2023 6th International Conference on Advanced Communication Technologies and Networking (CommNet) | 979-8-3503-2939-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/CommNet60167.2023.10365296

Automating the Creation of Graph-Based NoSQL Databases in the Context of Big Data

Fouad ELOTMANI[1], Redouane ESBAI[1], and Mohamed ATOUNTI[1]

[1]Mohammed Premier University, Morocco
Email: f.elotmani@ump.ac.ma

***Abstract*—This paper focuses on the transformation of a relational database to a NoSQL database using Model-Driven Architecture (MDA). The research question addressed in this paper is how to automate the process of migrating a relational database to a NoSQL database using MDA. The main findings of this research show that MDA can be used to transform a relational database schema into a NoSQL database schema. The results also show that the transformation process can be automated using Atlas Transformation Language (ATL). The conclusion of this paper is that MDA can help organizations to save time and resources by automating the transformation process, which is critical in modern data-driven applications.**

***Index Terms*—Relational database, NoSQL database, MDA, ATL.**

I. Introduction

This paper aims to investigate how Model-Driven Architecture (MDA) is used to transform a relational database into a NoSQL database. Transforming data from a relational to a NoSQL database is an essential task in modern data-driven applications. This paper explains the importance of this transformation and how MDA can be used to automate the process.

The importance of transforming from a relational database to a NoSQL database lies in the evolving landscape of modern data management and application development. Here are several key reasons highlighting the significance of this transformation:

- **Flexible Schema:**
  Relational Databases: Follow a rigid, predefined schema requiring a fixed structure for data. NoSQL Databases: Offer a flexible schema, allowing for dynamic and evolving data models. This flexibility is particularly advantageous in scenarios where data structures are subject to frequent changes.
- **Scalability:**
  Relational Databases: Scaling vertically by adding more powerful hardware is a common practice, but it has limitations. NoSQL Databases: Excel in horizontal scalability, making it easier to handle large volumes of data and increased traffic by distributing the workload across multiple servers.
- **Handling Unstructured Data:**
  Relational Databases: Primarily designed for structured data, and handling unstructured data can be challenging. NoSQL Databases: Excel in managing unstructured or semi-structured data, making them suitable for applications dealing with diverse data types, such as social media content, log files, and user-generated content.
- **Agile Development and Faster Time-to-Market:**
  Relational Databases: Changes to the database schema can be time-consuming and may require downtime during updates. NoSQL Databases: Allow for agile development, enabling developers to iterate quickly and release updates without significant disruptions.
- **Cost Efficiency:**
  Relational Databases: Scaling vertically can become expensive as it involves investing in more powerful hardware. NoSQL Databases: Horizontal scaling on commodity hardware tends to be more cost-effective, especially for large-scale applications.
- **Polyglot Persistence:**
  Relational Databases: Often necessitate a single database technology for the entire application. NoSQL Databases: Facilitate polyglot persistence, enabling the use of multiple databases tailored to specific data storage requirements within a single application.
- **Performance Optimization:**
  Relational Databases: Might face performance bottlenecks when dealing with large datasets or high transaction volumes. NoSQL Databases: Designed to optimize performance for specific use cases, providing efficient data retrieval and storage mechanisms.
- **Adaptability to Evolving Business Requirements:**
  Relational Databases: Changes to the data model may require substantial effort and planning. NoSQL Databases: Accommodate changes in data models more seamlessly, aligning with the dynamic nature of evolving business requirements.

This research is driven by the imperative for a more streamlined and automated method of transitioning from a relational database to a NoSQL database. The conventional process involves manual steps and is time-intensive, demanding substantial human effort and resources. This paper makes dual contributions. Firstly, it explores the potential of Model-Driven Architecture (MDA) as a promising avenue for automating the transformation process. Secondly, it furnishes an ATL code, presenting a tangible solution for converting a relational

database schema into a NoSQL Graph-based database schema.

A. Importance of the transformation from a relational database to a NoSQL database

Relational databases and NoSQL databases differ significantly in terms of data modeling, querying and scalability [1].

- **Data Modeling:** Relational databases use a structured data model based on tables with fixed columns and rows. Data is organized into tables with relationships between them, and the data is normalized to minimize redundancy. In contrast, NoSQL databases use a schemaless data model that allows for flexible and dynamic data structures. The data is typically organized into collections or documents, and relationships between them are not explicitly defined.
- **Querying:** Relational databases use SQL (Structured Query Language) to query data, which is a standard language used by most relational database management systems. SQL is used to retrieve and manipulate data from tables using a set of predefined commands. NoSQL databases, on the other hand, use different query languages depending on the type of database. For example, MongoDB uses a document-based query language that allows for more complex queries than SQL.
- **Scalability:** Relational databases are vertically scalable, meaning that they can only scale up by adding more resources to a single server. This can become expensive and limit the scalability of the database. NoSQL databases are horizontally scalable, meaning that they can scale out by adding more servers to the database cluster. This makes them more costeffective and scalable for large amounts of data.

B. Differences between Relational and NoSQL Databases

Advantages and Disadvantages: Relational databases have the advantage of being reliable, consistent, and having a well established ecosystem of tools and frameworks. They are best suited for applications that require strong data consistency, transactions, and complex queries. However, they can be rigid and difficult to scale.

NoSQL databases have the advantage of being flexible, scalable, and able to handle unstructured and semi-structured data. They are best suited for applications that require high scalability, low latency, and fast data processing. However, they can be complex to manage, and the lack of data consistency can be challenging for some applications.

In conclusion, both relational and NoSQL databases have their strengths and weaknesses, and the choice of database depends on the specific requirements of the application. Understanding the differences between the two types of databases is essential for selecting the appropriate database model for the task at hand. [2]

C. NoSQL Graph Databases

NoSQL graph databases, a specialized subset within the NoSQL domain, proficiently manage vast and interconnected datasets by prioritizing the relationships between entities. Specifically, semantic graph databases like RDF triplestores exhibit adeptness in integrating diverse data from multiple sources and extracting new insights from existing information. Operating without rigid schemas, these databases offer dynamic data handling capabilities, particularly beneficial for relationship-focused analytics and real-time big data analysis. Their adaptability in integrating varied data sources eliminates the need for extensive schema modifications, reducing complexities and integration costs. Compliant with global standards such as W3C, they ensure data interoperability and streamline sharing. Leveraging inference capabilities, these databases unveil implicit connections within data, enabling organizations to derive invaluable insights crucial for informed decision making processes [3].

II. Related Work

Alotaibi and Pardede introduced structured methodologies for transforming relational databases (RDB) into diverse NoSQL schemas, encompassing document-based, column-based, and graph-based databases [4]. Their work aimed to address the lack of standardized frameworks for NoSQL databases by devising specific transformation rules. This influential study validated its proposed rules through practical case studies involving MongoDB, Cassandra, and Neo4j.

Sellami, Nabli, and Gargouri delved into the transformation of data warehouse schemas into NoSQL graph databases, emphasizing the scalability and flexibility inherent in NoSQL databases, particularly within OLAP systems [5]. They proposed novel transformation rules targeted at converting multidimensional conceptual models into NoSQL graph-oriented models.

Aftab, Iqbal, Almustafa, Bukhari, and Abdullah focused on automating the transformation of unstructured NoSQL databases into structured SQL-based relational databases [6]. They introduced an efficient method for this conversion, showcasing significant performance enhancements compared to existing methodologies.

Oliveira, de Souza, Moreira, and Seraphim conducted a systematic literature review on techniques for mapping between relational and graph-oriented databases [7]. Their study highlighted various methodologies attempting mapping and migration processes, aiming to amalgamate strengths and address limitations observed in current methodologies.

Abdelhedi, Ait Brahim, Atigui, and Zurfluh proposed an automatic MDA-based approach to translate conceptual models expressed in UML into NoSQL physical models [8]. Addressing the challenge of storing Big Data in NoSQL systems, their approach offers mapping rules from a conceptual level to a physical one. It relies on an intermediate logical model compatible with various NoSQL system types (column, document, and graph-oriented), providing flexibility to choose the system that best aligns with business rules and technical constraints.

III. Transformation from Relational to NoSQL Graph Database using UML-Based Model Transformation

To effectively transform a relational database into a NoSQL graph database, we employ a UML-based Model-Driven Architecture (MDA) approach. Our process revolves around the conversion of a UML class diagram model (`UMLModel`) representing a relational database into a NoSQL graph database model (`GraphModel`). The UML metamodel incorporates classes and associations, while the graph metamodel involves nodes and edges [9].

A. Language of Transformation Between Metamodels:

The language of transformation between metamodels constitutes a critical aspect of model-driven engineering, providing a systematic framework for converting models across diverse representations. Notably, Query/View/Transformation (QVT) and Atlas Transformation Language (ATL) stand out as prominent languages in this domain.

QVT, a standardized family of languages defined by the Object Management Group (OMG), offers a robust solution for specifying precise mappings and transformations. Its declarative nature allows modelers to express transformations concisely, capturing the intricacies of different metamodels. QVT plays a crucial role in maintaining fidelity during model conversions, ensuring that the essential characteristics and relationships of the original models are preserved.

ATL complements QVT by providing a model-to-model transformation language specifically tailored for the Eclipse Modeling Framework (EMF). It excels in enabling the definition of transformations between models conforming to EMF-based metamodels. ATL utilizes a rule-based paradigm, where transformations are specified through a set of rules defining how elements in the source metamodel correspond to elements in the target metamodel. This makes ATL particularly effective for scenarios where EMF is prevalent.

Together, QVT and ATL contribute significantly to the efficiency and accuracy of metamodel transformations. These languages empower model-driven development by automating complex processes and ensuring a seamless transition between different metamodels, fostering a more adaptable and agile development paradigm.

B. UML source meta-model

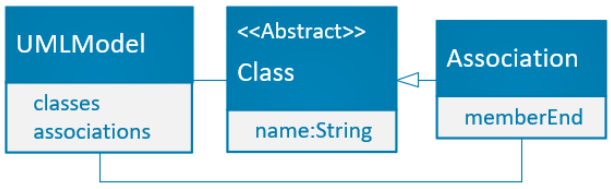

Fig. 1: Simplified UML source meta-model

This simplified UML meta-model (Fig. 1) includes a `Class` with a `name` attribute and an `Association`with a `memberEnd` reference representing the association ends. The `UMLModel` class includes references to classes and associations.

C. Graph-oriented target meta-model

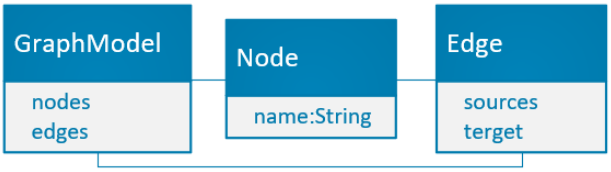

Fig. 2: Simplified Graph-oriented target meta-model

This simplified graph metamodel (Fig. 2) includes a `Node` with a `name` attribute and an `Edge` with `source` and `target` references representing the connections between nodes. The `GraphModel` class includes references to nodes and edges.

D. ATL Transformation: UML to Graph-Oriented Database.

We will apply our transformation on the UML class diagram model (`UMLModel`) representing a relational database into a NoSQL graph (`GraphModel`). The transformation involves creating nodes for each class in the UML model and edges for associations.

The ATL code :

```
-- Transformation Module
module UMLToGraph;

create OUT : GraphModel from IN : UMLModel;

-- Rule for transforming classes
rule Class2Node {
    from
        c: UML!Class
    to
        n: Graph!Node (
            name <- c.name
        )
}

-- Rule for transforming associations
rule Association2Edge {
    from
        a: UML!Association
    to
        e: Graph!Edge (
            source <-
                a.memberEnd->at(1).type.name,
            target <-
                a.memberEnd->at(2).type.name
        )
}

-- Main entry point
rule UMLToGraphRule {
    from
        umlModel: UMLModel
    to
```

```
        graphModel: GraphModel
    do {
        -- Apply transformation rules
        Class2Node;
        Association2Edge;
    }
}
```

- `Class2Node` rule transforms UML classes into nodes in the graph model.
- `Association2Edge` rule transforms UML associations into edges in the graph model.
- `UMLToGraphRule` is the main entry point for the transformation, where the individual rules are applied.

E. Example of UML to Graph Model Transformation

We present a hypothetical example based on the simplified metamodels and transformation rules discussed earlier.

UML Model (Input):

```
@startuml
class Person {
    - name: String
    - age: int
}

class Address {
    - street: String
    - city: String
}

Association:
Person -- Address
@enduml
```

Graph Model (Output, Hypothetical Result):

```
// Nodes
Node Person {
    name: "Person"
}

Node Address {
    name: "Address"
}

// Edges
Edge PersonToAddress {
    source: Person
    target: Address
}
```

IV. Result

In this hypothetical example, applying the ATL code gives the following result :

- The UML class diagram comprises two classes: `Person` and `Address`, along with an association between them.
- The transformation rules generate nodes in the graph model for each class and edges for the associations between classes.
- The resulting graph model contains two nodes, one for each class, and an edge representing the association between `Person` and `Address`.

V. Conclusion

The transformation from a relational database to a NoSQL graph database using a model-driven architecture presents diverse and multifaceted challenges, as summarized in Table 1. This tabulated overview delineates critical complexities, emphasizing the need for a strategic and comprehensive approach to navigate the transformation successfully.

Complexity	Description	Numerical Value
Data Model Transformation	The need to map the existing relational schema to a graph model, identifying entities, relationships, and properties.	4
Query Language Shift	Developers accustomed to SQL queries will need to adapt to the graph query language (e.g., Cypher for Neo4j). Training and skill development may be required.	3
Data Migration Challenges	Moving data from a relational database to a graph database involves careful consideration of data types, relationships, and ensuring data integrity during the transition.	5
Performance Tuning	While graph databases excel in certain types of queries, performance may still be a concern for specific use cases. Optimizing queries and indexing strategies may be necessary.	4
Schema Evolution	The flexibility of NoSQL databases can simplify schema evolution, yet maintaining consistency and managing changes in a distributed environment requires careful planning.	3
Application Code Modification	Existing application code reliant on SQL-based queries will need modifications to adapt to the new graph database query language and data model.	4
Skill Set Transition	Developers, administrators, and team members may need to acquire new skills related to graph databases, including understanding graph theory and the specific features of the chosen graph database.	5
Tooling and Ecosystem Differences	Graph databases have unique tools and ecosystems. Integrating these into existing development and deployment pipelines may necessitate adjustments.	3
Cost Considerations	Evaluating the cost implications of the new database system, including licensing fees, infrastructure requirements, and ongoing maintenance costs.	4
Testing and Quality Assurance	Rigorous testing is essential to prevent data inconsistencies, query errors, or performance issues post-transformation. Quality assurance processes need adaptation.	4

TABLE I: Complexities associated with database transformation from a relational database to a NoSQL graph database

In conclusion, the complexities associated with the transformation are varied and demand a strategic and comprehensive approach. While challenges exist, addressing each dimension with careful planning, skill development, and a nuanced under-

standing of organizational context is essential for a successful transition. This table provides a structured overview of the complexities, allowing stakeholders to prioritize and address each aspect effectively.

REFERENCES

[1] Douglas Kunda and Hazael Phiri. A comparative study of nosql and relational database. 1:1–4, Dec. 2017.

[2] Kosovare Sahatqija, Jaumin Ajdari, Xhemal Zenuni, Bujar Raufi, and Florije Ismaili. Comparison between relational and nosql databases. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 0216–0221, 2018.

[3] Jaroslav Pokornỳ. Graph databases: their power and limitations. In *Computer Information Systems and Industrial Management: 14th IFIP TC 8 International Conference, CISIM 2015, Warsaw, Poland, September 24-26, 2015, Proceedings 14*, pages 58–69. Springer, 2015.

[4] Obaid Alotaibi and Eric Pardede. Transformation of schema from relational database (rdb) to nosql databases. *Data*, 4(4), 2019.

[5] Amal Sellami, Ahlem Nabli, and Faiez Gargouri. Transformation of data warehouse schema to nosql graph data base. In Ajith Abraham, Aswani Kumar Cherukuri, Patricia Melin, and Niketa Gandhi, editors, *Intelligent Systems Design and Applications*, pages 410–420, Cham, 2020. Springer International Publishing.

[6] Shah Nazir, Zain Aftab, Waheed Iqbal, Khaled Mohamad Almustafa, Faisal Bukhari, and Muhammad Abdullah. Automatic nosql to relational database transformation with dynamic schema mapping. *Scientific Programming*, 2020:8813350, 2020.

[7] Anderson Tadeu de Oliveira, Adler Diniz de Souza, Edmilson Marmo Moreira, and Enzo Seraphim. Mapping and conversion between relational and graph databases models: A systematic literature review. In Shahram Latifi, editor, *17th International Conference on Information Technology–New Generations (ITNG 2020)*, pages 539–543, Cham, 2020. Springer International Publishing.

[8] Fatma Abdelhedi, Amal Ait Brahim, Faten Atigui, and Gilles Zurfluh. Umltonosql: Automatic transformation of conceptual schema to nosql databases. In *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, pages 272–279, 2017.

[9] Yelda Unal and Halit Oguztuzun. Migration of data from relational database to graph database. In *Proceedings of the 8th International Conference on Information Systems and Technologies*, pages 1–5, 2018.