Enhancement of a Transformation Algorithm to Migrate SQL Database into NoSQL Graph Database

HIRA LAL BHANDARI
ROSHAN CHITRAKAR

ABSTRACT

With the proliferation of data, relational database management systems face the challenge of managing big volumes of data because of the absence of dynamic data models, high functioning, high scalability, high availability, and low cost. The NoSQL technology has the features which RDBMS lacks. Therefore, many organizations are migrating from RDBMS to NoSQL databases. The migration from RDBMS to NoSQL is not always effective if the data is lost, if the data is replicated while transmission, and if the performance is degraded. For this reason, enhancement of transformation algorithm proposed by Shabana Ramzan et al. has been performed to implement the methodology which is used to successfully migrate data from structured and normalized SQL database MySQL to NoSQL graph database Neo4j. SchemaCrawler engine has been used to metamodeling from source to destination database and execute the transformation rules. This study has also evaluated and validated the performance of the enhanced algorithm after successful migration of data. The NoSQL graph database Neo4j handles the most common form of data that are unstructured, semi-structured, and structured as well as they support the ACID property of relational databases. This feature is not supported by other NoSQL databases such as Oracle NoSQL, MongoDB, and HBase databases.
LIST OF ABBREVIATIONS

CSV  Comma-separated values
DBMS  Database Management System
IDE  Integrated Development Environment
JSON  JavaScript Object Notation
Neo4j  Network Exploration and Optimization “for” Java
NOSQL  Not Only Structure Query Language
RDBMS  Relational Database Management System
SQL  Structured Query Language

1. INTRODUCTION

In NoSQL, databases are not built principally on tables, and commonly do not have the concept of using Structured Query Language for data manipulation (Moniruzzaman & Hossain 2013) unlike a Relational Database Management System. Nowadays, non-relational model NoSQL database systems are fast prominence as an alternative model for handling the data due to many advantages (Ba, Di & Ba 2013) which are: easily scalable, high performance, highly available, dynamic data model, fast development cycle, low cost, and so on. NoSQL databases are in different types according to their database modeling ways. Different forms of databases are: key-value, document, column oriented and graph oriented (Khare 2016).

A graph data storage is a system that incorporates graph constructions as nodes and edges, to get representation and storage of data. The mostly preferred graph data model is considered to be labeled property archetypal (Pokorný et al. 2017). The NoSQL graph database nowadays is gaining more popularity due to its flexible structure of representing almost all kinds of databases such as organized, semi-organized and unorganized. There is a growing status of data generation and almost all of them are connected because of rapid expansion of data from web and social networks, and therefore, most of the companies compulsorily need to migrate/transform their large volume of data systems into new data management systems which system enables to handle a large volume of highly connected data.

Ramzan et al. (2019) proposed an algorithm to transfer data from SQL database to NoSQL graph database. They also presented a framework that implements a transformation algorithm using SiTra tool to automatically map metadata and transfer database of SQL server to Oracle NoSQL. The algorithm implemented has used schema and object modeling between SQL database, that is, SQL server and NoSQL database, that is, Oracle NoSQL and transformed dataset from SQL to NoSQL. But we have identified a few problems viz. (1) The instantiation of table from source database to destination database Oracle NoSQL cannot be useful for that of MySQL to Neo4j graph database; (2) Schema mapping and data modeling of database from SQL server to Oracle NoSQL has different in that of MySQL to Neo4j; (3) Mapping column from MySQL Server to Oracle NoSQL and MySQL to Neo4j have different rules; and (4) Implementation of transformation algorithm using SiTra tool requires different transformations rules instead of using SchemaCrawler for the data migration.

And hence, this research work aims to enhance the transformation algorithm proposed by Ramzan et al. This study also presents the evaluation of the efficiency with different key matrices and validation of database using data manipulation language (DDL) and data definition language (DML).

2. RELATED WORKS

Hira Lal Bhandari*, and Roshan Chitrakar (Journal, Bhandari & Chitrakar 2020) have introduced a conversation, investigation, and correlation among seven distinct methods of information movement from SQL data set to NoSQL data set. Using SysGauge, a system tool, the migration was carried out with the appropriate tools and frameworks for each method, and the outcomes were evaluated, analyzed, and verified. Different parameters like speed, execution time,
maximum CPU usage, and maximum memory usage have been used in the analysis and comparison of databases.

Rahma S. Al Mahruqi, Manar H. Alalfi, and Thomas R. Dean (2020) demonstrates a semi-automated framework for the analysis and migration of three existing web applications in order to migrate the PHP code to interact with the migrated database and extract, classify, translate, migrate, and optimize the queries. He describes the four phases of the strategy: The schema is moved in the first stage, and the data from the SQL database is moved into the new NoSQL schema in the second stage. The migration and optimization of the SQL queries into MongoDB actions that are equivalent to them is the next step. In the last stage, the adaption of the application code utilizes the deciphered questions. This approach gives the accompanying commitments: assessing and executing existing pattern relocation calculations proposed by Arora et al. and Jia et al. to move the mapping from MySQL to NoSQL and relocating and improving the implanted SQL questions to communicate with the new data set framework and changing the application code to utilize the deciphered inquiries.

J. Christina* and S. Saranya** (Christina & Saranya 2018) have implemented the extended form of the existing algorithm from data migration from MySQL to MongoDB and presented the comparative study on their performances. The existing algorithm proposed by Liana et al. unfilled a structure which apparatuses an algorithm with automatic mapping from MySQL to MongoDB. The algorithm has used the MySQL INFORMATION_SCHEMA that gives entree to database metadata and represents the entity relationship in MongoDB using embedding and linking strategies (structure mode).

Jaejun Yoo et al. (Yoo, Lee & Jeon 2018) have presented the naive migration method based on table-level de-normalization, novel migration method based on column-level de-normalization and atomic aggregates in their paper. They have also performed comparative study on the performance of those three migrating concepts. They have used separate algorithms for respective concepts.

Ying-Ti Liaoa, Jiacheng Zhou, et al. (Liao et al. 2016) have utilized a data adapter system that is layered between applications and databases and highly modularized. Database transformation and queries can be carried out simultaneously from the application. To access both a relational and a NoSQL database, the system makes it easy to use a SQL interface that parses query statements. They enable applications to conduct queries regarding whether or not target data are being transformed and provide a mechanism for controlling the process of database migration. There is a patch mechanism for synchronizing inconsistent tables after data transformation.

Fatima Jamal Al Shekh Yassin (Yassine & Awad 2019) presents research by simulating a comparison of the performance of retrieving data from various MongoDB representations. The conclusion is based on some instances in which it is preferable to keep all of the data in one document rather than using multiple collections to represent massive amounts of data with intricate relationships. The MySQL server database is made up of five tables that hold information about employees and have various kinds of relationships between them. Using NoSQL DB Mongo, which stores data as documents, the data was represented. The MySQL DB and three distinct Mongo designs were compared in terms of data reading performance. The three designs were chosen to cover the different relationships between documents in MongoDB: reference relationships between two documents, each of which has a different level of embedding, fully reference relationships between five documents, and fully embedded relationships between two documents.

Alza A. Mahmood (Aftab et al. 2020) has presented an automated algorithm and developed a prototype for the structural type conversion from SQL data form to NoSQL data form with the automatic transfiguration features available in an algorithm. Different structure has been defined under different choices, such as Auto, Embedded and Linking.

Shabana Ramzan et al. (Ramzan et al. 2019) have proposed a method that has two building blocks: data transformation and data cleansing module. This study has been performed in two phases: (i) Migration of SQL database to Oracle NoSQL database with model-model conversion and (ii) Improvement of data quality through data cleansing ability to prepare it for big data analysis. Experiment has been performed to show the successfully transformation from structured database to non-SQL database and improving the data excellence.
Roberto De Virgilio et al. (De Virgilio, Maccioni & Tortone 2014) have provided a tool for automatically converting a SQL database into a graph database. The application’s persistent layer can be converted using this tool from a relational to a graph format, which can be very helpful for the flexible and effective management of graph-based data like social and semantic Web data. The Tinkerpop1 framework is used to implement the abstract interface over graph databases in R2G, which was developed in Java. There are four main components in R2G: i) the Metadata Analyzer, which checks the schema and constraints of the source database and creates the corresponding schema graph; ii) the Data Mapper (DM), which creates the data mapping; iii) the Query Mapper (QM), who is in charge of translating queries; and iv) the Graph Manager (GM), which actually moves the data and runs queries over the target database using mappings created by the DM and QM. The output of MA is utilized by the DM and the QM.

A method for migrating data from a Relational database to a NoSQL graph database was proposed in the paper by Mehak Gupta and Dr. (Mrs.) Rinkle Rani (Gupta 2014). Data can be transformed from a relational database to a graph database using foreign key relations in a relational database. A procedure for switching inquiries from SQL over completely to encode question language has likewise been proposed. A similar examination of SQL and CQL is performed based on the execution time in milliseconds.

3. METHODOLOGY

The work presented in this work is the enhancement of the transformation algorithm proposed by Shabana Ramzan et al. (Ramzan et al. 2019) to renovate data from MySQL server to Neo4j NoSQL diagram database. Automatic transformation of data from MySQL server to Neo4j is presented in the study. Model-model transformation method has been taken into implication. It is a method for creating software that uses models. The inputting model, the renovation explanation, the conversion mechanism, and the translation procedures are all necessary components for model transformation. Figure 1 depicts the entire model transformation framework.

Here we have presented the diagrammatic representation of migration procedure. The diagram clearly shows the process to convert and migrate the database from MySQL to Neo4j graph database.
From Figure 2 we can summarize the process flow of the application as per mentioned below:

1) SchemaCrawler engine extracts metadata and capture data interacting with SQL database.

2) Metamodelling of the source database i.e. MySQL is performed.

3) Now SchemaCrawler engine executes the transformation rule with the incorporation of enhanced algorithm.

4) SchemaCrawler establishes the connectivity with Neo4j using Java driver.

5) SchemaCrawler maps metadata from source model and convert data into respective elements as mentioned in the transformation rules.

6) Through metamodelling data is mapped and converted to destination database model, that is, the NoSQL graph model.

If we want to get efficient ETL (Extract, Transform, Load) from SQL database to NoSQL graph database, each table data and properties should be accessed as minimum as possible. Therefore, we inspect metadata before transformation of migration step forward. We should also identify relationships between tables, recursive relationship, primary key, foreign and other attributes that are not part of primary key or foreign key. SchemaCrawler is used to achieve all those aforementioned tasks (Mohamed, Abdel-Fattah & Abdelgaber 2021). Figures 3 and 4 clearly depicts meta modeling structure of relational and non-relational graph databases respectively.

Here in the existing model to model transformation technique, SchemaCrawler tool has been used as engine to reduce the time for metadata and data extraction from MySQL and then migration of that data to NoSQL graph database Neo4j. Due to some major limitations of SiTra as mentioned by D. H. Akehurst et al. [Akehurst et al. 2006], the tool ‘SchemaCrawler’ is worth considered for the efficiency of the migrating process. The limitations are:
• SiTra can’t be utilized when there is more than one decision that ought to guide to a similar objective item. This is due to the fact that SiTra cannot be used to determine which rule should map to the same target object.
• Additional constraint concerns the recursive invocation of procedures.
• SiTra and ETL give a direct hint of what rules and data sources have made what yields.

Although different tools available such as SchemaSpy online, SchemaCrawler is selected for meta modeling of MySQL database and Neo4j graph database as well as executing the transformation rules to transform the dataset due to support of following features:

• SchemaCrawler provides extracted information about database schemas in various formats (e.g., text, json, csv, html5) (SchemaCrawler – Free database schema discovery and comprehension tool n.d.).
• The API allows to access to SQL database in a homogeneous way (SchemaCrawler – Free database schema discovery and comprehension tool n.d.).
• SchemaCrawler is not only a diagramming tool, but also a tool for discovering and understanding database schemas, with dominant scripting capabilities using JavaScript, Groovy, Ruby or Python (Database Schema Drawing tools – Database Processing n.d.).
• It is highly customizable where more use cases can be achieved using the boilerplate code provided in the example distribution (Gowrisangar n.d.).
• For programmers, there is a rich Java API for database object metadata and integration with Maven repositories (Explore Your Database Schema with SchemaCrawler – DEV Community n.d.).

There are different ways to analyze the metadata of a schema in a database, including SchemaCrawler. SchemaCrawler is portrayed as ‘Open-source Java API that produces working with database metadata as simple as working with plain ancient Java objects’ and is an app (SchemaCrawler Alternatives and Similar Software|AlternativeTo n.d.). There are three choices to SchemaCrawler for Windows, Linux and Mac. The leading SchemaCrawler elective is DbSchema. It is not free. A free option is the SchemaSpy tool. We here demonstrate some different techniques so that we can remark the important key points on why we choose SchemaCrawler instead of DbSchema and SchemaSpy. Table 1 illustrates the differences clearly.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>SchemaCrawler</th>
<th>DbSchema</th>
<th>SchemaSpy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Free</td>
<td>Commercial</td>
<td>Free</td>
</tr>
<tr>
<td>Export type</td>
<td>CSV, HTML, JSON, Plain text</td>
<td>HTML, MS Excel, PDF</td>
<td>HTML</td>
</tr>
<tr>
<td>Runs on</td>
<td>Linux, Mac OS, Windows</td>
<td>Windows</td>
<td>Linux, Mac OS, Windows</td>
</tr>
</tbody>
</table>

Most popularly used two different tools are SchemaCrawler and SchemaSpy in metadata modeling and reengineering tool for the visualization of metadata of database schema. SchemaCrawler engine is used to map schema of SQL database and NoSQL graph database and then SchemaCrawler execute the transformation rules to migrate data set from MySQL server to Neo4j NoSQL graph database due to the following reasons (SchemaCrawler – Free database schema discovery and comprehension tool, n.d.):

i. SchemaCrawler provides support of almost any database that has JDBC driver.
ii. SchemaCrawler can be utilized with any working framework that upholds Java SE 8, CompactProfile 2, or later.
iii. SchemaCrawler is as well Java API, which makes it as easy to work with database metadata as it does for plain old Java objects.
iv. SchemaCrawler can be utilized with any working framework that upholds Java SE 8, CompactProfile 2, or later.

With the assistance of various Java APIs, SchemaCrawler is integrated with the same Java applications. In order for us to select from or even make changes to your database, a live connection to the script context is established. Working with database metadata is made
as simple as working with standard Java objects with SchemaCrawler, a Java API. When programmatically determining the capabilities of a particular RDBMS and locating the names and types of tables and columns in the database, we require access to the metadata of the database in order to dynamically generate SQL statements. Thus we can make the implementation of the MDD (Model Driven Development) approach to migrate data by incorporating these two main aspects.

a) The tool ‘SchemaCrawler’ is used programmatically using spring boot framework.

b) Plain old Java object (POJOs) as well as Java APIs are used for database mapping and instantiation for Meta modeling.

3.1 ENHANCEMENT OF TRANSFORMATION ALGORITHM

Improvement for transformation algorithm proposed by Shabana Ramzan et al. (2019) has been done as per the table shown below. Improvement has been done in this way so as to implement the algorithm to accomplish the conversion. The technique to perform translation of dataset from normalized structure of MySQL to NoSQL graph database Neo4j is same that was used to implement a transformation algorithm proposed by Shabana Ramzan et al. (2019). Table 2 shows how enhancement is done to implement the migration process.

<table>
<thead>
<tr>
<th>ALGORITHM PROPOSED BY SHABANA RAMZAN ET AL. TO MIGRATE DATA FROM MS SQL SERVER TO ORACLE NoSQL</th>
<th>ALGORITHM AFTER MODIFICATION TO MIGRATE DATA FROM MYSQL SERVER TO NEO4J GRAPH DATABASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Construct: Make a lean of the instances in the target tabular from those in the source table.</td>
<td>2. Construct: Creating a rundown of targeted nodes from the source table’s items.</td>
</tr>
<tr>
<td>3. Representing: Target tabular instances map to objects in the source table.</td>
<td>// Table object of SQL table is addressed as a named hub in a diagram data set</td>
</tr>
<tr>
<td>4. Construct: make an article for every section of the source table.</td>
<td>3. Mapping: Target node objects map to objects in the source table.</td>
</tr>
<tr>
<td>5. Mapping: Map from target field object to source column object.</td>
<td>// A node in a graph database has been used to map the source table’s schema.</td>
</tr>
<tr>
<td>6. If value is true</td>
<td>4. Construct: make an article for every section of the source table.</td>
</tr>
<tr>
<td>7. Pick the rule for the transformation.</td>
<td>5. Mapping: Target node property to object map from source column.</td>
</tr>
<tr>
<td>8. // the standard is appropriate or not when more than one rule is free for a similar sort of item.</td>
<td>// Oracle’s NoSQL source column is mapped with a table object, whereas the graph database’s source column is mapped with a node property.</td>
</tr>
<tr>
<td>9. endif</td>
<td>6. If value is true</td>
</tr>
<tr>
<td>10. Construct: The object for the target table is created.</td>
<td>7. Pick the rule for the transformation.</td>
</tr>
<tr>
<td>11. Construct: Every column’s target object is made.</td>
<td>// transformation rule that has been modified in accordance with the data set must be mapped and transformed from a SQL to a NoSQL graph database, whereas the previous algorithm utilized transformation rule that was transferred from an Oracle non-structured database to a relational database.</td>
</tr>
<tr>
<td>12. Select primary key.</td>
<td>8. // the standard is appropriate or not when more than one rule is free for a similar sort of item.</td>
</tr>
<tr>
<td>13. Keep a background marked by changed objects // to prevent the creation of target objects in duplicate.</td>
<td>9. endif</td>
</tr>
<tr>
<td></td>
<td>10. Create: The instance with the targeted labeled node is created.</td>
</tr>
<tr>
<td></td>
<td>// In a graph database, the object that represents the source table is labeled a node.</td>
</tr>
<tr>
<td></td>
<td>11. Construct: target hub property of every section is made. // the section is planned and changed as hub property in a diagram data set.</td>
</tr>
<tr>
<td></td>
<td>12. Select primary key.</td>
</tr>
<tr>
<td></td>
<td>13. Keep a background marked by changed objects // to prevent the creation of target objects in duplicate.</td>
</tr>
</tbody>
</table>

We have come across a crucial improvements in the algorithm (Ramzan et al. 2019) as shown in the right column of the above table. The improvements have been done from the following important aspects:
The enhanced algorithm uses this transformation rule below shown. This rule has an association with target model and the source model. Transformation rule has been pinpointed below:

Change rules

1. **ER-Normal Models: Rule of Transformation**
   a. Table become nodes
   b. Table name as node label
   c. Segments are changed over completely to hub properties
   d. If necessary, convert the values
   e. Keys from other countries n, n: 1) into connections, segment name into relationship-type

2. **Relationships are represented by join tables.**
   a. Additionally, two FKs and additional tables without domain identity (PK)
   b. Sections transforms into relationship properties

3. **ER-Normal Models: Rules for Cleanup**
   a. Eliminate specialized IDs (auto-increasing PKs)
   b. Maintain domain IDs like ISBNs
   c. Include lookup field indexes.
   d. Change the labels for the property Name, REL_TYPE, and Label.

4. **The Import Tool for Relational Database Management Systems consists of:**
   a. JDBC for database connection independent of vendor
   b. Database metadata can be extracted using SchemaCrawler.
   c. Transform a graph model import using rules
   d. A way to change the default behavior that is optional
   e. Simultaneously scans tables for relationships and nodes.

### 3.2 DATA DESCRIPTION

The research is basically the improvement of transformation algorithm proposed by Shabana Ramzan et al. (2019) move the informational collection from MySQL to Neo4j. The data set used to migrate from MySQL to Neo4j is structured. All the datasets of MySQL are structured. Here in the Table 3, we have 5 different data sets for the migration purpose:

<table>
<thead>
<tr>
<th>SN</th>
<th>DATA SET</th>
<th>SOURCE</th>
<th>FORMAT</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Classicmodels</td>
<td><a href="https://www.mysqltutorial.org/mysql-sample-database.aspx">https://www.mysqltutorial.org/mysql-sample-database.aspx</a></td>
<td>.sql</td>
<td>205 KB</td>
</tr>
<tr>
<td>2</td>
<td>Sampledb</td>
<td><a href="https://sample-videos.com/download-sample-sql.php">https://sample-videos.com/download-sample-sql.php</a></td>
<td>.sql</td>
<td>4,340 KB</td>
</tr>
<tr>
<td>3</td>
<td>Northwind</td>
<td><a href="https://en.wikiversity.org/wiki/Database_Examples/Northwind/">https://en.wikiversity.org/wiki/Database_Examples/Northwind/</a></td>
<td>.sql</td>
<td>64 KB</td>
</tr>
<tr>
<td>4</td>
<td>Hrdb</td>
<td><a href="https://www.sqltutorial.org/sql-sample-database/">https://www.sqltutorial.org/sql-sample-database/</a></td>
<td>.sql</td>
<td>23 KB</td>
</tr>
<tr>
<td>5</td>
<td>chinook</td>
<td><a href="https://github.com/lerocha/chinook-database/releases?fbclid=IwAR34vMltOsbXO_bBu2Rkw3dvWVQqboP18IF1c9JfW5jCQ2ENyyBYv7_8">https://github.com/lerocha/chinook-database/releases?fbclid=IwAR34vMltOsbXO_bBu2Rkw3dvWVQqboP18IF1c9JfW5jCQ2ENyyBYv7_8</a></td>
<td>.sql</td>
<td>1,827 KB</td>
</tr>
</tbody>
</table>

Table 3 Data Source.
The structured databases sampledb, northwind, hard, and classic models are made up of a number of tables, relationships, and constraints. The CMS as a model is discussed in this section. The relationship between the various entities, primary and foreign keys, and data types is depicted in the Figure 5.

The database for the classic model is essentially a sample database of scale car model retailers. customers, products, sales orders, and sales order line items, among other things, are included (MySQL Sample Database, n.d.).

![Figure 5 Retailer of Scale Models of Classic Cars Database.](image)

Database: sampledb

Schematic view of a single table, user_details, with fifty thousand records, as shown in the Table 4.

<table>
<thead>
<tr>
<th>FIELD</th>
<th>TYPE</th>
<th>NULL</th>
<th>KEY</th>
<th>DEFAULT</th>
<th>EXTRA</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_id</td>
<td>int</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>Username</td>
<td>varchar(255)</td>
<td>YES</td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>first_name</td>
<td>varchar(50)</td>
<td>YES</td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>last_name</td>
<td>varchar(50)</td>
<td>YES</td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gender</td>
<td>varchar(10)</td>
<td>YES</td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td>varchar(50)</td>
<td>YES</td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td>tinyint</td>
<td>YES</td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4 Schematic View of sampledb Database.
Database: northwind

Figure 6 shows relationship of data with different types of relations.

Database: hrdb

Database hrdb has different 7 tables. The database schematic view is presented in the Figure 7.
Database: chinook

Database chinook has different 11 tables. The database details has been presented in the Figure 8 in Entity Relationship Diagram form.
Briefly discussed are the fundamental components of the proposed method for transferring data from the MySQL server to Neo4j:

**SchemaCrawler:** SchemaCrawler is an open-source Java API that makes it simple to work with database metadata and regular Java objects. In addition to documenting tools for the schema, SchemaCrawler is a tool for locating and comprehending database schema. It is intended that the output be distinguished from other database schemas. It is bundled with drivers for some commonly used RDBMS systems for convenience, but it supports nearly all databases that have a JDBC driver. It works with any working framework that upholds Java.

**Transformation module:** A crucial component of the model-driven engineering paradigm is model transformation. The domain’s conceptual models can be created and played with in this automated manner. For this reason, the social Meta model is looked and separated, and planned with the Meta model of the NoSQL diagram information base so relocation can be performed from the SQL data set to the NoSQL chart data set.

### 3.4 TRANSFORMATION EVALUATION METRICS

Different cost metrics have been used to evaluate the DBMS’s performance. Additionally, these metrics are utilized to assess and verify the validity of database transformation procedures. The matrices used by Management and Url (2018) and Shabana Ramzan et al. (2019) are (a) schema information preservation, (b) data equivalence, and (c) system performance. These metrics are also used to verify the process of checking that software achieves its goal without any bugs. Performance evaluation with these metrics ensures whether the product that is developed is in right track or not. That means the process of evaluating metrics verifies whether the developed product meets the requirements that are pinpointed in the 'Statement of the Problem'.

1) **Schema information preservation:** Schema information is compared between source data store and target data store as mentioned in Fahrner & Vossen (1995) and Ramzan et al. (2019). The comparison is performed with semantics used in both kinds of store. Assuming the linguistic structure and semantics of the source model idea have been utilized accurately, then the diagram will be right. The execution of migration tool from MySQL to Neo4j developed has proved the syntactically and semantically migration process is valid. Migration from MySQL to Neo4j includes conversion and transformation of syntax and semantics from relational database MySQL into Neo4j graph database correctly. A formal foundation for accurate schema transformations has been developed by Pasteur, Koch and Robert in the Xiaolei Qian Computer Science Laboratory (Pasteur & Koch 1941). The formalization of schema transformations as signature interpretations, the formalization of accurate schema transformations as information-preserving signature interpretations, and the sanctification of schemas as abstract data types all fall under this category.

![Figure 9 The Process of Schema Mapping for Data Transformation (Legler & Naumann 2007).](image-url)
Data conversion and schema mapping are depicted in Figure 9. Schema matching is the first step after selecting the source and target schemas. Experts specify correspondences between various schema elements in this step. Tools that are only partially automated will probably help with this. Schema mapping is the outcome of this step. The subsequent step deciphers this planning as an information change. Typically, this transformation is carried out using the relevant data model (SQL, CQL). The target schema is created by transforming the source data using querying. Transformation queries can be used to hypothetically rewrite the query against the target schema or to populate the target database using a materialized approach. Casually, the understanding of diagram planning faces a few challenges.

**Source schema interpretation:** During transformation, it is necessary to recognize and preserve associations between data elements (relationships, nesting, and foreign keys) in the source schema.

**Target schema conformance:** The result of conversion must match the target schema.

**User intention:** Professional users only provide informal correspondences ('arrows') between schemas. Even under the requirement that all correspondence must be included in the interpretation of such mappings, there are usually several alternatives. Correctly guessing what you meant is only possible based on good heuristics.

Migration correction is checked with the help of queries so that all records from source database to destination are migrated successfully as used in Queiroz et al. (2022).

We find out whether all records of database from MySQL to Neo4j are migrated by executing query on both SQL and Cypher in MySQL and Neo4j respectively. The SQL query and cypher query for four different databases has been used to evaluate the correctness of schema and data set from MySQL to Neo4j.

1. To count entire set of records in a specific database of MySQL.
   SQL Query:
   ```sql
   SELECT SUM(TABLE_ROWS) FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = 'DATABASE_NAME';
   ```

2. To count entire set of nodes in a specific database of Neo4j.
   Cypher Query:
   ```cypher
   MATCH (n) RETURN count(*)
   ```

When we execute the data migration tool from MySQL to Neo4j, we have come into the following types of transformations:

(i) Table has been transformed into labelled nodes in Neo4j.
(ii) Source table object is mapped target node objects.
(iii) Column object is mapped to target node property.
(iv) Source table object is mapped to target labelled node object.
(v) Each record from all table(s) in database has been converted into respective nodes in Neo4j; that is, all tables, relationship types, columns from MySQL to Neo4j are mapped and hence migrated all data into Neo4j.

2) Data equivalence: The source and target databases will be accessed in order to evaluate the results. The information returned by the questions will be contrasted with approving the produced patterns. With the use of set of queries such as insertion, deletion, retrieval and updating for a MySQL; create and match for a Neo4j, we evaluate datasets as well as the primary features of the database.

Using a set of predefined queries on databases, we evaluated the proposed system: classicmodels, sampledb, northwind and hrdb. Results are physically surveyed to guarantee information identicalness.

Query roles are defined as follows:

1. **Query 1: INSERT record**
   In MySQL, the INSERT query has been used to add a row of data. The create operation in Neo4j is used to create or insert a new node.
SQL Query:
```
INSERT INTO table_name VALUES(val1, val2, ..., valN);
```

Cypher Query:
```
CREATE (n:node_name {attribute1: val1, attribute2: val2, attribute3: val3, ..., attributeN: valN}) RETURN n
```

2. **Query 2: DELETE**
   
   This inquiry is utilized to eliminate the single line. The delete command is what Neo4j uses to get rid of a specific node.

   SQL Query:
   ```
   ALTER TABLE TABLE_NAME DROP FOREIGN KEY TABLE_FK_KEY;
   DROP INDEX INDEX_NAME ON TABLE_NAME;
   DELETE FROM TABLE_NAME WHERE KEY_FIELD='VALUE';
   ```

   Cypher Query:
   ```
   MATCH (e: NODE_LABEL_NAME {KEY: VALUE}) DETACH DELETE e
   ```

3. **Query 3: RETRIEVE**
   
   The single row is retrieved by this query. Data was retrieved by Neo4j using the MATCH operation and the WHERE conditional.

   SQL Query:
   ```
   SELECT * FROM TABLE_NAME WHERE KEY_FIELD=VAL;
   ```

   Cypher Query:
   ```
   MATCH (c: NODE_LABEL_NAME) WHERE c.NODE_LABEL_NAME = VALUE RETURN c
   ```

4. **Query 4: MULTIPLE-RETRIEV**
   
   This question recovers numerous lines. Neo4j utilized the MATCH order to recover different records immediately.

   SQL Query:
   ```
   SELECT * FROM TABLE_NAME;
   ```

   Cypher Query:
   ```
   MATCH (n) WHERE n: NODE_LABEL_NAME RETURN n
   ```

5. **Query 5: RETRIEVE WITH SINGLE JOIN**
   
   This query pulls records from a relational database as a join query. Neo4j has a parent-younger relationship that serves as a join rather than a go-along. This query includes both the crucial key from the parent table column and the crucial key from the child table.

   SQL Query:
   ```
   SELECT * FROM TABLE1, TABLE2 WHERE TABLE1.PK=TABLE2.FK;
   ```

   Cypher Query:
   ```
   MATCH (n1: NODE_LABEL_NAME)-[r]->(n2: NODE_LABEL_NAME) RETURN n1, n2, r
   ```

6. **Query 6: RETRIEVE WITH MULTIPLE JOIN**
   
   This query reads rows from multiple child tables in addition to the parent table.

   SQL Query:
   ```
   SELECT DISTINCT c.TABLE_NAME, c.TABLE_NAME FROM TABLE_NAME AS c
   JOIN TABLE_NAME AS o ON (o.COLUMN_VALUE = c.COLUMN_VALUE)
   JOIN COLUMN_VALUE AS p ON (p.COLUMN_VALUE = c.COLUMN_VALUE);
   WHERE c.COLUMN_VALUE = VAL;
   ```

   Cypher Query:
   ```
   MATCH (c: NODE_LABEL_NAME),(o: NODE_LABEL_NAME),(p: NODE_LABEL_NAME)
   WHERE c.KEY=VALUE return DISTINCT c.FIELD_NAME, c.FIELD_NAME
   ```

7. **Query 7: UPDATE**
   
   The row is updated by this query. Neo4j uses a set operation to update the row based on the key value in the node.

   SQL Query:
   ```
   UPDATE TABLE_NAME SET COLUMN_NAME='VALUE';
   ```

   Cypher Query:
   ```
   MATCH (c: NODE_LABEL_NAME (FIELD_NAME: "VALUE")) SET c.FIELD_NAME = "VALUE";
   ```
3) System Performance Comparison: By evaluating the system’s performance, the quality of the conversion or translation is guaranteed. Question tumbled by time, the times that inquiry takes to execute, is utilized as an exhibition assessment metric by the vast majority of the DBMS. The exhibition and various parts of the usefulness of chart information stores are inspected and estimated through the created set of questions. By measuring the time spent on each query, the queries are used to assess the system’s performance. Windows 11 and an i3 CPU operating at 1.80 GHz were the subjects of our testing. sampledb, northwind, classicdb and hardb are the respective names of database. The execution times (in milliseconds) of each query for the source and destination databases are shown in Tables 6–10.

The measurement is the average amount of time that has passed since we ran each query three times. There is a plausible of reserving results from the data set, so we have changed the worth in the predicate prior to presenting the question each time. The assessment results are displayed in Figures 10–14.

4. EXPERIMENT AND ANALYSIS

This section presents the RDBMS model utilized for the relocation process. This section likewise portrays the information demonstrating and execution of both MySQL and Neo4j data sets. This section presents the results of the experiment. With qualitative and quantitative analysis, we observe the data migration process. As shown in Figure 2, the qualitative analysis is a proof of concept that involves migrating an existing relational database to a NoSQL graph database with the assistance of the SchemaCrawler Engine in accordance with the rule of Model Transformation from Relational to NoSQL graph database. The qualitative analysis aims to determine whether the migrated data have an effect on the applications’ ability to carry out a variety of SQL operations. In the quantitative examination, our goal is to comprehend and foresee conduct by the utilization of numerical estimations, factual displaying, and research. There are a number of uses for quantitative analysis, including efficiency, performance evaluation, and measurement. To play out this appraisal, we carried out an exploratory set-up with information and assessed the complete movement time for the various informational indexes. The size of the MySQL database, the operating system, the hardware configuration, and the related complexity of the tables all influence the data migration process.

For migrating database we have taken different dummy databases available online. The data set belongs to .sql format. We have imported those datasets in the MySQL server before migrating them from MySQL to Neo4j. Migration is performed in these two ways:

(i) In the first phase, we map and extract Metadata from MySQL and retrieve database from with SchemaCrawler engine.

(ii) In the second phase, SchemaCrawler extract data set from MySQL and transfer that data into Neo4j with Model to Model Migration process.

To perform the complete migration procedure using enhanced form of the Shabana et al. (Ramzan et al. 2019) Transformation algorithm, first we come through the process of meta modeling and extracting data set from MySQL to SchemaCrawler using existing plain Java code as well as JDBC drivers. To perform the migration of SQL data sets from MySQL to Neo4j has been implemented with plain Java code as mentioned in the APPENDIX A. SchemaCrawler engine has been presented as intermediary for the data migration. It executes the transformation rules with certain directed transformation rules so that the data set from MySQL to Neo4j is performed.

4.1 EXECUTION OF SPRINGBOOT PROJECT WITH INTelliJ

SchemaCrawler engine (tool) has been used to implement model driven approach through model driven development. Model driven development is in the implementation with plain Java code to develop the automatic migration tool (software) to migrate and convert relational database from MySQL to Neo4j. The approach has been come into full-fledged implementation using springboot framework. Implementation of the tool maps schema and then extracting metadata and data from MySQL and directly transfer the relational database into Neo4j. Program code snippet has been placed in the APPENDIX A and the details of execution has been explained and clearly depicted in the APPENDIX B.
There are four different databases of MySQL. These relational databases of MySQL have been converted and successfully migrated to Neo4j. Details of the databases to be migrated from MySQL and results of the transformation time of respective database has been shown in Table 5.

<table>
<thead>
<tr>
<th>DATABASE</th>
<th>NUMBER OF RECORDS</th>
<th>TRANSFORMATION TIME (IN SECONDS)</th>
<th>RELATION (YES/NO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classicmodels</td>
<td>3,864</td>
<td>11 sec.</td>
<td>Yes</td>
</tr>
<tr>
<td>Sampled</td>
<td>50,000</td>
<td>12 sec.</td>
<td>No</td>
</tr>
<tr>
<td>Northwind</td>
<td>932</td>
<td>10 sec.</td>
<td>Yes</td>
</tr>
<tr>
<td>Hrdb</td>
<td>136</td>
<td>8 sec.</td>
<td>Yes</td>
</tr>
<tr>
<td>Chinook</td>
<td>15,587</td>
<td>8 sec.</td>
<td></td>
</tr>
</tbody>
</table>

After the successful execution of the MySQL to Neo4j migrator tool with the help of SchemeCrawler engine, we come to the successful migration from the following important points:

- There are different four databases of MySQL databases, for example, classicmodels, sampledb, northwind and hrdb used for migrating purpose.
- Migration of each database in .sql format from MySQL to Neo4j has been performed successfully.
- Resultant data values after transformation and conversion from MySQL to Neo4j is consistent and accurate.
- Evaluation of consistency and accuracy of data transformation procedure is performed and explained in section 4.2.

4.2. EVALUATION OF MIGRATING PROCESS

Model to Model transformation is evaluated by using different key metrics as per the quantitative as well qualitative evaluation methods (Ramzan et al., 2019).

1) Schema information preservation

Migration correction is checked with the help of queries so that all records from source database to destination is migrated successfully as used in (Queiroz et al., 2022).

We find out whether all records of database from MySQL to Neo4j are migrated by executing query on both SQL and Cypher in MySQL and Neo4j respectively. The SQL query and cypher query for three different database has been performed as shown below:

A. **Database**: classicmodels

SQL Query:
```
SELECT SUM(TABLE_ROWS) FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = 'classicmodels';
```

Cypher Query:
```
MATCH (n) RETURN count(*)
```

B. **Database**: sampledb

SQL Query:
```
SELECT SUM(TABLE_ROWS) FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = 'sampledb';
```

Cypher Query:
```
MATCH (n) RETURN count(*)
```

C. **Database**: northwind

SQL Query:
```
SELECT SUM(TABLE_ROWS) FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = 'northwind';
```

Cypher Query:
```
MATCH (n) RETURN count(*)
```
D. Database: hrdb

SQL Query:
```
SELECT SUM(TABLE_ROWS) FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = 'hrdb';
```

Cypher Query:
```
MATCH (n) RETURN count(*)
```

2) Data equivalence

By running queries against the source and destination databases, the migration process’ outcome has been assessed. The produced schemas have been verified against the data that the queries returned. With the aid of a set of queries (insert, delete, retrieve, update for relational databases, and build and match for NoSQL graph databases), we have assessed datasets and the essential characteristics of databases.

Using a series of predefined queries on the databases classicmodels, sampledb, northwind, and hrdb, we tested the suggested solution. To make sure the data are equivalent, the results have been manually examined as shown in the Tables 6–10.

<table>
<thead>
<tr>
<th>NO.</th>
<th>QUERY</th>
<th>ELAPSED TIME MySQL</th>
<th>ELAPSED TIME NEO4j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Insert</td>
<td>32 ms</td>
<td>4.33 ms</td>
</tr>
<tr>
<td>2.</td>
<td>Delete</td>
<td>703 ms</td>
<td>74 ms</td>
</tr>
<tr>
<td>3.</td>
<td>Retrieve</td>
<td>20 ms</td>
<td>46.67 ms</td>
</tr>
<tr>
<td>4.</td>
<td>Multiple retrieve</td>
<td>3 ms</td>
<td>18.33 ms</td>
</tr>
<tr>
<td>5.</td>
<td>Retrieval with single join</td>
<td>47 ms</td>
<td>11.33 ms</td>
</tr>
<tr>
<td>6.</td>
<td>Retrieval with multiple join</td>
<td></td>
<td>1360 ms</td>
</tr>
<tr>
<td>7.</td>
<td>Update</td>
<td>47 ms</td>
<td>51.33 ms</td>
</tr>
</tbody>
</table>

Table 6 Query Elapsed Time in Milliseconds for Database classicmodels.

<table>
<thead>
<tr>
<th>NO.</th>
<th>QUERY</th>
<th>ELAPSED TIME MySQL</th>
<th>ELAPSED TIME NEO4j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Insert</td>
<td>0.0 ms</td>
<td>52 ms</td>
</tr>
<tr>
<td>2.</td>
<td>Delete</td>
<td>16 ms</td>
<td>6 ms</td>
</tr>
<tr>
<td>3.</td>
<td>Retrieve</td>
<td>16 ms</td>
<td>1209 ms</td>
</tr>
<tr>
<td>4.</td>
<td>Multiple retrieve</td>
<td>47 ms</td>
<td>1900 ms</td>
</tr>
<tr>
<td>5.</td>
<td>Retrieval with single join</td>
<td>No join</td>
<td>No join</td>
</tr>
<tr>
<td>6.</td>
<td>Retrieval with multiple join</td>
<td>No join</td>
<td>No join</td>
</tr>
<tr>
<td>7.</td>
<td>Update</td>
<td>0.0 ms</td>
<td>287 ms</td>
</tr>
</tbody>
</table>

Table 7 Query Elapsed Time in Milliseconds for Database Sampled.

<table>
<thead>
<tr>
<th>NO.</th>
<th>QUERY</th>
<th>ELAPSED TIME MySQL</th>
<th>ELAPSED TIME NEO4j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Insert</td>
<td>46 ms</td>
<td>216 ms</td>
</tr>
<tr>
<td>2.</td>
<td>Delete</td>
<td>31 ms</td>
<td>28 ms</td>
</tr>
<tr>
<td>3.</td>
<td>Retrieve</td>
<td>0.0 ms</td>
<td>215 ms</td>
</tr>
<tr>
<td>4.</td>
<td>Multiple retrieve</td>
<td>31 ms</td>
<td>156 ms</td>
</tr>
<tr>
<td>5.</td>
<td>Retrieval with single join</td>
<td>47 ms</td>
<td>199 ms</td>
</tr>
<tr>
<td>6.</td>
<td>Retrieval with multiple join</td>
<td>130 ms</td>
<td>212 ms</td>
</tr>
<tr>
<td>7.</td>
<td>Update</td>
<td>63 ms</td>
<td>330 ms</td>
</tr>
</tbody>
</table>

Table 8 Query Elapsed Time in Milliseconds for Database northwind.
From the above five results of query elapsed time in MySQL and Neo4j, we come to the following conclusion:

**a.** Query elapsed time for Insert, Delete and Retrieval with single join in Neo4j is better than in MySQL

**b.** Query elapsed time for single record retrieval, multiple record retrieval, update and retrieval with multiple join in MySQL is better than in comparison to Neo4j.

### 3) System Performance Comparison

Quality has been assessed by using evaluation of system performance. Query execution time has been implemented as a performance assessment metrics by many of the database systems. A set of queries has been developed to examine and measure the performance and various aspects of the functionality of graph data stores.

The query statements have been executed to assess the system functioning by assessing the execution time of each query. We have implemented tests on Core i3 1.80 GHz processor with 8 GB RAM and Windows 10. Each query for sourced and targeted databases is shown in for databases: classicmodels, sampledb, northwind, hrdb and chinook.

<table>
<thead>
<tr>
<th>NO.</th>
<th>QUERY</th>
<th>ELAPSED TIME MySQL</th>
<th>ELAPSED TIME NEO4j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Insert</td>
<td>16 ms</td>
<td>56 ms</td>
</tr>
<tr>
<td>2.</td>
<td>Delete</td>
<td>381 ms</td>
<td>60 ms</td>
</tr>
<tr>
<td>3.</td>
<td>Retrieve</td>
<td>0.0 ms</td>
<td>16 ms</td>
</tr>
<tr>
<td>4.</td>
<td>Multiple retrieve</td>
<td>0.0 ms</td>
<td>32 ms</td>
</tr>
<tr>
<td>5.</td>
<td>Retrieval with single join</td>
<td>0.0 ms</td>
<td>17 ms</td>
</tr>
<tr>
<td>6.</td>
<td>Retrieval with multiple join</td>
<td>0.0 ms</td>
<td>11 ms</td>
</tr>
<tr>
<td>7.</td>
<td>Update</td>
<td>78 ms</td>
<td>143 ms</td>
</tr>
</tbody>
</table>

### Table 9 Query Elapsed Time in Milliseconds for Database hrdb.

<table>
<thead>
<tr>
<th>NO.</th>
<th>QUERY</th>
<th>ELAPSED TIME MySQL</th>
<th>ELAPSED TIME NEO4j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Insert</td>
<td>63 ms</td>
<td>305Ms</td>
</tr>
<tr>
<td>2.</td>
<td>Delete</td>
<td>15 ms</td>
<td>61 Ms</td>
</tr>
<tr>
<td>3.</td>
<td>Retrieve</td>
<td>0.0 ms</td>
<td>160 ms</td>
</tr>
<tr>
<td>4.</td>
<td>Multiple retrieve</td>
<td>0.0 ms</td>
<td>124 ms</td>
</tr>
<tr>
<td>5.</td>
<td>Retrieval with single join</td>
<td>16 ms</td>
<td>183ms</td>
</tr>
<tr>
<td>6.</td>
<td>Retrieval with multiple join</td>
<td>16 ms</td>
<td>156036 ms</td>
</tr>
<tr>
<td>7.</td>
<td>Update</td>
<td>16 ms</td>
<td>294 ms</td>
</tr>
</tbody>
</table>

**Figure 10** Query Execution Time for classicmodels Database.
Figure 11 Query Execution Time for sampledb Database.

Figure 12 Query Execution Time for northwind Database.

Figure 13 Query Execution Time for hrdb Database.

Figure 14 Query Execution Time for chinook Database.
In the above diagrams, query elapsed time for insert, delete, retrieve, multiple retrieve, retrieval with single join, retrieval with multiple join and update have been plotted for every database: classicmodels, sampledb, northwind, hrdb and chinook. From the above presented plotted diagrams, the summary of the conclusion is:

a. Performance of cypher query is better for single record insertion, single record deletion and record retrieval with single join in comparison to MySQL SQL query.

b. On the other hand, performance of SQL query for single record retrieval, multiple record retrieval, update and retrieval with multiple in better in that of cypher query in neo4j.

4) Other validation techniques

Here we have summarized the validation techniques from different studies to migrate SQL to NoSQL so that the work we have performed is validated and verified.

a. The conference paper presented by (Feng & Huang 2022) proposed the method to migrate databases successfully from relational database to a graph database. The conversion accuracy has been measured through the analysis and comparison of data integrity, time cost, result validity. This has shown that the integrity and operability of the database before and after conversion are consistent.

b. The paper by (Putrama & Martinek 2022) has validated the conversion results by comparing the total number of records in the SQL database with the number of nodes and edges created in the graph database. The nodes properties are validated for consistency using a probabilistic data structure.

Finally we come to the conclusion that the migrated result from source database, that is, MySQL, to a graph database, that is, Neo4j, is validated. The above-shown metrics 1, 2 and 3 have proven the integrity and consistency of the result while migrating and converting the database from SQL to NoSQL graph database.

5. CONCLUSION AND FUTURE WORK

This research has worked on the enhancement of a transformation algorithm proposed by Shabana Ramzan et al. to convert and transfer databases from the relational database MySQL Server to the NoSQL graph database Neo4j. The migration performed is Model to Model data transformation. The migration model uses four basic components: input model, transformation description, and transformation engine and transformation rules to convert and migrate the database. The transformation module has automatically executed. The developed system completed the migration by mapping schema and extraction of data from MySQL Server database to Neo4j graph database. SchemaCrawler Tool has been used to execute transformation rules and perform automatic migration from MySQL to Neo4j. This migrating model has performed the migrating data, there is need of metamodeling from source database to target database. Future work of this study will be developing a transformation tool to migrate huge database from relational database to Neo4j using SchemaCrawler by solving the problem of heap memory issues while converting database into Neo4j.

The limitation of the research work is that if the data migration process is interrupted by any condition, it cannot be resumed from that point. The embed document limit (nested documents) is applied at once for all table which enforces all the tables to have same nested documents. The data modeling for nested documents is not available as a result for embedding documents it embeds all the data from particular table. The number of query limit and threads is handled automatically by system. The data migration is done within a system only, that is, it cannot migrate the data from one system to another system. This system that has been developed with an enhanced form of transformation algorithm proposed by Shabana Ramzan et al. smoothly works for small set of SQL data sets. The major limits of this research study are pointed below:

a. Migration can be performed for small datasets of SQL database.

b. Neo4j used in this study is Community version i.e. Neo4j 3.1.3

SchemaCrawler engine is used to execute transformation rule and map schema and extract data.
ADDITIONAL FILES
The additional files for this article can be found as follows:

- **Appendix A.** Code Snippets. DOI: https://doi.org/10.5334/dsj-2024-035.s1
- **Appendix B.** Execution of Code to Migrate Data from MySQL to SchemaCrawler. DOI: https://doi.org/10.5334/dsj-2024-035.s2

COMPETING INTERESTS
The authors have no competing interests to declare.

AUTHOR AFFILIATIONS
Hira Lal Bhandari orcid.org/0000-0002-2437-3157
Faculty of Science, Health and Technology, Nepal Open University, Lalitpur, Nepal
Roshan Chitrakar orcid.org/0000-0002-8967-1454
Nepal College of Information Technology, Lalitpur, Nepal

REFERENCES
Akehurst, D H et al. 2006 SiTra: Simple transformations in Java. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 4199 LNCS, pp. 351–364. DOI: https://doi.org/10.1007/11880240_25
Christina, A J and Saranya, S 2018 Extending the Existing Data Migration Algorithm for MySQL To MongoDB and Analyzing Its Performance, 6(8), 440–446.


Pokorný, J et al. 2017 Graph databases: Their power and limitations, pp. 58–69. DOI: https://doi.org/10.1007/978-3-319-24369-6_5

Putramo, I M and Martinek, P 2022 An automated graph construction approach from relational databases to Neo4j. IEEE Joint 22nd International Symposium on Computational Intelligence and Informatics and 8th International Conference on Recent Achievements in Mechatronics, Automation, Computer Science and Robotics, CINTI-MACRo 2022 – Proceedings, pp. 131–136. DOI: https://doi.org/10.1109/CINTI-MACRo57952.2022.10029438


Ramzan, S et al. 2019 Intelligent data engineering for migration to NoSQL based secure environments. IEEE Access, 7, pp. 69042–69057. DOI: https://doi.org/10.1109/ACCESS.2019.2916912


SchemaCrawler Alternatives and Similar Software|AlternativeTo n.d. Available at https://alternativeto.net/software/schemacrawler/?fbclid=IwAR2axgn-v7monmm78l1ee9s5t4hraxtpap6dd0vmc3zaphnfa7sgqukne [Last accessed 10 February 2024].
