Educational Administration: Theory and Practice
2024, 30(5), 1374-1383
ISSN:2148-2403
https://kuey.net/ **Research Article**

Enhancing Data Consistency Through Referential Integrity In Document Oriented NoSQL Databases

Dindoliwala Vaishali J.[1*], Morena Rustom. D.[2]

[1*]Assistant Professor, C. B. Patel Computer College & J. N. M. Patel Science College, Bharthana, Vesu, Surat, India.
vaishali_1331@yahoo.co.in
[2]Professor, Department of Computer Science, Veer Narmad South Gujarat University, Surat, India.
rdmorena@rediffmail.com

Citation: Dindoliwala Vaishali J., et al (2024), Enhancing Data Consistency Through Referential Integrity In Document Oriented NoSQL Databases, *Educational Administration: Theory and Practice*, *30*(5), 1374-1383, Doi: 10.53555/kuey.v30i5.3096

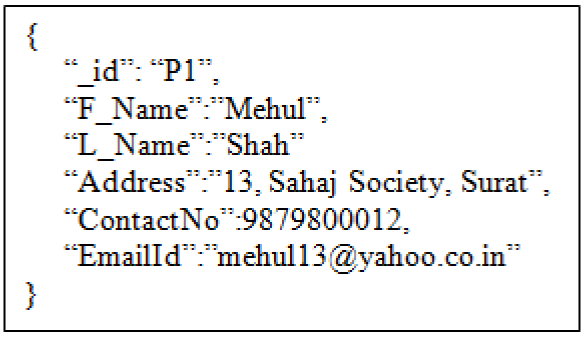
ARTICLE INFO **ABSTRACT**

The purpose of this paper is to present an efficient strategy for managing the data integrity among the documents of document-oriented NoSQL databases in distributed environment. Most of the document-oriented NoSQL databases do not provide any built-in mechanism for maintaining data integrity through establishing document relationships through managing referential integrity constraints which restrict applications to use these databases where data integrity and consistency is essential. In document-oriented NoSQL databases, generally data consistency is negotiated or it has been given not much important as relational database systems. So, we have proposed various strategies that not only manage data integrity in terms of establishing referential integrity constraints but also maintain data consistency among the documents of document-oriented NoSQL databases. For deciding the efficient strategy among the proposed strategies, various database operations are executed in a document-oriented NoSQL database and the results are obtained and analysed by measuring the execution time.

Keywords—Embedded Document, Data Integrity, NoSQL, Referencing Document, Referential Integrity Constraint

I. INTRODUCTION

The rapid growth in technology and web-based applications as well as increase in amount of data in every day, it is important to select the perfect database system which works efficiently and is able to handle large volume of data in big data environments. Relational databases are unable to handle big data and are unable to scale (Gessert et al., 2016). As a result, databases referred to as NoSQL has emerged to overcome these limitations of relational databases (Dagade et al., 2015). The important feature of NoSQL is that it does not provide rigid schema as the case with relational databases. It also provides the sharding mechanism through which horizontal scalability and availability is provided. There are various categories of NoSQL databases like key-value store databases, column store databases, document store databases and graph databases (Gessert et al., 2016; Karande, 2018; Gyorodi et al., 2015; Tiwari et al., 2017). Each of these categories has their own way of handling data in the NoSQL databases. Among these several categories of NoSQL databases, we have focused on document-oriented data model for our further research work due to its flexibility for defining database schema, scalability and high availability.
NoSQL document-oriented databases store data in form of documents. Group of documents is referred to as a collection in document-oriented NoSQL databases. The structure of a document may vary from document to document within a collection of document-oriented NoSQL databases. These documents are encapsulated and encoded in JSON format where each document contains keys and their associated values. Each document has a unique document identifier (Dindoliwala & Morena, 2019). Following is an example of a person document in document-oriented NoSQL databases.



```
{
    "_id": "P1",
    "F_Name":"Mehul",
    "L_Name":"Shah"
    "Address":"13, Sahaj Society, Surat",
    "ContactNo":9879800012,
    "EmailId":"mehul13@yahoo.co.in"
}
```

Fig. 1 Sample document in a document-oriented NoSQL database

In document-oriented NoSQL databases, generally data consistency is compromised or receives less attention. This will lead to higher performance in these databases but may result in poor data quality. Also, the applications that deal with critical transactions or where consistency among the data and sensitive data handling is required, relational databases are still preferred over NoSQL databases. So, our aim is to study how data integrity is handled in document-oriented NoSQL databases and to provide an efficient strategy that manages data integrity through managing referential integrity constraints among the documents of document-oriented NoSQL databases in a distributed environment. The rest of the paper is organized as follows: section II describes how data integrity is managed in document-oriented NoSQL databases, section III is on related work, section IV describes our proposed methodology to maintain data integrity in document-oriented NoSQL databases, section V is the discussion on experimental results and performance analysis and section VI is the conclusion.

II. DATA INTEGRITY MANAGEMENT IN DOCUMENT ORIENTED NOSQL DATABASES

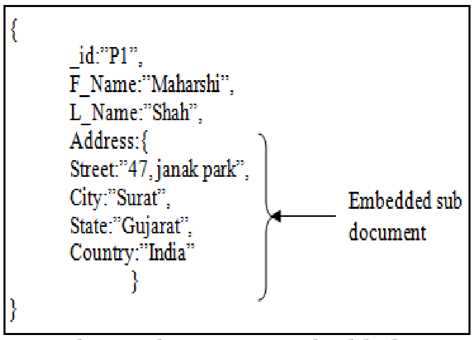
The term data integrity refers to the correctness and consistency of the data stored in a database. A good database will have built-in capabilities to impose data integrity whenever required. We have studied some popular document-oriented NoSQL databases such as MongoDB and CouchDB and describe how data integrity is maintained in such document-oriented NoSQL databases.

A. Data Integrity through Schema Validation

Document oriented NoSQL databases provide schema validation mechanisms for ensuring data integrity. A popular document-oriented NoSQL database, MongoDB, uses schema validation mechanism to provide various validations on the fields of a collection such as data type mechanism for fields, field value should be within a certain range, field value should be from the given set of values, required fields for a collection and so on. Through schema validation, one can also enforce fixed schema to a collection of a database. One can provide these schema validations while creating a collection in a database or one can also provide them on existing collections (https://www.mongodb.com/blog/post/mongodb-36-json-schema-validation-expressive-query-syntax). Thus, using schema or document validations, there is no need to handle schema validation code within the application code which simplifies the application code and minimizes errors associated with it. Also, one can control the insertion and modification of documents that has incorrect field's data type or values. CouchDB provides validation functions that check the structural constraints before inserting new documents or modifying the existing documents. These functions can be assigned to a collection. If the validation function fails then the modifications will not be accepted (Apache CouchDB Release, 2024).

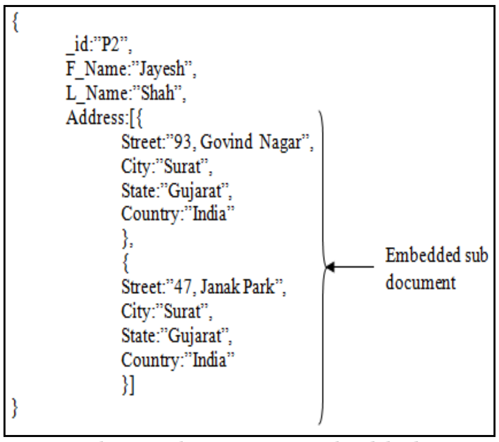
B. Data Integrity through Embedded Document Approach

In document-oriented NoSQL databases, each document of a collection has a unique document identifier termed as "_id". Embedded document approach is used to define data integrity through maintaining relationships among the documents of a collection. In this approach, documents can be embedded into another document. That means related documents will be within a single document in a collection. This generally provides a denormalized approach of storing data (Davoudian et al., 2018 &Vera et al., 2015). One can establish one to one or one to many types of relationships among the documents of a collection using embedded document approach. Fig. 2 and Fig. 3 show examples of embedded document approach to maintain one to one and one to many relationships among the documents of a collection respectively.



```
{
        _id:"P1",
        F_Name:"Maharshi",
        L_Name:"Shah",
        Address:{
        Street:"47, janak park",
        City:"Surat",
        State:"Gujarat",
        Country:"India"
                }
}
```

Fig. 2 One to One Relationship using Embedded Document Approach



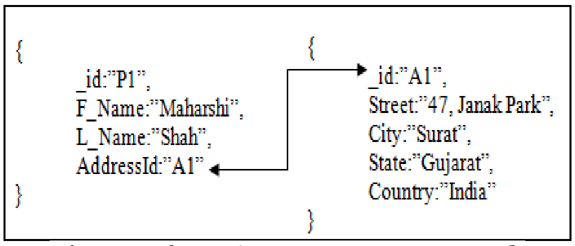
```
{
        _id:"P2",
        F_Name:"Jayesh",
        L_Name:"Shah",
        Address:[{
                Street:"93, Govind Nagar",
                City:"Surat",
                State:"Gujarat",
                Country:"India"
                },
                {
                Street:"47, Janak Park",
                City:"Surat",
                State:"Gujarat",
                Country:"India"
                }]
}
```

Fig. 3 One to Many Relationship using Embedded Document Approach

The problem associated with this approach is that it is generally used when limited amount of information is going to be stored in an embedded document. It may also happen that some of the information is going to be duplicated in embedded sub documents which results in utilizing more space. Due to duplication of information in embedded sub document, the other problem is that if some information is going to be changed then those changes should have to be reflected in all embedded documents. Other problem is that if the whole document is going to be deleted then all the information embedded within the document will also be lost.

C. Data Integrity through Referencing Document Approach

To overcome the limitations associated with embedded document approach, document-oriented NoSQL databases provide referencing document approach for establishing relationships among the documents of collections of a database. Using referencing document approach, one can save "_id" field of one document into another document which provides reference to the documents of collections. It provides normalization that maintains the relationships among the documents (https://docs.mongodb.com/manual/core/data-model-design/#data-modeling-embedding). The advantage of this approach as compared to embedded document approach is that the duplication of information can be avoided. That is the information is saved in a document and then it can be referenced in another document using that document's "_id" field (A MongoDB White Paper, 2018).



```
{                                   {
        _id:"P1",                    _id:"A1",
        F_Name:"Maharshi",           Street:"47, Janak Park",
        L_Name:"Shah",               City:"Surat",
        AddressId:"A1"               State:"Gujarat",
}                                    Country:"India"
                                    }
```

Fig. 4 Referencing Document Approach

One can establish many to one or many to many types of relationships among the documents using referencing document approach in document-oriented NoSQL databases (A MongoDB White Paper, 2018). The problem associated with this type of approach is that if the parent document is deleted then still its reference will exist in the child document which results in invalid document reference. Document oriented NoSQL databases do not provide cascade update or cascade delete mechanisms which are there in relational databases (Dindoliwala & Morena, 2018).

III. RELATED WORK

Much research work has been done in comparing the performance of NoSQL and SQL databases based on various operations performed on it. But we have not seen much work done in analysing the effect of applying data integrity by maintaining relationships among the documents of document-oriented NoSQL databases. (Georgiev, 2013) has discussed how to manage semantic relationship among the documents of collection in document-oriented databases. He had used MapReduce programming model for the implementation of his methodology. His methodology detects invalid document references which are generated due to invalid transactions performed on a database or due to program code errors. He has applied his implementation for the post-factum detection of foreign key inconsistency in the social book application to detect and remove leaking data. His methodology does not provide any suggestions about the cause for the generation of these inconsistent documents. The authors (Pokorny et al., 2017) have mainly focused on Neo4j graph database and try to represent database schema and integrity constraints on it. They have designed a language called a constraint language which follows the same principle which is used in SQL. They have implemented three types of constraints - node property uniqueness, mandatory properties and property value limitations. Node property uniqueness contains a single property value or two independent property values. Mandatory properties must be assigned for nodes to specify a node with a particular label must have assigned a value. Property value limitations specify type for a property value. They have also defined required relationships. They have developed interface for creation of explicit integrity constraints management. They have setup constraints on an existing database and enabled them and measured the time required for constraint checking. The authors (Rabuzin et al., 2016) have also implemented check integrity constraints for the graph database Neo4j. They have developed a web application which is built using Spark, a Java Web framework and Apache Tinkerpop, a graph computing framework for graph databases which includes classes and methods for performing Gremlin queries. For creating a check constraint, a user has to define constraint name, the node type, its property and the check type. This constraint will be stored in the list of database constraints. Whenever a new node is created, the methodology checks whether any constraint exists on that node and according to that appropriate action will be taken. Apart from this, no other domain constraint has been implemented. (Weintraub and Gudes, 2018) have presented a protocol that allows cloud users to verify integrity of data on the cloud databases. They have also discussed that till now there is no solution available for managing integrity of data in column-oriented NoSQL databases and their innovative approach will overcome these limitations. Their novel approach is also inspired by the relational databases' probabilistic approach. In their approach, they have used hash functions, secret keys, data authentication, data encryption and bloom filters. For performing experimental evaluation, they have implemented a prototype and used Cloud BigTable as a database and YCSB framework. They have measured the performance mainly for two types of workloads i.e., workload A and workload B. In workload A, predefined number of rows is inserted while in workload B, predefined number of rows is retrieved from the database. Apart from this, there is no other data integrity mechanism is provided. In his research work, (Raja, 2012) has designed an interface for managing referential integrity constraints for key value store databases. He has used Apache Cassandra, a cloud database management system to evaluate the performance of his API. He has proposed four solutions for storing and accessing the constraints metadata for improving the performance of the database. In all the solutions, he has measured response time and throughput for performing database operations such as insert, update and delete. He concluded that all the solutions have different trade-offs between the performance, metadata management and requirement of disk space. It is dependent on the requirement of an application to select the best solution among these four solutions. (Headley, 2017) had built a tool called as rest-hapi where entity relationships are defined as a part of model configuration and various methods are supported for insertion, updation and querying relational data. This tool supports one to one, one to many and many to many relations among the documents. These relations are created by adding necessary schema fields. He had managed many to many relationships using junction tables. But his approach takes more time to read data. His tool combines the flexibility feature of MongoDB with the relational structure.

IV. PROPOSED METHODOLOGY

In document-oriented NoSQL databases, there is no method for managing data integrity in terms of maintaining relationships among the documents of collections and it does not have any built-in mechanism for ensuring consistency of the database by maintaining data integrity in terms of referential integrity constraint. One has to verify association among the documents by writing application code even though embedded or referencing document approach is used. So, to keep the database in consistent state, there should be a mechanism of maintaining relationship among the documents through the referential integrity constraints. As being a distributed system, NoSQL database systems store, process and provide huge amount of data that may be accessed by thousands of users. Thus, being a distributed system, not only storage of data but also storage of referential integrity constraints metadata also plays an important role. We have presented three strategies for storing and accessing referential integrity constraint metadata efficiently for document-oriented NoSQL databases in a distributed environment. Each strategy has its own methods to store and access referential integrity constraints metadata. We have developed the proposed framework using C#.NET and Visual C++. This framework manages various databases of document-oriented NoSQL databases, various collections of databases, documents within the collections, referential integrity constraints among the documents of collections and database operations like create, insert, update and delete for document-oriented NoSQL databases. The proposed framework architecture is shown in below Fig. 5.

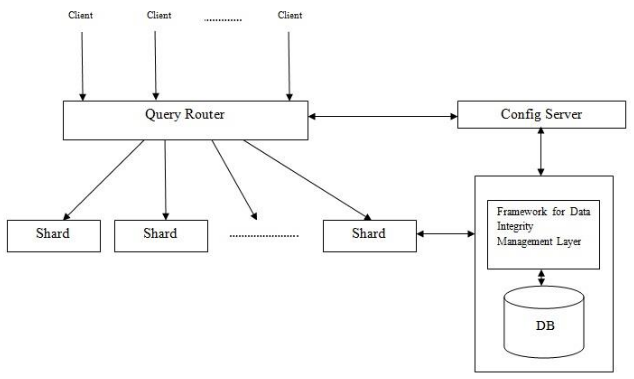


Fig. 5 Proposed Framework Architecture

The main components of this framework are shards, query router, config server and clients. The client component is responsible for generating various client requests for performing various database operations like insert, update or delete. Shards act as database servers that execute various clients' requests. The query router provides an interface through which various clients can communicate with the system for executing database operations. It will deliver the clients' requests to the shards for execution and will also act as a load balancer for delivering the clients requests to the shards. The config server stores the metadata about the shards, metadata of various databases and their collections as well as referential constraints created on collections within the ConfigDB database.
In our proposed framework, each document of a collection has a unique document identifier which is termed as "_id". This "_id" field is used to establish a referential integrity constraint among the documents of collections of a database. The referential integrity constraints can be set while creating collections or they can be set after creating collections or on existing collections of a document-oriented NoSQL database. Each referential integrity constraint metadata requires unique name of the constraint, the name of the collection in which referential constraint is going to be created (referencing collection), the name of the field in referencing collection on which referential constraint is going to be created, the name of the collection to be referenced by the referencing collection, the name of the field to be referenced by the referencing collection and cascade rule for performing cascade operations for update and delete operations.

We have proposed three strategies for storing and accessing referential integrity constraints in different manner in a distributed environment for document-oriented NoSQL databases. The aim of providing these strategies is to minimize overall execution time for executing various database operations requested by various clients in a distributed environment and to provide data consistency through managing data integrity.

In our first strategy, referential integrity constraints metadata are stored along with the actual documents within the collections. They are stored as the top document in the collections which has "_id" which is set to -1. This document stores all the referential integrity constraints of a database in the form of an array. Each of the constraints within an array has their own fields to manage referential integrity constraints. Since the referential integrity constraints metadata are stored in all the collections of a database, there will also be duplication of referential integrity constraints metadata and whenever new database constraints are created, existing constraints are updated or deleted for collections, these changes should be reflected in a document which has "_id" as -1 in all collections of a database.

In our second strategy, referential integrity constraints metadata of a database are stored in a separate collection in the ConfigDB database. So, whenever any modification takes place in metadata of referential integrity constraints, these modifications will be at one place only. Whenever database operations like insert, update or delete are going to be performed on a database collection, the referential integrity constraints metadata are retrieved and accessed from this place by the shards to validate the referential constraints for the documents of document-oriented NoSQL databases. And then based on the retrieved constraints metadata, appropriate actions will be taken by the shards.

In our third strategy, again the referential integrity constraints metadata of a database are stored in a separate collection in the ConfigDB database. But in this strategy, these referential integrity constraints metadata are cached by the query router from the ConfigDB database. The query router will then deliver these constraints metadata to the shards along with the clients' requests. This reduces the overall execution time of client request by a shard as shards do not need to retrieve the referential integrity constraints metadata.

V. EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS

We have simulated the system that provides thread safe environment and works like a document-oriented NoSQL database system that handles various clients' requests. We have also implemented a process priority mechanism which gives the real time priority to our process among the other system processes to achieve maximum CPU utilization. To measure the performance of our proposed strategies, we have performed the experimental testing for the university database that includes the collections Discipline, Course, Student, Enrolment and StudentFees and we have also set referential integrity constraints among these collections of a database. To conduct experimental testing, we have taken around 22 documents of various disciplines in the "Discipline" collection, 500 documents of various courses in the "Course" collection of various disciplines and 5000 documents for "Student", "Enrolment" and "StudentFees" collections individually. For choosing an efficient strategy among the specified strategies, we have considered the collections that have the maximum number of requests to execute. So, we have only focused on collections "Student", "Enrolment" and "StudentFees" as they have 5000 requests for each database operations. We have run several tests and calculated the average time taken in milliseconds for executing the requests for each collection when no referential integrity constraints are applied as well as for our proposed strategies and compared them.

Table I shows the average time taken in milliseconds for executing 5000 clients' requests for "Student", "Enrolment" and "StudentFees" collections for each insert, update and delete operations individually when no referential integrity constraints are applied on the collections of a database. Fig. 6 shows graphical representation of comparison of time taken to execute insert, update and delete operations of Table I.

Table I. Average Execution time in milliseconds for each operation when no referential integrity constraints are applied

	Operations		
Collections	**Insert**	**Update**	**Delete**
Student	1041.37	2588.16	1932.02
Enrolment	436.34	1025.02	811.75
StudentFees	492.46	1175.79	881.67

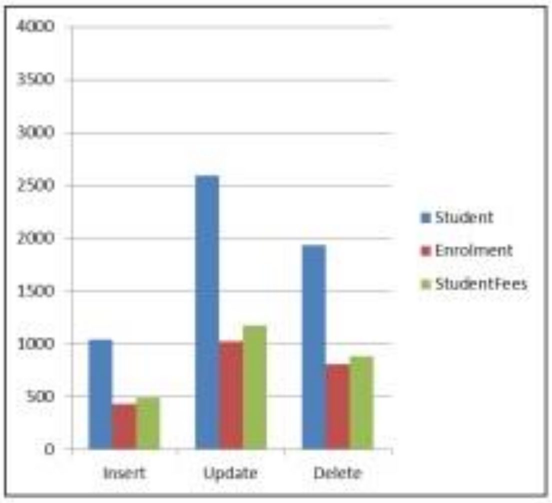


Fig. 6 Comparison of Average Execution time for insert, update and delete operations

Table II shows the average time taken in milliseconds for executing 5000 clients' requests for "Student", "Enrolment" and "StudentFees" collections for each insert, update and delete operations individually for our first strategy. Fig. 7 shows graphical representation of comparison of time taken to execute insert, update and delete operations of Table II.

Table II. Average Execution time in milliseconds for each operation for First Strategy

	Operations		
Collections	**Insert**	**Update**	**Delete**
Student	2957.49	3970.20	3223.04
Enrolment	3849.82	2703.76	1545.76
StudentFees	2313.16	3109.96	1830.30

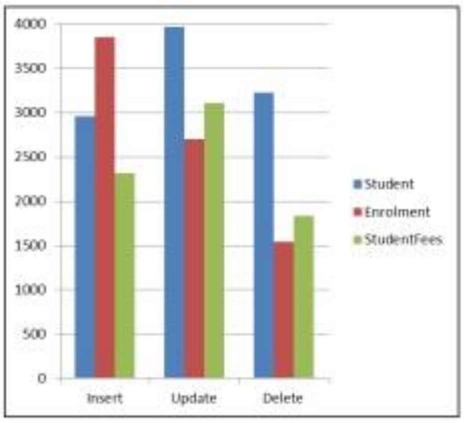


Fig. 7 Comparison of Average Execution time for insert, update and delete operations

Table III shows the average time taken in milliseconds for executing 5000 clients' requests for "Student", "Enrolment" and "StudentFees" collections for each insert, update and delete operations individually for our second strategy. Fig. 8 shows graphical representation of comparison of time taken to execute insert, update and delete operations of Table III.

Table III. Average Execution time in milliseconds for each operation for Second Strategy

	Operations		
Collections	**Insert**	**Update**	**Delete**
Student	2615.72	3451.28	3179.44
Enrolment	3790.20	2622.77	1495.10
StudentFees	2280.84	2998.23	1735.40

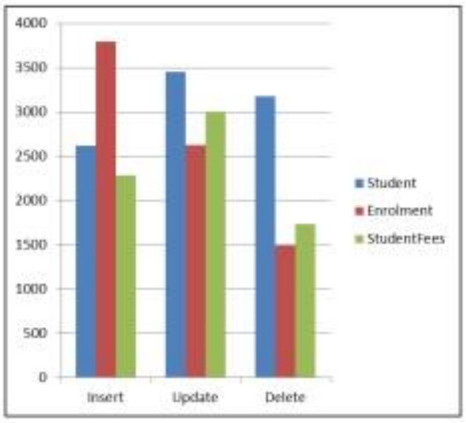


Fig. 8 Comparison of Average Execution time for insert, update and delete operations

Table IV shows the average time taken in milliseconds for executing 5000 clients' requests for "Student", "Enrolment" and "StudentFees" collections for each insert, update and delete operations individually for our third strategy. Fig. 9 shows graphical representation of comparison of time taken to execute insert, update and delete operations of Table IV.

Table IV. Average Execution time in milliseconds for each operation for Third Strategy

	Operations		
Collections	**Insert**	**Update**	**Delete**
Student	1464.31	2903.01	2602.79
Enrolment	3310.29	1740.71	960.76
StudentFees	1672.59	2144.44	1180.44

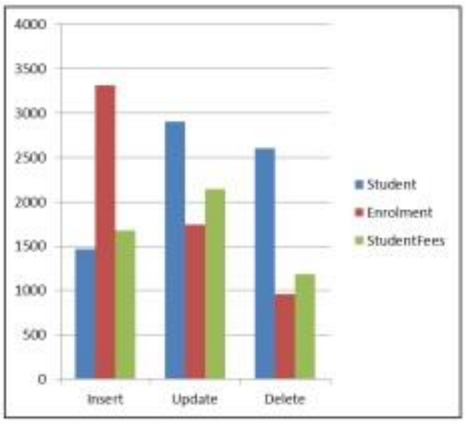


Fig. 9 Comparison of Average Execution time for insert, update and delete operations

From the above results, it is clear that the time for executing various database operations is least in the case when no referential integrity constraints are applied. But when referential integrity constraints are applied, the execution time will increase. So, to minimize this execution time, we have proposed three strategies. From Table II, Table III and Table IV, it is clear that the execution time for "Enrolment" collection is maximum as compared to other two collections because two referential integrity constraints are defined within it and before insertion, it has to check two referential integrity constraints. Also, the updation and deletion time for "Student" collection is maximum as compared to other two collections because before performing updation or deletion of a document within a "Student" collection, it has to check whether any dependency of "_id" of a "Student" document exists in other documents of some other collection and if dependency exists then according to the set value of cascade rule, appropriate action will be taken.

Using results of Table I, Table II, Table III and Table IV, we have made comparison of average execution time required in milliseconds for 5000 insert, update and delete operations respectively for a database which is shown in Table V and its graphical representation is shown in Fig. 10.

Table V. Comparison of Average Execution time in milliseconds for various Strategies for each operation

	Operations			
Strategies	**Insert**	**Update**	**Delete**	**Average Time**
Without Referential Integrity Constraints	656.72	1596.32	1208.48	1153.84
Strategy 1	3040.16	3261.30	2199.70	2833.72
Strategy 2	2895.59	3024.09	2136.65	2685.44
Strategy 3	2149.06	2262.72	1581.33	1997.71

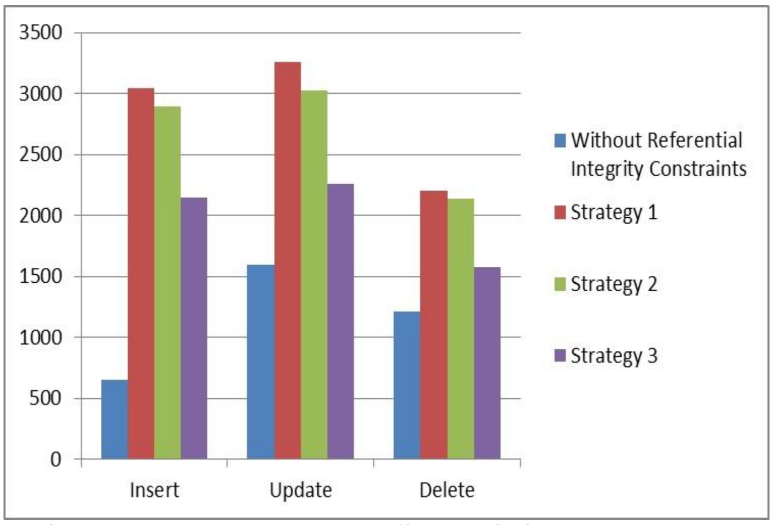


Fig. 10 Comparison of average execution time in milliseconds for various Strategies for each operation

From Table V, it is clear that the third strategy has the lowest average execution time as compared to other two strategies for executing database operations such as insert, update and delete when referential integrity constraints are applied. Using Table V, we have also calculated the amount of percentage gain in performance of insert, update and delete operations for the third strategy in comparison with first and second strategies which is shown in Table VI.

Table VI. Performance Gain of Strategy 3 in comparison with Strategy 1 and Strategy 2

	Performance Gain	
Operations	**% gain w.r.t Strategy 1**	**% gain w.r.t Strategy 2**
Insert	29.31%	25.78%
Update	30.61%	25.17%
Delete	28.11%	25.99%

From the above table, we can say that the third strategy is an efficient strategy among the three proposed strategies for maintaining data integrity by means of applying referential integrity constraints in the collections of document-oriented NoSQL databases. The third strategy lowers the extra overhead of processing referential integrity constraints as compared to our first and second strategies because the query router component caches the metadata of referential integrity constraints. Shards do not require connecting to the ConfigDB database each time to execute client request which lowers the overall execution time.

VI. CONCLUSION

The document-oriented NoSQL databases provide flexible database schema, horizontal scalability, high availability and high performance. But still the integrity of data is in question for these types of databases and this is the reason that still the applications are using relational databases where data integrity and consistency is more important. In most of the document-oriented NoSQL databases, the maintained by writing the application code only that may create inconsistency in the data. So, to avoid this inconsistency among the data, we have proposed three strategies to ensure integrity of data through the implementation of referential integrity constraints mechanism. Among these specified strategies, the strategy that caches referential integrity constraints metadata from the ConfigDB works faster than the other two strategies because the shards do not require to connect with the ConfigDB each and every time for accessing referential integrity constraints metadata for executing each client request which saves the time and improves the performance of the database operations such as insert, update and delete. Using this methodology, the big data quality can be improved because it checks referential integrity among the data before executing each insert, update and delete operations due to which such type of databases can be used in the applications like banking systems, accounting systems etc. where quality of methodology, one can still have the benefit of flexibility of schema and high availability of data with improved consistency for the document-oriented NoSQL databases.

REFERENCES

1. Gessert, F., Wingerath, W., Friedtich, S., & Ritter, N. (2016, November). NoSQL Database Systems: A Survey and Decision Guidance, Springer Link, 32, 353-365.
2. Dagade, V., Lagali, M., Avadhani, S., & Kalekar, P. (2015). NoSQL Databases – A Survey, International Journal of Emerging Technology in Computer Science & Electronics, 14 (2), 897-901.
3. Karande, N. (2018, February). A Survey Paper on NoSQL Databases: Key-Value Data Stores and Document Stores, International Journal of Research in Advent Technology, 6 (2), 153-157.
4. Gyorodi, C., Gyorodi, R., Pecherle, G., & Olah, A. (2015, June). A Comparative Study: MongoDB vs. MySQL, The 13th International Conference on Engineering of Modern Electric Systems, Oradea.
5. Tiwari, S., Akkalakshmi, M., & Bhagavatula, K. (2017, April). Analysis of NoSQL Databases: MongoDB, HBase, Neo4J, International Journal of Engineering and Technology, Special Issue, 234-239.
6. Dindoliwala, V. J., & Morena, R. D. (2019, January). A Survey on NoSQL Document Store Databases – MongoDB, CouchDB and RavenDB, National Conference on Emerging Technologies in IT, 122-131.
7. Apache CouchDB Release 3.3.3 (2024), User Guides. https://docs.couchdb.org/_/downloads/en/latest/pdf/
8. Davoudian, A., Chen, L., & Liu, M. (2018, April). A Survey on NoSQL Stores, ACM Computing Surveys, 51 (2), Article 40, 40-43.
9. Vera, H., Boaventura, W., Holanda, M., Guimaraes, V., & Hondo, F. (2015). Data Modeling for NoSQL Document-Oriented Databases, SIMBig, 129-135.
10. RDBMS to MongoDB Migration Guide, Considerations and Best practices. (2016, June). A MongoDB White Paper.
11. Dindoliwala, V. J., & Morena, R. D. (2018), Comparative Study of Integrity Constraints, Storage and Profile Management of Relational and Non-Relational Database using MongoDB and Oracle, International Journal of Computer Sciences and Engineering, 6 (7), 831-837.
12. Georgiev, K. (2013, January). Referential Integrity and Dependencies between Documents in a Document Oriented Database, GSTF Journal on Computing (JoC), 2 (4), 24-28.
13. Pokorny, J., Valenta, M., & Kovacic, J. (2017). Integrity constraints in graph databases, The 7th International Symposium on Frontiers in Ambient and Mobile Systems, 975 -981.

14. Rabuzin, K., Konecki, M., & Sestak, M. (2016, October). Implementing Check Integrity Constraint in Graph Databases, Proceedings of 82nd the IIER International Conference, Berlin, Germany, 19-22.
15. Weintraub, G., & Gudes, E. (2018, July). Data Integrity Verification in Column-Oriented NoSQL Databases, IFIP Annual Conference on Data and Applications Security and Privacy, Bergamo, Italy, 165-181.
16. Raja, H. (2012). Referential Integrity in Cloud NoSQL Databases, Master's Thesis, Victoria University of Wellington.
17. Headley, J. (2017, February). The Problem with MongoDB. https://hackernoon.com/the-problem-with-mongodb-d255e897b4b
18. https://docs.mongodb.com/manual/core/data-model-design/#data-modeling-embedding
19. https://www.mongodb.com/blog/post/mongodb-36-json-schema-validation-expressive-query-syntax