2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT)
Bangalore, India. Oct 7-9, 2022

Transforming NoSQL Database to Relational Database: An Algorithmic Approach

Aniketh Anchalia
Department of Computer Science and Engineering
MS Ramaiah Institute of Technology
(Affiliated to VTU)
Bengaluru, India
animketh@gmail.com

Ankit Paudel
Department of Computer Science and Engineering
MS Ramaiah Institute of Technology
(Affiliated to VTU)
Bengaluru, India
ankitpdl2000@gmail.com

Dr. Sanjeetha R
Department of Computer Science and Engineering
MS Ramaiah Institute of Technology
(Affiliated to VTU)
Bengaluru, India
sanjeetha.r@msrit.edu

Anirudh Kakati
Department of Computer Science and Engineering
MS Ramaiah Institute of Technology
(Affiliated to VTU)
Bengaluru, India
anirudhkakati@gmail.com

2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT) | 978-1-6654-6855-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/GCAT55367.2022.9972168

***Abstract* - Use of NoSQL to maintain databases has gained popularity in recent times. However, traditional relational databases provide structured data which helps in quick response time to execute certain queries. Currently there exist many algorithms that convert relational to unstructured databases, but vice versa is not available extensively. In this paper we propose an algorithm to convert NoSQL databases to relational databases, i.e., conversion of MongoDB databases to MySQL databases. Five datasets taken from GitHub, namely mobile accessories, bookstore, university database, restaurants and ecommerce store, are used as case studies to show the performance when stored as MongoDB database and as MySQL database. The input is given in JSON format and the output is obtained as MySQL tables. The execution time of the proposed algorithm and the time taken to execute queries are tabulated. It is observed that the execution time of the group-by queries improves by 75.33% when the database is converted to SQL.**

Index Terms – NoSQL, Relational Database, MongoDB, MySQL

I. Introduction

The basic purpose of databases is to arrange large volumes of data so that it can be rapidly retrieved by users. As a result, they must be small, and well-structured and data extraction speed must be fast and efficient.

MySQL is a relational database management system that is free and open-source. MySQL includes stand-alone clients that permit users to interact. SQL can be used to interact directly with a MySQL database, but MySQL is most commonly used in conjunction with other tools to create applications that require relational database capability. Some of the major features available in NoSQL incorporate cross-platform support, updatable views, performance schema, ACID compliance, built-In replication support, high availability, etc.

MongoDB is a cross-platform document-oriented database application that is open source. It is a NoSQL database application that works with JSON-like documents and optional schemas. MongoDB allows you to search by field, range, or regular expression. Queries can retrieve specific fields from documents and can also include JavaScript functions defined by the user. The major features provided by MongoDB include indexing replication, load-balancing, file-storage, aggregation, server-side JavaScript execution, capped collection, transactions, etc. Relational Databases like MySQL have the advantage of the splitting table and forming relations to provide a proper structure to the database, also helping increase the speed of data extraction from the database. RDBMS works best with data that can be subdivided across multiple tables. RDBMS has better data integrity and security whereas data from tens of thousands of MongoDB installations have been stolen due to MongoDB's default security setting, which allows anyone to have full access to the database.

Although MongoDB and NoSQL databases provide more data flexibility and diversity, SQL databases are essential when data needs to be structured and comply with ACID properties. Greater transaction reliability is provided by SQL databases. An algorithm to convert NoSQL databases, specifically MongoDB databases, to MySQL databases is presented in this work. The presented algorithm can handle even nested JSON documents in MongoDB, giving consistent data in MySQL.

II. Related Works

With rapid increase in the amounts and size of data, NoSQL databases are increasingly being implemented. There is extensive research in the conversion of data being stored in SQL supported databases to NoSQL databases. A hybrid database architecture-promoting data adapter system was put out [5]. Both the advantages of RDBMS, like their ACID properties, and NoSQL are maintained through an intermediate layer that facilitates concurrent transactions with other layers and keeps track of all ongoing transactions. A schema conversion model for SQL to NoSQL [6] is

established that can provide superior efficiency of join queries with nesting tables, and a graph transformation method that offers properly nested sequences in order to contain all of the join query's required material in a table. By inserting middleware (Metadata) between the application layer and the database layer [9], a solution for integrating SQL and NoSQL databases is suggested. For handling large amounts of data like Big Data, NoSQL databases are utilized [10] and conversion techniques are proposed. A unified query middleware system [11], that can conduct joins, aggregation and filtering on data from numerous sources and allows federated analytical queries using standard SQL syntax across numerous data sources, is described. In order to make the transition from relational DBMS to NoSQL DBMS as convenient as possible, RDBMS to NoSQL technique through schema migration and query mapping [13] was proposed.

Numerous studies comparing and contrasting SQL and NoSQL databases have been conducted. For evaluating the efficiency of NoSQL databases [3][4], basic query operations were performed and the results analyzed. The sharding mechanism and MongoDB's automatic load balancing capability is outlined [4]. The time taken for insertion, retrieval [7][15], updating, deletion [7] and joining [15] was analyzed for MongoDB and MySQL. When time taken is plotted against varying sizes of the databases, for selection operations, SQL is found to be more efficient whereas for insertion, updating, deletion and joining operations, MongoDB is the one that was found to yield better efficiency. CRUD operations for MySQL, document-based MySQL and CouchDB were evaluated based on the time taken [8]. For insertion NoSQL approach was found to perform better. For single-update MySQL was found to yield better results. For multiple updates, selection with joins and soft delete operations MySQL performed better when the number of elements was small but performance deteriorated as the number became large. For simple select and hard delete operations CouchDB gave better results. The advantages and disadvantages of SQL and NoSQL databases were identified [12][14]. The analysis has found that SQL is the better choice when ACID features are significant. NoSQL databases, with their adaptable model, provide more scalability and are useful when huge amounts of data are involved. Not having a standard query language may be a reason for customers being hesitant to employ NoSQL databases.

In most applications, relational databases are used, and they perform effectively when dealing with smaller volumes of data. Developing relational databases from NoSQL databases hasn't been the subject of significant study. A general methodology for converting various NoSQL databases to RDBMS was suggested [1] for Neo4j and MongoDB databases. A dynamic schema mapping approach for conversion of NoSQL to SQL was presented [2], by converting MongoDB databases to MySQL and PostgreSQL databases.

The existing NoSQL to SQL conversion methods cannot work with nested JSON documents as input. The proposed methodology aims to handle that.

III. Experimental Setup

The study was conducted in phases, each phase contributing to the transformation of data objects from a non-relational data model into a relational data model, as shown in Figure 1.

The JSON dataset file is passed to the Data Integration phase of the study where the data is preprocessed by replacing the attribute value with no entries by NULL values and forming a dictionary of the attributes with attribute values as key-value pairs. These data entries are added to the single dictionary file from their presence in multiple JSON object structures. If the attribute value of the attribute in the dictionary is NULL for over 75% of the data entries, that particular attribute is removed from all the datasets considering the inadequate amount of data evaluation information present in the given datasets for the attribute.

The integrated data after preprocessing is passed through the most focal phase of the study. In this phase, the integrated data in dictionary form is translated into a Relational Database where if the attribute is of type dictionary, then its value is appended to the list with the same key. In presence of any nested attributes of the form list, the list data is appended to the list in its entirety, of the same key. On conversion of all the attributes into the list, the lists are converted into a table structure for all the individual entries if the list entry is a dictionary on its own it is represented in a separate table with the inclusion of a surrogate key common to both the tables to maintain the lossless join property of the data. All the data

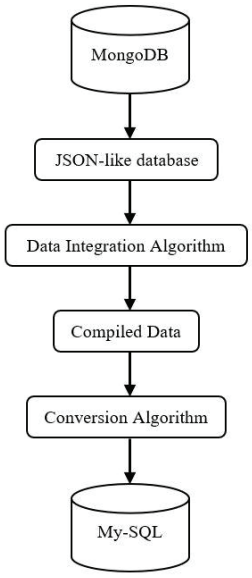

Fig. 1. Flow Diagram for the conversion from MongoDB to MySQL

entries of the list are converted into tables following the same procedure and are mapped to the SQL in the form of relations by making use of the conversion algorithm to generate the relational model of the data transformed from the non-relational datasets.

IV. Implementation

Python 3 was used for algorithm development. The relational and non-relational databases used were MongoDB and MySQL, respectively. You can use any IDE that supports Python 2.7 or above. In all the algorithms, a JSON object will be referred to as a dictionary. Before using the algorithm, a global object storing the integrated data must be made. The study's development was divided into two algorithms:

A. Data Integration

This algorithm's main goal is to take a dictionary and append the required information to the global object. For each key in the first iteration of the algorithm, an empty list is created for that attribute and inserted to the global dictionary. Then determine whether or not the global object's length is 0. If it is not 0, check if all of the input's attributes are present in the global object. If an attribute is missing, a list is created and n NULLs are added to it, where n is the length of the list of the first attribute. Two scenarios may arise while looping through the input:

1. The attribute is not a dictionary- Append the value of the attribute to the list of that attribute
2. The attribute is a dictionary - If the attribute is not present, create a new dictionary and repeat the above process. Later, add this dictionary to the global object. If the attribute is present, then iterate through that dictionary and repeat the process mentioned in step 1

Continue in this manner until all of the input has been iterated through. The next step is to verify consistency. The dictionary's keys are repeatedly examined to see if each list of an attribute's values is the same length as the list of the first value. To the list of lower lengths, "NULL" is added. The algorithm is shown below:

```
dict ← a global object/dictionary
    //Purpose: To integrate all the data and store it in one container
    //Input: in_dict, a dictionary/JSON object
    //Output: The integrated data in dict from various in_dict
    dataIntegration(in_dict)
    If length of dict != 0
    .   ForEach attribute in in_dict
    .   .   If the attribute is not in dict
    .   .   .   Create a list
    .   .   .   Append n number of 'NULL' to the new list where
    .   .   .   n is the length of the first attribute's list. Add this
    .   .   .   attribute to dict with value new list
    .   .   End If
    .   End ForEach
    End If
    ForEach attribute in in_dict
    .   If the attribute is not of type dictionary/JSON
    .   .   Append the value to the list of that attribute
    .   End If
    .   Else
    .   .   If an attribute is of type dictionary and it is not in dict
    .   .   .   Create a new dictionary dict_x and insert the
    .   .   .   attribute in it using the data integration algorithm
    .   .   .   After inserting the attribute, dict_x is appended to
    .   .   .   the dict
    .   .   End If
    .   .   Else
    .   .   .   If the attribute is present in dict
    .   .   .   .   Check if a new attribute is there and add
    .   .   .   .   'NULL'
    .   .   .   End If
    .   .   .   Else
    .   .   .   .   Append the value to the existing list
    .   .   .   End Else
    .   .   End Else
    .   End Else
    End ForEach
    If there is a missing attribute in the in_dict that is present in the
    dict
    .   Then insert a list of 'n' 'NULL' in the missing attribute
    .   value in dict
    .   If the attribute in dict is not of type dictionary
    .   .   If the length of the attribute is less than the length of
    .   .   the first attribute in dict
    .   .   .   Append 'NULL' to the list of that attribute in dict
    .   .   End If
    .   .   Else If the length of the attribute is greater than the
    .   .   length of the first attribute in dict
    .   .   .   Append 'NULL' to the list of the first attribute in
    .   .   .   dict
    .   .   End Else
    .   End If
    .   Else
    .   .   Iterate through that attribute which is of type
    .   .   dictionary
    .   .   ForEach attribute in this dictionary
    .   .   .   If the length of the attribute is less than the length
    .   .   .   of the first attribute in dict
    .   .   .   .   Append 'NULL' to the list of that attribute
    .   .   .   .   in dict
    .   .   .   End If
    .   .   .   Else If the length of the attribute is greater than
    .   .   .   the length of the first attribute in dict
    .   .   .   .   Append 'NULL' to the list of the first
    .   .   .   .   attribute in the dict.
    .   .   .   End Else
    .   .   End ForEach
    .   End Else
    End If
End
```

B. Conversion Algorithm

This algorithm's objective is to update the MySQL database with the combined data from several objects. The database administrator is in charge of creating the tables and adding various constraints and authentication. By counting the number of attributes that have a dictionary as their value, it is simple to determine how many tables need to be constructed.

The created tables are joined together using a foreign key. The attributes that have more than 75% 'NULL' values are discarded. Tuples are created, each of which correspond to a database row. The tuples can be added to the table one at a time or by making a list of them and inserting it all at once. A new list of tuples must be created if an attribute's value is a dictionary, and this new list of tuples will be added to the relevant table of the nested dictionary. The algorithm is shown below:

*//**Purpose:** To append the integrated data to the relational database*
*//**Input:** The global dictionary dict*
*//**Output:** The relational database*
//Checking if all attributes have 'NULL' lesser than 75% of the length of one of the attributes
*a <= 75/100 * (length of the first attribute in dict)*
***ForEach** attribute in dict*
*. **If** the attribute is not of type dictionary*
*. . **If** the length of the number of 'NULL's in the attribute > 'a'*
. . . Remove that attribute from dict
*. . **End If***
*. **End If***
*. **Else***
. . Iterate through each attribute of this dictionary and remove
. . the attributes that have 'NULL' values greater than 'a'.
*. **End Else***
End ForEach
Establish a connection to the database
Initialise list L to store the attributes of list type and P to store attributes of dictionary types
***ForEach** attribute in dict*
*. **If** the attribute type is a dictionary*
. . Then append the attribute to the list P
*. **End If***
*. **Else***
. . Append the attribute to the list L
*. **End Else***
End ForEach
Create a table for list L and have an additional column that will be the surrogate key for the table.
Create N number of tables where N is the length of the list P.
For each table create a new column for the primary key as done in the previous step, which will be the foreign key for the ith table referencing the column in the previous table in list P. The first table of list P would reference the column created for list L.
Create n lists where n is the number of tables.
For** (i← 0 ; i < length(first attribute's list) ; i++) **do
. Create list C.
. Append each attribute's ith value from the corresponding list
. to C
. Convert C to a tuple and append it to the appropriate table list.
End For
End

V. Results

Dataset 1: A MongoDB collection of mobile accessories was taken [16]. The dataset consists of key-value pairs. The keys of the dataset are ("name", "type", "price", "rating", "warranty_years", "for", "_id") not in the same order in every object as shown in Figure 2. "_id" is a nested JSON key which contains the attribute "Soid". The dataset was passed through both the algorithms and the corresponding output is shown in Figure 3. There are two tables as there is a nested JSON object "_id". The first table consists of attributes ("pid", "name", "type", "price", "rating", "warranty_years", "for"). "pid" is a PRIMARY KEY added to the table. The second table formed due to "_id" contains the attributes ("pid", "id"). "pid" is a PRIMARY KEY of the table and also a FOREIGN KEY referencing "pid" of table one as shown in Figure 3. Separating the "_id" attribute can help us during data analysis by not considering that table. Adding "pid" as PRIMARY KEY and FOREIGN KEY helps in the GROUP BY queries of MySQL.


```
{
    "name": "AC3 Series Charger",
    "type": "charger",
    "price": 19,
    "rating": 2.8,
    "warranty_years": 0.25,
    "for": "ac3",
    "_id": {
        "$oid": "507d95d5719dbef170f15bf9"
    }
}
{
    "_id": {
        "$oid": "507d95d5719dbef170f15bfa"
    },
    "name": "AC3 Case Green",
    "type": "case",
    "color": "green",
    "price": 12,
    "rating": 1,
    "warranty_years": 0
}
{
    "name": "AC3 Case Black",
    "type": "case",
    "color": "black",
    "price": 12.5,
    "rating": 2,
    "warranty_years": 0.25,
    "available": false,
    "for": "ac3"
}
{
    "name": "AC3 Case Red",
    "type": "case",
    "color": "red",
    "price": 12,
    "rating": 4,
    "warranty_years": 0.25,
    "available": true,
    "for": "ac3"
}
```


Fig. 2. JSON dataset of mobile accessories

pid	name	type	price	rating	warranty_years	for	color	available
1	AC3 Series Charger	charger	19	2.8	0.25	ac3	NULL	NULL
2	AC3 Case Green	case	12	1	0	NULL	green	NULL
3	AC3 Case Black	case	12.5	2	0.25	ac3	black	0
4	AC3 Case Red	case	12	4	0.25	ac3	red	1

pid	id
1	507d95d5719dbef170f15bf9
2	507d95d5719dbef170f15bfa
3	NULL
4	NULL

Fig. 3. MySQL database of mobile accessories

Dataset 2: A MongoDB collection of books was taken [17]. The keys of the JSON collection are (“id”, “name”, “description”, “price”, “author”, “type”, “img”, “inCart”, “category”) as shown in Figure 4 The dataset was passed through both the algorithms and the corresponding output is shown in Figure 5. There are no nested JSON objects thus resulting in only one table. The attributes of the table are (“pid”, “id”, “name”, “description”, “price”, “author”, “type”, “img”, “inCart”, “category”). “pid” is added as the PRIMARY KEY of the table. The Database Administrator can decide which attribute to keep as the PRIMARY KEY. As there are no nested JSON objects, there is only one table as shown in Figure 5.


```
{
    "id": 1,
    "name": "The Power of HABIT",
    "description": "The Power of HABIT: Why We Do What We Do in Life and Business. A young woman walks into a laboratory. Over the past two years, she has transformed almost every aspect of her life. She has quit smoking, run a marathon, and been promoted at work. The patterns inside her brain, neurologists discover, have fundamentally changed.",
    "price": 16.33,
    "author": "Charles Duhigg",
    "type": "hardcover",
    "img": "https://images-na.ssl-images-amazon.com/images/I/51ejXdSceNL._AA300_.jpg",
    "inCart": false,
    "category": "business"

    "id": 2,
    "name": "Think and Grow Rich",
    "description": "'Think and Grow Rich!' explains entrepreneur Andrew Carnegie's secret to success, revealed to Napoleon Hill during private interviews with Carnegie, the richest man of his time, and during more than 20 years of research into the lives and philosophies of more than 500 of the most successful people in America. This timeless classic presents a systematic nuts-and-bolts approach to developing the skills and mindset required to achieve exceptional success in any field or endeavor, personal or professional. Hill explains in detail 13 steps required to achieve those goals. The book contains numerous self-tests and checklists.",
    "price": 8.98,
    "author": "Napoleon Hill",
    "type": "hardcover",
    "img": "https://images-na.ssl-images-amazon.com/images/I/51ZoxHoBGtL._SX315_BO1,204,203,200_.jpg",
    "inCart": false,
    "category": "business"

    "id": 3,
    "name": "The 7 Habits of Highly Effective People",
    "description": "The 7 Habits of Highly Effective People: Powerful Lessons in Personal Change. The 7 Habits of Highly Effective People, the beloved classic that has sold over 20 million copies worldwide, is celebrating its 25th anniversary with this reissue! With a new foreword, the wisdom of the 7 Habits still holds true after all these years. The 7 Habits have become so famous because they work. They have been integrated into everyday thinking by many millions of people. The reason: They work. Habit 1: Be Proactive. Habit 2: Begin with the End in Mind. Habit 3: Put First Things First. Habit 4: Think Win/Win. Habit 5: Seek First to Understand, Then to Be Understood. Habit 6: Synergize. Habit 7: Sharpen the Saw. The book presents a principle-centered approach for solving personal and professional problems.",
    "price": 11.48,
    "author": "Stephen R. Covey",
    "type": "paperback",
    "img": "https://images-na.ssl-images-amazon.com/images/I/51Myx6jMqjL._AA300_.jpg",
    "inCart": false,
    "category": "business"
```


Fig. 4. JSON dataset of bookstores

pid	id	name	description	price	author	type	img	inCart	category
1	1	The Power of HABIT	The Power of HABIT: Why We Do What We Do in Life a...	16.33	Charles Duhigg	hardcover	https://images-na.ssl-images-amazon.com/images/I/5...	0	business
2	2	Think and Grow Rich	'Think and Grow Rich!' explains entrepreneur Andre...	8.98	Napoleon Hill	hardcover	https://images-na.ssl-images-amazon.com/images/I/5...	0	business
3	3	The 7 Habits of Highly Effective People	The 7 Habits of Highly Effective People: Powerful ...	11.48	Stephen R. Covey	paperback	https://images-na.ssl-images-amazon.com/images/I/5...	0	business

Fig. 5. MySQL database of bookstores

Dataset 3: A MongoDB collection of a university database was taken [16]. The dataset consists of keys (“Name”, “Age”, “School”, “Sex”, “DoB”). “Schools” is a nested JSON object consisting of attributes (“College”, “High_School”) as shown in Figure 6. The dataset was passed through both the algorithms and the corresponding output is shown in Figure 7. There are two tables as there is a nested JSON object “Schools”. The attributes of table one are (“pid”, “Name”, “Age”, “Sex”). The key DoB has been discarded as it had more than 75% NULL values. “pid” is the PRIMARY KEY of table one and is added. The second table consists of attributes (“pid”, “College”, “High_School”). “pid” is the PRIMARY and FOREIGN KEY of the second table as shown in Figure 7.


```
{
    'Name': "Liam",
    'Age': 20,
    'Schools': {
        'College': 'Adelphi University'
    }
}
{
    'Name': "Olivia",
    'Age': 24,
    'Schools': {
        'College': 'Adrian College',
        'High_School': 'Phillips Academy Andover'
    },
    'Sex': 'Female'
}
{
    'Name': "Noah",
    'Age': 25,
    'Schools': {
        'High_School': 'Groton School',
        'College': 'Alma College'
    },
    'Sex': 'Male'
}
{
    'Name': "Emma",
    'Schools': {
        'College': 'Belmont Abbey College'
    },
    'Sex': 'Female',
    'Dob': '2001-01-05'
}
{
    'Name': "Ava",
    'Schools': {
        'High_School': 'Trinity School'
    },
    'Sex': 'Female'
}
```


Fig. 6. JSON dataset of University database

pid	name	age	sex
1	Liam	20	NULL
2	Olivia	24	Female
3	Noah	25	Male
4	Emma	NULL	Female
5	Ava	NULL	Female

pid	College	High_School
1	Adelphi University	NULL
2	Adrian College	Phillips Academy Andover
3	Alma College	Groton School
4	Belmont Abbey College	NULL
5	NULL	Trinity School

Fig. 7. MySQL database of University database

Dataset 4: A MongoDB collection of Restaurants was taken [16]. The keys of this collection are (“URL”, “_id”, “address”, “address line 2”, “name”, “outcode”, “postcode”, “rating”, “type_of_food”) as shown in Figure 8. The dataset was passed through both the algorithms and the corresponding output is shown in Figure 9. There are two tables as there is a nested JSON object “_id”. Table one consists of attributes (“pid”, “URL”, “address”, “address line 2”, “name”, “outcode”, “postcode”, “rating”, “type_of_food”) where the attribute “pid” is added as the PRIMARY KEY of the table. Table two consists of the attributes (“pid”, “oid”). “pid” is the PRIMARY and FOREIGN KEY of the table referencing the “pid” of table one. Separating the “_id” attribute can help us during data analysis by not considering that table as shown in Figure 9.


```
{
    "URL": "http://www.just-eat.co.uk/restaurants-cn-chinese-cardiff/menu",
    "_id": {
        "$oid": "55f14312c7447c3da7051b26"
    },
    "address": "228 City Road",
    "address line 2": "Cardiff",
    "name": ".CN Chinese",
    "outcode": "CF24",
    "postcode": "3JH",
    "rating": 5,
    "type_of_food": "Chinese"
}
{
    "_id": {
        "$oid": "55f14312c7447c3da7051b27"
    },
    "address": "376 Rayleigh Road",
    "address line 2": "Essex",
    "name": "@ Thai",
    "outcode": "SS9",
    "postcode": "5PT",
    "rating": 5.5,
    "type_of_food": "Thai"
}
{
    "URL": "http://www.just-eat.co.uk/restaurants-atthairestaurant/menu",
    "address": "30 Greyhound Road Hammersmith",
    "address line 2": "London",
    "name": "@ Thai Restaurant",
    "outcode": "W6",
    "postcode": "8NX",
    "rating": 4.5,
    "type_of_food": "Thai"
}
```


Fig. 8. JSON dataset of Restaurants

pid	URL	address	address2	name	outcode	postcode	rating	type_of_food
1	http://www.just-eat.co.uk/restaurants-cn-chinese-c	228 City Road	Cardiff	.CN Chinese	CF24	3JH	5	Chinese
2	NULL	376 Rayleigh Road	Essex	@ Thai	SS9	5PT	5.5	Thai
3	http://www.just-eat.co.uk/restaurants-atthairestau	30 Greyhound Road Hammersmith	London	@ Thai Restaurant	W6	8NX	4.5	Thai

pid	oid
1	55f14312c7447c3da7051b26
2	55f14312c7447c3da7051b27
3	NULL

Fig. 9. MySQL database of Restaurants

Dataset 5: A MongoDB collection of an e-commerce store was taken [16]. The dataset consists of keys (“additional Features”, “os”, “battery”, “camera”, “connectivity”, “description”, “display”, “hardware”, “id”, “images”, “name”, “storage”). There are six nested JSON objects (“battery”, “camera”, “connectivity”, “display”, “hardware”, “storage”) as shown in Figure 10. The dataset was passed through both the algorithms and the corresponding output is shown in Figure 11. There are 7 tables due to the presence of 6 nested JSON objects. The first table consists of attributes (“pid”, “additionalFeatures”, “os”, “description”, “id”, “images”, “name”). “pid” is the PRIMARY KEY of the table. Table two consists of the attributes in the key “battery”, which are (“pid”, “type”, “standByTime”). “pid” is a FOREIGN KEY referencing the “pid” of table one. Table three consists of attributes in the key “camera”, which are (“pid”, “features, “primary”). “pid” is a FOREIGN KEY referencing the “pid” of table two. Table four consists of attributes in the key “connectivity”, which are (“pid”, “bluetooth, “cell”, “gps”, “infrared”, “wifi”). “pid” is a FOREIGN KEY referencing the “pid” of table three. Table five consists of attributes in the key “display”, which are (“pid”, “screenResolution”, “screenSize”). “pid” is a FOREIGN KEY referencing the “pid” of table four. Table six consists of attributes in the key “hardware”, which are (“pid”, “accelerometer, “audioJack”, “cpu”, “fmRadio”, “physicalKeyboard”, “usb”). “pid” is a FOREIGN KEY referencing the “pid” of table five. Table seven consists of attributes in the key “storage”, which are (“pid”, “hdd”, “ram”). “pid” is a FOREIGN KEY referencing the “pid” of table six. The links can either be created on the first table or as shown in Figure 11.


```
{
  "additionalFeatures": "Front Facing 1.3MP Camera",
   "os": "Macintosh OS X 10.7",
  "battery": {
      "type": "Lithium Ion (Li-Ion) (7000 mAH)",
      "standbytime" : "24 hours"
  },
  "camera": {
      "features":  "Video",
      "primary": "5.0 megapixels"
  },
  "connectivity": {
      "bluetooth": "Bluetooth 2.1",
      "cell": "T-mobile HSPA+ @ 2100/1900/AWS/850 MHz",
      "gps": True,
      "infrared": False,
      "wifi": "802.11 b/g"
  },
  "description": "Flipkart Apple iBook is the best in class computer for your professional and personal work.",
  "display": {
      "screenResolution": "WVGA (1280 x 968)",
      "screenSize": "13.0 inches"
  },
  "hardware": {
      "accelerometer": True,
      "audioJack": "3.5mm",
      "cpu": "Intel i7 2.5 GHz",
      "fmRadio": False,
      "physicalKeyboard": False,
      "usb": "USB 3.0"
  },
  "id": "flipkartproduct_1",
  "images": "img/apple-laptop.jpg",
  "name": "Flipkart.com : Apple iBook",
  "storage": {
      "hdd": "750GB",
      "ram": "8GB"
  }
}
```


Fig. 10. JSON dataset of ecommerce store

pid	additionalFeature	os	description	id	images	name
1	'Front Facing 1.3MP Camera'	Macintosh OS X 10.7	Flipkart Apple iBook is the best in class computer...	flipkartproduct_1	img/apple-laptop.jpg	Flipkart.com : Apple iBook

pid	type	standbytime
1	Lithium Ion (Li-Ion) (7000 mAH)	24 hours

pid	features	primary
1	Video	5.0 megapixels

pid	bluetooth	cell	gps	infrared	wifi
1	Bluetooth 2.1	T-mobile HSPA+ @ 2100/1900/AWS/8	True	False	802.11 b/g

pid	screenResolution	screenSize
1	WVGA (1280 x 968)	13.0 inches

pid	accelerometer	audioJack	cpu	fmRadio	physicalKeyboard	usb
1	True	3.5mm	Intel i7 2.5 GHz	False	False	USB 3.0

pid	hdd	ram
1	750GB	8GB

Fig. 11. MySQL database of ecommerce store

The algorithm execution time taken by each of the five datasets are 5.789, 5.450, 8.072, 4.197, 8.544 seconds respectively. The time taken is determined when the dataset has completed both the algorithms. The time taken to create the tables is not included as that task is left to the Database Administrator. The time taken by each of the datasets is represented by a graph as shown in Figure 12. We can see that as the number of nested JSON objects increases, the time also increases. There was removal of an attribute in Dataset 3 thus resulting in a bit more time than expected.

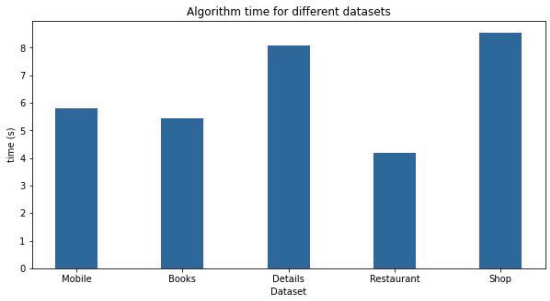

Fig. 12. Algorithm execution time for different datasets

When we convert nested JSON datasets, we get multiple relations and to match the equivalent selection query in MongoDB, selection with JOIN is done in MySQL. For a single relation dataset, simple selection is done.

Dataset 1 Mobile:

- *MySQL:* SELECT * FROM mobile NATURAL JOIN oid2 where 1;
- *MongoDB:* db.mobile.find({}).pretty();

Dataset 2 Books:

- *MySQL:* SELECT * FROM books where 1;
- *MongoDB:* db.books.find({}).pretty();

Dataset 3 University:

- *MySQL:* SELECT * FROM details1 NATURAL JOIN schools1 WHERE 1;
- *MongoDB:* db.university.find({}).pretty();

Dataset 4 Restaurant:

- *MySQL:* SELECT * FROM rest NATURAL JOIN oid WHERE 1;
- *MongoDB:* db.rest.find({}).pretty();

Dataset 5 Shop:

- *MySQL:* SELECT * FROM ecomm NATURAL JOIN battery NATURAL JOIN camera NATURAL JOIN connectivity NATURAL JOIN display NATURAL JOIN hardware NATURAL JOIN storage WHERE 1;
- *MongoDB:* db.ecomm.find({}).pretty();

The time taken to print all the documents using the JOIN query was tested and the results were promising. The time taken for the queries in MySQL by the datasets are 0.00216, 0.00394, 0.00216, 0.002236, 0.00424 seconds respectively, as shown in Figure 13.

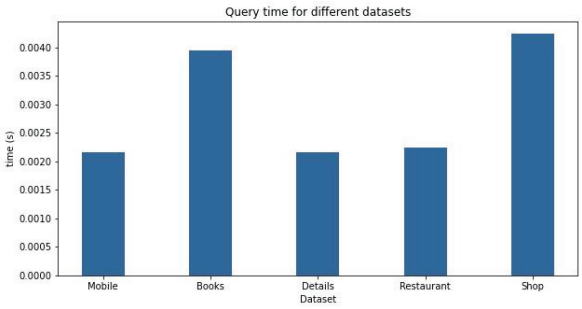

Fig. 13. Query time for different datasets

TABLE I. EXECUTION AND SELECTION QUERY TIMES FOR THE DATASETS

	Algorithm Execution Time (s)	SQL Selection Query Time (s)	NoSQL Selection Query Time (s)
Dataset 1	5.789	0.00216	0.00200
Dataset 2	5.450	0.00394	0.00128
Dataset 3	8.072	0.00216	0.00105
Dataset 4	4.197	0.00224	0.00176
Dataset 5	8.544	0.00424	0.00265

Dataset 5 took 8.5 seconds to complete the entire process, while others took close to 5 seconds to complete. For the selection with JOIN query the maximum time was taken by dataset 5 as well. SQL was found to give slower query time than NoSQL, as shown in Table I.

The three datasets Mobile (dataset 1), Books (dataset 2) and Restaurant (dataset 4) were taken, and a group-by query was run for each of them, in both MySQL and MongoDB. The data was inserted multiple times so as to increase the size of the dataset to analyse the group by query time. The results have been listed.

Dataset 1 Mobile: The same data of 4 rows was inserted twice. A group-by query was run based on the "type" key:

- *MySQL:* SELECT type, COUNT (*) FROM `mobile` WHERE 1 GROUP by type having type = 'case'
- *Mongo:* db.mobile.aggregate([{ $group : {_id : "$type", count : {$sum : 1}}}, {$match : {_id : "case"}}]);

type	COUNT(*)
case	6

Fig. 14. GROUP BY query result on mobile dataset

The query in SQL took 0.00096 seconds whereas for MongoDB it took 0.004 seconds, an improvement of 76%.

Dataset 2 Books: The same data of 3 rows was inserted 4 times. A group-by query was run based on the "id" key:

- *MySQL:* SELECT name, COUNT (*) from books where 1 GROUP BY id
- *Mongo:* db.books.aggregate([{$group : {_id : "$name", count : {$sum : 1}}}]);

name	COUNT(*)
The Power of HABIT	4
Think and Grow Rich	4
The 7 Habits of Highly Effective People	4

Fig. 15. GROUP BY query result on books dataset

The query in SQL took 0.0015 seconds whereas for MongoDB it took 0.005 seconds, an improvement of 70%.

Dataset 4 Restaurant: The same data of 3 rows was inserted 9 times. A group-by query was run based on the "name" key:

- *MySQL:* SELECT name, rating, COUNT (*) FROM `rest` WHERE 1 GROUP by name HAVING rating >=5
- *Mongo:* db.rest.aggregate([{$match : {rating : {$gte : 5}}}, {$group : {_id : {name : "$name", rating : "$rating"}, count : {$sum : 1}}}]);

name	rating	COUNT(*)
.CN Chinese	5	9
@ Thai	5.5	9

Fig. 16. GROUP BY query result on restaurant dataset

The query in SQL took 0.0016 seconds whereas for MongoDB it took 0.008 seconds, an improvement of 80%. It is shown in Figure 17.

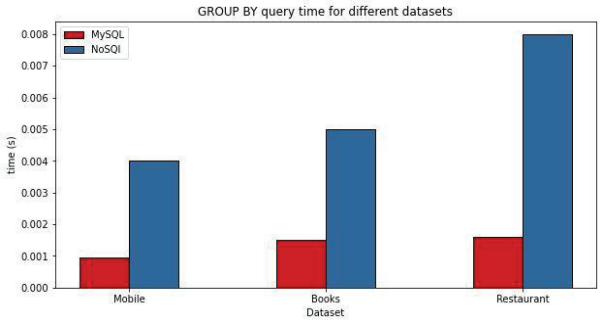

Fig. 17. GROUP BY query time in MySQL and MongoDB for different datasets

TABLE II. COMPARISON OF GROUP BY QUERY TIMES IN MYSQL AND MONGODB

	MySQL (s)	MongoDB (s)
Mobile (dataset 1)	0.00096	0.004
Books (dataset 2)	0.0015	0.005
Restaurant (dataset 4)	0.0016	0.008

MySQL performs better for group-by queries against the equivalent queries in MongoDB, as shown in Table II. An average improvement of 75.33% was found when converted to SQL.

VI. CONCLUSION AND FUTURE WORK

Despite NoSQL databases rapidly growing with ever-growing data in the world, numerous business cases reveal that the data supplied by NoSQL sources is tiny when compared to SQL sources for some systems. A conversion mechanism for NoSQL databases to Relational databases was developed in this study. The model was run for five datasets and the execution and query times were tabulated. The query time of Group-by query was compared for SQL and NoSQL. SQL was found to be faster by 75.33%. Thus, with this approach, a NoSQL database can be converted to a MySQL database with consistent data for future analysis, and data warehousing.

This research was solely based on the conversion of key-value NoSQL database type to a relational database, but as a diversification to this, other NoSQL databases that are Column-oriented, Graph-based, and Document-oriented can be converted into relational databases.

REFERENCES

[1] Maity, Biswajit, Anal Acharya, Takaaki Goto, and Soumya Sen. "A framework to convert NoSQL to relational model." In Proceedings of the 6th ACM/ACIS International Conference on Applied Computing and Information Technology, pp. 1-6. 2018.

[2] Aftab, Zain, Waheed Iqbal, Khaled Mohamad Almustafa, Faisal Bukhari, and Muhammad Abdullah. "Automatic NoSQL to Relational Database Transformation with Dynamic Schema Mapping." Scientific Programming (2020)

[3] Győrödi, Cornelia, Robert Győrödi, George Pecherle, and Andrada Olah. "A comparative study: MongoDB vs. MySQL." In 2015 13th International Conference on Engineering of Modern Electric Systems (EMES), pp. 1-6. IEEE, 2015

[4] Patil, Mayur M., Akkamahadevi Hanni, C. H. Tejeshwar, and Priyadarshini Patil. "A qualitative analysis of the performance of MongoDB vs MySQL database based on insertion and retrieval operations using a web/android application to explore load balancing—Sharding in MongoDB and its advantages." In 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics, and Cloud) (ISMAC), pp. 325-330. IEEE, 2017

[5] Ghule, Sanket, and Ramkrishna Vadali. "Transformation of SQL system to NoSQL system and performing data analytics using SVM." In 2017 International Conference on Trends in Electronics and Informatics (ICEI), pp. 883-887. IEEE, 2017

[6] Zhao, Gansen, Qiaoying Lin, Libo Li, and Zijing Li. "Schema conversion model of SQL database to NoSQL." In 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, pp. 355-362. IEEE, 2014

[7] Shivam Kumar Giri, Chidanandan V., Narendran Rajagopalan. "Research of NOSQL and SQL and Designing a Better Database Schema for Databases", International Journal of Engineering and Advanced Technology (IJEAT), Volume-8 Issue-6S, August 2019

[8] Cornelia A. Gyorödi, Diana V. Dumse-Burescu, Doina R. Zmaranda, Robert S. Gyorödi, Gianina A. Gabor and George D. Pecherle. "Performance Analysis of NoSQL and Relational Databases with CouchDB and MySQL for Application's Data Storage", Appl. Sci. 2020, 10, 8524; doi:10.3390/app10238524

[9] Sanobar Khan, Prof. Vanita Mane, "SQL Support over MongoDB using Metadata", International Journal of Scientific and Research Publications, Volume 3, Issue 10, October 2013

[10] Md. Saiful Islam, "Techniques for Converting Big Data from SQL to NoSQL Databases"

[11] Huiran Zhang, Cheng Zhang, Rui Hu, Xi Liu, and Dongbo Dai. "Unified SQL Query Middleware for Heterogeneous Databases", IWECAI 2021 Journal of Physics: Conference Series, doi:10.1088/1742-6596/1873/1/012065

[12] Twana Asaad Taha. "The SQL vs NoSQL Differences and Similarities", International Journal of Scientific & Engineering Research, Volume 10, Issue 8, August-2019

[13] Neelima Kuderu, Dr. Vijaya Kumari. "Relational Database to NoSQL Conversion by Schema Migration and Mapping", International Journal

of Computer Engineering In Research Trends, Volume 3, Issue 9, September-2016, pp. 506-513
[14] Sourav Mukherjee, “The battle between NoSQL Databases and RDBMS”, University of Cumberlands Chicago, United States, 2019
[15] Mohd Muntjir, Mohd Junedul Haque, Dr.Hussain Abu Sorrah. “Inclusive Assessment of SQL Database and MongoDB Database with Latest Evaluation”, International Journal of Scientific & Engineering Research, Volume 7, Issue 3, March-2016
[16] https://github.com/ozlerhakan/mongodb-json-files/tree/master/datasets
[17] https://github.com/yugabyte/yugastore/blob/master/models/sample_data.json