

## An ontology-based approach to designing a NoSQL database for semistructured and unstructured health data

Poly Sil Sen<sup>1</sup> · Nandini Mukherjee<sup>2</sup>

Received: 5 December 2022 / Revised: 5 December 2022 / Accepted: 18 March 2023 © The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2023

#### **Abstract**

With the advent of ICT-based healthcare applications, various formats of health data are generated every day in huge volume. Such data, consisting of unstructured, semi-structured and structured data, has every characteristic of Big data. NoSQL databases are generally preferred for storing such type of health data with the objective of improving query performance. However, for efficient retrieval and processing of Big Health Data and for resource optimization, suitable data models and design of the NoSQL databases are important requirements. Unlike relational databases, no standard methods or tools exist for NoSQL database design. In this work, we adopt an ontology-based schema design approach. We propose that an ontology, which captures the domain knowledge, be used for developing a health data model. An ontology for primary healthcare is described in this paper. We also propose an algorithm for designing the schema of a NoSQL database, keeping in mind the characteristics of the target NoSQL store, using a related ontology, a sample query set, some statistical information of the queries, and performance requirements of the query set. The ontology proposed by us for primary healthcare domain and the above mentioned algorithm along with a set of queries are used for generating a schema targeting MongoDB datastore. The performance of the proposed design is compared with a relational model developed for the same primary healthcare data and the effectiveness of our proposed approach is demonstrated. The entire experiment has been carried out on MongoDB cloud platform.

Keywords Health Data · Big data · NoSQL Data Model · Health Ontology · Schema Design

#### 1 Introduction

Health Data is well known to be complex, sensitive, and voluminous having unmanageable structures, ownership issues and data variety. In the last two decades, deliberate attempts have been made to store health data effectively on electronic media and make it available to the people in need. There are various formats of health data starting from

heavy data, data from different medical devices, charts, graphs, images, audio, video files and sensed data—all are part of health data. Nowadays, lengthy textual comments are also used to check the mental fitness of a person. In short, health data is polymorphic. Thus, it is obvious that the storage, management, and processing of such different formats of data will vary extensively. Storing such variety of health data types with different volumes in a single data store and satisfying the performance requirements is a challenge.

structured, semi-structured to unstructured data. Text

Following the above discussion, it is understood that the data in healthcare domain has every property of Big data, i.e. volume, variety, velocity and veracity. Therefore, usual processing of data using a relational model often leads to various performance issues. Many of these performance issues arise due to the inefficient storage of Big Health Data. In usual scenarios, the data are retrieved by users at different levels (patients, care givers, hospital

✓ Poly Sil Sen
 poly.silsen@uem.edu.in
Nandini Mukherjee
 nandini.mukhopadhyay@jadavpuruniversity.in

Published online: 08 April 2023

- Computer Science Technology, University of Engineering and Management, New Town Road, Kolkata 700160, West Bengal, India
- Department of Computer Science Engineering, Jadavpur University, Raja S.C. Mallick Road, Kolkata 700032, West Bengal, India



Table 1 Relational Design for Primary Healthcare (showing only needed attributes)

Person (personId, name, occupation, socioEconomicStatus, address, contactNo, email, ageApproximate, foodHabbit)

Patient(patienttId,personId,familyHistoryDetail)

Doctor (doctorId,personId, specialization, worksForOrganization)

Complaint(complaintId,patientId, complaintDetails)

TreatmentEpisode(treatmentEpisodeId,patientId, complaintId, doctorId, treatmentEpisodeStatus)

Visit (visitId, patientId, complaintId, doctorId,treatmentEpisodeId, investigationId,treatmentPlanId, diagnosisInVisit)

Invsetigation(investigationId, patientId, complaintId, doctorId, treatmentEpisodeId, visitId, investigationDetail, investigationFile, investigationResult)

ContinuousMonitoring(continuousMonitoringId,patientId, complaintId, doctorId, treatmentEpisodeId, visitId,investigationId, continuousMonitoringDetails)

AbstractedSensorData(abstractedSensorDataId, patientId, complaintId, doctorId, treatmentEpisodeId, visitId,investigationId, continuousMonitoringId,abstractedSensorDataFile)

TreatmentPlan(treatmentPlanId, patientId, complaintId, doctorId, treatmentEpisodeId, visitId, treatmentPlanDetails)

Medication(medicationId,patientId, complaintId, doctorId,treatmentEpisodeId, visitId,treatmentPlanId, medicationName)

Surgery(surgeryId,patientId, complaintId, doctorId,treatmentEpisodeId, visitId,treatmentPlanId, surgeryDetails)

Therapy(therapyId,patientId, complaintId, doctorId,treatmentEpisodeId, visitId, treatmentPlanId,therapyDetails)

administration and government) running their queries on health data store. Some of the queries must run fast to take care of emergency situations. Response time of such queries is very important.

It is evident from many studies that NoSQL databases are able to handle Big data having complex data representations. Query performance and horizontal scalability are improved with the use of NoSQL databases. However, designing a NoSQL database is not trivial. The approach followed for designing relational databases, i.e. starting with a conceptual model, then to a logical model and finally implementing a physical model, is not always effective for NoSQL databases. Relational database design approach generally focus on optimising storage. On the other hand, NoSQL database design needs to focus on improving query performance, and hence, it follows a query-driven approach. Understanding the detailed application query pattern is a critical first step for query-driven database design.

Alongside, in order to achieve an efficient NoSQL database design, conceptual modeling plays a major role by capturing the semantics of the applications. In this regard, developing a generic ontology for healthcare domain and integration with the design process is an important requirement. Ontology is a repository of shared knowledge on a specific domain which has understandable and computer processable, accepted representation. Thus, an ontology relevant to the application is required for developing a suitable data model for NoSQL database design.

In this paper, a NoSQL database design approach is proposed on the basis of the related domain knowledge stored in the form of an ontology (web ontology file), a sample set of queries, statistical data about the queries and performance requirements of the queries. The process is depicted in Fig. 1 and put forward as an algorithm. In order to demonstrate the effectiveness of the approach, query response times from the resulting database are compared with the response times from a relational design on the same domain using the same set of queries. To the best of our knowledge, this topic has not been given much importance by the people in the research domain. Though developers from software industries design the data model meticulously by studying the application domain, the data characteristics, various storage options etc.

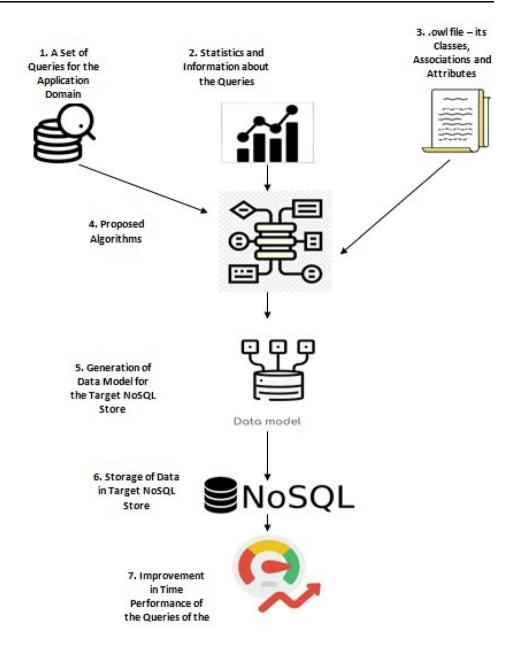
The rest of the paper is organized as follows. In the next section, related work in this domain is discussed. In the subsequent sections, first the proposed approach is discussed and an overview of the proposed algorithm is presented, which is followed by a brief description of a healthcare ontology specifically focusing on primary healthcare domain. In the Case Studies Section, the ontology and a set of queries are used for few case studies based on a comparison of our proposed approach to NoSQL schema design and a relational model for the same domain. Observations are noted, and the paper concludes with future work in the Conclusion Section.

## 2 Related work

This work involves various research domains, starting with ICT-based applications, sensor, ontology, cloud, NoSQL databases and other related topics. For the purpose of understanding the requirements of health data storage, we



**Fig. 1** The overall process of data model development for a NoSQL store



also studied health data standards. One important requirement of remote healthcare delivery applications is storing and management of e-health sensor data. Thus, this section attempts to give an overview of the contemporary research works, which have been carried out in the domains related to the work presented in this paper.

## 2.1 ICT- based healthcare applications

Some study on remote healthcare delivery has been made in [1]. In this paper, the authors describe a kiosk-based healthcare delivery framework and elaborate the requirements for implementation of this framework. In [2], the authors surveyed the evolution of the tele medicine landscape, discussed its utilization during the COVID-19 pandemic and how they expect this technology to be implemented in the post pandemic world. However, the primary focus of this article is the state-of-the-art in USA. Importance of ICT-based healthcare system has also been discussed in [3] and [4].

In the past two decades, much progress has been made in e-healthcare applications and remote patient monitoring. In the work of Celesti et al, remote patient monitoring and rehabilitation services are proposed by satellite care places or directly at patients' homes with the help of Cloud computing, Internet of Things (IoT) and Big data analytics technologies. Patients having motor disabilities can be assisted remotely. The paper provides an implementation



of Tele-Rehabilitation as a Service. The service generates healthcare Big data from remote rehabilitation devices needed by patients which need to be processed in the Cloud of the care provider. Performances of four major NoSQL stores including Cassandra, Hbase, Neo4J, MongoDB are compared and the document-based approach of MongoDB is found to perform best in general [5].

### 2.2 Sensor

Sensors play important role in the modern Internet-based ICT applications. Sensors continuously generate and feed data over a time period. There is a temporal factor which is involved with sensor data, hence a flexible schema having integration factor with temporal data is needed. Mehmood et al observes that, all NoSQL databases have one or the other unique advantages, but "document-oriented storage, as MongoDB provides, is considered robust for handling multiple structural information to support IoT goals.. This.. favours Java Script Object Notations (JSON) documents to support dynamic schemas; hence provide integration to different data types besides scalability features". In contrast, column oriented databases provide attribute level access in place of tuple level access. In the work of Mehmood et al, three ways of storing sensor data is suggested - document per second, document per minute and document per event [6].

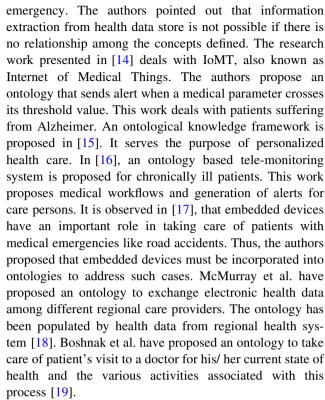
#### 2.3 Cloud

In the research works presented in [7] and [8], storage of health data in cloud including real time data has been discussed. Their main goal is to serve different health care service providers, managing disparate representation of information, data recovery and security. The authors also discuss various services provided by cloud. They also discuss aging of data and long time storage of data.

## 2.4 Ontology

Health data knowledge domain has to be studied for primary healthcare and remote healthcare for this work. Ontology accommodates the differences of multiple health data standards. It also uses the needed portions of health data standards and introduces some more needed concepts and terminologies like sensor etc.

Schneider et al acknowledge that ontology helps in data model adaptability, reuse and thus reduce operational and developmental cost. It integrates Big data systems maintaining semantic interoperability. It provides an answer to semantic heterogeneity [12]. In [13], an ontology is proposed that is used to specify actors which can access the information provided by embedded devices during medical



The health related ontologies discussed above cover a small sub-domain of ontology. While studying the literature on health ontologies, it has been observed that no general health ontology is proposed to meet the requirement of e-healthcare. Ontologies used in healthcare related applications are either too specific focusing on a single application or too general focusing on a generic domain. Hence, we propose a generic healthcare ontology to facilitate design of data models.

## 2.5 NoSQL data stores

A survey of Big data technologies has been published in [9]. The primary objective here is to conduct a comprehensive investigation of state-of-the-art storage technologies available for Big data. The paper presents a taxonomy for Big data storage technologies which provide assistance to data analysts and researchers to understand and select a storage mechanism suitable for their needs. The authors use Brewer's CAP theorem to evaluate the performance of different storage architectures and compare and analyze the existing approaches.

Cloud-centric Big data placement strategies together with the data storage methodologies have been discussed in [10]. The paper attempts to highlight the actual correlation between the strategies and methodologies in order to support Big data management. The authors of this paper focus on the management aspects which are seen under the prism of non-functional properties.



In the work of [11], the authors focus on various aspects of handling Big data in healthcare. It presents the current state-of-the-art technology used for storing health records and health data models. The research challenges in Big data acquisition, storage, processing and security have also been discussed in this book.

## 2.6 Data modelling in NoSQL data store

Although research targeting NoSQL data modeling is very limited, Paolo et al. has proposed a generic method for NoSQL data modeling [20]. It is considered that the target data store will be a set of collections. Each collection will be a set of blocks. Each block will be a set of entities. Group of needed entities are made to form aggregates in this method. Aggregates are partitioned as and when required. Implementation is done by mapping to the target data store. The implementation domain has been chosen to be representation and management of aggregates for games.

Imam et al. has proposed an automatic schema generation model for NoSQL document oriented databases on the basis of relationship prioritization and semantic mapping of related documents. It uses user requirements after balancing and prioritizing. Embedding and referencing are decided automatically by the proposed method. The process accepts list of entities and the create and update operations as input. The approach takes care of consistency, availability and scalability. However, this approach is based on only create and update operations, it ignores the most important operation operation for healthcare application, that is Read operation. The generated schemas have been tested for security and performance. The implementation of this approach has been done in MongoDB and Couchbase [21].

Similar work has been carried out in [22] without performance testing in the domain of university social media network which has been developed by programmers of the university. The ERD model and the UML model have been used as the basis of this work. User entities and user dependent entities are found in this approach. Data access patterns, nature of data to be used, rate of updates on a particular field, relationships among the entities are considered to be important in designing the schema in this work. Embedding, referencing, bucketing and general approach are considered in this work. General approach is considered as the case when none of the other approaches are applicable.

A study of the requirement and possibility of automated schema design for NoSQL store has been done in [23]. The objective of this study is to optimize query performance and minimize space overhead. An automated cost driven approach for NoSQL store is found to be the better

approach on the basis of a given workload based on query and storage costs.

A procedure to generate schema for NoSQL data store is suggested using UML of the domain under study in [24]. In this paper, cardinality, constraints are considered for mapping. It covers model partitioning and region mapping considering multiple data stores.

In the work of [25], automatically schema is designed for Document data store using query workload and multiple parameters like query cost, storage cost, degree of heterogeneity, and average depth of the document etc.

Mior et al. have proposed a procedure for design of schema for Cassandra using conceptual schema, statistic, query workload, space constraint. It uses a number of possible designs to suggest a schema design and a query execution plan [26].

The above stated approaches do not use ontology for understanding the domain knowledge. None of the works are done in the healthcare domain. The works do not concentrate on read performance most importantly. The proposed processes do not provide a generic approach for a query set.

In our previous work, we have proposed an algorithm that takes an ontology, a set of queries and performance requirement of the queries and provides schema for a NoSQL data store. The performance of the algorithm has been compared theoretically with relational design for the same set of queries [27].

Healthcare domain has many entities with complicated relationships among them. Since, read performance is given the most importance, embedding is preferred in comparison to referencing and bucketing. Referencing and bucketing approaches are considered only in special cases. 'Normalization can be used as space is not a problem anymore [22]'— this comment cannot be agreed upon, because normalization increases time of reading considerably in distributed NoSQL stores. The proposals are intuitively followed in this work.

#### 2.7 Overall observation of related work

It is observed that none of the above works consider queries, specifically the read queries to be run on the target NoSQL store. Queries are very important for designing schema of a distributed data store. There must be a formal input of domain information to the process of automatic schema generation in terms of ontology. Ontology is dynamic, human understandable, computer interpretable and can be used again and again for sharing and interoperability among different users and also for accommodating multiple (at times incompatible) needed data formats.



Although NoSQL databases are in use for quite sometime and many implementations of health care applications are available that use one or the other NoSQL data stores, a standard, well accepted procedure to map the data requirements to the physical data model of a target NoSQL store is not available in general. The physical data model of the target NoSQL store is designed on the basis of the application requirements and the services required from it. There is no work that tries to optimize the timing requirements or the resource usage of an application. This paper focuses on developing a design approach based on a set of queries and an ontology, so that the query performance improves, while taking into account the semantics of the application.

## 3 Proposed approach for NoSQL database design

In this section, we describe our approach to NoSQL database design. Although, the present work focuses on healthcare applications, our approach is generic and will be applicable in other domains as well.

## 4 Choosing a NoSQL data store

There are various NoSQL data stores which are frequently used by the developers and their types are designed for a specific group of use cases. Therefore, choosing the right data store is required while designing a data model for an application. For example, document databases are useful for applications which need flexible schema with complex queries and leverage complex indexes (multikey, full text search etc.). On the other hand, column-oriented databases are suitable for data having high write speeds with relatively less velocity reads, data extractions by columns using row keys and no ad-hoc query patterns. Graph database work well with applications requiring traversal between data points [28]. To avoid complexities of the approach, in this paper, we consider document databases, in particular MongoDB database, which has been used for implementation most frequently. This is due to the fact that MongoDB satisfies our requirements and it is such a NoSQL store that can accommodate all the other models using JSON (actually BSON format). However, the steps described below will also be applicable for column-oriented databases and graph databases with modifications.

<sup>&</sup>lt;sup>1</sup> The class or entity set C contains the aliases of the entities, for example "Visit" entity may be called "Encounter", "History" may be called "MedicalHistory".



## 5 Input to the process

For every database application, the workload is properly known in terms of a query set, requirements of the queries in respect of timing and resources, whether the queries mostly read or write data (read-heavy or write-heavy), the size of the data that is dealt with, frequency of the queries used, performance requirement from the queries. Hence, the set of queries is an input to our algorithm.

Alongside, we also consider the ontology for this particular domain. The owl file of the ontology considered for this purpose is studied to retrieve set of classes / entities, set of object properties (associations), and the set of data properties (attributes). The names of the data attributes may contain the substring "Id" to indicate it is an identifier of the entity or class concerned. On the other hand, the annotation property may also contain indication of the identifier.

Based on the above knowledge, the steps of the process are implemented as follows:

## 5.1 Step I

The query set is ordered in descending order of frequency of running the query, timing requirement from the query.

From the, ontology file, a data structure M is generated that contains information on the association among the entities. Such an association may be dealing with one, two or multiple entities or classes. The aliases of the entity names are also included in C.  $^{1}$ 

#### 5.2 Step II

If the query deals with write heavy data like sensor data then such data is kept separately.

## 5.3 Step III

For a query from the query set, perform the following steps:

- Retrieve the entities, attributes and conditional attributes used in the query. The entities are found from the set of classes or entities. If the names are not found then the alias list can be checked.
- 2. If the number of entities in the query is 1, then all the attributes and conditional attributes of the query are placed in one collection.
- If the number of entities is 2, then the M data structure having the association among the classes are considered.

- If the two entities have some association and the name of the association has the sub-string "contains" or "has", then the collection is formed with the name of the having or containing class and all the attributes of both classes are kept in the collection.
- If the name of the association does not contain the sub-strings specified above, then all the needed attributes of both entities are kept in one collection with the name of the collection formed from the contributing entities.
- 4. If the number of entities is more than 2, then also the association among the classes are studied from the data structure *M* and a collection is made with all the attributes of the associated classes.

In all the above cases, key of the collection is set to be the set of Id(s) of the classes or entities involved. The Ids are found from the ontology by searching the substring "id" from the related attributes.

## 5.4 Step IV

If there are more queries in the query set, then the next query is selected. If there is a suggested collection that satisfies the requirement of the newly selected query, then no collection is required for the new query and the procedure is continued.

If the new query requires adding few attributes to an already created collection, then the attributes are added accordingly.

Else, the control goes back to StepIII.

## 6 Health ontology

A healthcare ontology has already been proposed in [29]. In this research work, the ontology is further developed and enhanced to capture the semantics required for primary healthcare applications. In the proposed ontology, a main class *Person* is created along with his / her demographic data and basic clinical parameters. The classes *Patient* and *Doctor* inherit the *Person* class. For a Patient, the medical history is stored, which include Family History, Personal History, Present History, Allergy History. *MenstrualObstericHistory* and *ImmunizationHistory* following the Minimum Data Set of Indian Standard of Electronic Health Records [30]. This part of the ontology is shown in Figs. 2 and 3.

A patient suffers from a disease and he or she has one or multiple symptoms which are considered as *Complaint*; *Complaint* details of a Patient are also stored. A patient with a *Complaint* visits a doctor and that initiates a

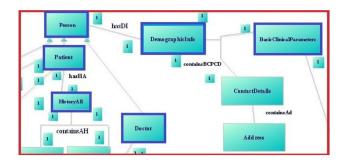


Fig. 2 Medical history of a patient and demographic information of a person

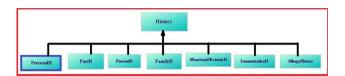


Fig. 3 Types of medical history

TreatmentEpisode. A TreatmentEpisode contains one or more Visits by the patient to the Doctor. In a Visit, certain clinical parameters are measured, such as height, weight, pulse, and blood pressure, which come under the class called MeasuredClinicalParameters. The relevant entities and relationships for representation of this part of the ontology are shown in Fig. 4.

Clinical Signs, Symptoms, are noted by the *Doctor* during a *Visit* and also assessment is done (classes are *ClinicalSigns*, *ClinicalSymptoms*, *ClinicalAssesments*). Doctor may prescribe investigations. *Investigation* class has two subclasses, (i) *ContinuousMonitoring* — when continuous monitoring for the patient is required, and (ii) *DiscreteMonitoring* — when time-based monitoring is done for the patient. *DiscretMonitoring* may be of two types — *OneTimeInvsetgation* (such as laboratory-based blood test) and *PeriodicInvestigation* (such as periodic temperature measurement). In our remote healthcare system, *ContinuousMonitoring* gets its data from *Sensors*.

Diagnosis is made on the basis of basic clinical parameters, measured clinical parameters, different types of clinical signs, symptoms and assessment, and results of the prescribed investigations. Accordingly, the doctor proposes a treatment plan. The class *TreatmentPlan* includes *Medication*, *Surgery* and/or *Therapy*. On the basis of the diagnosis, medication, surgery, therapy etc. are prescribed and stored in the above classes.

The above mentioned classes include attributes. The attributes and initial medical details are studied from [31], 32] and from other health data standards and included in our proposed ontology.

In the next section, we further discuss different parts of the ontology as per the requirements. In all the



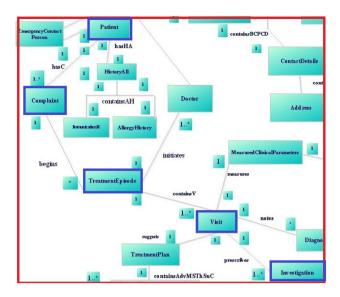


Fig. 4 Investigation of a patient

figures showing classes or entities, the classes used for a query are marked with rectangles with border in dark blue colour.

## 7 Case studies

We now apply our approach described in the Proposed Approach for NoSQL Database Design Section in the healthcare domain. The ontology discussed in the Health Ontology Section is input to the process. We consider altogether nine queries for designing the NoSQL database. In order to demonstrate the efficiency of the approach, we also implement a relational design and execute each of the nine queries on the NoSQL implementation, as well as on the relational implementation. It may be noted here that in order to achieve a fair basis of comparison, both designs have been implemented on the same data store, here MongoDB. A set of experiments have been conducted to compare these two implementations. The experiments are described in this section as a few case studies.

## 7.1 Scenario

In the case studies, a situation is considered when a *Patient* suffering from one or more *Complaint* goes to a *Doctor* to get herself treated. *Doctor* requires some attributes of *BasicClinicalParameter* Class like *age*, *gender*, *blood-Group*, *allergyDetails* etc. Doctor also requires *allergyHistory* of the *Patient*. In this process, *MedicalHistory* or *History* of a *Patient* is noted. *Doctor* advises a set of *Investigation* to the *Patient*. Before or after, the *invResults* (investigation results) are studied, *Doctor* advises

Medication, Surgery, Therapy etc. depending on the specific Complaint. Investigation may include AbstractedSensorData from ContinuousMonitoring or DiscreteMonitoring entities/ classes.

The following set of queries is considered for the experiments.

## 8 Set of queries

The set of queries considered in this work in the context of the above scenario is mentioned below in SQL format (for using a standardized representation):

- Select \* from Doctor as D, Person as P where D.specialization = 'cardiologist' and D.perId= P.perId;
- Select P.name, P.occupation, P.socioEconomicStatus, P.foodHabit from Patient as Pt, Person as P where P.perId = Pt.perId and Pt.ptid = 'a string';
- Select I.invFile from Patient as Pt, Investigation as I where Pt.ptId = I.ptId Patient as Pt where ptId = 'a string' and invDet = 'ECG';
- 4. Select ptId, familyHistoryDetail from Patient as Pt where ptId = 'abc' and familyHistoryDetail like %diabetes%;
- Select asdFile from ContinuousMonitoring as CM, Patient as Pt where Pt.ptId = CM.ptId and readingTakenFor = '1 hr' and cMDet = 'bp';
- 6. Select TE.trtEpStatus, Pt.familyHistoryDetail, V.diagInVDescription, I.invRes, TP.tpDet, TP.advDet, TP.suggestedCareDet, M.medicationName, S.surgeryDetail, TH.therapyDetail from Patient as Pt, Complaint as C, TreatmentEpisode as TE, Doctor as D, Visit as V, Investigation as I, TreatmentPlan TP, Medication as M, Surgery as S, Therapy as TH where Pt.ptId = 'a string' and C.compliantDet like %breathing problem% and Pt.ptId= TE.ptId and TE.drId= D.drId;
- 7. Select \* from Doctor as D, Person as P where D.worksForOrganization = 'an org' and D.perId = P.perId;
- 8. Select P.name, C.complaintDet from Person as P, Patient as Pt, Complaint as C where P.perId=Pt.perId and Pt.ptId = C.ptId and Pt.ptId = 'a string';
- 9. Select count(\*) from Person as P, Patient as Pt, Visit as V where P.ageAppr<= 5 and V.diagnosisInV = 'malnutrition' and P.perId = Pt.perId and Pt.ptId = V.ptId;



#### Table 2 Algorithm Output for nine queries

- Query 1 PersonDoctor(doctorId, personId, name, address, contactNo, specialization, worksForOrganization)
- Query 2 PersonPatient(personId, patienttId, name, occupation, socioEconomicStatus, foodHabit)
- Ouery 3 PatientInvestigation(patienttId, investigationFile,investigationDetail)
- Query 4 Patient(patienttId, familyHistoryDetail)
- Query 5 PatientAbstractedS ensorData (patienttId, abstractedSensorDataFile, continuousMonitoringDetail, readingTakenFor)
- Query 6 PaCoDTrEViInDiTrPMeSuTh (patienttId, doctorId, complaintDetail, treatmentEpisodeStatus, familyHistoryDetail, diagnosisInVisit, investigationResult, treatmentPlanDetail, medicationName,surgeryDetail,therapyDetail)
- Query 7 DoctorPerson (personId,name, address, contactNo worksForOrganization )
- Query 8 PersonPatientComplaint(personId, patientId, name, complaintDet )
- Query 9 PersonPatientVisit (patientId, personId, ageApproximate, diagnosisInVisit )

## 9 Relational database design

In this subsection, a relational design of the domain of this work is presented. In the given schema, the primary keys are indicated with single underline and foreign keys are indicated with double underline. It is to be noted that only the needed attributes are shown in the schema. A table may have other attributes as well.

## 9.1 NoSQL sesign

Table 2 presents the denormalized design resulting from our approach using the health ontology and the nine queries listed above. NoSQL databases mostly follow BASE property. Here, more importance is given to Availability in this design.

### 10 Experimental set up

The experiments described in the next few subsections have been conducted on MongoDB shared cloud platform with 512 MB RAM and 3 replica sets. A standard Internet connection of a mobile has been used to store and retrieve health data in MongoDB cloud. Actual health data collected in a previous real life project have been used to synthetically generate health data in huge volumes. ECG files and sensor data have been obtained from the abovementioned project. Sensor data have been stored following the format of real life sensor data.

While performing the experiments, nine queries have been chosen to run in the experimental set up explained here. The queries work with the classes *Person*, *Patient*, *Doctor*, *DemographicInfo*, *HistoryAll*, *Complaint*, *TreatmentEpisode*, *Visit*, *Investigation*, *ContinuousMonitoring*, *AbstractedSensorData*, *Sensor* etc.

The queries are run with varying number of records depending on the size of the data that could be stored in cloud following the outputs of the process proposed in the previous section. The time for each query execution is measured in seconds. The same query is run 5 times (general query) and 10 times (for sensor and ECG data) and each time the average of all these observations is taken.

The entire procedure is repeated with the NoSQL design and the relational design for the same query set. Performances are observed in similar manner and graphs were plotted with number of records present in collection on x-axis and time required to retrieve the records (documents) on y-axis.

In our work, we have chosen MongoDB for easy integration with the language used accommodating all types, sizes, and formats of data and providing similar performance. There is a conscious effort from MongoDB cloud service providers and the team to make the job of a developer easy by providing blogs and documentation, and helping with troubleshooting. Sensor data and images can be efficiently stored in MongoDB along with different types of text. The document model with JSON documents stored as BSON objects completely suits the type, size, format, and variety of health data.

GridFs is used in MongoDB to store and retrieve BSON files of size greater than 16 MB. The files used in this work are less than 16 MB in size, and for this reason, GridFS has not been used to store and retrieve files in this work. Buckets are aggregation of data in MongoDB with the entries kept in group by a field. Buckets can be used to store sensor data files. Storage management will be better in that way. The disadvantage of using bucket is that fast searching, prompt insertion and infrequent updates are not efficiently supported. Thus, in this work, bucket has also not been used. However, in future, performance may be tested with other options, such as GridFS, buckets etc.

In the following subsections, we consider different cases where queries are run on the NoSQL design which is an outcome of our algorithm and the relational design (both implemented on MongoDB database) and their performances are compared. The experiments and the results obtained in these cases are discussed below.



## 10.1 Information on type of data working with

Few things are noted here. Changes in demographic information, like *name*, *dateOfBirth* etc. are not very frequent. If a *Patient* undergoes many *Investigations*, then this data becomes write heavy. Sensor data is generally write heavy for critically ill *Patients* and also at the time of emergency. For a patient who is critically ill, sensor data can be kept in bucket provided by MongoDB. In order to access sensor data fast, it can be kept within collections of MongoDB, if the file size is less.

## 10.2 Simple query execution with no join

A query is run to find whether a patient is diabetic (has a history of diabetes) or not when his/ her patient id is provided. In our proposed ontology [29], *Patient* is an entity, which is associated with *HistoryAll* using *hasHA* object property as shown in Fig. 2.

The information regarding a diabetic patient is stored in the *PersonalHistory* of a *Patient. PersonalHistory* is contained within *MedicalHistoryAll* class. Patient Collection in MongoDB will always keep history data and all other needed basic clinical parameters following the algorithm specified in the Prposed Approach for NoSQL Database Design section. In relational design, since there is a 1:1 relationship between a patient and his or her medical history, therefore, these fields are stored with the patient entity. Hence, no join is required for this query in case of relational implementation. Thus, both implementations result similar performance. The purpose of observing the performance of this query is to observe the performance of MongoDB cloud in general for a query where no join is needed.

Records of patients are varied form one hundred thousand to eight hundred thousand records in the query where

it is found out whether a patient is diabetic or not. The graphs depicted in Figs. 5 and 6 show the percentage of Virtual Memory used and the CPU usage respectively for the above query. The query is run varying the number of records (here the number of patients) from hundred thousand to eight hundred thousand. The percentage of virtual memory usage has been shown to vary between 54.5 and 57.4 s approximately. The percentage of CPU usage varies from 1.3 to 2.15 s approximately. Thus, both virtual memory and CPU usage are found to be increasing with the increase of number of records.

## 10.3 General queries

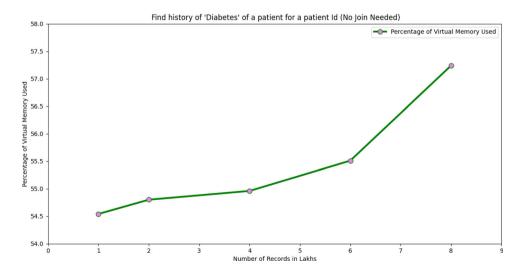
The following two queries are general queries which require join in case of relational design.

In the first case, demographic details of a patient having a certain patient id is searched. Figure 2 shows the part of the ontology related to this query. Here, *Person* is an entity with attributes or data properties *name*, *Address*, *contactNumber*, *email* (if any), *foodHabit*, *socioEconomicStatus*, and *occupation*. *Patient* is a subclass of *Person*. If relational design is used then a join operation is needed for *Person* and *Patient*.

The query execution times for both, NoSQL implementation and corresponding relational implementation are plotted against varied number of records in the database. The graph shown in Fig. 7 depicts this result.

For execution of this query, number of records (number of patients) varied from one hundred thousand records to four hundred thousand. The time required to run the query in case of four hundred thousand records for both relational and denormalized design is 3.26 and 3.51 s respectively. As observed from the graph, time required for running this query on relational design is always higher in comparison with the denormalized design.

**Fig. 5** Percentage of virtual memory used for a query with no join





**Fig. 6** Percentage of CPU processing used for a simple query that does not require join

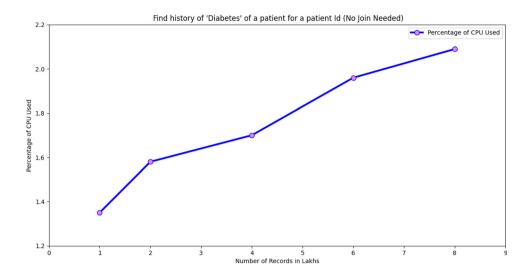
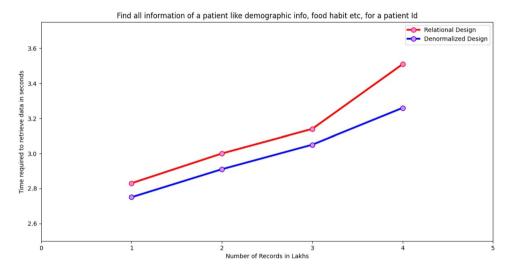


Fig. 7 General queries-A



In the second such query, details of complaints of a patient is searched for a given patient id. Patient and Complaint are two different entities. The Complaint entity contains complaintDetail of a Patient (patient\_Id is a foreign key in the Complaint class in case of the relational design) along with other important data properties of Complaint (Fig. 2 and Fig. 4). This query needs name attribute of *Person* class also. As in the previous case, this query is run on the NoSQL implementation (the schema contains all attributes of the patient class and the complaint class) and the relational implementation (where a join of two entities is required) and the execution times on the data store with varying number of records are shown in Fig. 8. The time required to run the query in case of four hundred thousand records for both relational and denormalized design is 3.22 and 3.37 s respectively. Here again, time required for relational design is much higher than the time taken for denormalized design.

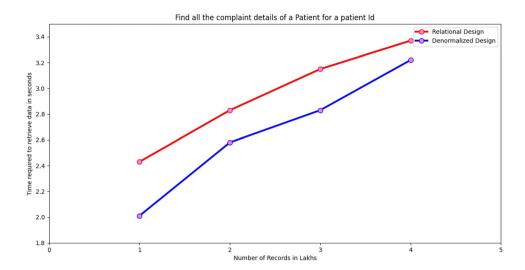
## 10.4 Queries returning multiple records from the data store

In this subsection, queries returning considerable number of records from a collection are studied. Thus, the queries require more time in seconds in comparison with other queries.

As a test case, a query is executed that will retrieve the details of all doctors having a certain specialization. The part of the ontology required here contains *Doctor* entity (a subclass of *Person*) containing data properties, like *doctor\_id*, *organization*, *specialization* etc. In order to get the basic demographic details of a *Doctor*, such as *name* and *address*, the data properties of *Person* entity need to be accessed. The entities and their associations required for this query are shown in Fig. 2. As *Person* and its subclass *Doctor*, both are required for this query and a join operation is necessary to run this query in relational design. The



Fig. 8 General queries -B



query returns all the records which match the specified criteria, hence longer time is needed compared to the previous cases. The results of executing this query on NoSQL and relation implementations are compared in Fig. 9.

For execution of this query, number of records (number of doctors) varied from one hundred thousand records to four hundred thousand. The time required to run the query in case of the denormalized and the relational design for four hundred thousand records are 27.8 and 31.23 s respectively. As observed from the graph, time required for running this query on relational design is always higher in comparison with the denormalized design.

Another query to search the details of all doctors belonging to an organization is executed in similar manner. The query returns all the records that match the specified criteria. Results of running this query are shown in Fig. 10.

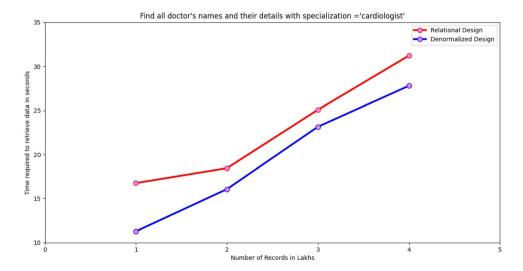
For execution of this query, number of records (number of doctors) varied from one hundred thousand records to four hundred thousand. The time required to run the query

in case of the denormalized and the relational design for four hundred thousand records are 28.63 and 41.99 s respectively. The time required for running this query on relational design is always higher in comparison with the denormalized design.

## 10.5 Query dealing with ECG files

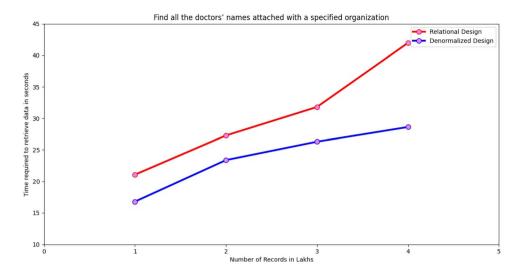
In this subsection, we deal with the query handling ECG files stored as image files. In the proposed ontology [29], Investigation is an entity having data properties investigation\_Id (alias invId), investigationDetail (alias invDet), investigation\_File (alias invFiles), investigationResult (alias invRes) etc. Patient is associated with Investigation through the hasDetailsOfInv object property as shown in Fig. 4. ECG file is an investigation\_File (invFiles). Since, the ECG files used here have size less than 16 MB, these files are stored as Binary data (if the file size is more than 16 MB then the file can be stored in GridFS). The entities like Patient, Complaint, Investigation

**Fig. 9** Queries returning multiple records -A

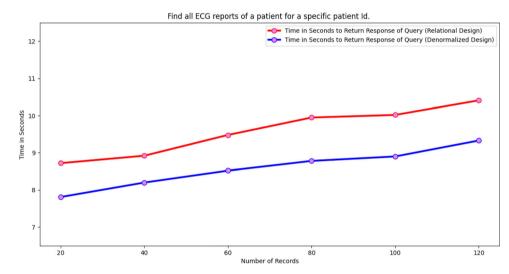




**Fig. 10** Queries returning multiple records- B



**Fig. 11** Query dealing with ECG image files



are needed for this query. In case of relational design, join operation is performed with these entities.

A query is executed to retrieve the ECG files related to a patient with a particular patient id. The time requirement for this query is a bit more than the time requirement of other queries returning a single record. The results of executing this query on NoSQL implementation and relational implementation are shown in Fig. 11 for varying number of records.

The query is run for 20 records to 120 records. Each of the ECG file occupies 2 MB approximately. MongoDB shared cloud has a space limitation of 512 MB. For this reason, more records could not be used for execution of this query. For execution of this query, number of records (number of patients) varied from one hundred thousand records to four hundred thousand. The time required to run the query in case of the denormalized and the relational design for 120 records are 9.33 and 10.41 s respectively. The time required for running this query on relational

design is always higher in comparison with the denormalized design.

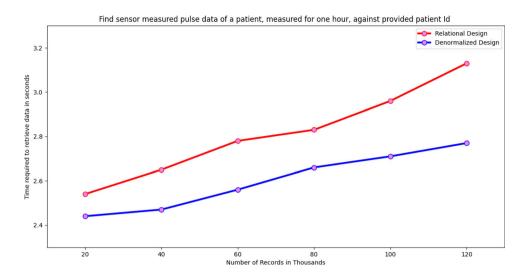
## 10.6 Query to retrieve sensor data

In this subsection, the results of executing a query to retrieve sensor pulse data is presented. In our ontology, an entity *ContinuousMonitoring* is created under *Investigation* class that deals with all the data obtained from continuous monitoring of a patient. In order to address the heterogeneity, sharing and other issues related with sensor data, we propose to create virtual sensors as discussed in [33]. Virtual sensors are represented as *AbstractedSensorData* which is associated with *ContinuousMonitoring* through the *containsASD* object property. *AbstractedSensorData* gets data from *SensorOutput* provided by *Sensor* entity as shown in Fig. 4.

We run a query in which pulse of a patient measured for one hour is retrieved when his/ her patient id is provided.



**Fig. 12** Query to Retrieve Pulse Data observed for an hour is retrieved



As in case of the ECG files, sensor data files are stored as binary data. The needed classes are *Patient*, *ContinuousMonitoring* and *AbstractedSensorData*. Thus, in case of relational implementation, join is performed with the related entities. Following the algorithm specified earlier, sensor files, being write heavy files, are stored separately. However, they are managed as attributes of the collection along with other needed attributes. In case of relational implementation, sensor data is stored separately from *Patient* data and while running query join operation is performed. But, in case of NoSQL implementation, a denormalised design is followed. The results of this execution are shown in Fig. 12.

The query is run for twenty thousand records to one hundred and twenty thousand records. Each sensor file occupy approximately 2 KB storage space. For this reason, maximum 120 thousand records could be accommodated in MongoDB cloud with 512MB memory limitation. The time required to run the query in case of the denormalized and the relational design for one hundred and twenty thousand records are 2.33 are 2.71 s respectively. The time required for running this query on relational design is always higher in comparison with the denormalized design.

## 10.7 Query requiring more than one join in case of relational design

In this subsection, we consider a query that requires multiple joins in case of relational design. In this query, treatment case history of a patient is retrieved when his/ her patient id is provided along with a complaint.

In the proposed ontology [29], a *Patient* is associated with *Complaint* entity through *hasC* object property. *Complaint* entity is associated with *TreatmentEpisode* entity through *begins* object property. *TreatmentEpisode* is associated with *Visit* entity through *containsV* object

property, and *Visit* is associated with *Investigation* through *prescribes* object property. On the basis of *investigationResult* and other data properties and object properties, *TreamentPlan* is advised to the patient. This part of the ontology is shown in Fig. 4. Thus, to satisfy the abovementioned query, the entities Patient, Complaint, TreatmentEpisode, Visit, Investigation, and TreatmentPlan need to be joined in relational implementation. In case of NoSQL implementation, a denormlized schema is created from the above entities.

The above query is executed with varying number of records on NoSQL and relational implementations. The execution times for this query on two different implementations are shown in Fig. 13.

For execution of this query, number of records (number of patients) varied from fifty thousand to four hundred thousand. The time required to run the query in case of the denormalized and the relational design for four hundred thousand records are 2.86 and 2.89 s respectively. The time required for running this query on relational design is always higher in comparison with the denormalized design. There is a point where the readings of both the plots are close to each other. The reason may be that on a shared cloud the load varies from time to time affecting the performance at some point of time.

## 10.8 A simple statistical query

In this subsection, performance of a simple statistical query is observed. In this query, number of babies suffering from "Malnutrition" is requested. Our ontology contains *diagnosisInV* in *Visit* entity that may have values "Malnutrition", if it is diagnosed for a patient. Number of such entries is counted for *age* (found from *dateOfBirth* data property) less than 5 years. The entities needed for this



query are *Patient*, *Complaint*, *TreatmentEpisode*, *Visit*. The *Visit* entity has the required *diagnosisInV* field.

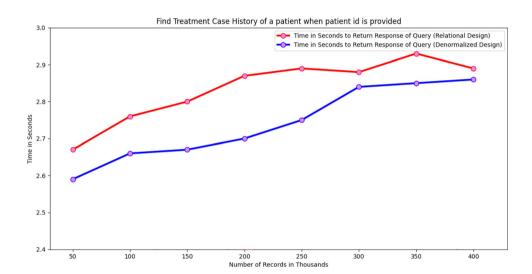
Figure 14 shows the execution times of this query on NoSQL and relational implementations for varying number of records. For execution of this query, number of records (number of patients) varied from one hundred thousand to four hundred thousand. The time required to run the query in case of the denormalized and the relational design for four hundred thousand records are 3.27 and 3.41 s respectively. The time required for running this query on relational design is always higher in comparison with the denormalized design. There is a point where the readings of both the plots are close to each other. Here again, use of shared cloud might have affected the performance at some point of time.

# **Fig. 13** Query Requiring Multiple Joins in case of Relational Design

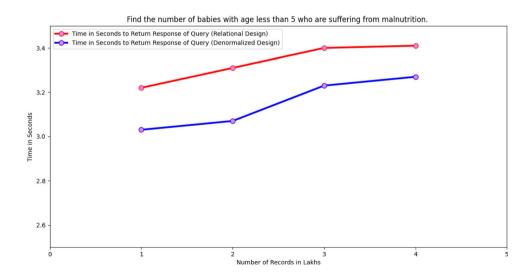
### 11 Observation

In all the cases discussed in the Case Studies Section, whenever a NoSQL implementation is generated using the approach discussed in this paper, the response times for the queries improve in comparison with the response times in the relational implementation. There is a straightforward reason for this. The relational implementation is designed to overcome the redundancies in data. On the other hand, a denormalization strategy is followed for NoSQL implementation with an objective to improve query response times.

In case of the proposed approach, all the collections are prepared in such a way that no join is required. It is to be kept in mind that if a query returns some data on which treatment of a patient and related decision making are



**Fig. 14** Query Handling Simple Count Operation





involved, then the focus should be on optimising query performance, and preference should be given to denormalization over the requirement for efficient resource usage.

Further, it is observed that the storage required is comparatively less in case of the proposed approach than the storage requirement in relational design.

Sensor files are either voluminous or of small size. Small-sized files can be stored along with other needed attributes in case of denormalized design. However, voluminous sensor data need to be stored in separate files.

For this work and in normal situations, smart devices with Internet connection and access to Cloud Services are sufficient to store and retrieve health data in cloud. However, it is noticed that the speed of Internet connection severely affects the execution performance of the queries.

## 12 Conclusion

No standard procedure of designing NoSQL store is available in the contemporary literature, which focus on the application and study of data domain and query characteristics. In this work, an effort has been made to formalize the procedure. The result shows that the queries are taking less time to retrieve needed data with our proposed design in comparison with the relational design implemented on the same MongoDB cloud. In future, the algorithm may be used on ontologies developed by others and ontologies provided by popular health data standards. In such cases, the algorithm may be tested on a different scale of implementation of health care ontologies. Experiments should also be carried out with other storage types, indexing in MongoDB data store, as well as other NoSQL data stores for further study and understanding the effectiveness of the proposed approach.

## **Appendix**

Ontology Details:

Entities or Classes

- Person A human being having ContactDetails and/ or other DemographicInfo details etc.
- 2. Patient A Person having Complaint, requirement of *TreatmentEpisode*.
- 3. Doctor A Person who treats a Patient for a Complaint
- 4. *Complaint* Any health problem requiring medical care

- 5. DemographicInfo Information related to a Person that identifies the person like, name, dob, gender etc. Address and ContactDetails
- 6. BasicClinicalParameter- bloodGroup, gender, dob etc. of a Person
- 7. *HistoryAll* All types of *MedicalHistory* (alias) of a *Patient*
- 8. *TreatmentEpisode* The beginning of a treatment episode for a *Complaint* of a *Patient*
- 9. *Visit* An encounter of a *Patient* with a *Doctor* when *diagnosisInV* many be taken.
- Investigation Laboratory, Pathology or Imaging test performed on a Patient on advise/ prescription of a Doctor
- 11. *ContinuousMonitoring* A type of of *Investigation* performed continuously for a duration
- 12. Sensor A physical device used for sensing
- 13. SensorOutput -obtained from Sensor
- 14. *AbstractedSensorData* contains details like parameter to be measured, unit of measurement etc.

Some Important Associations or Object Properties

- 1. hasDI -A Person has DemographicInfo details
- 2. containsBCPCD- DemographicInfo contains BasicClinicalParameter and ContactDetails.
- 3. containsAd ContactDetails contains Address
- 4. hasHA Patient has HistoryAll
- 5. hasC Patient has Complaint
- 6. contains AH History All contains all types of History including Allergy History, Immunization History, Family History etc.
- 7. begins Complaint begins a TreatmentEpisode
- 8. *containsV TreatmentEpisode* contains one or more than one *Visit*
- 9. *prescribes* -zero or more *Investigation* is prescribed in a *Visit*
- 10. containsASD ContinuousMonitoring contains AbstractedSensorData
- 11. uses AbstractedSensorData uses SensorOutput
- 12. comesFrom- SensorOutput is obtained from Sensor

Author contributions Both authors contributed equally.

Funding No funding has been received for this work

**Data availability** The datasets used and/or generated during the current study are available from the corresponding author on reasonable request.

**Code availability** The code written during the current study is available from the corresponding author on reasonable request.



#### **Declarations**

Conflict of interest The authors declare that they have no competing interests.

Ethical approval Not Applicable

Consent to participate Not Applicable

Consent for publication Consent Provided

## References

- Mukherjee, N., Bose, S., Ray, H.S.: A framework for delivering iot services with virtual sensors: Case study remote healthcare delivery. In: Interoperability in IoT for Smart Systems, pp. 137–152 (2020). Taylor & Francis Book Chapter, CRC Press. https://doi.org/10.1201/9781003055976-8
- Kichloo, A., Albosta, M., Dettloff, K., Wani, F., El-Amir, Z., Singh, J., Aljadah, M., Chakinala, R.C., Kanugula, A.K., Solanki, S., Chugh, S.: Telemedicine, the current covid-19 pandemic and the future: a narrative review and perspectives moving forward in the USA. Fam Med Community Health 8(3):e000530 (2020). https://doi.org/10.1136/fmch-2020-000530
- Moorthy, R., Udupa, S., Bhagavath, S.A., Rao, V., et al.: Centralized and automated healthcare systems: A essential smart application post covid-19. In: 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS), pp. 131–138 (2020). IEEE
- Sil Sen, P., Banerjee, S., Mukherjee, N.: Ontology for preliminary detection of covid-19. In: Information and Communication Technology for Competitive Strategies (ICTCS 2020), pp. 349–361 (2022). Springer
- Celesti, A., Lay-Ekuakille, A., Wan, J., Fazio, M., Celesti, F., Romano, A., Bramanti, P., Villari, M.: Information management in iot cloud-based tele-rehabilitation as a service for smart cities: Comparison of nosql approaches. Measurement 151, 107218 (2020). https://doi.org/10.1016/j.measurement.2019.107218
- Mehmood, N.Q., Culmone, R., Mostarda, L.: Modeling temporal aspects of sensor data for mongodb nosql database. Journal of Big Data 4(1), 8 (2017). https://doi.org/10.1186/s40537-017-0068-5
- Wang, L., Alexander, C.: Medical applications and healthcare based on cloud computing. International Journal of Cloud Computing and Services Science (IJ-CLOSER) 2 (2013). doi: https:// doi.org/10.11591/closer.v2i4.3452
- Council, C.S.C.: Impact of Cloud Computing on Healthcare Version 2.0. https://www.omg.org/cloud/deliverables/CSCC-Impact-of-Cloud-Computing-on-Healthcare.pdf (2017)
- Siddiqa, A., Karim, A., Gani, A.: Big data storage technologies: a survey. Frontiers Inf Technol Electronic Eng 18, 1040–1070 (2017). https://doi.org/10.1631/FITEE.1500441
- Mazumdar, S., Seybold, K.K.E.A.D.: A survey on data storage and placement methodologies for cloud-big data ecosystem. J. Big Data 6(15), 1040–1070 (2019). https://doi.org/10.1186/ s40537-019-0178-3
- Mukherjee, N., Neogy, S., Chattopadhyay, S.: Big Data in Ehealthcare: Challenges and Perspectives (1st ed.). Chapman and Hall/CRC Press (2019). https://doi.org/10.1201/9781351057790
- Schneider, T., Hashemi, A., Bennett, M., Brady, M., Casanave, C., Graves, H., Grüninger, M., Guarino, N., Levenchuk, A., Lucier, E., Obrst, L., Ray, S., Sriram, R., Vizedom, A., West, M., Whetzel, T., Yim, P.: Ontology for big systems: the ontology summit 2012 communiqu. Appl. Ontol. 7, 357–371 (2012). https://doi.org/10.3233/AO-2012-0111

- Zeshan, F., Mohamad, R.: Medical ontology in the dynamic healthcare environment. In: Proceedings of the 3rd International Conference on Ambient Systems, Networks and Technologies (ANT 2012), the 9th International Conference on Mobile Web Information Systems (MobiWIS-2012), Niagara Falls, Ontario, Canada, August 27-29, 2012, pp. 340–348 (2012). doi: https:// doi.org/10.1016/j.procs.2012.06.045
- Rhayem, A., Mhiri, M.B.A., Gargouri, F.: Healthiot ontology for data semantic representation and interpretation obtained from medical connected objects. In: 14th IEEE/ACS International Conference on Computer Systems and Applications, AICCSA, pp. 1470–1477. IEEE Computer Society. doi: 10.1109/ AICCSA.2017.171. https://doi.org/10.1109/AICCSA.2017.171
- Dang, J., Hedayati, A., Hampel, K., Toklu, C.: An ontological knowledge framework for adaptive medical workflow. J. of Biomed. Informatics 41(5), 829–836 (2008). https://doi.org/10. 1016/j.jbi.2008.05.012
- Lasierra, N., Alesanco, A., Guillén, S., García, J.: A three stage ontology-driven solution to provide personalized care to chronic patients at home. J. Biomed. Informatics 46(3), 516–529 (2013). https://doi.org/10.1016/j.jbi.2013.03.006
- Edoh, T.: Internet of things in emergency medical care and services. In: Farhadi, H. (ed.) Medical Internet of Things (m-IoT). IntechOpen, Rijeka (2019). Chap. 1. doi: https://doi.org/10.5772/intechopen.76974
- McMurray, J., Zhu, L., McKillop, I., Chen, H.: Ontological modeling of electronic health information exchange. J. Biomed. Informatics 56, 169–178 (2015). https://doi.org/10.1016/j.jbi. 2015.05.020
- Boshnak, H., Abdelgaber, S., Yehia, E., Abdo, A.: Ontology-based knowledge modelling for clinical data representation in electronic health records. Int. J. Comput. Sci. Information Security 16, 68–86 (2019)
- Atzeni, P., Bugiotti, F., Cabibbo, L., Torlone, R.: Data modeling in the nosql world. Comput. Stand. Interfaces 67, 103149 (2020). https://doi.org/10.1016/j.csi.2016.10.003
- Imam, A.A., Basri, S., Ahmad, R., Watada, J., Gonzaez-Aparicio, M.T.: Automatic schema suggestion model for nosql documentstores databases. J. Big Data 5(1), 46 (2018). https://doi.org/10.1186/s40537-018-0156-1
- Imam, A.A., Basri, S.B., Ahmad, R., Watada, J., Aparicio, M.T.G., Almomani, M.A.T.: Data modeling guidelines for nosql document-store databases. Int. J. Adv. Comput. Sci. Appl. 9, 2018 (2018)
- Mior, M.J.: Automated schema design for nosql databases. In: Proceedings of the 2014 SIGMOD PhD Symposium. SIG-MOD'14 PhD Symposium, pp. 41–45. Association for Computing Machinery, New York, NY, USA (2014). doi: https://doi.org/10.1145/2602622.2602624
- Daniel, G., Gomez, A., Cabot, J.: Umlto[no]sql: Mapping conceptual schemas to heterogeneous datastores. 2019 13th International Conference on Research Challenges in Information Science (RCIS), 1–13 (2019). doi: https://doi.org/10.1109/RCIS. 2019.8877094
- Moditha Hewasinghage, A.A. Sergi Nadal: DocDesign 2.0: Automated database design for document stores with multi-criteria optimization. https://upcommons.upc.edu/handle/2117/343145 (2020)
- Mior, M.J., Salem, K., Aboulnaga, A., Liu, R.: Nose: Schema design for nosql applications. IEEE Trans. Knowl. Data Eng. 29, 2275–2289 (2017)
- Sen, P.S., Mukherjee, N.: Ontology-based data modeling for nosql databases: A case study in e-healthcare application. SN Comput. Sci. 4(1), 3 (2023). https://doi.org/10.1007/s42979-022-01405-5



- Pore, A.: How to Choose the Right NoSQL Database for Your Application? urlhttps://www.dataversity.net/how-to-design-schema-for-your-nosql-database/ (2018)
- Sen, P.S., Banerjee, S., Mukherjee, N.: Ontology-driven approach to health data management for remote healthcare delivery. In: Proceedings of the 7th ACM Workshop on ACM Mobile Health 2017. MobileHealth'17 (2017). doi: https://doi.org/10.1145/ 3084035.3084039
- of India, G.: Electronic Health Record Standards for India, August 2013. https://www.nhp.gov.in/NHPfiles/ehr\_2013.pdf (2013)
- Basra, R.: EHR Standards for India: A beginning of Integrated Indian Healthcare System. https://healthfore.wordpress.com/ 2013/11/13/ehr-standards-for-india-a-beginning-of-integrated-indian-healthcare-system/ (2013)
- 32. WHO: Integrated Management of Neonatal and Childhood Illness (2003). https://www.jknhm.com/zip/IMNCI%20%20Students'%20Handbook.pdf
- Bose, S., Gupta, A., Adhikary, S., Mukherjee, N.: Towards a sensor-cloud infrastructure with sensor virtualization, pp. 25–30 (2015). doi: https://doi.org/10.1145/2757743.2757748

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the

accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Poly Sil Sen received her M.E. degree in Information Technology from the Maulana Abul Kalam Azad University of Technology, West Bengal, India, in 2007. Since 2022, she has been Assistant Professor in the Department of Computer Science & Technology and Computer Science & Information Technology, University of Engineering and Management, Kolkata, India. She worked as the Assistant Professor of Information Technology, Techno Main Salt Lake from 2009 to 2022. Her research areas include health data storage and management, ontology and NoSQL schema design.

Nandini Mukherjee received the Ph.D. degree in computer science from the University of Manchester, Manchester, U.K., in 1999. She was the recipient of a Commonwealth Scholarship for her doctoral study in the U.K. She served in industry for nearly three years. She has been a faculty member with the Department of Computer Science and Engineering, Jadavpur University, Kolkata, India, since 1992, where she is currently a Professor and Head of the Department. She has served as the Director of the School of Mobile Computing and Communication, Jadavpur University for almost six years. She is an active researcher in her research areas, which currently include high performance parallel computing, wireless sensor networks and Internet of Things. Prof. Mukherjee is a Senior Member of the IEEE Computer Society and a member of ACM.

