A Conceptual Model of Semi−structured Data for Big Data Applications

SHADY HAMOUDA
Liwa College, Abu Dhabi, UNITED ARAB EMIRATES

Abstract: The vast influx of data has posed significant challenges for major corporations. This deluge has sparked numerous issues within big data, prompting a need for research and industrial solutions. Among the crucial challenges is the requirement for semi-structured data storage and handling to manage large volumes of data with adaptable schemas. Recently, researchers have proposed features and specifications for designing schemas tailored to semi-structured data formats. However, these proposals have not undergone evaluation based on semi-structured properties. Consequently, this study assesses these features within the context of semi-structured properties for big data applications. The findings indicate that semi-structured properties partially or fully support some of these features.

Key-words: Conceptual Model, Semi-Structured Data, Schema, Document-Oriented Data, Big Data.

Received: March 13, 2024. Revised: August 6, 2024. Accepted: September 11, 2024. Published: November 1, 2024.

1. Introduction

According to Stanescu, et al. [1], the handling of the increasing volume of data and storing and analyzing this data are pressing concerns in big data. Semi-structured data has been widely used in areas such as data integration, data distribution, data warehousing, management, information retrieval, and knowledge management for large amounts of semi-structured data on the web [2].

Assunção, et al. [3] identified another important issue related to the method of handling and processing semi-structured data with a flexible schema. Semi-structured data are required to store and handle large amounts of data with flexible schemas, which have emerged as one of the biggest data models for handling large amounts of data. Information technology in big organizations is trying to shift from structured data to semi-structured data.

Currently, a Relational Database Management System (RDBMS) is inefficient in handling applications and software requirements of big data, such as supporting horizontal scaling for a distributed environment and the inability to achieve effective data [4, 5]. Therefore, many organizations are looking forward to the next generation of data management to support their business applications" is almost identical to the following: "Nowadays, big organizations are looking forward to NoSQL databases as the next generation of data management to support their business applications" [6, 7]

These issues and challenges have led to the development of Not only SQL (NoSQL) databases as a new technology to overcome the limitations of the relational database, such as designing a schema without strict constraints [8, 9]. In addition, NoSQL databases can accept all types of structured, semi-structured, and unstructured data and have many features, such as support for distributed systems, flexible schema, horizontal scalability, and easy replication [10, 11].

Nowadays, big organizations are looking forward to NoSQL databases as the next generation of data management to support their business applications [6]. This is due to the exponential increase in the amount of data and the need for a flexible schema with semi-structured data.

One of the most powerful types of NoSQL databases is the document-oriented database, which supports a flexible schema that used to store, retrieve, and manage data using a semi-structured data format, as well as provide high performance, availability, and scalability [1]. Also, the document-oriented database can define a field into a document, and the application can query this document by using the fields. In addition, document-oriented databases can create an index over fields, and these indexes can help optimize queries used as a reference to the fields [12]. This capability means that a document-oriented database can be more suitable than other NoSQL databases for storing a large amount of data that needs to be retrieved based on more complex criteria [13].

In the current era, with the advent of new applications and the necessity of integrating diverse data types into big data applications, structured data faces numerous constraints and regulations. These challenges stem from the need to store and process various data types within a single application, as well as the demand for developing applications capable of managing large volumes of data with varied uses.

Currently, document-oriented databases are becoming increasingly popular due to their scalability, availability, and performance. These databases store data in semi-structured data formats such as XML, JSON, and BSON, which do not require a formal structure. However, designing a schema for a document-oriented database can be challenging due to the lack of specifications for conceptual models.

2. A Semi-structured Data Model

A semi-structured data is a form of structured data that can deal with any data type without a formal structure. This data format can store data in XML, JSON, and BSON. The following are models that have been proposed to handle semi-structured data.

2.1 ERX

ER for XML (ERX) is a conceptual model used to overcome complex XML processors that can be adopted to visually describe XML document structures from ER models [14, 15]. ERX has many major drawbacks in representing the properties of semi-structured data, such as the irregular structure of data, mixed content, abstraction, and heterogeneity.

2.2 ORA-SS

The object relationship attribute model for semi-structured data (ORA-SS) was designed to provide the appropriate conditions for nested relationships with semantic data [14]. ORA-SS is a semantically rich data model used for semi-structured data and has four basic concepts: object classes, relationship types, attributes, and references. It consists of four diagrams: schema, instance, functional dependency, and inheritance diagrams. However, ORA-SS schema diagrams, as traditional databases, may contain redundancies and suffer from abnormal updating [16]. In addition, ORA-SS does not directly support the representation of non-hierarchical relationships and mixed content in a conceptual-level semi-structured data model [17].

2.3 XER

The extensible ER (XER) model is used to describe an XML document structure through conceptual models. It can automatically generate XML document definitions and schemas from the ER model [14]. However, XER does not support most semi-structured semantic properties.

2.4 EREX

Ganguly and Sarkar [14] defined EReX as a conceptual model extended to XML, which is added to the ER model to provide different features of XML.

- Each entity types are formed according to the type of relationship.
- The roles of entity types in relationship types are specified and categorized by the constraints.
- The constraint for the participants in a relationship type is specified by a thick solid line.

EReX does not have cardinality and an array relationship and does not explicitly support content with the separation of structure.

2.5 XUML

XUML aims to support the design of XML documents and XML databases and the information integration of XML [18]. It does not cover the properties of self-evolution, abstraction, cardinality, and explicit separation of structure and content.

2.6 XSEM

The idea of XSEM is to divide the XML conceptual modeling process into two parts. The first part consists of designing an overall conceptual schema using an extension of the classical ER model. The second part involves designing a hierarchical organization of the structures from the first part using a hierarchical model [19]. XSEM has drawbacks in representing some of the most important properties of semi-structured data, such as inheritance, heterogeneity, and array relationship.

2.7 GOOSSDM

The concept of the graph object-oriented semi-structured data model (GOOSSDM) is to use object-oriented data models to represent a semantic graph. It supports hierarchical and non-hierarchical relationships and provides an abstraction conceptual model with graphical constructs to the designer and the user.

2.8 GN-DTD

Graphical notations data type documentation (GN-DTD) proposes to arrange the content of XML documents to provide an improved understanding of DTD structures and enhance the XML design and normalization process. GN-DTD was developed to represent and support XML structures and capture the semantics of XML documents [20].

Numerous properties of conceptual models for semi-structured data need to be addressed to handle semi-structured data. These properties can include lenient structure, lenient participation/instances, hierarchical and non-hierarchical structures, ordering, irregular structure of data, disjunction, self-evolution, mixed content, abstraction, explicit separation of structure and content, partial relationship/participation, heterogeneity, *n*-array relationship, inheritance, reuse potential, constraints, functional dependencies, and symmetric and recursive relationships [14].

The study of Hamouda,2019; proposal specifications and features for the document-oriented database, which is the basic concept to design a schema. In conclusion, Various models have been proposed to handle semi-structured data, but there is a need to evaluate their compatibility with document-oriented databases.

This study discusses the data format and modelling of semi-structured data to evaluate the semantic properties of the conceptual model with semi-structured modelling. The result shows that none of the existing semi-structured models can handle a large volume of data with a flexible schema. Therefore, this study evaluates these specifications and features of a semi-structured data format based on the proposed schema for a document-oriented database to evaluate the semantic properties of the conceptual model with semi-structured modelling. The result of this study has been shown which feature can be partial or full supported by the proposed schema.

3. Design a Schema for Semi-structure Data (Semischema)

This study proposes a new schema named "Schema for Semi-Structured Data" (SemiSchema). The SemiSchema consists of specifications and features that provide new features and specifications to improve the concepts and schema of semi-structured data presented in previous studies, such as Arora and Aggarwal [21], Atzeni, et al. [22], Hashem and Ranc [23], Bhogal and Choksi [24]. For instance, SemiSchema provides new features for semi-structured data, such as flexible schema and time stamps, which are not covered by previous semi-structured data models.

Ferro, et al. [25] explained that the relationships of document--oriented database can be represented by embedded or reference documents. Embedded documents store related data in a single document; this denormalized data allows systems to query and update related data in a single database operation. The embedded model is used when a one-to-many relationship exists between entities; child documents always appear within the parent document. Embedding delivers good performance for read operations, but documents may grow after creation, thereby possibly affecting write performance.

In contrast, the reference document stores the relationships between data by providing links from one document to another and is normalized data. The reference document is used when numerous many-to-many relationships exist. A reference document can provide more flexibility than an embedded document.

3.1 Semischema Components

Previous studies have highlighted the properties of semi-structured data in general without addressing the representation of each component needed to design a document-oriented data schema. The schema not only includes collections, documents, and key values, but also a significant amount of business logic implemented in the form of stored procedures and functions. This section outlines the main components of SemiSchema as follows:

i. a collection representing the database table,
ii. a document representing the table record,
iii. key-value pairs, as a key represents the attribute and value represents the data type of this attribute according to the type of value,
iv. the array data type can be used to represent multiple values or many documents.

v. embedded and reference documents representing the relationship types between the collection.

3.2 Semischema Specifications

The SemiSchema specifications represent notations for a semi-structure data schema based on the ER schema as follows:

Table 1: SemiSchema Specifications

SEMISCHEMA Components	SEMISCHEMA Specification	Description and condition
Key-value	{K1,…..,Ki}	The attributes of each collection are described by using K, which represents the name of the attribute. These attributes are listed inside two brackets with a comma between them.
	$\underline{K}$	The primary key is represented by an underline of the attribute.
	$\underline{K}$	The foreign key is represented by a dashed line of the attribute.
	K@	The unary relationship is described by adding @ after the attribute K and the name of the relationship.
	K#	The derived attribute is described by a hash after the attribute K.
Key-value with array data type	MV[K1,…,Ki]	The multi-valued attribute is described by the name of the attribute with an array data type, while Ki represents all the multi-values.
Collection	CA{ K1, …,Ki }	CA: the name of the composite attribute. Ki : a set of key names listed in the document with a comma between them. i : number of composite attributes

Embedded document in the collection	Collection_name{
	K1, . .
	Embedded document {
	K1,…..,Ki},
	Kn
	}
		Creates a document for all main collections with all the keys of this collection. Identifies a primary key for main collection levels.
		- The embedded document is stored in the related document.
		- Adds the embedded document with its key values as an array of embedded documents.
		- Adds a primary key for each embedded document.
		collection{k1.ki,embedded document[{k1..kj}]}

Embedded document and Reference document	Collection1{
	K1,
	Collection2
	[{kp,,,ki},,,{kp..ki}] }
	Collection2{
	K1;
	Collection1
	[{kp,,,ki},,,{kp..ki}]
	}
		This type of relationship is identified by creating two collections named Collection 1 and Collection 2, then storing the primary key of the related document of Collection 1 inside Collection 2, and vice versa. If the document of Collection 1 is stored inside Collection 2, then the name of the document root becomes Collection 1.
		Kp: primary key.
		Ki: number of keys.
		Collection name [{K1,…. Ki},…..,{K1,…. Ki}]

3.3 Semischema Features

This study considers two dimensions of big data: volume and semi-structured data. The contribution of this study is that it addresses issues of big data, such as data volume, and represents a semi-structured data schema.

This study highlights that JSON is more compact than XML for semi-structured data and plays an important role in representing semi-structured data models. However, there is a lack of research on how to represent a conceptual model for semi-structured data using the JSON format. To overcome this problem, the SemiSchema applies the existing features of semi-structured models and adds two new features required to support a big data application: a flexible schema and a timestamp. A flexible schema is needed to respond to changing business requirements and to yield dynamic data that can adapt to the demands of scalability, high availability, and storage of immense amounts of data.

Additionally, the timestamp feature is required for semi-structured data. Real-time applications need a way to evolve with time, and the timestamp data type is better and more efficient than the date-time data type. These features can address the issue of how to handle and process semi-structured data with a flexible schema, as presented by (Assunção, Calheiros et al., 2015). Additionally, these features can resolve challenges faced by current big organizations, such as how to handle increasing data volumes with flexible data schemas (Kanade et al., 2014; Madison et al., 2015).

The semi-structure features that are applied in a SemiSchema can be described as follows:

i. No strict structure: A SemiSchema has no strict formatting, which means that similar document attributes in each document are not important, and new attributes can be added at any time without any strict structure.
ii. No strict participation/instance: In a SemiSchema, each key may correspond to any kind of data type, and the same key may have different values. Therefore, no strict participation for a SemiSchema exists.
iii. Hierarchical structure: A SemiSchema can support tree data structures of hierarchical relationships. The concept of embedding a document into a collection considers the hierarchical structure between two collections, and each collection may have many embedded documents.
iv. Non-hierarchical structure: The main collection (parent) stores each tree node in a document, and each tree node stores the ID of the main document. Thus, each collection may have many references to other collections with no hierarchical structure.
v. Ordering: The concept of the SemiSchema is a flexible schema; the order of key-value in the document will depend on how to insert the key value into the document. Therefore, this feature is supported by the SemiSchema and can be implemented through the application.
vi. The irregular structure of data: The SemiSchema stores data in a semi-structured data format, and thus stores data as a key–value. The values of this key can be any data type, and the key-value pairs are stored within documents. As each document has a flexible schema, it thus allows the storage of data in an irregular structure.
vii. Disjunction: The SemiSchema is less homogeneous, and this disjunction is represented by an embedded document, the main document ID associated with the embedded document ID, and the embedded ID that can access all the embedded fields.
viii. Self-evolving: Self-evolving considers the main concept of the SemiSchema, as each document has a self-description that allows each key to be self-described.
ix. Mixed content: The SemiSchema allows each key to have different content in each document, and this content can be blended into the same document without a structure. The concept of key-value entails accepting any kind of data without any constraint or definition of the type of key because the constraint of value will be responsible for the application.
x. Abstraction: The SemiSchema hides the complexity of data. The key can be accessed without any details of the value type as the content in each key is not important,
xi. The explicit separation of structure and content: The logical structure of documents is represented in a separate hierarchy by considering the content of the key-value series.
xii. Partial relationship/participation: In the schema of a document, a relationship can be represented by using references and embedded documents that can identify the parent and child of each document. The main collection includes the parent and the embedded documents, which are considered the child. The reference model between collections represents the participation between the parent and child.
xiii. *N*-array relationship: In many-to-many SemiSchema relationships and multi-value attribute representation in array values; this feature makes JSON better than other formats used in the document-oriented database.
xiv. Inheritance: The SemiSchema supports inheritance through the embedded document, as the main document can describe the common properties, and the embedded document can be represented by the sub-properties of the main document data.
xv. Reuse potential: The reference linking of the relationship between two documents describes their reuse potential.

xvi. Constraints: One of the advantages of the SemiSchema is its flexible schema, such that dealing with different data types or with each document may have different fields depending on the system. Moreover, if any constraint needs to be applied, it will be implemented by a programming application.

xvii. Cardinality: The SemiSchema supports cardinality features, like the relational database, which retains relationships between the tables through embedded and reference approaches.

xviii. Functional dependencies: The data are organized into the concept of the key- value. Each key is used to store the value, which can be determined by the key. Each document also has a primary key used to identify all document keys. Therefore, all document fields are functionally dependent on the primary key document. The SemiSchema supports the concept of functional dependencies.

xix. Symmetric relationship: Each value in the document relates to the key, and each key symmetrically relates to the value.

xx. Recursive relationship: This feature describes the relationships between the documents. A unary relationship is stored in the related collection, which is described as a recursive relationship.

xxi. Flexible schema: This new feature is required for big data not covered by any semi-structured model or relational database, as well as for new business requirements and changes. In the SemiSchema, the schema can be changed at any time. The SemiSchema allows the addition of any field in any document without constraint and permits each document to have different numbers of fields. It also enables the changing of relationships before or after implementation.

xxii. Timestamp: This feature is required for semi-structured data. Real-time applications need a way to evolve with time. The time-stamp data type is better and more efficient than the date-time data type. The SemiSchema provides time-stamp data for each document that it supports.

A document-oriented database is designed for storing, retrieving, and managing document-oriented or semi-structured data. The central concept of a document-oriented database is the notion of a document. Where the contents within the document are encapsulated or encoded in a standard format such as JSON or BSON, And XML.

4. Case Study: W3school Schema

The case study is the schema of the W3school website (http://www.w3schools.com/). The W3school schema, presented in Figure 1, can be described as the `PRODUCT` has `CATEGORIES` and `SUPPLIERS`, and `ORDER` has `ORDERDETAILS` and `SHIPPERS` to the `CUSTOMER` through the `EMPLOYEES`.

The SemiSchema were applied to the schema shown in Figure 1, and the output of this schema, presented in Figure 2, is as follows:

i) Created new collections for the main strong entity, which are `PRODUCT`, `ORDER`, and `EMPLOYEES`.

ii) Mapped the relationship between `PRODUCT` and `CATEGORY` by store `CATEGORIES` as embedded documents in the `PRODUCT` collection. Also, mapped the relationship between `PRODUCT` and `SUPPLIERS` by store `SUPPLIERS` as embedded documents in the `PRODUCT` collection

iii) Mapped the relationship between `ORDER` and `ORDERDETAILS` by creating an embedded document for `ORDERDETAILS` in the `ORDER` collection. Also, the relationship between `ORDER` and `SHIPPERS` was mapped to create embedded documents for `SHIPPERS` in the `ORDER` collection.

iv) Mapped the relationship between `ORDERS` and `CUSTOMERS` by creating an embedded document for `CUSTOMERS` in the `ORDERS` collection.

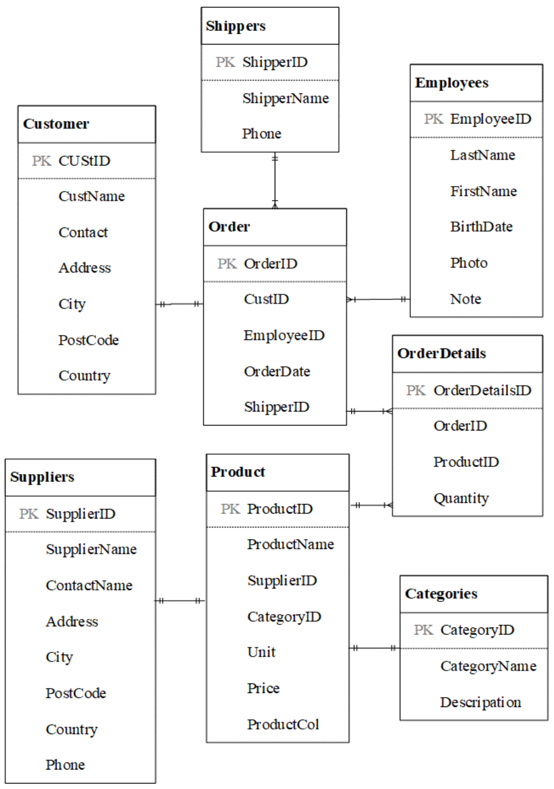

Figure 1: ER schema for W3schools [26].

```
Product{productId,
productname,unite,price,productsCol,
Category{CategoryID,CategoryName,Des
cripation},
Suppliers{SupplierID,SupplierName,Co
ntactName,Address,City,PostCode,Coun
try,Phone}}

Order{OrderID,EmployeeID,ProductID,O
rderID,
OrderDetails{OrderDetailsID,ProductI
D,Quantity},   ShipperID,ShipperName,
Phone},
Customer{CustomerID,CustomerName,Con
tactName,Address,City,PostCode,Count
ry},
Employee{EmployeeID,LastName,FirstNa
me, BirthDay,Photo,Notes} }
```

Figure 2: The SEMISCHEMA for W3schools.

As Figure 2 shows, the ER schema first depicted in Figure 1 has been mapped in it entirety without missing any specification. The SemiSchema of Figure 2 contains three collections `PRODUCT`, `ORDER`, and `EMPLOYEE`, as the `PRODUCT` collection contains `CATEGORY` and `SUPPLIERS` entities as embedded documents based on the TRs. Also, the `ORDER` collection contains `ORDERDETAILS`, `SHIPPERS`, and `CUSTOMERS` entities as an embedded document based on the SemiSchema. The evaluation of this case study assessed the flexibility of the schema by checking whether the features of the SemiSchema were compatible with any business requirement changes.

5. Evaluate the Flexibility of the Schema

The flexibility of the schema refers to the evaluation of the capability to meet and respond to business change requirements during the development process [27]. Accordingly, this evaluation can enable organizations to achieve flexibility in software development for managing unpredictable and changing conditions [28].

This evaluation tested the flexibility of the SemiSchema using the W3schools schema in Figure 2 in order to determine whether it could keep up with new business requirements. In a relational database, the new field should add to all table records to change its schema. By contrast, a SemiSchema allows the addition or alteration of the data for a specific document in any structure without changing the database schema. A SemiSchema thereby permits the application to use the required data while ignoring the unrequired data.

In the W3schools schema (Figure 1), the relationship between `PRODUCTS` and `SUPPLIES` is one to one. In this case, the W3schools schema needs to change the business requirements by allowing one `PRODUCT` to have many `SUPPLIES` or each `SUPPLIER` to provide many `PRODUCTS`; however, this new requirement creates difficulty in changing the relational database schema. To incorporate this requirement in the relational database, a new table should be added to allow the `PRODUCT` to have many `SUPPLIERS`. The relational database schema will not allow the same `PRODUCT` to have many `SUPPLIERS` because it is fixed and has change constraints.

The previous scenario indicates that the change requirements will affect the database schema, query level, and reporting level. Given that the relational database needs to change, all queries related to `PRODUCT` and `SUPPLIER` need to be redesigned and recoded. By contrast, a SemiSchema supports a flexible schema with semi-structured data that can add or change relationships between entities without adverse effects.

In the previous case in Figure 2, the relationships between `PRODUCTS` and `SUPPLIERS` can be changed without affecting the schema because the `PRODUCT` collection stores the `SUPPLIERS` as embedded documents and lists `SUPPLIERS` for each product. In a SemiSchema, mapping one-to-one or one-to-many relationships can occur via the embedded documents.

Therefore, this schema can store many `SUPPLIES` as embedded documents into each `PRODUCT`, as shown in Figure 2. In addition, the second evaluation assessed the flexible schema feature and data based on the semi-structure features. Ultimately, the issue confronting the current application is how fast development processes can respond to meet users' changed requirements. The flexibility of the schema is presented to integrate the changes in business requirements with the existing code after the completion of the changes.

The flexible schema feature may be suitable for handling the large volume and variety of data of modern applications. It also allows the dynamic modification of the schema without interruption, as well as simplified application design and a reduction in the overall efforts needed to develop applications.

6. Conclusion

This study addresses issues related to the specifications and features of semi-structured data that are not presented in previous models. These specifications and features can address most of the features of semi-structured data models while adding two new features – a flexible schema and a timestamp – that are needed for big data applications. To define a schema for semi-structured data, the Schema for Semi-Structured Data components, specifications, and features can be used as a conceptual model to design a schema for semi-structured data.

The schema will store data as a key-value concept by regarding the key as a field and the value as a field. These values can be of any data type, such as string, integer, array, and nested document. This property ensures flexibility and scalability in organizing and handling different types of data. In future work, this study can be extended to evaluate the schema of semi-structured data for a real-time application and assess the features of semi-structured data.

References

[1] L. Stanescu, M. Brezovan, and D. D. Burdescu, "Automatic mapping of MySQL databases to NoSQL MongoDB," in *Computer Science and Information Systems (FedCSIS), 2016 Federated Conference on*, 2016, pp. 837-840: IEEE.

[2] L. Zhang, N. Li, and Z. Li, "An overview on supervised semi-structured data classification," in *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*, 2021, pp. 1-10: IEEE.

[3] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. Netto, and R. Buyya, "Big Data computing and clouds: Trends and future directions," *Journal of Parallel and Distributed Computing,* vol. 79, pp. 3-15, 2015.

[4] C. A. Győrödi, D. V. Dumşe-Burescu, D. R. Zmaranda, R. Ş. Győrödi, G. A. Gabor, and G. D. J. A. S. Pecherle, "Performance analysis of NoSQL and relational databases with CouchDB and MySQL for application's data storage," vol. 10, no. 23, p. 8524, 2020.

[5] M. Dahiya, S. Sharma, and S. Grima, "Big Data Analytics Application in the Indian Insurance Sector," in *Big Data Analytics in the Insurance Market*: Emerald Publishing Limited, 2022, pp. 145-164.

[6] W. Qi, M. Sun, S. R. A. J. J. o. M. Hosseini, and Organization, "Facilitating big-data management in modern business and organizations using cloud computing: a comprehensive study," pp. 1-27, 2022.

[7] N. Deepa *et al.*, "A survey on blockchain for big data: approaches, opportunities, and future directions," 2022.

[8] A. Erraji, A. Maizate, M. Ouzzif, and Z. I. J. E. T. BATOUTA, "Migrating Data Semantic from Relational Database System To NOSQL Systems to Improve Data Quality for Big Data Analytics System," vol. 107, no. 1, p. 19495, 2022.

[9] M. A. Fouly, T. H. A. Soliman, and A. I. Taloba, "Developing an Efficient Secure Query Processing Algorithm for Unstructured

Data on Encrypted Databases," in *2022 10th International Japan-Africa Conference on Electronics, Communications, and Computations (JAC-ECC)*, 2022, pp. 252-257: IEEE.

[10] J. Chaudhary, V. Vyas, and C. Jha, "Qualitative Analysis of SQL and NoSQL Database with an Emphasis on Performance," in *IOT with Smart Systems: Proceedings of ICTIS 2022, Volume 2*: Springer, 2022, pp. 155-165.

[11] F. Tesone, P. Thomas, L. Marrero, V. Olsowy, and P. Pesado, "A Comparison of DBMSs for Mobile Devices," in *Computer Science–CACIC 2021: 27th Argentine Congress, CACIC 2021, Salta, Argentina, October 4–8, 2021, Revised Selected Papers*, 2022, pp. 201-215: Springer.

[12] H. Hashem and D. Ranc, "Evaluating NoSQL document oriented data model," in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, 2016, pp. 51-56: IEEE.

[13] A. Corbellini, C. Mateos, A. Zunino, D. Godoy, and S. Schiaffino, "Persisting big-data: The NoSQL landscape," *Information Systems,* vol. 63, pp. 1-23, 2017.

[14] R. Ganguly and A. Sarkar, "Evaluations of Conceptual Models for Semi-structured Database System," *International Journal of Computer Applications,* vol. 50, no. 18, 2012.

[15] A. Sengupta, S. Mohan, and R. Doshi, "XER-extensible entity relationship modeling," in *Proceedings of the XML 2003 Conference*, 2003, pp. 140-154.

[16] A. Qtaish, M. T. J. J. o. I. Alshammari, and K. Management, "A narrative review of storing and querying xml documents using relational database," vol. 18, no. 04, p. 1950048, 2019.

[17] S. Hamouda, R. Sughayyar, and O. E. Elejla, "Semi-Structured Schema for a Big Data (S-SSBD)," in *KEOD*, 2021, pp. 202-209.

[18] H. Liu, Y. Lu, and Q. Yang, "XML conceptual modeling with XUML," in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 973-976: ACM.

[19] M. Necasky, "XSEM: a conceptual model for XML," in *Proceedings of the fourth Asia-Pacific conference on Comceptual modelling-Volume 67*, 2007, pp. 37-48: Australian Computer Society, Inc.

[20] Z. Zainol and B. Wang, "GN-DTD: Graphical notations for describing XML documents," in *Advances in Databases Knowledge and Data Applications (DBKDA), 2010 Second International Conference on*, 2010, pp. 214-221: IEEE.

[21] R. Arora and R. R. Aggarwal, "Modeling and querying data in mongodb," *International Journal of Scientific and Engineering Research,* vol. 4, no. 7, 2013.

[22] P. Atzeni, F. Bugiotti, L. Cabibbo, and R. Torlone, "Data modeling in the NoSQL world," *Computer Standards & Interfaces,* 2016.

[23] H. Hashem and D. Ranc, "Evaluating NoSQL document oriented data model," in *Future Internet of Things and Cloud Workshops (FiCloudW), IEEE International Conference on*, 2016, pp. 51-56: IEEE.

[24] J. Bhogal and I. Choksi, "Handling big data using NoSQL," in *Advanced Information Networking and Applications Workshops (WAINA), 2015 IEEE 29th International Conference on*, 2015, pp. 393-398: IEEE.

[25] M. Ferro, E. Silva, R. J. D. Fidalgo, and K. Engineering, "Astar: A modeling language for document-oriented geospatial data warehouses," p. 102174, 2023.

[26] L. Rocha, F. Vale, E. Cirilo, D. Barbosa, and F. Mourão, "A Framework for Migrating Relational Datasets to NoSQL1," *Procedia Computer Science,* vol. 51, pp. 2593-2602, 2015.

[27] S. Rathor, D. Batra, and W. Xia, "What Constitutes Software Development Agility?," 2016.

[28] A. Gamal, S. Barakat, and A. J. J. o. b. i. Rezk, "Standardized electronic health record data modeling and persistence: A comparative review," vol. 114, p. 103670, 2021.