements of the application. When modeling, it's essential to consider how the data will be used in the application. Decisions regarding the design of data models for MongoDB applications are centered around the organization of documents and how the application depicts relationships between data. This flexibility enables the mapping of documents to entities or objects. Each document can align with the data fields

of the represented entity, even if it varies substantially from other documents in the collection.

The creation of the new document-oriented database follows a specific syntax, which incorporates the object concept by adding the notions of aggregation, composition, and inheritance via objectID, using the principles of referencing and embedded documents. A comparative study was conducted across various approaches, focusing on architecture, data manipulation speed, and the number of records. The migration approach proposed in this study demonstrates remarkable effectiveness compared to the methods utilized by MySQL, MongoVue, and other approaches cited in related works. The comparative analysis of performance metrics, including selection, insertion, and deletion operations, clearly indicates the superiority of the migration approach.

Firstly, the migration approach showcases reduced speed for selection, insertion, and deletion operations. This means that when querying data (selection), inserting new data, or removing existing data, the migration approach exhibits faster execution times compared to the methods employed by MySQL, MongoVue, and other approaches.

Additionally, the migration approach offers more efficient storage, resulting in a minimized number of records. By optimizing the storage mechanism, the migration approach ensures that the data is stored in a compact and organized manner, thereby reducing the overall number of records needed to represent the same dataset. This not only saves storage space but also contributes to improved data management and retrieval efficiency.

Overall, the superior performance of the migration approach underscores its effectiveness in achieving faster operation execution and more efficient data storage, positioning it as a favorable choice for database management tasks compared to the alternatives. A prototype was developed, demonstrating the entire migration process automatically without human intervention, showcasing the effectiveness of this approach.

Conflicts of Interest

The authors declare no conflict of interest.

Author Contributions

Conceptualization, A. El Alami and Y. Khourdifi; methodology, A. El Alami; software, A. El Alami and Z. Ait El Mouden; validation, A. El Alami and M. Lahmer; formal analysis, A. El Alami and M. L. Hasnaoui; investigation, A. El Alami; resources, A. El Alami; data curation, A. El Alami; writing—original draft preparation, A. El Alami and Y. Khourdifi; writing—review and editing, A. El Alami and Y. Khourdifi; visualization, A. El Alami and Z. Ait El Mouden; supervision, A. El Alami; project administration, A. El Alami; funding acquisition, A. El Alami.

References

[1] N. K. sung, et L. D. sik, “Bigdata Platform Design and Implementation Model”, *Indian J. Sci. Technol.*, Vol. 8, No. 18, août 2015.

[2] P. S. Sen, et N. Mukherjee, “An ontology-based approach to designing a NoSQL database for semi-structured and unstructured health data”, *Cluster Comput.*, avr. 2023.

[3] W. Khan, T. Kumar, C. Zhang, K. Raj, A. M. Roy, and B. Luo, “SQL and NoSQL database software architecture performance analysis and assessments—a systematic literature review”, *Big Data Cogn. Comput.*, Vol. 7, No. 2, 2023.

[4] T. Taipalus, “Database management system performance comparisons: A systematic literature review”, *J. Syst. Softw.*, p. 111872, oct. 2023.

[5] A. Meier and M. Kaufmann, *SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management*, Springer Vieweg, 2019.

[6] E. A. Lee, R. Akella, S. Bateni, S. Lin, M. Lohstroh, and C. Menard, “Consistency vs. availability in distributed real-time systems”, *Intelligent Computing*, Vol. 2, pp. 13, 2023.

[7] S. Zhao, et E. de Angelis, “Performance-based Generative Architecture Design: A Review on Design Problem Formulation and Software Utilization”, *J. Integr. Des. Process Sci.*, Vol. 22, No. 3, p. 55-76, 2019.

[8] S. M. L. Hahn, I. Chereja, et O. Matei, “Evaluation of Transformation Tools in the Context of NoSQL Databases”, *Lecture Notes in Networks and Systems*, p. 146-165, 2021.

[9] T. T. Le, and X. L. Pham, “Towards NoSQL databases: Experiences from actual projects”,In; *Proc. of 2022 3rd International Conf on Big Data Analytics and Practices (IBDAP)*, pp. 15-20, 2022.

[10] H. A. Eldrrat, and A. M. Maatuk, “Data Migration from Conventional Databases into NoSQL: Methods and Techniques”, In: *Proc. of 2023 IEEE 3rd International Maghreb Meeting of the Conf on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA)*, pp. 370-375, 2023.

[11] Q. Meng, K. Zhang, H. Pan, M. Yuan et B. Ma, "Design and Implementation of Key-Value Database for Ship Virtual Test Platform Based on Distributed System", *Communications in Computer and Information Science*, p. 109-123, 2023

[12] R. H. P. Prakoso, and F. N. Azizah, "The Study of Data Modeling Methodologies For Column-Oriented Databases", In: *Proc. of 2023 IEEE International Conference on Data and Software Engineering (ICoDSE)*, Toba, Indonesia, pp. 238-243, 2023.

[13] M. D. Ordoñez, D. S. R. Baena, et B. Y. Casalilla, "A new approach for the construction of historical databases—NoSQL Document-oriented databases: the example of AtlantoCracies", *Digit. Scholarship Humanities*, avr. 2023.

[14] M. Elsabagh, "NO SQL Database: Graph database", *Egyptian Journal of Artificial Intelligence*, Vol. 1, No. 1, 2022.

[15] K. Seguin, "The Little MongoDB". *Self-Published*, 2013.

[16] N. Bansal, S. Sachdeva, et L. K. Awasthi, "Database Migration Tools: From RDB to NoSQL Database", *Frontiers in Artificial Intelligence and Applications*, 2022.

[17] M. J. V. Cedeño, et H. V. Nolivos, "Methodology for the migration of NoSQL databases to SQL databases for their subsequent integration in servers set in relational data models", In: *vi int. Scientific conv. Universidad técnica de Manabí : Advances Basic Sci., Inform. Appl. Eng*, Portoviejo, Ecuador, 2024.

[18] K. Rajaram, P. Sharma, and S. Selvakumar, "DLoader: Migration of Data from SQL to NoSQL Databases", In: *Proc. of the International Conf on Cognitive and Intelligent Computing: ICCIC 2021*, Vol. 2, pp. 193-204, 2023.

[19] Y. Khourdifi, A. Elalami, M. Bahaj, M. Zaydi, et O. Er-Remyly, "Framework for integrating healthcare big data using IoMT technology", In: *Computational Intelligence for Medical Internet of Things (MIoT) Applications*, p. 191-210, 2023.

[20] S. Dwivedi, R. Balaji, P. Ampatt, et S. D. Sudarsan, "A Survey on Security Threats and Mitigation Strategies for NoSQL Databases", In: *Information Systems Security*, pp. 57-76, 2023

[21] Y. D. Reddy, et A. P. Sajin, "An Efficient Traffic-Aware Partition and Aggregation for Big Data Applications using Map-Reduce", *Indian J. Sci. Technol*, mars, Vol. 9, No. 10 2016.

[22] W. C. Chung, H. P. Lin, S. C. Chen, M. F. Jiang, and Y. C. Chung, "J ackhare: a framework for sql to nosql translation using mapreduce", *Automated Software Engineering*, Vol. 21, No. 4, pp. 489-508, 2014.

[23] A. Karras, C. Karras, A. Pervanas, S. Sioutas, et C. Zaroliagis, "SQL Query Optimization in Distributed NoSQL Databases for Cloud-Based Applications", *Algorithmic Aspects of Cloud Computing*, p. 21-41, 2023.

[24] S. Sicari, A. Rizzardi, and A. C. Porisini, "Security&privacy issues and challenges in NoSQL databases", *Comput. Netw*, Vol. 206, 2022.

[25] C. J. F. Candel, D. S. Ruiz, et J. J. G. Molina, "A unified metamodel for NoSQL and relational databases", *Inf. Syst*, p. 101898, 2021.

[26] J. V. L. Cabral, V. E. R. Noguera, R. R. Ciferri, et D. Lucrédio, "Enabling schema-independent data retrieval queries in MongoDB", *Inf. Syst.*, p. 102165, 2023.

[27] A. Erraji, A. Maizate, et M. Ouzzif, "multi-criteria analysis between nosql databases categories toward a complete migration from relational database", *J. Theor. Appl. Inf. Technol.*, Vol. 100, No. 1, p. 9, 2022.

[28] M. Khan, F. Zaman, M. Adnan, A. Imroz, M. Rauf, and Z. Phul, "Comparative Case Study: An Evaluation of Performance Computation between SQL and NoSQL Database", *SJHSE*, Vol. 1, No. 2, pp. 14-23, 2023.

[29] A. E. Alami, and M. Bahaj, "The Road to a Full Migration of Relational Database (RDB) to Object Relational Database (ORDB): Semantic Enrichment, Target Schema, Data Mapping", *Int. J. Adv. Inf. Sci. Technol*, Vol. 3, No. 10, p. 8, 2014.

[30] A. E. Alami and M. Bahaj, "Framework for a complete migration of relational databases to other types of databases(object oriented OO, object-relational OR, XML)", In: *Proc. of 2016 IEEE/ACS 13th International Conf of Computer Systems and Applications (AICCSA)*, Agadir, Morocco, pp. 1-7, 2016.

[31] A. E. Alami and M. Bahaj, "Migration of the Relational Data Base (RDB) to the Object Relational Data Base (ORDB). World Academy of Science Engineering and Technology International Journal of Computer", *Information Science and Engineering*, Vol. 8, No. 1, 2014.

[32] A. E. Alami, and M. Bahaj, "Schema and Data Migration of a Relational Database RDB to the Extensible Markup Language XML", *Int. J. Comput. Inf. Eng.*, Vol. 9, No. 7, pp. 1756-1761, 2015.

[33] MongoDB, "MongoDB Documentation", [Online]. Available: https://www.mongodb.com/docs/.

[34] M. Bahaj, and A. Elalami, "The migration of data from a relational database (RDB) to an object relational (ORDB) database", *J. Theor. Appl. Inf. Technol*., Vol. 58, No. 2, 2013.

[35] A. E. Alami, and M. Bahaj, "Migration of a relational databases to NoSQL: The way forward", In: *Proc. of 2016 5th International Conf on Multimedia Computing and Systems (ICMCS)*, Marrakech, Morocco, pp. 18-23, 2016.

[36] A. E. Alami, and M. Bahaj, "The migration of a conceptual object model COM (conceptual data model CDM, unified modeling language UML class diagram...) to the Object Relational Database ORDB", *MAGNT Res. Rep*. (ISSN. 1444-8939), Vol. 2, No. 4, pp. 318-332, 2018.

[37] H. Matallah, G. Belalem, and K. Bouamrane, "Comparative study between the MySQL relational database and the MongoDB NoSQL database", *Int. J. Softw. Sci. Comput. Intell*, Vol. 13, No. 3, pp. 38-63, 2021.

[38] I. MongoDB, "MySQL To MongoDB Migration Guide." https://www.mongodb.com/basics/mysql-to-mongodb.