Article

An Intelligent Approach for Handling Complexity by Migrating from Conventional Databases to Big Data

Shabana Ramzan [1], Imran Sarwar Bajwa [1,*] and Rafaqut Kazmi [2]

[1] Department of Computer Science & IT, Islamia University of Bahawalpur, Bahawalpur 63100, Pakistan; shabana@gscwu.edu.pk
[2] School of Computing, University of Technology Malaysia, Johor 81310, Malaysia; rafaqutkazmi@gmail.com
* Correspondence: imran.sarwar@iub.edu.pk

Received: 26 October 2018; Accepted: 14 November 2018; Published: 3 December 2018

Abstract: Handling complexity in the data of information systems has emerged into a serious challenge in recent times. The typical relational databases have limited ability to manage the discrete and heterogenous nature of modern data. Additionally, the complexity of data in relational databases is so high that the efficient retrieval of information has become a bottleneck in traditional information systems. On the side, Big Data has emerged into a decent solution for heterogenous and complex data (structured, semi-structured and unstructured data) by providing architectural support to handle complex data and by providing a tool-kit for efficient analysis of complex data. For the organizations that are sticking to relational databases and are facing the challenge of handling complex data, they need to migrate their data to a Big Data solution to get benefits such as horizontal scalability, real-time interaction, handling high volume data, etc. However, such migration from relational databases to Big Data is in itself a challenge due to the complexity of data. In this paper, we introduce a novel approach that handles complexity of automatic transformation of existing relational database (MySQL) into a Big data solution (Oracle NoSQL). The used approach supports a bi-fold transformation (schema-to-schema and data-to-data) to minimize the complexity of data and to allow improved analysis of data. A software prototype for this transformation is also developed as a proof of concept. The results of the experiments show the correctness of our transformations that outperform the other similar approaches.

Keywords: big data; complexity; NoSQL databases; Oracle NoSQL; data migration

1. Introduction

The modern information systems have to deal with high-dimension data in terms of gigantic size, and the heterogenous and complex nature of the data. Similarly, the cloud applications and social media applications also have to store, manage and process a massive amount of data. However, the Relational Databases (RDBs) have fixed schema and allow storage and handling of only structured data in the form of tuples or relations [1]. Additionally, the RDBs only provide vertical scalability (vertical scalability allows only vertical growth of a data-structure by adding only new records at run-time.) at higher hardware cost but no horizontal scalability (horizontal scalability allows horizontal growth of a data-structure by also allowing the addition of fields at run-time.) is provided by the RDBs. Since horizontal scalability is needed by today's software applications to handle high-speed heterogenous data; currently, the relational databases have to face various challenges at the application development level and operational level. At the application development level, the system developer needs high coding velocity to handle large number of users; however, such capability is not available in relational databases. Additionally, modern complex and heterogenous data needs horizontal scaling but that feature is also not provided by the relational databases and consequently, they fail to cope with the needs of modern data-intensive software applications.

Once of the key challenges in recent times has been to handle high-speed data, as there is a rapid increase in digital information, exponentially growing (see Figure 1) to Petabytes (PB) PB = 1000 TB) from Terabytes (TB) 1TB = 1000 GB, and even to Exabytes (EB) 1EB = 1000 TB as shown in Figure 1. John Gantz and David Reinse also predicted this phenomenon [2]. Typical relational database systems have shown their limits for such exponential growth of data. The shortcomings of typical relational databases are addressed by Big data solutions such as NoSQL databases [3–5]. Here, NoSQL stands for "Not Only SQL". Such databases are currently the main focus of research due to the fast and persistent growth of data. The NoSQL was introduced in 1998 by Carlo, and the name given to his relational database solution that was due to not using Structured Query Language (SQL) [6]. The idea of NoSQL was redefined in 2009 and became the competitor of RDBs. Now they have become the backbone of large-sized enterprises such as Google, Twitter, Facebook, Amazon, etc. due to its peculiar features such as high availability (when a data is automatically distributed evenly across a cluster with no single master.), efficient performance, horizontal scalability, and the support of a variety of data models and queries. Moreover, the rapid growth of cloud computing has highlighted the problems that are endured in handling large volumes of data. However, NoSQL databases can handle "Big Data" problems efficiently rather than RDBs. These databases are becoming popular because they are providing a high level of scalability. Additionally, they are very efficient in handling the unstructured data to facilitate universal data communication [7] in modern information systems. Relational databases follow the ACID (Atomicity, Consistency, Isolation, Durability) and BASE (Basically available, Soft-state, Eventual-consistency) properties. Whereas, NoSQL databases exist in a spectrum between ACID and BASE alliance.

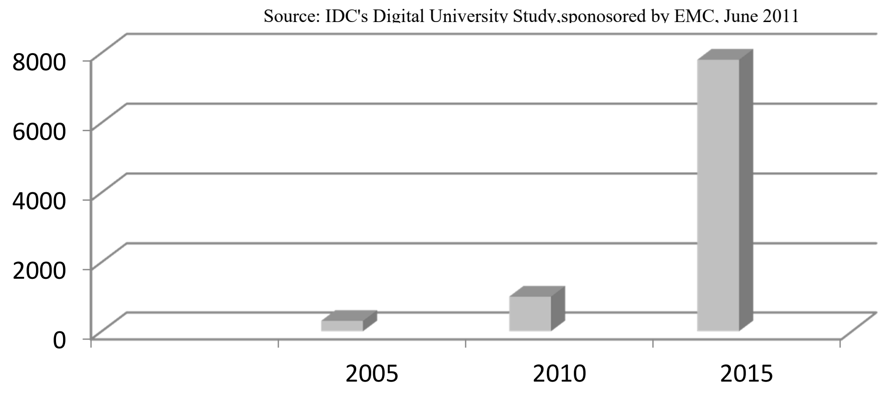


Figure 1. Exponential growth of digital information, going to Exabyte.

The NoSQL databases have typically four different models: (1) key-value store, (2) column store, (3) graph store, and (4) document store [8]. Each NoSQL database model has its own distinct schema of storing data [9]. The most simple and flexible model is a key value store that is used in our study and an overview of key-value stores is given below:

1.1. Key-Value Stores

A Key-value store is a database that stores data in the form of associative arrays known as a hash or a dictionary. Each dictionary has a collection of records that have different fields of data. They store data as a key-value (record) pair as shown in Figure 2. Each value is stored and retrieved through a unique key. A value is a data of an arbitrary type, size and structure. Here, value can be anything such as a number, text, image, programming code (such as PHP), markup code (such as HTML), etc. They do not have any query language, only use get, put and delete operations [10]. A key can be simple (filename, hash or URL) or a composite key (such as in Oracle NoSQL) [11].

A set of operations are used to interact with key-value stores such as Get operation is used to retrieve a value that is stored against a unique key and put operation is used to insert the key-value pair. However, manipulation of multiple values in a single operation is not allowed by these single-key operations. These operations facilitate the users that do not have proper knowledge of query language to easily retrieve data. A key-value store handles the process data retrieval manually at the application level. Here, lookup structures are used that are based on keys such as Log-Structured Merge-trees (LSM-trees) and Distributed Hash Tables (DHTs) [12], and are highly suitable for applications that can access data through a single key, such as web session information, user profile/configuration and online shopping cart. Key-value stores are the only databases that provide efficient data retrieval and storage mechanisms to cloud-based applications [13]. Key-value stores provide features like easy partitioning and high scalability. Figure 2 shows a storage model of a typical key-value store.

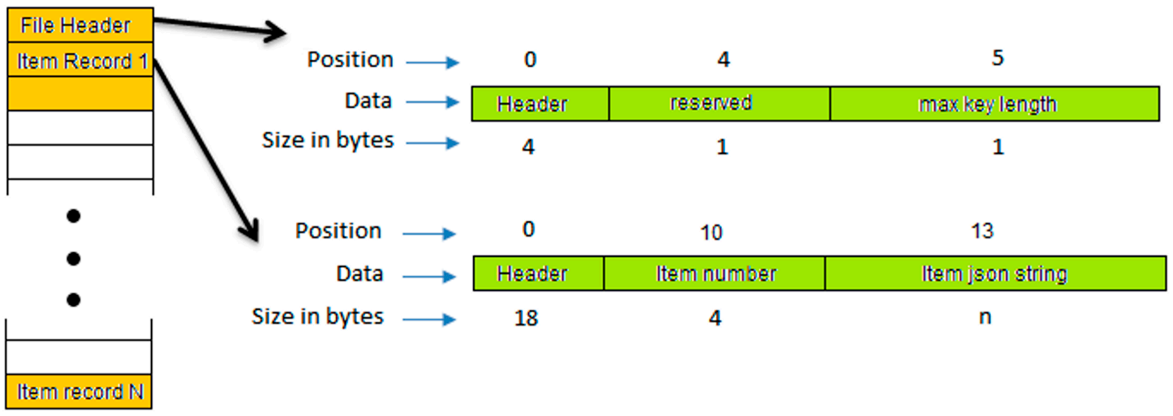


Figure 2. Key-Value Store data model.

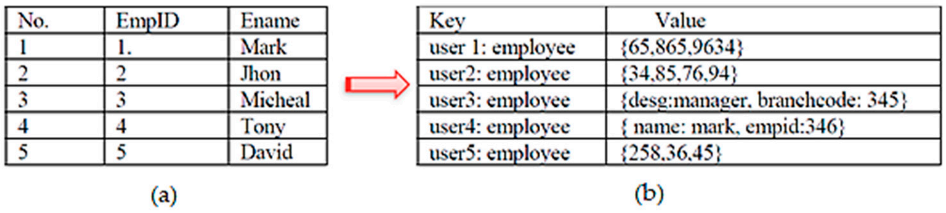
On the base of storage models, key-value stores can be divided into three types of stores including permanent, temporary and hybrid. In permanent stores, all the data is stored on the hard disk but I/O operations are slow. The temporary key-value store ensures fast data access as all data is stored in memory; however, if the system is down, the data will be lost. Whereas, a hybrid store is a combination of the positive features of both permanent and temporary stores as it supports data storage in memory and when specified conditions are met, the date is written to the hard disk. The most popular key-value stores are hyperdex, Redis, Riak, Oracle NoSQL, BerkelyDB, Yahoo Pnuts and Project Voldemort. The following section provides an overview of Oracle NoSQL.

1.2. Oracle NoSQL

Oracle NoSQL is a distributed type of key-value store. It provides important features like horizontal scalability, monitoring, transactional semantics for improved data manipulation, and simple administration of data. The Oracle NoSQL has a very simple data model. Each row is a key-value pair; value is associated with a unique key. Value is of arbitrary length. It has tables, rows and fields which are equivalent to tables, rows and columns of relational databases but has a different concept. The following are the key features of Oracle NoSQL stores:

- Oracle NoSQL table is schema free but relational databases' tables have predefined schema.
- Each column has a separate schema but in relational databases each table has a schema.
- Each row in Oracle NoSQL database can have unrelated fields but in relational databases each row is a collection of related items.

Figure 3 shows the relational and Oracle NoSQL key-value store databases. Oracle Berkeley DB Java Edition high-availability storage engine is the basis of Oracle NoSQL. It provides sharding, replication, transparent load balancing, high availability and fault tolerance. It is a free schema and supports various programming languages such as C++, C, C#, Ruby, Scala, Java, Javascript(Node.js), and Python. Oracle NoSQL supports simple data types (java string float, integer, long, boolean, double) as well as complex data types (array, enum, fixed binary, map, records).



No.	EmpID	Ename
1	1.	Mark
2	2	Jhon
3	3	Micheal
4	4	Tony
5	5	David

(a)

Key	Value
user 1: employee	{65,865,9634}
user2: employee	{34,85,76,94}
user3: employee	{desg:manager, branchcode: 345}
user4: employee	{ name: mark, empid:346}
user5: employee	{258,36,45}

(b)

Figure 3. Relational database (**a**) and Oracle NoSQL Key-value store database (**b**).

Considering the challenges of big data, the modern organizations are rapidly shifting to NoSQL databases from conventional RDBs. Relational to relational database like MySQL to Oracle conversion is possible because they are based on mathematical theory [14]. On the other hand, NoSQL databases are non-relational and their scheme design is completely different. So RDB-trained staff has difficulty in converting existing RDB system to NoSQL databases [15,16]. However, the lack of methodological and tool support for automated migration from RDB to NoSQL has been a real challenge in recent times. In this paper, a methodology is presented to transform the existing relational database into a NoSQL database. It automatically transforms both the data and schema. The proposed approach transforms the MySQL database into an Oracle NoSQL database by handling the complexity of data.

The rest of the paper is organized as follows. Section 2 introduces related work in the fields of RDBs, NoSQL and migration between these two generations of databases. Section 3 describes the used approach and Section 4 discusses the implementation details of our approach. The results of experiments and discussion is given in Section 5 along the evaluation details. Finally, Section 6 concludes the results with possible future work.

2. Related Work

A data transformation from a relational database to an NOSQL database depends on different factors such as mapping styles, query structures, storage structures, etc. Additionally, querying data from a relational database and a NoSQL database at the same time is difficult but now it is applicable by using the data adapter technique [17]. Here, a method DB converter is described that transforms relational data into NoSQL data for querying results. However, this conversion is temporary, only for query execution [17], since NoSQL databases are efficient in data storage and provide high levels of scalability and availability. There are several different studies on NoSQL databases [4], such as BigTable [13], Cassandra [18], HBase [15], MongoDB [19] and for big data [20]. It is studied that the schema conversion from relational databases to NoSQL is difficult because relational databases use JOINs but the NoSQL databases do not support it. In NoSQL databases, nesting tables are used as an alternative to JOINs. This method is designed to improve the performance of cross table query. In nesting table technique, the parent-child layer is designed with references as relationships between the tables. Here, the referred table is defined as a child and the other one is defined as a parent [21]. The cross table query is important in SQL databases, but in NoSQL, a question arises on how to use JOIN type or alternative queries to retrieve data from NoSQL databases. Column-oriented databases provide a solution for these types of queries because column-oriented databases have a design principle of DDI (Denormalization, Duplication and Intelligent keys). This method works as: initially, denormalization of the database and its transformation into a big table; then identification of unique keys in a big table; and finally, the selection of the most suitable key as primary key. In this method, MySQL database is transformed into a column-oriented database [22]. Most of the web-based applications and Content Management System (CMS) solutions are using relational databases for data management, but users of internet and clouds are growing rapidly, so it is difficult for relational databases to handle the huge data traffic. The designed approach transforms the real CMS SQL database to a NoSQL database [23]. This approach has two steps, first to denormalize the SQL database and then to choose a unique identifier key as a primary key for a big table. In this approach, MySQL database is migrated to a column-oriented Hbase database.

Another method is designed to transform data from a relational database (MySQL) to NoSQL (MongoDB). Migration from relational to NoSQL has a few steps; initially, MySQL database connection is created, after connectivity, the details of the database are accessed through prototype software. In next step, mapping is performed between the relational database MySQL to NoSQL MongoDB [24,25].

For transformation from RDBs to NoSQL, another application is developed which deals with the transformation of relational database schema to NoSQL schema. This application is able to handle both the DDL and DML commands of relational schema and transform these commands into equaling commands of NoSQL [26]. To access the NoSQL database, a subset of SQL commands is used. CQL is the query language for Cassandra, where CQL and SQL are quite similar. Cassandra and MongoDB are integrated because MongoDB is capable of performing complex queries. Therefore, authors designed a system for translation of SQL commands to NoSQL. This system is implemented by middleware in C# [27]. Table 1 shows the comparison of existing approaches. The majority of the researchers tend to use HBase and MonogoDB as a target database but no one used Oracle NoSQL. The facts tabulated in the following table clearly show the research gap that currently no approach or tool supports automated transformation of MySQL to Oracle NoSQL for both data and schema transformation.

Table 1. Comparison of transformation approaches.

Source Database	Target Database	Schema Conversion	Data Conversion	Conversion Time	Data Set	Technique	Study Reference
MySQL	MongoDB	Yes	No	No	72 Tables	Transform algorithm	Zhao et al.
MySQL	HBase	Yes	No	No	Hush database 1 thousand transactio-ns)	Automatic transformation Mechanism based on NoSQL DDI Design Principle.	Lee et al.
MySQL	MongoDB	Yes	Yes	No	Two datasets 1. Twitter App. 2.W3Scho-ols App	framework (1) migration module (2) mapping module	Rocha et al.
MySQL	MongoDB	No details about schema conversion	Yes	No	—	Migration Methodology (1) Extracting logical structure (2) Mapping between databases.	Hanine et al.
SQL	any key-oriented NoSQL DB	Yes	No	No	European Air quality database	Transformation layer	Schreiner et al.
SQL	HBase	Yes	No	No	15 GB Dell DVD Store relational database	Heuristic based approach	Serrano et al.
RDB	HBase	Yes	No	No	RDB schema with 7 tables.	Extracting conversion rules and applied conversion rules.	Ouanouki et al.
RDB	HBase	Yes	No	No	—	heuristic-based approach	Li et al.
RDB	document-oriented NoSQL,	Yes	Yes	No	Different databases of different sizes.	Column-level Denormalization and Atomic Aggregates	Yoo et al.
MySQL	**Oracle NoSQL**	**Yes**	**Yes**	**Yes**	**Five different databases**	**Automatic Transformation**	**Proposed Scheme**

There is another methodology for the conversion of a relational database to HBase in four steps [28]. First, create a single merge table in HBase and convert all one-to-one and one-to-many relationships into that table. Second, merge neighboring tables through a recursive method. Third, a row key design, and fourth, create access patterns views. The set of rules for schema conversion between an existing relational database to Hbase are defined [29]. First, experimentally justify the need of conversion rules by observing the conversions without conversion rules. This first experiment is used as a baseline of the second experiment. Second, the experiment is performed convert the existing relational database application to Hbase using a first list of conversion rules. This conversion proves that the conversion rules reduce the difficulty of the whole conversion process. Another approach is presented for RDBs to NoSQL migration that has two phases [30], the first phase transforms relational database schema to HBase schema and also provides guidelines to develop HBase application. In the second phase, schema mappings are used to create a set of programs to automatically transform the data of the source database to the target database. Similarly, this proposed a solution for migrating RDBMS schema to document oriented NoSQL database schema [16]. This method provides atomicity using atomic aggregate and avoids join operations. It uses the column-level denormalization in order to minimize the disadvantages of table-level denormalization.

Data extraction from Big data has been one of the major research challenges [31,32] in recent times. Since, Big data has discrete and heterogenous types of data, this challenge becomes more difficult. However, various contributions [32–34] are made to address this challenge. Suciu [33] discussed the extraction of knowledge from Big data and [34] discussed how conceptual modeling can help in addressing this challenge. The NoSQLayer tool presented is proposed for migrating from a relational database to a NoSQL database; this approach has two modules. The data migration module migrates the SQL database to the NoSQL database. Here, the metadata of the MySQL database is accessed during Java Metadata API. The data mapping module transforms data from a relational database to a NoSQL database seamlessly [35].

To the best of our knowledge, there is no approach or tool available to handle the complexity of automatic conversion of a RDB (such as MySQL) to Big data solutions (such as Oracle NoSQL database) and the major contribution of this paper is to present a novel approach that is intelligent enough to handle the complexity of data and automatically transform MySQL database to Oracle NoSQL for both data and schema conversion.

3. Used Approach for Handling Complexity of RDB to Big Data Conversion

The used approach is based on a rule-based system that has two modules: The first module handles the conversion of a relational database (such as MySQL) schema to a NoSQL database schema which is very flexible in nature (Schema Conversion). Whereas, the second module handles the conversion of the data from the relational database (such as MySQL) to NoSQL database (such as Oracle NoSQL). The working of the first module in the proposed approach is shown in Figure 4, that performs the schema transformation.

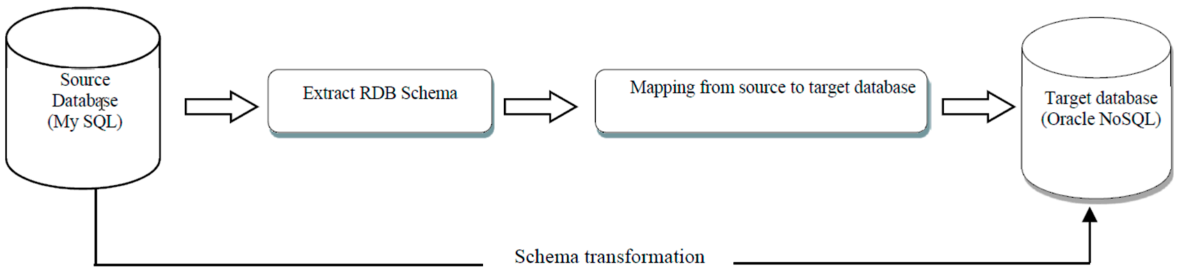

Source Database (My SQL)
Extract RDB Schema
Mapping from source to target database
Target database (Oracle NoSQL)
Schema transformation

Figure 4. Schema transformation module.

The following text explains the components used in both modules (schema transformation and data transformation) and their working.

3.1. Schema Transformation

This module handles the transformation of MySQL schema to Oracle NoSQL schema. For this purpose, a relational database in MySQL is taken as an input and parses in Java to extract metadata of MySQL databases such as table-names, their attribute-names, attribute-data-types, relationship-names, indexes from the database, etc. Here, the JDBC driver and the Java metadata base class library is used to access the schema of tables from MySQL database. For relationship metadata extraction, the primary and foreign key constraints of each table were used as the relationship information can be helpful for schema conversion. Here, Java metadata class library provides different methods to extract schema information in different aspects, e.g., if we want to get information about a primary key then we use metadata's primary key function. The methods used for this approach are listed in Table 2. When all required metadata of the tables' schemas are extracted, the mapping given in Table 3 is used to transform the MySQL metadata to Oracle NoSQL key-value store.

Table 2. List of methods used in schema transformation.

Methods	Description
getTables()	This method returns all the tables of the database. The list returned by this method is traversed to get information of each table.
getColumns()	The names of all attributes that are defined by the parameters and their characteristics are retrieved through this method.
getMetaData()	This method returns all other information about relational databases, such as data type and constraints.
getIndexInfo()	All indexes that are created on the relational database return through this method.
getImportedKeys()	This method retrieves a description of the primary key columns that are referenced by a table's foreign key columns.
Getprimary keys()	This method retrieves a description of the primary key columns of the given table.

This mapping is implemented in Java to accomplish the schema level transformation of MySQL to Oracle NoSQL database. Once the schema transformation is accomplished, the data level transformation is carried out; this is described in the following section.

Table 3. Migration mapping RDBs to Oracle NoSQL.

MySQL	Oracle NoSQL
Table	Record or Table
Column Name	Field Name
Column Data Type	Field Type
Column	Field
Users	Users
Permisssions	Priviliges
Index	Index
JOIN	Parent-Child link
Foreign Key	Reference (parent-child link)

In our approach, we have created an online test preparation of a student database in MySQL. A subset of this MySQL database is shown in Figure 5 that is used to explain the schema conversion methodology of our approach.

Relational database has an important feature of JOIN. But Oracle NoSQL database uses parent-child relationship instead of JOIN. MySQL database Student table (parent table) is linked with Marks table (child table) and Marks table (parent table) is linked with Course table (child table) and Course table (parent table) is linked with lecturer table (child table) and lecturer table (parent table) is linked with Classes (child table). In Figure 6, we see that Student is a parent table, Marks is a child table as well as the Course is a sub-child table and Course table is a parent table and Lecturer is a child table and classes are sub-child table.

In Oracle NoSQL key-value store, these tables are stores in the form of Avro schema. Avro schema supports both APIs of Oracle NoSQL, and that is why the entire key-value store is based on this schema. Figure 7 shows the Avro Schema of tables stored in Oracle NoSQL store and Figure 8 shows Avro Schema of Marks table. Table API of the Oracle NoSQL key-value store is just a front-end layer to provide a user friendly environment.

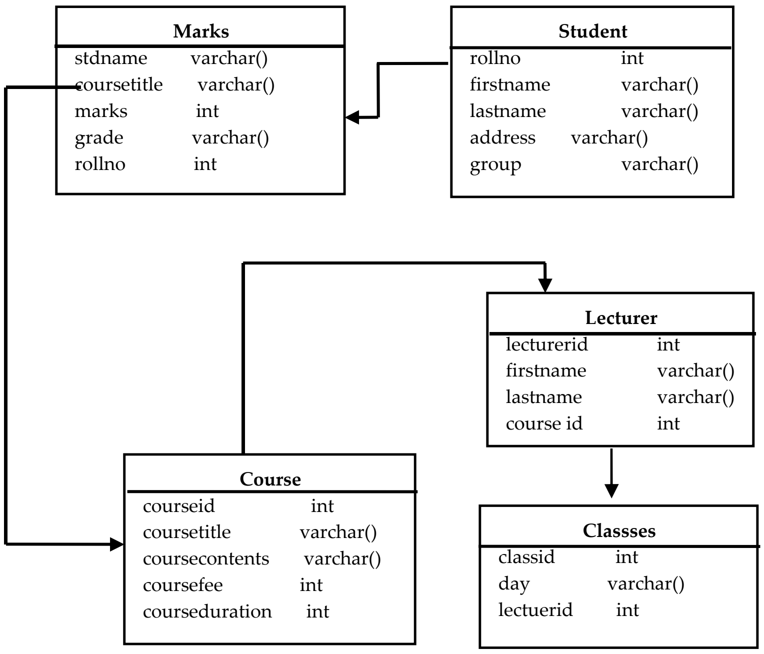


Figure 5. Database model used as test database.

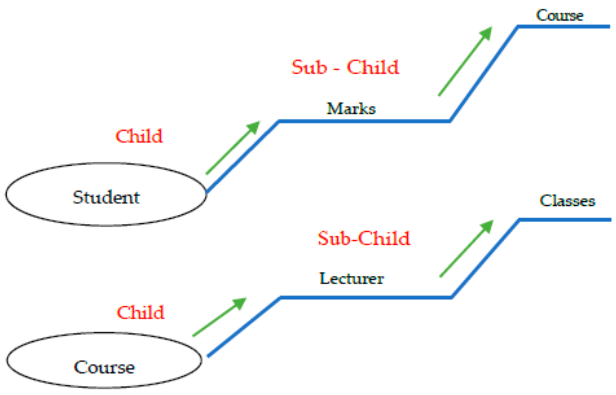
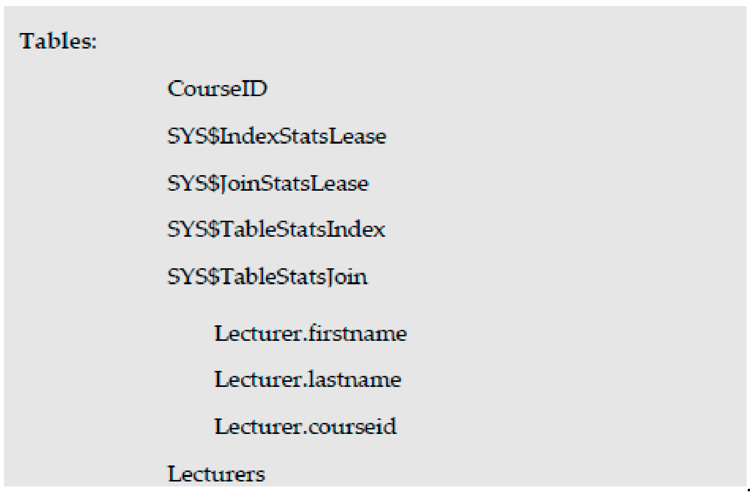


Figure 6. Relationships of Oracle NoSQL database tables.



Tables:

CourseID

SYS$IndexStatsLease

SYS$JoinStatsLease

SYS$TableStatsIndex

SYS$TableStatsJoin

Lecturer.firstname

Lecturer.lastname

Lecturer.courseid

Lecturers

Figure 7. Detail of tables stored in Oracle NoSQL store.

```
{
 "Type"   : "Table",
"Name" :  "Marks",
"Owner": "null",
"shardkey": ["Coursetitle"],
"primarykey": ["Coursetitle"],
"fields": [{
          "name": "StdName",
          "type": "Integer",
          "nullable": "true",
          "default": null
           },{
          "name": "Coursetitle",
          "type": "String",
          "nullable": "true",
          "default": null

          "name": "Mark",
          "type": "Integer",
         "nullable": "true",
           "default": null
           },{
          "name": "Grade",
          "type": "STRING",
           "nullable": "true",
          "default": null
          }, {

          "name": "RollNo",
          "type": "INTEGER",
          "nullable": "true",
          "default": null
         }    ]
}
```

Figure 8. Avro Schema of Marks Table.

3.2. Data Transformation

For data transformation, a table in source MySQL database is selected from which the data is to be extracted and transformed into JSON format for final storage in an oracle NoSQL database. For this purpose, the ETL (Extract, Transform and Load) methodology [31] is used. The data transformation module is shown in Figure 9.

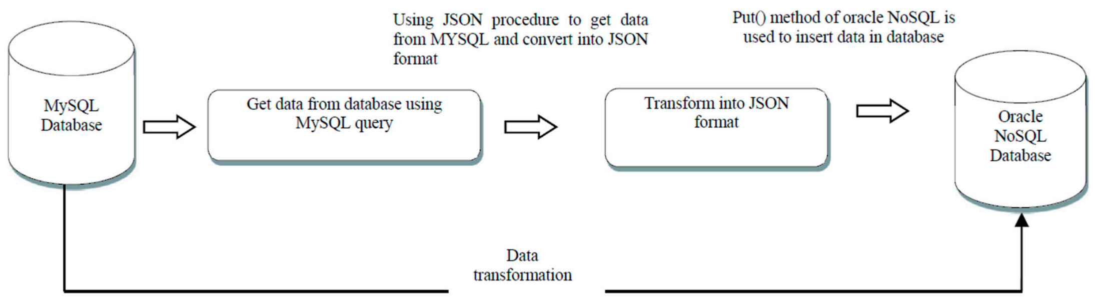

Using JSON procedure to get data from MYSQL and convert into JSON format
Put() method of oracle NoSQL is used to insert data in database
MySQL Database
Get data from database using MySQL query
Transform into JSON format
Oracle NoSQL Database
Data transformation

Figure 9. Data transformation module.

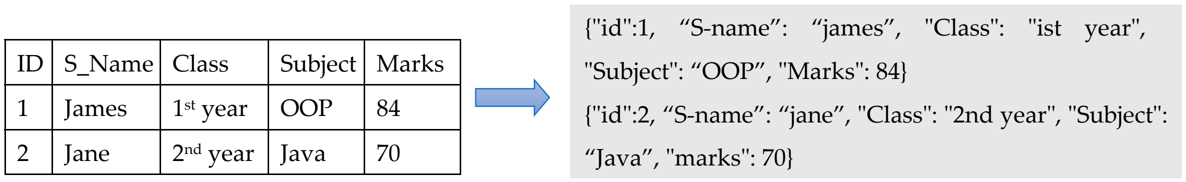
JSON format is a light weight, data interchange format, and is easy to understand. JSON is used in Oracle NoSQL database as a data type and also as a schema; the schema created through JSON is called Avro Schema. Before inserting data into a key value store, it is necessary to create an Avro schema for the record which defines the structure of the data.

In the first step, the required table is selected, from which data will be extracted. The next step is to convert data of the table row by row from MySQL database and store it in a text file; when all the data of the table is transformed, then this text file is called by another module that stores it into the Oracle NoSQL database. For data transformation, we need to know about some data types used in MySQL database to compare it with data types of the Oracle NoSQL database as shown in Table 4.

Table 4. Data type comparison.

MySQL	Oracle NoSQL
int, bigint	Integer
Long	Long
Array	Array
Bolean	Bolean
Float	Float
Double	Double
String	String, Java String
BLOB	Binary

In the data transformation process, every column data type of the MySQL database table is compared with the Oracle NoSQL data types. Such a comparison helps in finding the exact match of MySQL data type. Figure 10 shows the mapping of category table from RDBs to Oracle NoSQL.



ID	S_Name	Class	Subject	Marks
1	James	1st year	OOP	84
2	Jane	2nd year	Java	70

{"id":1, “S-name”: “james”, "Class": "ist year", "Subject": “OOP”, "Marks": 84}
{"id":2, “S-name”: “jane”, "Class": "2nd year", "Subject": “Java”, "marks": 70}

Figure 10. Mapping of relational table to Oracle NoSQL table.

The designed software prototype transforms all data of the required table into JSON format and temporarily stores this data into a text file. After converting data into JSON format, the next step is to insert data into Oracle NoSQL destination schema table. Finally, put() function is called to read JSON data from the file and store it into an Oracle NoSQL schema table.

4. Implementation Details

The approach discussed in the previous section is implemented in Java as Eclipse plugin. The implemented system starts working with the connectivity with a source relational database. After connectivity, two options will be displayed for conversion/transformation process. One is Schema Conversion and the other is Data Conversion. Which option is used depends on the user or administrator. If the user selects schema conversion, then the first step is to select the required database to transform into the Oracle NoSQL database. The next step is hidden from the user, which actually maps the databases and makes a conversion. In the last step, the transformed database schema will be displayed on the form. After creating schema from the relational to NoSQL database, a user can also transform data from MySQL to NoSQL. The second option is Data Conversion, if the user goes for this option, firstly, he will select the database and the required table from the database afterwards. The next step is to transform the data from the relational database to the Oracle NoSQL database. Consequently, a stored procedure is designed which transforms relational database data into Avro schema base JSON data. The entire working of the proposed system is shown in Figure 11.

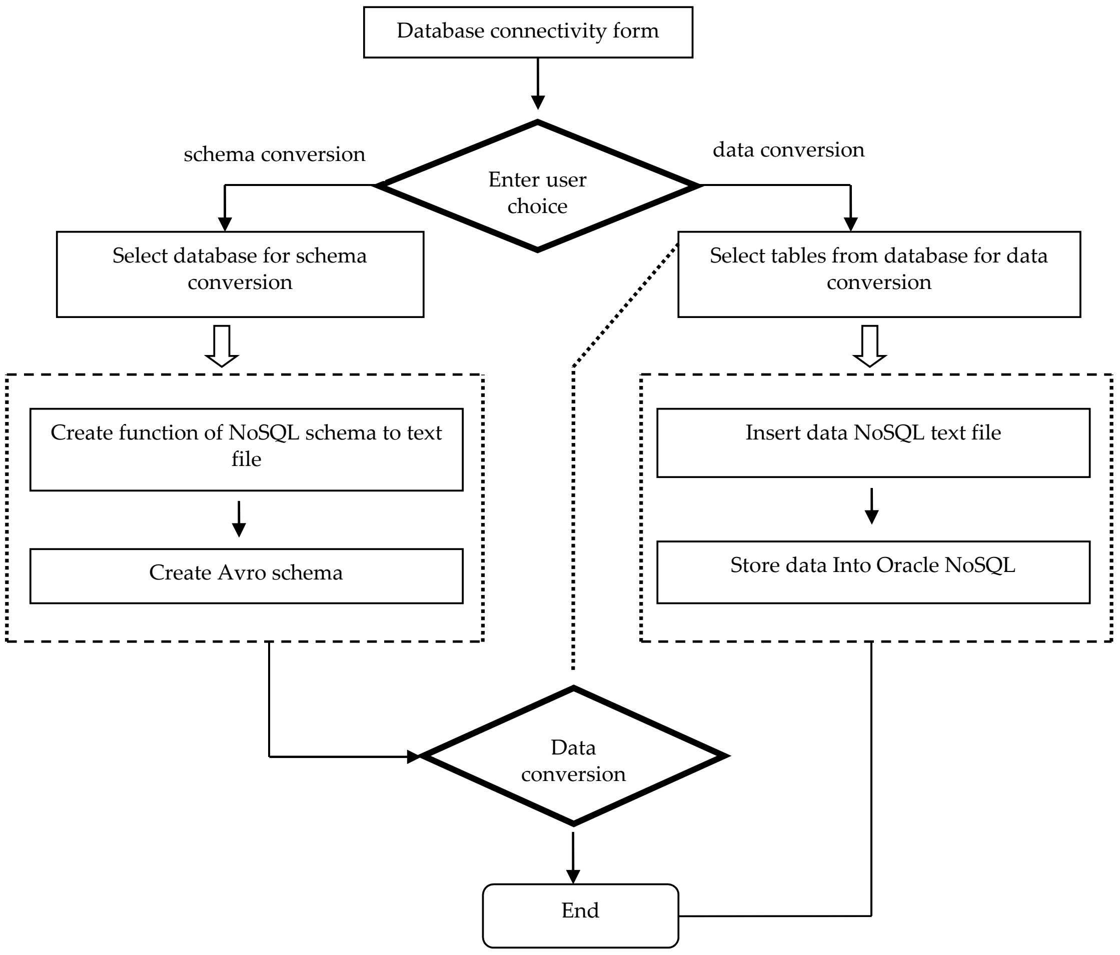

Database connectivity form
schema conversion
Enter user choice
data conversion
Select database for schema conversion
Select tables from database for data conversion
Create function of NoSQL schema to text file
Create Avro schema
Insert data NoSQL text file
Store data Into Oracle NoSQL
Data conversion
End

Figure 11. Framework of the proposed approach.

4.1. Module A. Creating Schema of Mysql Table and Store It in the File

The function is used, which transforms the MySQL database tables into Avro Schema as shown in Figure 7. This function has the following steps.

1. Get metadata of MySQL database tables:

In this step, after database connectivity, MySQL tables are selected one by one and a function of Java metadatabase class library is called, which selects the table and its attributes.

2. Create a file:

For this step, a function of file Java class library is used, and to write data to this file PrintWriterfunction or PrintWriter Java class library are used.

3. Get keys from metadata:

In this part, the primary and foreign keys of are table are used for primary and foreign key mapping with NoSQL Avro Schema.

4. Map the data types of both databases and create fields in avro style:

In this step, the mapping process is defined, col.next() built-in function is used to select the column names of the tables one by one and compare the data type of MySQL table with Oracle NoSQL data types.

5. Create JOIN like parent-child relationship:

The fk.next() is a result set which hold the foreign key details and its corresponding tables. The table name of the foreign key is selected as a parent table name for the under-process table.

According to Oracle NoSQL, a parent-child relationship is like:

Parenttable.childtable (attributes with data types and primary keys of both tables).

4.2. Module B. Schema from File to NoSQL

In this module, the file is called, which has temporarily stores the schema of tables; after that, an object of Oracle NoSQL key-value store is created to access the NoSQL database. In the next step, an object of Table API is created for new table creation. Now the function runs while looped and gets data from files and sends it to KVstoreexecuteSync() function for table creation in the Oracle NoSQL store.

4.3. Module C. Transform Mysql Data into Oracle NoSQL Data

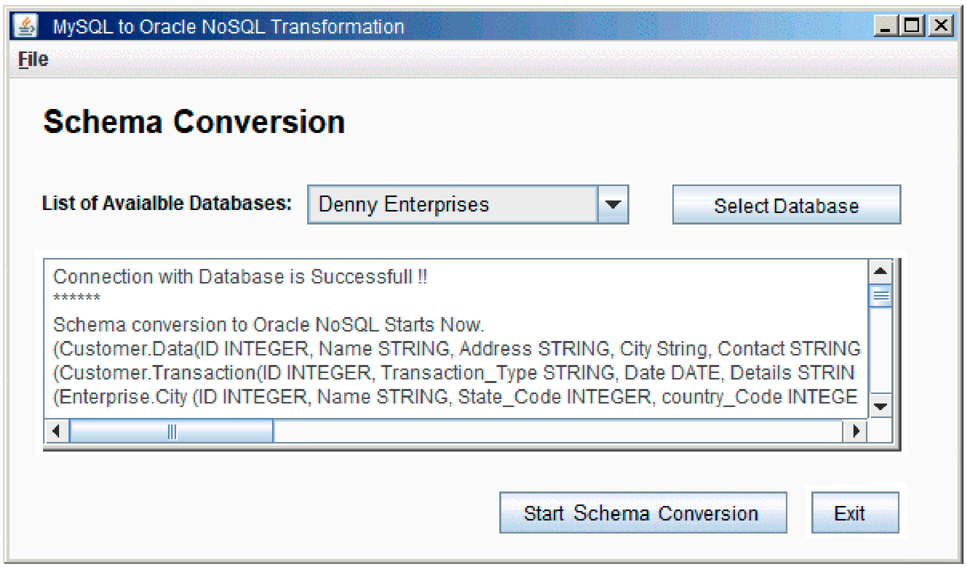
This module performs two tasks:

1. Create procedure: A procedure is created in a generalize format to get data from the database and create its JSON schema.
2. Call procedure to Transform data into JSON: The above function is called and executed, and then it gets data from the database and creates its JSON values row by row. Completing this task, the data that comes in the procedure is stored in a file and the data of this file will be sent to NoSQL database to store data in the database.

5. Results and Discussion

The implemented system was tested with a number of examples to verify the working of the tool and the accuracy of the transformation output. Here, MySQL is used as a source database and Oracle NoSQL is used as a target database. The experiment is performed on different datasets of databases. Our developed system has two parts, one is schema conversion, and the other is data transformation. In the Schema Conversion part, the software will work according to these steps:

1. In this step, Oracle NoSQL is started by running a set of commands in the CLI interface.
2. When designed software starts running, the user connects to a MySQL database through the software.
3. After MySQL connectivity, the next step is to select the required database from the connected databases of MySQL for conversion into the NoSQL database.
4. When a database is selected, then the software will give two options, one schema conversion and the other is data conversion; when the user clicks on the Schema Conversion button, then software will automatically create the schema of database tables. If there is a relationship between tables, then all tables linked with one another will be selected. The software will automatically convert this relationship into a parent-child relationship of Oracle NoSQL database. This parent-child relationship is an alternative for JOINs.
5. For schema conversion, a function is called, which get the table schema from a MySQL database and then stores it in a text file in the application. After completing this function, another function is executed; it gets data from the file and starts mapping the MySQL table schema, its attributes and data type with oracle NoSQL attribute style and data type. Later on, this table schema converted into Avro Schema and is stored into the Oracle NoSQL database. Avro schema is used in the Oracle NoSQL database for creating a record or table schema in which the data will be stored. Details of converted tables will be displayed in the form as shown in Figure 12, if any error is found in the conversion process, it will also be displayed in the form.



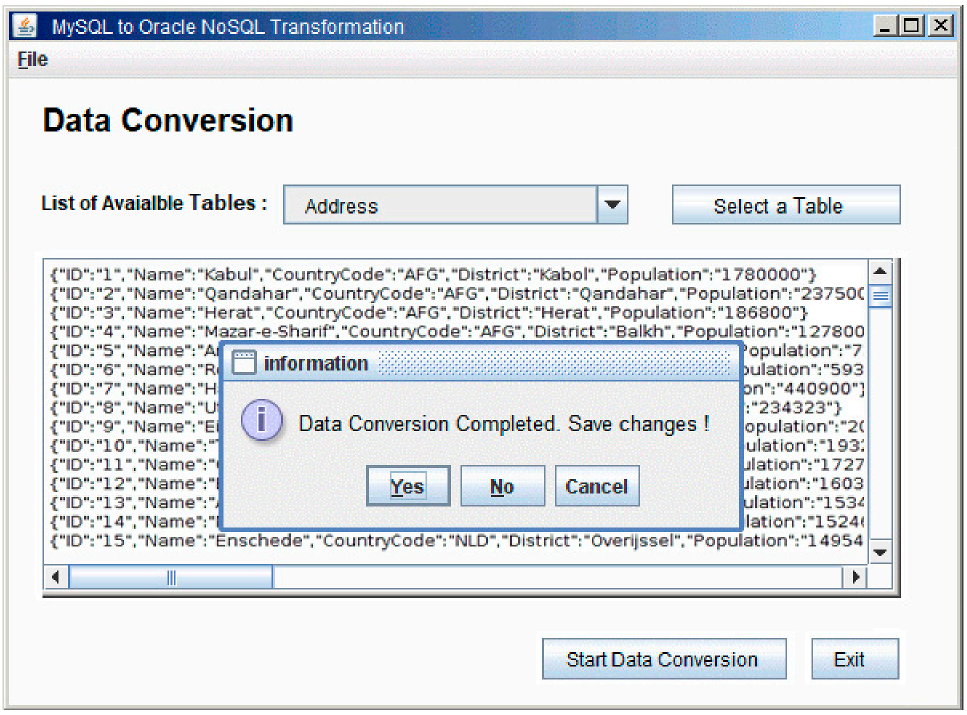
```
Connection with Database is Successfull !!
******
Schema conversion to Oracle NoSQL Starts Now.
(Customer.Data(ID INTEGER, Name STRING, Address STRING, City String, Contact STRING
(Customer.Transaction(ID INTEGER, Transaction_Type STRING, Date DATE, Details STRIN
(Enterprise.City (ID INTEGER, Name STRING, State_Code INTEGER, country_Code INTEGE
```

Figure 12. Schema conversion details.

In the data transformation part, all steps are the same except for Step.5. The Step.5 of data transmission is started when a table is selected from the given tables. After selecting the table, and clicking on the Data transformation button, a function is called to get the data of the table from MySQL database and store it in a text file. After that, another function is called which gets data from a text file and stores it in specific schema table of the Oracle NoSQL database. The form will display the details of the converted table. The conversion detail of the city table is shown in Figure 13.

The proposed system is tested on a Ci5 2.4 Ghz processor with 4GB RAM with Ubuntu 14 OS on VM; five different databases are tested in the proposed system to check the effectiveness of the proposed system. The databases used for the evaluation of our proposed methodology are given below:

- World: This database has three tables (Country, City, Language) [size: 1.2 GB]
- OnlineQuiz: This database has five tables (Category, SubCategory, Quiz, Users, TestDetails.. [size: 2.3 GB]
- Accounts and products: This database has six tables (User, Accounts, Transactions, redemption, ebaycard, products). [size: 3.2 GB]
- Employees: This database contains five tables (Emp, Dept, Products, Sale, Accounts). [size: 1.4 GB]
- Classicmodels: This database has eight tables (customers, offices, emp, orderdet, order, payments, productline, products). [size: 1.1 GB]
- Denny Enterprises: This database has seven tables (Customer, Transactions, City, Products, Payment, Stock, Order). [size: 1.78 GB]



```
{"ID":"1","Name":"Kabul","CountryCode":"AFG","District":"Kabol","Population":"1780000"}
{"ID":"2","Name":"Qandahar","CountryCode":"AFG","District":"Qandahar","Population":"23750(
{"ID":"3","Name":"Herat","CountryCode":"AFG","District":"Herat","Population":"186800"}
{"ID":"4","Name":"Mazar-e-Sharif","CountryCode":"AFG","District":"Balkh","Population":"127800
{"ID":"5","Name":"A                                                        opulation":"7
{"ID":"6","Name":"R                                                        ulation":"593
{"ID":"7","Name":"H                                                        n":"440900"}
{"ID":"8","Name":"U                                                        :"234323"}
{"ID":"9","Name":"E                                                        opulation":"2(
{"ID":"10","Name":"                                                        ulation":"193:
{"ID":"11","Name":"                                                        lation":"1727
{"ID":"12","Name":"                                                        lation":"1603
{"ID":"13","Name":"                                                        ulation":"1534
{"ID":"14","Name":"                                                        lation":"1524(
{"ID":"15","Name":"Enschede","CountryCode":"NLD","District":"Overijssel","Population":"14954
```

Data Conversion Completed. Save changes !

Figure 13. Data conversion details.

Figure 13 shows the screenshot of the data conversion module that allows the user to select a table name of the source database and it converts it into the Oracle NoSQL database table. Here, for the conversion, the approach discussed in Section 3.2 is applied and the results of the conversion are also shown in Figure 13.

5.1. Evaluation Methodology

The working, and the results, of the presented approach have been discussed in the previous section. To evaluate the performance of our approach, an evaluation methodology was designed to find how accurately the relational database schema and data is transformed into Key-value NoSQL format. An evaluation methodology, for the performance evaluation of intelligent tools, is used, and was originally proposed by Hirschman, L., Thompson in 1995 [36]. The following section describes the evaluation methodology used to evaluate the performance of our approach.

5.1.1. Criterion for Evaluation

A criterion was defined for the quantitative evaluation of the designed approach to find how accurately it transforms the source database to the target database. Accuracy of the transformation is measured by finding how close the output is of our approach to the opinion of a human expert (named total results). In this study, the opinion of a human expert for the target input was taken and used as a total result for the sake of evaluation.

5.1.2. Method of Evaluation

For the quantitative evaluation of the results of the used approach, each correct transformation (tables, fields, views, keys, etc.) was matched with the expert's opinion ($N_{total\_transformations}$). The results of all transformations were matched as all the transformations that matched the expert's opinion were declared correct ($N_{correct\_transformations}$), and otherwise, were considered incorrect ($N_{incorrect\_transforamtions}$).

5.1.3. Measures of Evaluation

A set of evaluation measures used in our evaluation methodology are: recall, precision, and F-Measure. The details of these three evaluation measures are given below:

Recall. The recall can be attributed as the completeness of the results produced by system. In our methodology, Recall (R) is calculated by finding the number of correct transformation from the total number of transformations. In Equation (1), $N_{correct\_transformations}$ is the number of correct transformations generated by the approach and $N_{total\_transformations}$ is the number of total correct transformations.

$$R = \frac{N_{correct\_transformations}}{N_{total\_transformations}} \tag{1}$$

Precision. The precision can be attributed to as the accuracy of the designed system. Precision is measured by comparing the designed system's number of correct results by all (incorrect and correct) results produced by the system, calculated as: In Equation (2), $N_{correct\_transformations}$ is the number of correct transformations generated by the approach and $N_{incorrect\_transformations}$ is the number of total incorrect transformations.

$$P = \frac{N_{correct\_transformations}}{N_{correct\_transformations} + N_{incorrect\_transformations}} \tag{2}$$

F-measure: The F-measure can be attributed as a harmonic mean of Precision and Recall. F-measure is the harmonic mean or the "standard" average of total, correct, and incorrect results. By using harmonic mean, Sasaki (2007) [24] calculated F-measure using the following formula:

$$F = \frac{2(P)(R)}{P + R} \tag{3}$$

5.2. Quantitate Evaluation

A set of five cases were selected to test the accuracy of the transformation. The selected cases have a set of MySQL databases with different numbers of respective tables in each database. All these five cases were processed with our tool for schema transformation and then data transformation. Table 5 shows the results of schema transformation whereas, each metadata element was considered on the transformation element.

Table 5. Calculate the values of P, R and F-measure.

Case	Total Transforma-tions	Correct Transforma-tions	Incorrect Transforma-tions	Missed Transforma-tions	Precision (P) %	Recall (R) %	F-Measure (F) %
1	24	21	2	1	87.50	91.30	89.35
2	37	33	3	1	89.18	91.66	90.40
3	20	18	1	1	90.00	94.73	92.30
4	26	23	1	2	88.46	95.83	91.99
5	19	16	2	1	84.21	88.89	86.48

Table 5 shows that in all five experiments of RDB to NoSQL migration, the rate of success of transformation in terms of Recall was as high as 88 to 96%. There were rare incorrect transformations as well. In our approach, the conversion rules are supporting the maximum type of conversions from RDB to NoSQL.

Figure 14 shows the results of recall, precision and F-measure of all five different case studies. The results of these measurements show the accuracy and performance of the system under the different database loads. The results shown in Figure 14 depict that the schema to schema transformation and data to data transformation are carried out successfully.

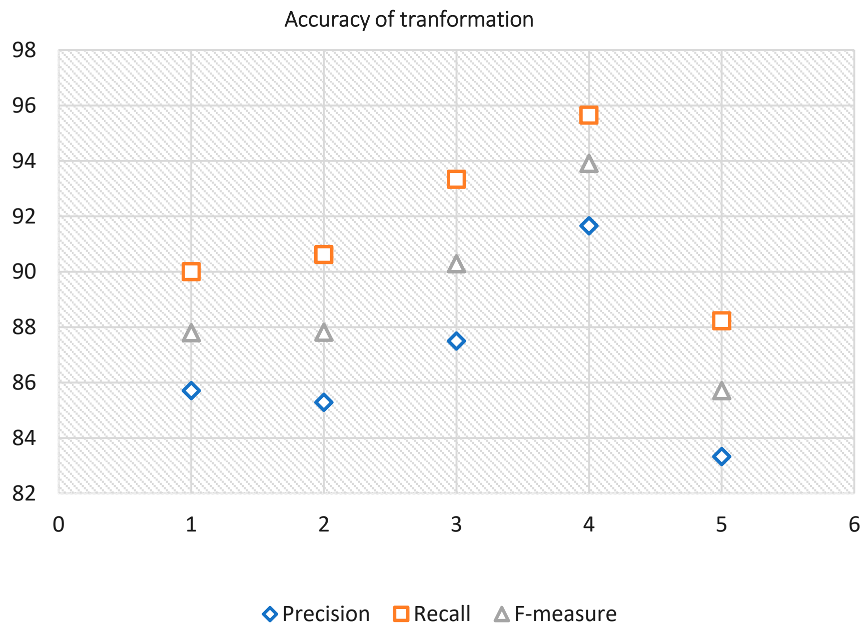


Figure 14. Evaluation results from Relational to NoSQL key-value store.

5.3. Qualitative Evaluation

These databases first tested on schema conversion and the details of this conversion are shown as a graph in Figure 15. In this graph, the world database has three tables, onlineQuiz has five tables and accounts has six tables; however, the conversion time taken for OnlineQuiz accounts for more than double the world class because the OnlineQuiz and accounts database have multi parent child relationship, and therefore, it takes the max time.

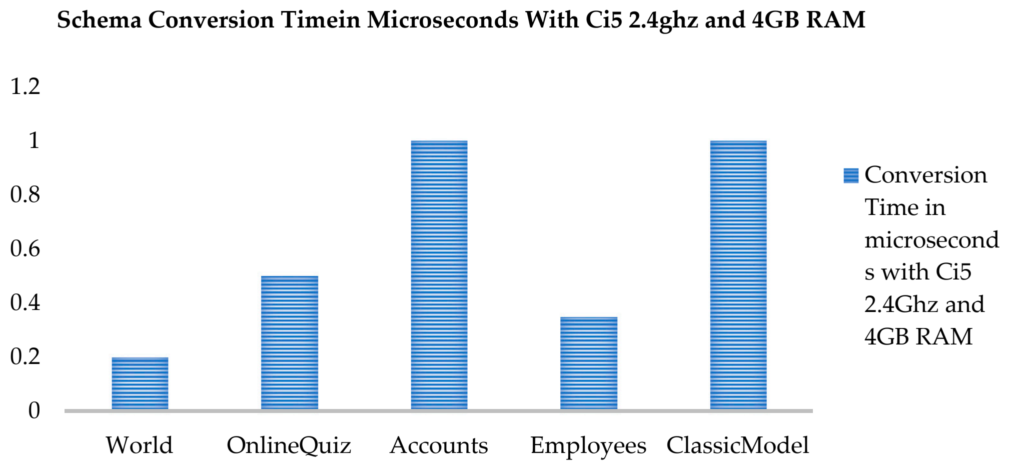


Figure 15. Schema Conversion time in microseconds.

The data conversion time of databases is shown in Figure 16. The single table data conversion time is the same, in the parent-child relationship there is a minor difference in time, but in the multi parent-child relationship, the time is double the parent-child relationship. This is because in the proposed software prototype, the process of mapping parent-child relationship is time consuming. In this process phase, the system finds all foreign keys which are linked with the child table then it creates NoSQL key value store (JSON) storage schema of data and sends it to databases.

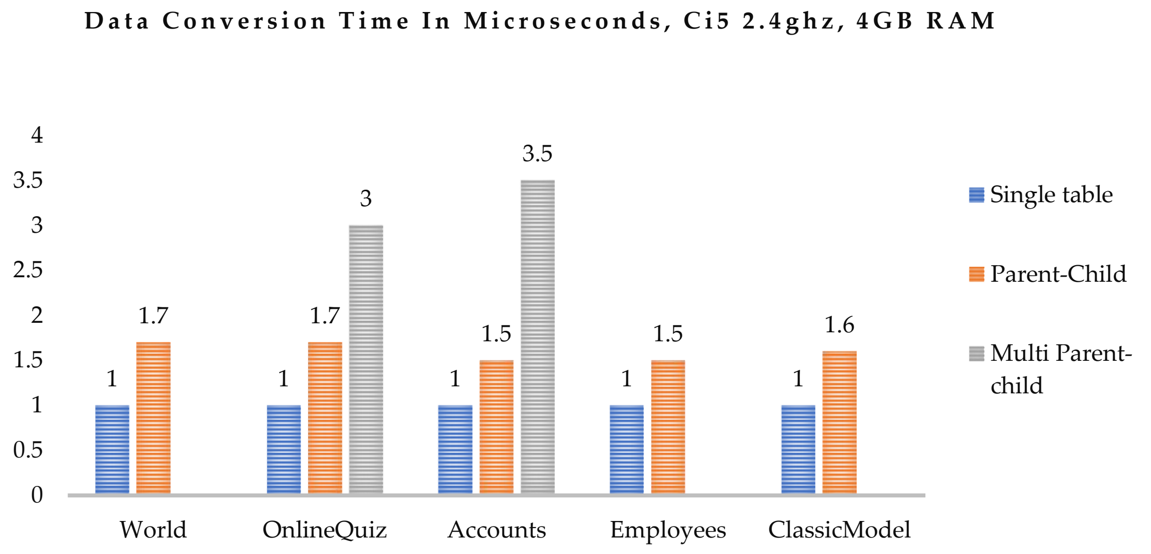


Figure 16. Data conversion time in microseconds.

We could not compare the results of our prototype tool to other tools as no other tool is available that can generate Oracle NoSQL database from the relational database. However, we have compared the results of our prototype tool to a few tools that migrate the relational database to different types of NoSQL databases. Table 6 shows a comparison of performance with the previous approaches:

Table 6. Comparison of our approach with previous approaches.

	Source Database	Target Database	Time	Dataset Size
NoSQLayer [35]	MySQL	MongoDB	1.66 min	50 K records
DigiBrowser [37]	MySQL	NoSQL	10 min	1.5 million records [4.2 GB]
ODBAPI [38]	MySQL	CouchDB	-	-
Kuderu, et al. [39]	RDB	NoSQL	13 min	5000 Transactions
Our Approach	MySQL	Oracle NoSQL	3.5 min	3.2 GB

In this paper, the used approach is novel and automatically transforms the existing database in MySQL to Oracle NoSQL database and provides a highly accurate transformation. The used approach uses a rule-based system to perform transformation at the schema level as well as at the data level. A software prototype for this transformation is also developed as a proof of concept. The results of the experiments show the correctness of our transformations, and outperforms the other similar approaches.

6. Conclusions and Future Work

This study has presented a system to automatically transform relational database into a NoSQL key-value store. The developed system does conversion at the schema level as well as at the data level. The user chooses the type of conversion one wants to perform. In the schema conversion part, the structure of the whole database tables with relationships will be converted to the Oracle NoSQL schema. In the data conversion part, the data of the required tables are converted to Oracle NoSQL supported data types. JSON schema is used for this conversion methodology. The software prototype is developed in Java language. The system has been implemented and evaluated on different sample databases. The results show that the transformation process is very efficient and accurate.

As a future direction, our approach will be able to enhance advance technologies to support all other relational databases and NoSQL databases. The transformation time can be further reduced by using direct entry method (from MySQL to Oracle NoSQL without using middle storage medium).

Here, a model transformation to map RDB elements to NoSQL elements can also improve the accuracy and efficiency of the said migration.

Author Contributions: S.R. contributed in design, implementation, and experimentation of this research and writing this manuscript. I.S.B. supervised this work and also edited this manuscript. R.K. contributed in experiments and evaluation of this research.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Codd, E.F. Relational database: A practical foundation for productivity. In *Readings in artificial Intelligence and Databases*; Elsevier: Heidelberg, Germany, 1988; pp. 60–68.
2. Gantz, J.; Reinsel, D. Extracting value from chaos. *IDC Iview* **2011**, *1142*, 1–12.
3. Hecht, R.; Jablonski, S. Nosql evaluation: A use case oriented survey. In Proceedings of the 2011 International Conference on Cloud and Service Computing (CSC), Hong Kong, China, 12–14 December 2011; pp. 336–341.
4. Han, J.; Haihong, E.; Le, G.; Du, J. Survey on NoSQL database. In Proceedings of the 2011 6th International Conference on Pervasive Computing and Applications (ICPCA), Port Elizabeth, South Africa, 26–28 October 2011; pp. 363–366.
5. Sadalage, P.J.; Fowler, M. *Nosql Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*; Pearson Education: Upper Saddle River, NJ, USA, 2012.
6. Strozzi, C. Nosql-a relational database management system. *Lainattu* **1998**, *5*, 2014.
7. Iwazume, M.; Iwase, T.; Tanaka, K.; Fujii, H.; Hijiya, M.; Haraguchi, H. Big data in memory: Benchimarking in memory database using the distributed key-value store for machine to machine communication. In Proceedings of the 2014 15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Las Vegas, NV, USA, 30 June–2 July 2014; pp. 1–7.
8. Mohamed, M.A.; Altrafi, O.G.; Ismail, M.O. Relational vs. NoSQL databases: A survey. *Int. J. Comput. Inf. Technol.* **2014**, *3*, 598–601.
9. Scherzinger, S.; Klettke, M.; Störl, U. Managing schema evolution in NoSQL data stores. *arXiv* **2013**, arXiv:1308.0514.
10. DeCandia, G.; Hastorun, D.; Jampani, M.; Kakulapati, G.; Lakshman, A.; Pilchin, A.; Sivasubramanian, S.; Vosshall, P.; Vogels, W. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS Oper. Syst. Rev.* **2007**, *41*, 205–220. [CrossRef]
11. Davoudian, A.; Chen, L.; Liu, M. A survey on NoSQL stores. *ACM Comput. Surv.* **2018**, *51*, 40. [CrossRef]
12. O'Neil, P.; Cheng, E.; Gawlick, D.; O'Neil, E. The log-structured merge-tree (LSM-tree). *Acta Inform.* **1996**, *33*, 351–385. [CrossRef]
13. Chang, F.; Dean, J.; Ghemawat, S.; Hsieh, W.C.; Wallach, D.A.; Burrows, M.; Chandra, T.; Fikes, A.; Gruber, R.E. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.* **2008**, *26*, 4. [CrossRef]
14. Li, N.; Xu, B.; Zhao, X.; Deng, Z. Database conversion based on relationship schema mapping. In Proceedings of the 2011 International Conference on Internet Technology and Applications (iTAP), Wuhan, China, 16–18 August 2011; pp. 1–5.
15. Vora, M.N. Hadoop-HBASE for large-scale data. In Proceedings of the 2011 International Conference on Computer Science and Network Technology (ICCSNT), Harbin, China, 24–26 December 2011; pp. 601–605.
16. Yoo, J.; Lee, K.-H.; Jeon, Y.-H. Migration from RDBMS to NoSQL using column-level denormalization and atomic aggregates. *J. Inf. Sci. Eng.* **2018**, *34*, 1–17.
17. Liao, Y.-T.; Zhou, J.; Lu, C.-H.; Chen, S.-C.; Hsu, C.-H.; Chen, W.; Jiang, M.-F.; Chung, Y.-C. Data adapter for querying and transformation between SQL and NoSQL database. *Future Gen. Comput. Syst.* **2016**, *65*, 111–121. [CrossRef]
18. Lakshman, A.; Malik, P. Cassandra: A decentralized structured storage system. *ACM SIGOPS Oper. Syst. Rev.* **2010**, *44*, 35–40. [CrossRef]

19. Chodorow, K. *Mongodb: The Definitive Guide: Powerful and Scalable Data Storage*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2013.
20. Manyika, J.; Chui, M.; Brown, B.; Bughin, J.; Dobbs, R.; Roxburgh, C.; Byers, A.H. *Big Data: The Next Frontier for Innovation, Competition, and Productivity*; McKinsey Global Institute: San Francisco, CA, USA, 2011.
21. Zhao, G.; Lin, Q.; Li, L.; Li, Z. Schema conversion model of SQL database to NoSQL. In Proceedings of the 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), Krakow, Poland, 4–6 November 2014; pp. 355–362.
22. Lee, C.-H.; Zheng, Y.-L. Automatic SQL-to-NoSQL schema transformation over the MYSQL and HBASE databases. In Proceedings of the 2015 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), Taipei City, Taiwan, 6–8 June 2015; pp. 426–427.
23. Lee, C.-H.; Zheng, Y.-L. SQL-to-NoSQL schema denormalization and migration: A study on content management systems. In Proceedings of the 2015 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Hong Kong, China, 9–12 October 2015; pp. 2022–2026.
24. Sasaki, Y. The Truth of the F-Measure. University of Manchester, Technical Report, Version. Available online: http://www.flowdx.com/F-measure-YS-26Oct07.pdf (accessed on 20 January 2018).
25. Hanine, M.; Bendarag, A.; Boutkhoum, O. Data migration methodology from relational to NoSQL databases. *Int. J. Comput. Electr. Autom. Control Inf. Eng.* **2016**, *9*, 2566–2570.
26. Schreiner, G.A.; Duarte, D.; dos Santos Mello, R. Sqltokeynosql: A layer for relational to key-based NoSQL database mapping. In Proceedings of the 17th International Conference on Information Integration and Web-Based Applications & Services, Brussels, Belgium, 11–13 December 2015; p. 74.
27. Rith, J.; Lehmayr, P.S.; Meyer-Wegener, K. Speaking in tongues: SQL access to NoSQL systems. In Proceedings of the 29th Annual ACM Symposium on Applied Computing, Gyeongju, Korea, 24–28 March 2014; pp. 855–857.
28. Serrano, D.; Han, D.; Stroulia, E. From relations to multi-dimensional maps: Towards an SQL-to-hbase transformation methodology. In Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing (CLOUD), New York, NY, USA, 27 June 27–2 July 2015; pp. 81–89.
29. Ouanouki, R.; April, A.; Abran, A.; Gomez, A.; Desharnais, J. Toward building rdb to hbase conversion rules. *J. Big Data* **2017**, *4*, 10. [CrossRef]
30. Li, C. Transforming relational database into HBASE: A case study. In Proceedings of the 2010 IEEE International Conference on Software Engineering and Service Sciences (ICSESS), Beijing, China, 16–18 July 2010; pp. 683–687.
31. Radonić, M.; Mekterović, I. Etlator-a scripting ETL framework. In Proceedings of the 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 22–26 May 2017; pp. 1349–1354.
32. Yang, C.; Huang, Q.; Li, Z.; Liu, K.; Hu, F. Big Data and cloud computing: Innovation opportunities and challenges. *Int. J. Dig. Earth* **2017**, *10*, 13–53. [CrossRef]
33. Suciu, G.; Dobre, C.; Suciu, V.; Todoran, G.; Vulpe, A.; Apostu, A. Cloud computing for extracting price knowledge from big data. In Proceedings of the 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS), Santa Catarina, Brazil, 8–10 July 2015; pp. 314–317.
34. Storey, V.C.; Song, I.-Y. Big data technologies and management: What conceptual modeling can do. *Data Knowl. Eng.* **2017**, *108*, 50–67. [CrossRef]
35. Rocha, L.; Vale, F.; Cirilo, E.; Barbosa, D.; Mourão, F. A framework for migrating relational datasets to NoSQL1. *Procedia Comput. Sci.* **2015**, *51*, 2593–2602. [CrossRef]
36. Hirschman, L.; Thompson, H.S. Chapter 13 evaluation: Overview of evaluation in speech and natural language processing. In *Survey of the State of the Art in Human Language Technology*; Cambridge University Press: New York, NY, USA, 1995.
37. Karnitis, G.; Arnicans, G. Migration of relational database to document-oriented database: Structure denormalization and data transformation. In Proceedings of the 2015 7th International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN), Riga, Latvia, 3–5 June 2015; pp. 113–118.

38. Sellami, R.; Bhiri, S.; Defude, B. ODBAPI: A unified REST API for relational and NoSQL data stores. In Proceedings of the 2014 IEEE International Congress on Big Data (BigData Congress), Anchorage, AK, USA, 27 June–2 July 2014; pp. 653–660.
39. Kuderu, N.; Kumari, V. Relational Database to NoSQL Conversion by Schema Migration and Mapping. *Int. J. Comput. Eng. Res. Trends* **2016**, *3*, 506–513. [CrossRef]