Data Modeling Guidelines for NoSQL Document-Store Databases

Abdullahi Abubakar Imam^{1,a,b}, Shuib Basri^{2,a}, Rohiza Ahmad^{3,a}, Junzo Watada^{4,a}, Maria T. Gonzlez-Aparicio^{5,c}, Malek Ahmad Almomani^{6,a}

^{a}CIS Department, Universiti Teknologi PETRONAS, Bandar Seri Iskandar, 31570, Perak, Malaysia
^{b}CS Department, Ahmadu Bello University, Zaria-Nigeria
^{c}Computing Department, University of Oviedo Gijon, Spain

***Abstract*—Good database design is key to high data availability and consistency in traditional databases, and numerous techniques exist to abet designers in modeling schemas appropriately. These schemas are strictly enforced by traditional database engines. However, with the emergence of schema-free databases (NoSQL) coupled with voluminous and highly diversified datasets (big data), such aid becomes even more important as schemas in NoSQL are enforced by application developers, which requires a high level of competence. Precisely, existing modeling techniques and guides used in traditional databases are insufficient for big-data storage settings. As a synthesis, new modeling guidelines for NoSQL document-store databases are posed. These guidelines cut across both logical and physical stages of database designs. Each is developed based on solid empirical insights, yet they are prepared to be intuitive to developers and practitioners. To realize this goal, we employ an exploratory approach to the investigation of techniques, empirical methods and expert consultations. We analyze how industry experts prioritize requirements and analyze the relationships between datasets on the one hand and error prospects and awareness on the other hand. Few proprietary guidelines were extracted from a heuristic evaluation of 5 NoSQL databases. In this regard, the proposed guidelines have great potential to function as an imperative instrument of knowledge transfer from academia to NoSQL database modeling practices.**

Keywords*—*Big Data; NoSQL; Logical and Physical Design; Data Modeling; Modeling Guidelines; Document-Stores; Model Quality

I. Introduction

With the rise in data sizes, types and rates of generation, i.e., big data, traditional datastores have become less capable for many reasons, such as structural rigidity and untimely response due to high access latency [1], [2], [3], [4], [5]. This unacceptable performance has led to a reevaluation of how such data can be efficiently managed in a new generation of applications where performance and availability are paramount [5], [6]. As a result, NoSQL (Not Only SQL) databases were introduced to augment the features of Traditional Databases (TD) with new concepts such as schema flexibility, scalability, high performance, partition tolerance, and other new extended features [7]. The schemas of such databases are enforced by client-side application developers rather than database engines, as in the case of TD [2], [8].

Consequently, several giant organizations, such as Google, Facebook, and Amazon, have adopted NoSQL technology for data management and storage [5]. However, the inherent complexity and unpredictable nature of todays data [9], along with the low competence level of data modelers [3], [10], [11], developer autonomy [1], [12] and inadequate modeling guidelines [13], have posed numerous challenges in NoSQL schema best-practice implementation. This has increasingly led to erroneous database modeling and designs [1], [14], [15], [16], [17], which defeats the notion of robustness in NoSQL databases and results in the production of low-performance, non-secure and less-durable systems.

For example, consider the security aspect of NoSQL document-oriented databases. The databases offer a query language or an Application Program Interface (API) that has the ability to retrieve the contents of any document in a collection. These APIs, although they provide flexibility in data access across heterogeneous platforms, can be used as breaking points by hackers when incorrectly implemented [18], [19]. Recently, Flexcoin, a United States bank, was attacked, and more than a half-million USD was lost [20]. In addition, an airport was completely shut down due to a system failure [21] in the UK, resulting in several flight cancelations. These tragic events were strongly attributed to improper database design, as discussed in Section 3. However, some of the latest reported security breaches are as follows: 1) schema: because of its flexibility, mere record insertion can automatically create a new schema within a collection, 2) queries: unsafe queries can be created via string concatenation, and 3) JavaScript (JS): the clause of db.eval(), $where takes in JS functions as parameters [18]. Such types of issues are what drew the attention of researchers to provide viable and substantial solutions. However, many of the solutions come in as testing tools for already developed databases [4], [22], [23] or are proprietary [10], [17], [24], [25], which opposes our understanding that the solutions should come at the earliest stage of design (data modeling). Clearly, there is a need for a standard guide in practice.

As such, a set of NoSQL modeling guidelines for the logical and physical design of document-store databases is proposed. In these guidelines, all possible relationships are retrieved, analyzed, categorized and prioritized. The resulting guidelines are expected to serve as an important tool of knowledge for beginners, intermediates or even advanced NoSQL database developers. For the actualization of this goal, we employ an exploratory approach for the investigation of existing works, empirical methods and expert consultations. We analyze how industry experts prioritize the guidelines and analyze the relationships between datasets on the one hand and error prospects and awareness on the other hand. Few proprietary guidelines were extracted and harmonized from a heuristic evaluation of 5 different existing NoSQL databases. In this regard, the proposed guidelines have great potential to function as an imperative instrument of knowledge transfer from academia to NoSQL database modeling practices.

The remainder of this paper is structured as follows. Section II reviews and analyzes existing works. Section III puts forward the proposed guidelines and their application scenarios. Section IV prioritizes guidelines in 3 different categories. Section V discusses the findings (limitations and potentials). Finally, Section VI concludes and highlights the future focus.

II. Related Works

The origin of Data Modeling (DM) in databases can be traced back to the mid-20th century as a technique for structuring and organizing data [33]. The exercise is astonishingly similar to construction designs where walls are planned, flows are optimized, and materials are chosen based on the type of utility that it will accommodate and the level of interaction needed between sections [34]. DM gained the attention of researchers in the field of information systems and data visualizations in the 1970s (see [35], [36]). In the late 1990s, a Unified Modeling Language (UML) [34] was introduced to consolidate the data modeling symbols and notations invented by [35], [36] into one standardized language, all for the purpose of simplifying data visualization and modeling in relational databases.

Now, with the emergence of unstructured, voluminous and complex datasets, i.e., big data, requirement to have more flexible and higher-performance databases have become essential [27], [28], [33], [37], which has given rise to the concept of NoSQL databases. The high flexibility of NoSQL databases makes data modeling even more challenging, as schemas are written and enforced by the client-side application developers rather than database engines, as in the case of RDBMS [12], [38], [26], [29]. This raises the question of competence, which may lead to the production of high- or low-quality models [10], [12]. A recent report by [20] shows how a low level of competence in NoSQL data modeling cost a United States-based company called Flexcoin a half-million US dollars. A hacker was able to make several transactions before the account-balance-document was updated (low consistency). In another case, an airport was completely shut down as a result of a major IT system failure in London [21], for which the experts assigned the blame to the poor back-end system design. These are officially reported instances, while several other cases, such as those discussed in [39], [40], do exist.

To mitigate these challenges, experts shared their experiences on the most common questions asked by the client-side application developers online. Some of these questions are (i) how to model one-to-N relationships in document databases, (ii) how to know when to reference instead of embedding a document, and (iii) whether document databases allow Entity Relationship modeling at all. In an attempt to address these and similar questions, experts highlighted the necessity of having a standardized modeling guide for these powerful data stores [10], [12], [17], [30]. This is partly because many of the questions keep reappearing repeatedly on multiple platforms or even the same platform.

In the words of William (Lead Technical Engineer at MongoDB) [10], guidance is strongly required for MongoDB developers, upon which few guidelines were produced to ease the modeling process. Moreover, Ryan CrawCuor and David Makogon [17] created a comprehensive presentation on how to model data in JSON. In addition, eBay [24] and Netflix [25] produced some guidelines for schema design in Cassandra. However, these guidelines, though comprehensive, are complex and designed for the referenced databases only, i.e., they are proprietary. Consequently, straightforward and more general guidelines are needed in practice.

In [8] and [12], reuse of existing modeling expertise (from RDBMS) is allowed to minimize the high level of competence required to model NoSQL databases. This was achieved using Idef1X (a standard data-modeling language) and Formal Concept Analysis (FCA). However, an experiment conducted by [13] evidently showed the limitation of the existing modeling expertise when applied to new-generation complex datasets (big data). Clearly, NoSQL databases need a different modeling approach to efficiently manage big data due to its diverse characteristics [32].

In [1], a cost-based approach for schema recommendation is proposed with the aim of replacing the rules of thumb currently followed by less competent NoSQL database designers. In this approach, the expected performance of the target application is estimated, upon which a candidate schema is recommended. The approach made schema modeling more stable and secure than before. However, more stages are added to the design processes, such as data analysis, application of the tool to propose schema, and then translation of the schema into real application. Moreover, the approach is applicable to column family databases only. In addition, the tool focuses only on the expected performance of candidate schema, despite the fact that NoSQL schema design is largely driven by the nature of the target data [16]. Alternately, an interactive, schema-on-read approach was proposed in [41] for finding multidimensional structures in document stores. [42] proposed a data migration architecture that migrates data from SQL to NoSQL document-stores while taking into account the data models of both the two categories of databases. Although these approaches yielded relatively good findings, more generic, simple, and data-driven guidance prepared for at least one category of NoSQL databases [12], [31], [32] is still needed for practitioners.

The heterogeneity of todays systems, data complexity growth, and lack of modeling expertise have been stated as motivations of the aforementioned works. These claims have been confirmed by error-rate reports [20], [21], [39], [40] in real-world of NoSQL-driven projects. Undoubtedly, there is a need for well-founded guidelines in practice. The following section presents the proposed guidelines, which were synthesized from empirical research and professional involvements.

III. Proposed Guidelines

In this section, the proposed guidelines which were synthesized from empirical work are introduced. The section is divided into four subsections. In Section 3.1 an example model from university social media networking system is described which was used for this research. Section 3.2 highlights, in summary, the empirical research upon which the proposed guidelines are built. Section 3.3 presents the guidelines and their respective explanations. Section 3.4 shows how the proposed guidelines can improve the model presented in Section 3.1.

A. An Example Model

To illustrate the proposed guidelines, a running example shown in Fig. 1 was used. The model describes entities and their connections of a university social media networking system which was developed by the university programmers. The modeling was done without considering the proposed guidelines and, as will be seen later, will improve when the proposed guidelines are applied.

The model shown in Fig. 1 follows the Entity Relationship Diagram (ERD) notations and symbols proposed by [35] and [36] which are the most popular relational database modeling technique in both the academia and industry.

Although a Unified Modeling Language (UML) [43] was introduce to standardize approaches and notations, the model in Fig. 1 adopted few fundamental symbols and notations from [35], [36], [43] for demonstration purposes. Rectangle, arrows, and curly and square brackets were used to show, conceptually, the activity flow. Generally, in ERD, rectangles are used to indicate entities while arrows correspond to data flows or connections between the entities. Moreover, notations such as curly and square brackets were used to indicate attribute and arrays of keys respectively.

The given model in Fig. 1 roughly describes a user entity and user-dependent entities. A user has direct entities such as contact info, basic info, friends and family, messages, and education and work. Each of these entities has other sub-entities which, as the tree expands; many entities repeatedly appear in different parent entities. For example, likers and commenters entities contain the same list of people as in friends & family entity. Furthermore, the list of people in friends and family entity are also the system users who are recorded in the User entity. Now, these repetitions might improve data availability but at the expense of consistency or speed during inserts, updates or deletes. This will be further explained later when the model in Fig. 1 is improved using our guidelines.

B. Empirical Research Background

The research background upon which the proposed guidelines are defined is described in this section. The widely acceptance and adoption of ERD model in relational databases is connected with its ease of comprehension and application onto structured datasets. In prior research, we thoroughly investigated the new generation datasets (big data) while taking into account the connection between NoSQL databases and the factors leading to their comprehension and proper modeling. Factors such as understanding, error probability, and ambiguity are experimented as well as other factors that motivated the guidelines propositions. The findings are summarized as follow.

- Understanding relates to the degree to which datasets and system requirements can be easily understood. Its a strong basis on which data is classified, categorized and modeled. In an experiment reported in [13] we introduced new cardinality notations and relationship styles for NoSQL databases. From our engagement with programmers regarding the new notations and

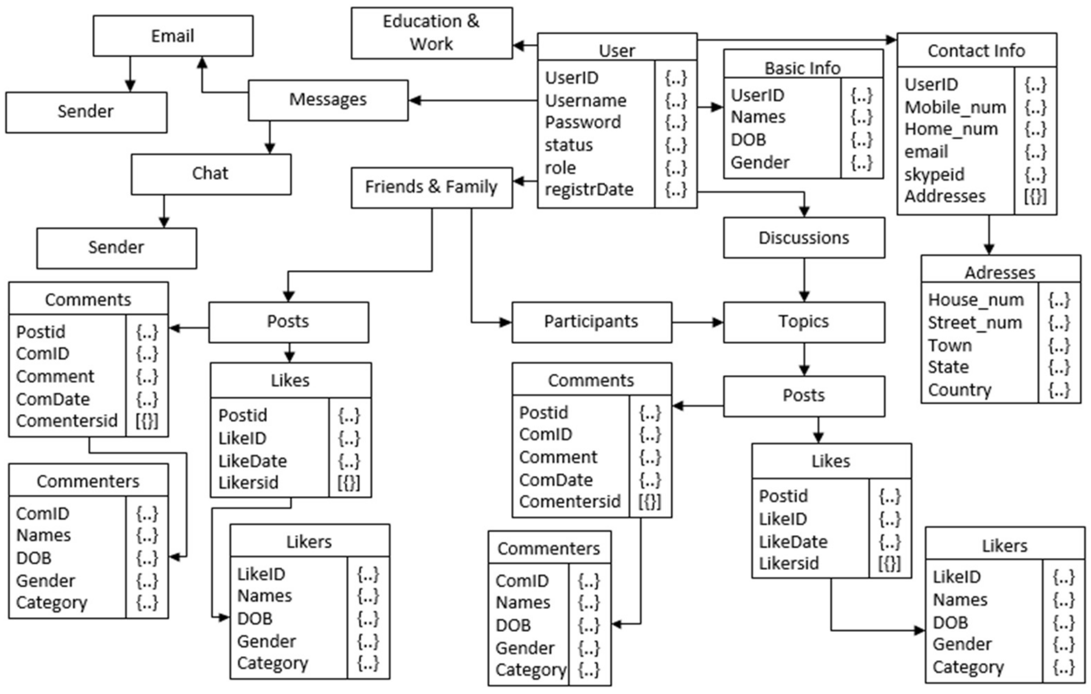

Fig. 1. A section of social media database model.

styles, we found a negative correlation with understanding how the new notations should be best implemented.

- Error probability in our case refers to the extent a programmer is able to classify datasets based on the new notations and styles without introducing errors. In a different experiment we have modeled the new notations using MongoDB database [44] while focusing on availability and consistency as the measurement factors. The results of this experiment trace error probability back to lack of understanding of data sets, knowledge of modeling, and expertise. We found that, modeling expertise and datasets complexity are the most important drivers to error probability in NoSQL database modeling.
- Ambiguity in relationships between datasets and system/business requirements are an important road block to the understanding NoSQL database design structure. As such, it was observed during our experiment that, the notations (*1:1*, etc.) used in relational databases have been in practice for decades which shows the level of knowledge of SQL modeling notations among practitioners; therefore, similar grammatical representation was adopted and extended for the new NoSQL notations (*1:F*, etc.) [13]. Out of 14 postgraduate students from University Technology PETRONAS, 12 students said that using similar notations will lessen the ambiguity as the focus would be on datasets analysis rather than introducing entirely new notations.
- Styles Application Scheduling (SAS) captures the awareness of most appropriate time to implement any of the modeling style like embedding, referencing or bucketing when modeling NoSQL database. We conducted an experiment on each of the modeling styles and found that, even though they are less ambiguously introduced, knowledge of when to apply which modeling style is still missing.
- Guidelines extraction from a heuristic evaluation of five different NoSQL databases [45]. We extracted the available modeling guidelines written by the technical team of databases such as MongoDB, Couchbase, Google Cloud Datastore, CouchDB, and MarkLogic. The extracted guidelines were harmonized and generalized for document-store databases.
- Expert consultations from three different SME companies across the globe were involved. In total, 9 different industry experts were requested to critically scrutinize and make recommendations on the proposed guidelines. The experts consisted of one independent programmer from Sweden, one researcher from Spain, two from Software Development Company (SDC) in Malaysia, two from Software Development Committee (SDC) Ahmadu Bello University Nigeria and three postgraduate students from Universiti Teknologi PETRONAS, Malaysia. 2919

Based on these six empirical insights into NoSQL relationship modeling, we define the proposed guidelines as presented in the next section.

C. The Guidelines

The proposed guidelines provide a set of recommendations on how to develop NoSQL document-store databases, each of which builds on empirical research [13], [45] summarized in the previous section. Modeling NoSQL databases has noticeably become more challenging as data increases in volume, variety and velocity, i.e. big data [10], [12], [17], [22]. The aim of these guidelines is to ease data modeling process by improving developers skills towards modeling document-store databases. This is hoped to maximize data retrieval and storage efficiency, minimize erroneous modeling, and reduce the time taken to model database, as well as improve data security. Hence, it is important to note that, the proposed guidelines build on insight which might be described differently using a different approach.

For better understanding and quick mapping, the proposed guidelines were categorized into four different categories as illustrated in Fig 2. This includes embedding, referencing, bucketing and general. In each of the categories, there is at least one important note which should be taken into account.

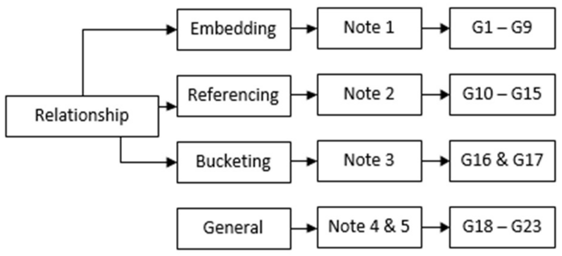

Fig. 2. Categorization of guidelines

At first, the notes as presented in Fig 2 are elaborated, which were followed by the proposed guidelines.

Note 1: Characteristically, embedding provides better read performance when retrieving data from document-store databases. However, when writing data to database, it can be exceedingly slower, unlike referencing which uses the concept of writing data horizontally into smaller files.

Note 2: Typically referencing provides better write performance. However reading data may require more round trips to the server.

Note 3: Bucketing enhances data retrieval by partitioning document with large contents into smaller affordable sizes.

Note 4: Normalizing data may help to save some space, but with the current advancement of technology, space is not a problem anymore.

Note 5: Finally, understand the data access patterns, the nature of the data to be used in the application, the rate of updates on a particular field, and the cardinality relationships between entities. Such information shapes the design and modeling structure of document-store databases.

The proposed guidelines are as follows. They adhere to the categorizations as depicted in Fig. 2. In the beginning, embedding is put forward.

1) Embedding: This section presents the first set of the proposed guidelines (G1 — G9) which aim to answer questions related to embedding (i.e. insertion of one document into another).

G1: Embed sub documents unless forced otherwise: For better system performance in terms of saving and retrieving speed, try to always embed child documents except when it is necessary to do otherwise. With embedding, there is no need to perform a separate query to retrieve the embedded documents [7].

G2: Use array concept when embedding: It is recommended to use array of embedded documents when modeling few relationships [10], [17].

G3: Define array upper bound in parent document: Avoid the use of unlimited array of ObjectID references in the many side of the relationship if it contains a few thousands of documents [17].

G4: Embed records which are managed together: when records are queried, operated and updated together, they should be embedded [13].

G5: Embed dependent documents: dependency is one of the key indicators to embed a document [17]. For example, order details are solely dependent to the order itself; thus they should be kept together.

G6: Embed one-to-one (explained in [13]) relationships: when modeling one-to-one relationship, embedding style should be applied.

G7: Group data with same volatility: data should be grouped based on the rate to which it changes [13]. For example, persons bio-data and status of several social media accounts. The volatility of social media status is higher than the bio-data which does not change quite often like email address or does not even change at all, explicitly, date of birth.

G8: Two-way embedding is preferred when *N* size is close to the *M* size in *N:M* relationship (presented in [13]): in *N:M* relationship, try establish a relationship balance by predicting the maximum number of *N* and maximum number of *M* [7], [13]. Two-way embedding is preferred when the *N* size is close to the *M* size.

G9: One-way embedding is preferred if theres a huge gap in size between *N* to *M*: if gap is for example 3 in N side and 300,000 in *M* side, then one-way embedding should be considered [13].

2) Referencing: Referencing can be explained as a process of connecting two or more documents together using a unique identifier [13]. The following guidelines (G10 G15) aim to answer questions related to referencing.

G10: Reference highly volatile documents: high volatility of document gives a good signal to reference a document instead of embedding. For example, lets consider a post made on a social media (Fig. 1), the likes tag changes so often; thus unbound it from the main document so that the main document is not always accessed each time likes button is hit.

G11: Reference standalone entities: avoid embedding a child document/object if it will be at one time accessed alone. Documents, when embedded, cannot be retrieved alone as a single entity without retrieving the main entity [10].

G12: Use array of references for the many side of the relationship: when a relationship is one-to-many as in [13] or a document is a standalone document, array of references are best recommended.

G13: Parent referencing is recommended for large quantity of documents: for instance, when the many side of a relationship is squillion (introduced in [13]), parent-referencing is preferred.

G14: Do not embed sub-documents if they are many: a key entity with many other sub-entities should adopt referencing rather than embedding [13]. This will minimize high-cardinality arrays [41].

G15: Index all documents for better performance: If documents are indexed correctly and projection spacefarers like the relationship styles discussed in [13] are used, the applications level joins are nothing to be worried about.

3) Bucketing: Bucketing refers to splitting of documents into smaller manageable sizes. It balances between the rigidity of embedding and flexibility of referencing [13].

G16: Combine embedding and referencing if necessary: embedding and referencing can be merged together and work perfectly [10]. For example, consider a product advert on Amazon website, there is the product information, the price which may change, and a list of comments and likes. This advert actually combines reasons to embed as well as to reference, thus merging the two techniques together can be the best practice in this case.

G17: Bucket documents with large content: to split a document into discreet batches such as days, months, hour, quantity etc, bucketing should be considered [13]. For example, the squillions (introduced in [13]) side of the relationship can be divided into 500 records per display as the case of pagination.

4) General: There are few guidelines that do not fall into any of the earlier discussed categories (embedding, referencing and bucketing). Such guidelines are grouped and presented as follows.

G18: Denormalize document when read/write frequency is very low: denormalize document only if it is not updated regularly. So, access frequency prediction should guide the decision to denormalize any entity.

G19: Denormalize two connected documents for semi-combined retrievals: Sometimes two documents are connected, but only one is to be retrieved and few fields from the second document, denormalization can help here [13]. For example, when retrieving a presentations session, a speakers name would need to be displayed as well but not all the speakers details, so, the second document (speaker) is denormalized to get only the name of the presenter and attach it to session document.

G20: Use tags implementation style for data transfer: if information is not sensitive, packaging it within tags like in XML document is recommended [46].

G21: Use directory hierarchies if security is a priority: apply role based authorization to each of the directories for access protection [19]. A user can have the privilege to access one directory or a collection of directories, depending on the users role.

G22: Use documents collections implementation style for better read/write performance: this is the same as G21, but with addition of better read/write performance.

G23: Use Non-visible metadata for data transfer between nodes or servers: in many cases, APIs dont have security mechanisms embedded in them [47]. So, encoding sensitive information before transfer and decoding upon arrival is strongly recommended. This will improve data security on the air.

TABLE I. Overview of the Proposed Guidelines

Gl	Embed sub-documents unless forced otherwise
G2	Use array concept when embedding
G3	Define array upper bound in parent document
G4	Embed records which are managed together
G5	Embed dependent documents
G6	Embed one-to-one relationships
G7	Group data with same volatility
GS	Two-way embedding is preferred when N size is close to the M size in N:M relationship
G9	One-way embedding is preferred if there's hug gap in size between N to M
G10	Reference highly volatile document
G11	Reference standalone entities
G12	Use array of references for the many side of the relationship
G13	Parent referencing is recommended for large quantity of entities
G14	Do not embed sub-documents if they are many
G15	Index all documents for better performance
G16	Combine embedding and referencing if necessary
G17	Bucket documents with large content
G18	Denormalize document when read write frequency is very low
G19	Denormalize two connected documents for semi-combined retrievals
G20	Use tags implementation style for data transfer
G21	Use directory hierarchies if security is a priority
G22	Use document collections implementation style
G23	Use Non-visible metadata for data transfer between nodes or server

The following section explains the application of the aforementioned guidelines.

D. Application

To demonstrate the proposed guideline, we will show how the original social media model (Fig. 1) can be transformed into a more stable model. In Fig. 3, we marked and labeled some areas of improvement on the same model using guideline identifies. A transformed model is presented as in Fig. 4 which results from the application of the proposed guidelines. The application of each of these guidelines is explained as follows.

In the original model, some modeling problems were identified such as too much redundancy of information which of cause leads to inconsistencies among entities. For example, there exists an entity of user which contains some information about users, this entity is fully repeated in places like "family & friends", "commenters", "likers" etc. in different branches of the model. The problem with this approach is that, updating a single attribute for instance will require updating all documents with the same attribute. Now, in a situation where an attribute changes so frequently and the affected documents are many, more serious issues like inconsistency, temporary insecurity (for access authorization) and performance deterioration may arise. Such events motivated many guidelines such as **G1** which recommends the embedment of all documents or **G6** which recommends embedding of a single document. To maintain the availability provided by duplicating users data even when its embedded into "User" entity, **G17** came in to take the few rarely changed attributes from the main document to the areas where they are accessed quite often. However, as "User" entity is bucketed, referencing became required (**G11**). Similarly, higher volatile documents like "Discussions" and "Posts" were bucketed from "User" entity and grouped based on the rate to which they change (**G7**) which allows write/update operations without necessarily accessing the parent documents. Also, **G11** was considered for independent access of "Discussions" and "Posts" since they may be accessed alone in most cases.

While referencing related documents, **G2** was used, which states the use of array concept when referencing documents using their IDs especially in the M (many) side of the relationship (**G12**), besides, upper bound was defined for any array of IDs (**G3**). But in a situation of a large number of entities like "comments", the spirit of parent referencing (**G13**) was followed.

By referring to the original model, since the write frequency is very high in the entities of "comments" and "likes", embedding was avoided (**G14**), instead we denormalized "commenters" and "likers" entities (**G19**) and reference each of them (**G10**) such that embedding and referencing can be combined (**G16**) using only the commenters name and ID, leading to achieving both availability and consistency at the same time. The rationale behind this is that, only commenters or likers name is often required for each comment or like. Therefore, for high availability, only the name of a user should be denormalized and for the consistency during updates, array of IDs can be used for more round trips.

On the section of "User" entity again, Basic Info" and "Contact Info" are not only dependent to "User" entity but they are also managed together. Since such information has low read/write frequencies (**G18**), putting them together based on their collectivity in management (**G4**) or based on their dependencies on one another (**G5**) will significantly minimize the number of round trips to the server for a single update. Given that, the predicted records for all the three entities ("Basic Info", "contact info" and "User") are almost at same level, two-way embedding is best recommended (**G8**) to permit connection from either directions. But in the case of "Posts"

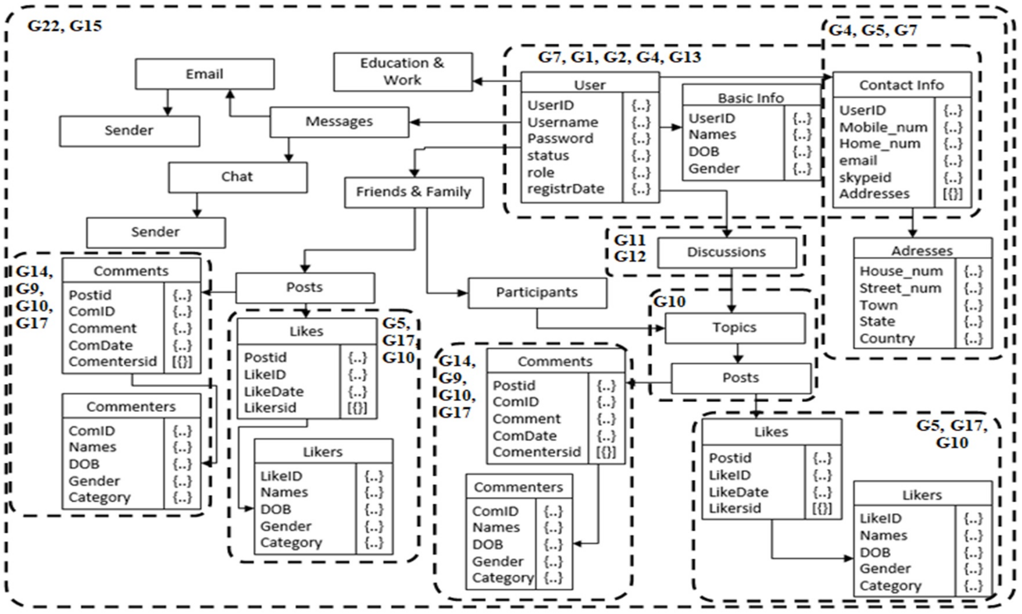

Fig. 3. A section of social media database including markers

and "Likes", One-way embedding is most preferred (**G9**) since the number of "likes" can be more than a million for a particular post.

In view of the fact that, performance is usually a priority requirement, indexing all documents (**G15**) is strongly recommended. Also, considering the node balance challenge posed by hierarchical data modeling style, document collection implementation style (**G22**) is maintained for many reasons such as horizontal schema flexibility as database scales up and down.

Although it is not frequently used, interfacing possibilities (data exchange) with other applications is an important aspect to consider right from modeling stage to avoid using proprietary data export format, **G20** proposes the use of tags formatting style such as XML which is open source and can be formatted (**G23**) and read by almost all programming languages. In many cases, web-services are allowed to determine everything including using special characters; this flexibility creates security vulnerabilities such as NoSQL injections via restful APIs. Such high expectations of security breaches motivated the use of hierarchical data modeling (**G21**) which ease the application of role based authorization on each node

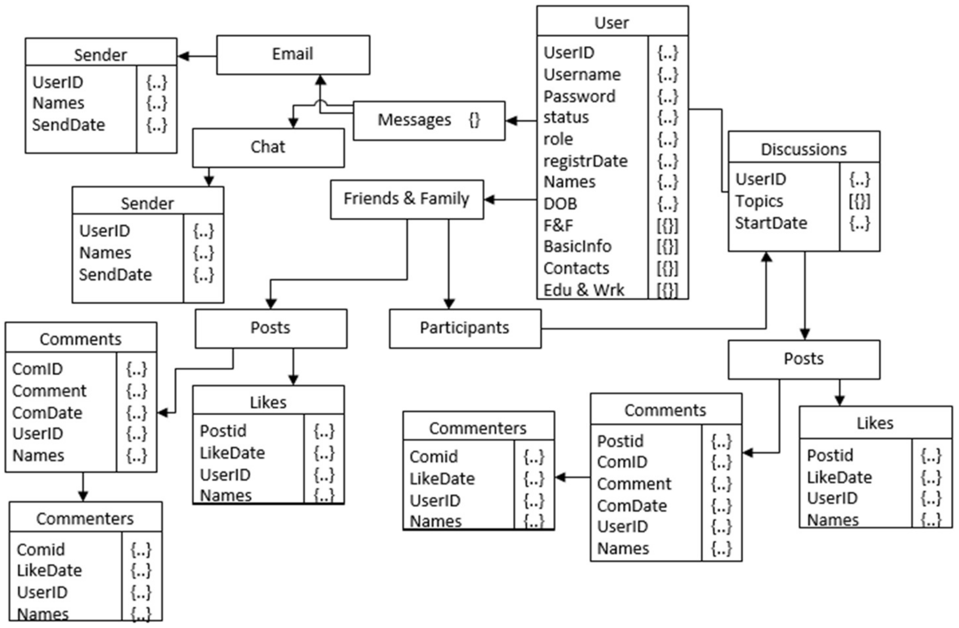

Fig. 4. The optimized model

of the tree, or **G22** which clusters documents into collection of documents at different stages.

It is important to note that not all the guidelines are applicable to the original model, some guidelines such as **G20** - **G23** were exemplified in a more generic way. This is because the original model did not interface with other models or applications. Also, the overall number of entities has been reduced from 24 in the original model to 17 in the transformed model as a result of prioritizing guidelines such as **G1**, **G4**, **G5**, **G7** etc. In summary, the original model is restructured and transformed to less redundant model with high availability and consistency without changing the model behavior.

IV. Prioritizing Guidelines

In the preceding section, we illustrated how each element of the proposed guidelines can be applied on a real datasets. However, in a situation where two or more guidelines are applicable, the modeler needs to be guided towards taking the most appropriate direction based on system requirements. For instance, while embedding dependent documents **(G7)** increases read/write performance, requirement to access document independently may necessitate referencing standalone entities **(G11)** or bucketing the frequently accessed entities **(G15)** into affordable elements. This is because, embedded child document cannot be retrieved alone without retrieving the parent document [32]. This situation is explained in the previous section, which clearly demanded more sensible priorities when applying the proposed guidelines.

Its important to note that, in as much as we tried to simplify the guidelines; their diverse nature significantly increases the challenge of resolving conflicts between them. For a given model, many conflicting guidelines can be applicable in one section, and many sections can adopt one guideline.

The scope of this paper does not include a more comprehensive prioritization which is theoretically motivated and empirically validated. Nevertheless, we have taken the following approach to arrive at some guidance on guideline application prioritization. At first, a presentation of the proposed guidelines was made to experts in Universiti Teknologi PETRONAS, Malaysia which led to comprehensive refinement of the guidelines. Secondly, SQL and NoSQL professionals in our network were contacted to take part in reviewing, analyzing and prioritizing the guidelines based on their expert opinions, this include five experts from Malaysia one from Sweden, one from Spain, and two from Nigeria. A total of nine professionals with an average modeling experience of 4 years complied with our request and assisted in refining the guidelines and prioritizing their application under different circumstances.

Each of the professionals contacted, receives a verbal or written presentation of the proposed guidelines from the researchers. After that, all professionals were asked to individually review each guideline and add or remove to/from the list. Next, each professional was also asked to rank the refined proposed guidelines with respect to three different categories, namely, availability (read operation), consistency (write and update operations) and cardinality notations using a scale of 1-23. For this scale, the ranking begins from 1 which indicates a perception of being the highest relative potential, while rank of 23 in the scale indicates the lowest relative potential. This inquiry guided the researchers to infer a priority scheme which can resolve conflicts among rival guidelines.

While ranking the guidelines, all participating experts were allowed to give equal rank to more than one guideline. However, for each participant, a constraint of a total number of 276 (= 1 + 2 + 3 + 4 + 23) assigned ranks was expected.

These assigned ranks were accumulated per guideline leading to results as presented in Table II. It can be seen from this table that, **G6** is considered to have the highest potential to improve data availability, as it has total rank scores of 12.

While, **G21** is deliberated to have the least potential to improve data availability with an average score of 202. The total scores of the remaining guidelines fall between these extremes.

On the other hand, since availability is not always a priority of all systems [4], the prioritization also considered an important database concept which is consistency in replicated, connected or dependent data. As such, another set of priority

TABLE II. Prioritizing Guidelines based on Availability (Read Operations)

No	Description	Total Scores (Rank)	Priority Level
G6	Embed one-to-one relationships	12	1
G1	Embed sub-documents unless forced otherwise	16	2
G17	Bucket documents with large content	30	3
G15	Index all documents for better performance	34	4
G2	Use array concept when embedding	52	5
G7	Group data with same volatility	54	6
G11	Reference standalone entities	67	7
G9	One-way embedding is preferred if there's hug gap in size between N to M	69	8
G3	Define array upper bound in parent document	6	9
G19	Denormalize two connected documents for semi-combined retrievals	92	10
G5	Embed dependent documents	97	11
G4	Embed records which are managed together	106	12
G22	Use document collections implementation style	123	13
G8	Two-way embedding is preferred when N size is close to the M size in N:M	I26	14
G12	Use array of references for the many side of the relationship	129	I5
G10	Reference highly volatile document	141	16
G13	Parent referencing is recommended for large quantity of entities	I57	17
G14	Do not embed sub-documents if they are many	161	18
G18	Denormalize document when read write frequency is very low	168	19
G23	Use Non-visible metadata for data transfer between nodes or server	186	20
G16	Combine embedding and referencing if necessary	I87	21
G20	Use tags implementation Style for data transfer	199	22
G21	Use directory hierarchies if security is a priority	202	23

TABLE III. PRIORITIZING GUIDELINES BASED ON CONSISTENCY (WRITE & UPDATE OPERATIONS)

No	Description	Total Scores (Rank)	Priority Level
G1	Embed sub-documents unless forced otherwise	17	1
G6	Embed one-to-one relationships	25	2
G4	Embed records which are managed together	28	3
G5	Embed dependent documents	35	4
G7	Group data with same volatility	50	5
G18	Denormalize document when read write frequency is way low	70	6
G10	Reference highly volatile document	79	7
G11	Reference standalone entities	91	8
G14	Do not embed sub-documents if they are many	93	9
G12	Use array of references for the many side of the relationship	95	10
G8	Two-way embedding is preferred when N size is close to the M size in X:M	99	11
G15	Index all documents for better performance	101	12
G3	Define array upper bound in parent document	103	13
G9	One-way embedding is preferred if there's hug gap in size between N to M	118	14
G13	Parent referencing is recommended for iarge quantity of entities	125	15
G16	Combine embedding and referencing if necessary	136	16
G2	Use array concept when embedding	138	17
G21	Use director.' hierarchies if security is a priority	152	18
G19	Denormalize two connected documents for semi-combined retrievals	169	19
G23	Use Non-visible metadata for data transfer between nodes or server	173	20
G22	Use document collections implementation style	190	21
G20	Use tags implementation style for data transfer	196	22
G17	Bucket documents with large content	201	23

list was debated; results of which is shown in Table III. This table suggests that, in consistency, **G1** has the highest potential to improve consistency among different clusters, documents or datasets as it has an accumulated score of 17 ranks. In contrast, **G17** is considered to have the lowest potential to do so with an accumulated score of 201 ranks. The remaining guidelines fall between the two extremes.

In addition to prioritizing guidelines for availability and consistency, cardinality can also be considered as an important factor for the categorization of the proposed guidelines. Thereafter, prioritize their application in each of the categories. To do so, the new generation cardinalities proposed by [13] were considered. These cardinalities have the potential to categorize complex datasets in seven different relationships such as one-to-one (*1:1*), one-to-few (*1:F*) etc. In line with this, our study reveals that, more than one guideline can be in the same priority level for a single cardinality as shown in Fig. 5.

In each of the cardinalities (in Fig. 5), guidelines are prioritize on a scale of seven (priority levels 1 — 7) which are color coded (light gray to dark gray). As it was mentioned before, professionals were allowed to allocate the same rank to more than one guideline, therefore, many guidelines were given the same level in the same category which indicates their potential equality in improving design performance.

In general, the suggested use of these rankings in three different categories (availability, consistency and cardinalities) is

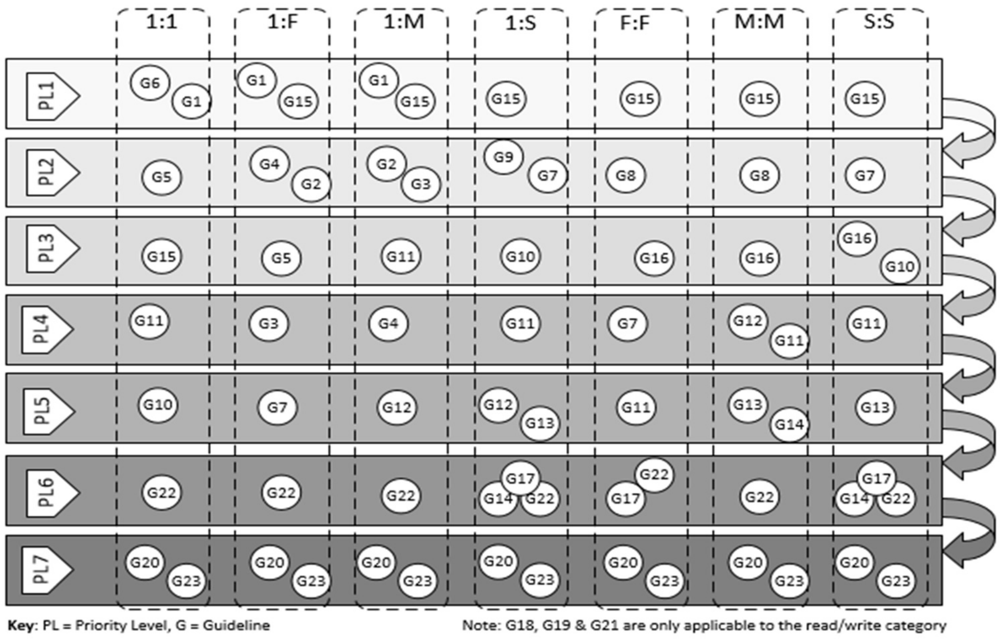

Fig. 5. Guideline Prioritization Based on Cardinalities

that guidelines with higher positions should be favored over the guidelines with lower positions or conflicting guidelines. For instance, while referencing standalone entities **(G11)** increases data availability for independent or round-trip queries, requirement to have high consistency may necessitate combining **G7** and **G15**. This means that, in the case of security access, authorization across cluster can be controlled (consistency) and the solo records within the main document can be bucketed into a different smaller document for independent retrieval (availability). In other words, the application of **G7** can interfere with the impact of applying **G11** or **G15** because it appears higher, but when categorized (availability, consistency and cardinalities), their levels of application changes based on the requirement.

It is worth mentioning that most of the elements in the presented guidelines were broadly reorganized by the experts as they have used some of them in their NoSQL modeling process already which led to better understanding on how best they can be prioritized.

V. Discussion

In this section, the proposed guidelines are investigated regarding two different aspects. First, some limitations of the proposed guidelines are discussed. Thereafter, several aspects of their potentials are elaborated.

A. Limitations

While the proposed guidelines are stronger in their foundation and more generalized than many existing proprietary guidelines, some limitations must be noted. The first limitation relates to the development of the proposed guidelines and their validity: the fundamental principles and the empirical insights that ground the introduction of said guidelines would have been more thorough and evolving if the number of professionals involved was greater than nine. However, the scarcity of NoSQL modelers (expert-level) made it difficult to find the typically used number of professionals. This is because NoSQL databases are new and used to manage new-generation datasets (big data), and they thus have not yet matured in academia and industry.

The second limitation is that the proposed guidelines assume that all modelers have basic SQL modeling skills. This means that the symbols, notations and terminologies proposed by [35] and [36] are prerequisite skills for the effective use of the proposed guidelines. People with no database modeling background may find it challenging to start modeling with the proposed guidelines. However, in the world of diversification, such individuals should also be considered in a more automated manner where a modeler answers a few questions and a suitable model is automatically produced, subject to an experts analysis. This will minimize errors in modeling, thereby producing more stable NoSQL models.

The third limitation relates to the guideline prioritization described in Section 4. The ranking was derived from a number of presentations and expert scorings. Although this could be seen as needing wider expert coverage, it also raises questions such as what other alternative ranking roots are available, for instance, through experimentation. Nevertheless, it seems less attractive at this stage to focus on producing very perfect guidance on how best the proposed guidelines can be prioritized and applied. This is why we have high expectations that the proposed guidelines will be further extended in the near future to cover more application scenarios, as professional have already inspired us with a few guidelines to be considered in the future.

B. Potential

This section continues to discuss the potential of the proposed guidelines beyond their detailed explanations (see Section 3.3) and application (see Section 3.4). The first modeling guidelines prepared to guide data modelers for NoSQL document-store databases, coupled with the increase in complexity of todays data (big data), greatly increase the potential to widely accept and adopt the proposed guidelines in both industry for practice and academia for learning.

On the technical side, two potential aspects are identified. First, the proposed guidelines can be the basis of *automating the modeling process from scratch*, which may not require more technical background. Second, if a model already exists, improvement might be required, as shown in Fig. 4, which resulted from applying the guidelines in Fig. 3. Instead of manually transforming the model using the proposed guidelines, the process can be intelligently automated to identify errors and mark them such that *existing models can be automatically transformed*. Solutions or approaches like these will require further in-depth and formal research on both aspects, as well as potentially more.

The proposed guidelines also point to more potentials for the *competence analysis* of modelers. This might be achieved by measuring the structures of the produced models, which might be based on some assumptions, such as to what extent the proposed guidelines considered the model requirements. Modelers with high levels of competence are likely to detect any model that deviates from the proposed guidelines. In an experiment that involved designing a complete mini-NoSQL-based system, a model was repeatedly redesigned for improvements as a result of a low level of competence, which can be associated with the lack of basic skills [1]. In this manner, the proposed model offers easier methods with simple language to identify difficulties associated with complex datasets as well as the best methods to relate the entities.

VI. Conclusion and Future Work

In this paper, the mismatch between proprietary recommendations for NoSQL document-store modeling and technical insight into NoSQL modeling practice are addressed. Prior empirical research and expert suggestions were consolidated, which led to the derivation of the proposed guidelines. Contrary to proprietary guidelines, our guidelines were built from a strong research foundation, which was practically motivated, empirically derived and conceptually validated. In contrast to the existing research on database modeling, our guidelines were made specifically for document-store NoSQL databases with simple and straightforward explanations. In this manner, the proposed guidelines addressed the practical modeling problems that are being faced by many modelers in industry. This fact among others was emphasized by the low competence level of casual NoSQL modelers [32], [10] and the high rates of errors, repetition and insecurity [19], [20].

In addition to these virtues, the proposed guidelines also revealed some limitations to reflect upon. More significantly, although the guidelines were prioritized based on three identified categories (availability, consistency and cardinalities), we believe that, as big data and NoSQL mature, several other categories will be harnessed, which may call for re-prioritization to suit new categories. Furthermore, humans (who are naturally prone to errors) are, to a large extent, involved in the application of the proposed guidelines, and as such, several automations are required to minimize possible human error, thereby producing more stable models. Such solutions are slotted into our future research schedule.

In addition to the future focuses mentioned earlier, the applicability and usability of the proposed guidelines are another important aspect. While considering other usability test approaches, such as that in [48], where the applicability of SEUAL quality was assessed, the proposed guidelines might be subjected to similar usability assessment in the future, particularly the use of a standard survey, which may result in further improvement of the proposed guidelines.

Finally, with high optimism, the proposed guidelines have great potential to function as an imperative instrument of knowledge transfer from academia to NoSQL database modeling practices, which may bridge the two disconnected communities (academia and industry) with respect to NoSQL database modeling.

ACKNOWLEDGMENT

The authors wish to acknowledge the support from Universiti Teknologi PETRONAS (UTP) for funding this research through Yayasan and Graduate Assistantship Scheme (UTP-GA).

REFERENCES

[1] Micheal J. Mior, K. Salem, A. Aboulnaga, and R. Liu, NoSE: Schema design for NoSQL applications, IEEE Trans. Knowl. Data Eng. From 2016 IEEE 32nd Int. Conf. Data Eng. ICDE 2016, vol. 4347, no. c, pp. 181192, 2016.

[2] H. Zhang, G. Chen, B. C. Ooi, K. L. Tan, and M. Zhang, In-Memory Big Data Management and Processing: A Survey, IEEE Trans. Knowl. Data Eng., vol. 27, no. 7, pp. 19201948, 2015.

[3] G. C. Everest, Stages of Data Modeling Conceptual vs . Logical vs . Physical Stages of Data Modeling, in Carlson School of Management University of Minnesota, Presentation to DAMA, Minnesota, 2016, pp. 130.

[4] M. T. Gonzalez-Aparicio, M. Younas, J. Tuya, and R. Casado, A New Model for Testing CRUD Operations in a NoSQL Database, in 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), 2016, vol. 6, pp. 7986.

[5] IBM, Why NoSQL? Your database options in the new non-relational world, Couchbase, no. March, p. 6, 2014.

[6] J. Bhogal and I. Choksi, Handling Big Data Using NoSQL, in Proceedings - IEEE 29th International Conference on Advanced Information Networking and Applications Workshops, WAINA 2015, 2015, pp. 393398.

[7] MongoDB, How a Database Can Make Your Organization Faster, Better, Leaner, MongoDB White Pap., no. October, p. 16, 2016.

[8] V. Jovanovic and S. Benson, Aggregate Data Modeling Style, Proc. South. Assoc. Inf. Syst. Conf. Savannah, GA, USA March 8th9th, pp. 7075, 2013.

[9] H. He and E. A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng., vol. 21, no. 9, pp. 12631284, 2009.

[10] Z. William, 6 Rules of Thumb for MongoDB Schema Design, MongoDB, 2014. [Online]. Available: https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-1. [Accessed: 23-Jan-2017].

[11] X. Wu, X. Zhu, G. Q. Wu, and W. Ding, Data mining with big data, IEEE Trans. Knowl. Data Eng., vol. 26, no. 1, pp. 97107, 2014.

[12] V. Varga, K. T. Jnosi, and B. Klmn, Conceptual Design of Document NoSQL Database with Formal Concept Analysis, Acta Polytech. Hungarica, vol. 13, no. 2, pp. 229248, 2016.

[13] A. A. Imam, S. Basri, R. Ahmad, N. Abdulaziz, and M. T. Gonzlez-aparicio, New Cardinality Notations and Styles for Modeling NoSQL Document-stores Databases, in IEEE Region 10 Conference (TENCON), Penang, Malaysia, 2017, p. 6.

[14] A. Ron, A. Shulman-Peleg, and A. Puzanov, Analysis and Mitigation of NoSQL Injections, IEEE Secur. Priv., vol. 14, no. 2, pp. 3039, 2016.

[15] M. Obijaju, NoSQL NoSecurity Security issues with NoSQL Database, Perficient: Data and Analytics Blog, 2015. [Online]. Available: http://blogs.perficient.com/dataanalytics/2015/06/22/nosql-nosecuity-security-issues-with-nosql-database/. [Accessed: 21-Sep-2016].

[16] M. J. Mior, Automated schema design for NoSQL databases, Proc. 2014 SIGMOD PhD Symp. - SIGMOD14 PhD Symp., pp. 4145, 2014.

[17] R. CrawCuor and D. Makogon, Modeling Data in Document Databases. United States: Developer Experience & Document DB, 2016.

[18] M. Chow, Abusing NoSQL Databases, Proceedings of DEF CON 21 Hacking Conference. 2013.

[19] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes, and J. Abramov, Security issues in NoSQL databases, in Proc. 10th IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communications, TrustCom 2011, 8th IEEE Int. Conf. on Embedded Software and Systems, ICESS 2011, 6th Int. Conf. on FCST 2011, 2011, pp. 541547.

[20] E. G. Sirer, NoSQL Meets Bitcoin and Brings Down Two Exchanges: The Story of Flexcoin and Poloniex, Hacking, Distributed, 2014. [Online]. Available: http://hackingdistributed.com/2014/04/06/another-one-bites-the-dust-flexcoin/. [Accessed: 31-Jul-2017].

[21] J. FORTIN and A. Cruz, System Failure at British Airways Shuts Down Flights Out of London, The New York Times, 2017. [Online]. Available: https://www.nytimes.com/2017/05/27/world/europe/british-airways-flights-heathrow-and-gatwick-airports-.html. [RAccessed: 01-Aug-2017].

[22] W. Naheman, Review ofNoSQL Databases and Performance Testing on HBase, 2013 Int. Conf. Mechatron. Sci. Electr. Eng. Comput., pp. 23042309, 2013.

[23] C. O. Truica, F. Radulescu, A. Boicea, and I. Bucur, Performance evaluation for CRUD operations in asynchronously replicated document oriented database, in Proceedings - 2015 20th International Conference on Control Systems and Computer Science, CSCS 2015, 2015, pp. 191196.

[24] J. Patel, Cassandra data modeling best practices, part 1, ebaytechblog, 2012. [Online]. Available: http://ebaytechblog.com/?p=1308. [Accessed: 02-Aug-2017].

[25] N. Korla, Cassandra data modeling - practical considerations @ Netflix, Netflix, 2013. [Online]. Available: http://www.slideshare.net/nkorla1share/cass-summit-3. [Accessed: 02-Aug-2017].

[26] N. Jatana, S. Puri, and M. Ahuja, A Survey and Comparison of Relational and Non-Relational Database, Int. J. , vol. 1, no. 6, pp. 15, 2012.

[27] C. JMTauro, A. S, and S. A.B, Comparative Study of the New Generation, Agile, Scalable, High Performance NOSQL Databases, Int. J. Comput. Appl., vol. 48, no. 20, pp. 14, 2012.

[28] R. April, NoSQL Technologies: Embrace NoSQL as a relational Guy Column Family Store, DBCouncil, 2016. [Online]. Available: https://dbcouncil.net/category/nosql-technologies/. [Accessed: 21-Apr-2017].

[29] S. Visigenic, ODBC 2.0 Programmers Manual, Version 2. United States: TimesTen Performance Software, 2000.

[30] G. Matthias, Knowledge Base of Relational and NoSQL Database Management Systems: DB-Engines Ranking per database model category, DB-Engines, 2017. [Online]. Available: https://db-engines.com/en/ranking_categories. [Accessed: 21-Apr-2017].

[31] Gartner and M. Fowler, The NoSQL Generation: Embracing the Document Model, MarkLogic Corp. Hype Cycle Big Data, no. May, 2014.

[32] P. Atzeni, Data Modelling in the NoSQL world: A contradiction?, Int. Conf. Comput. Syst. Technol. - CompSysTech16, no. June, pp. 2324, 2016.

[33] P. Lake and P. Crowther, A History of Databases: Concise guide to databases: a practical introduction, Springer-Verlag London, vol. 17, no. 1, p. 307, 2013.

[34] K. Dembczy, Evolution of Database Systems, Intell. Decis. Support Syst. Lab. Pozna_n Univ. Technol. Pol., vol. 16, p. 139, 2015.

[35] P. P.-S. Chen, The Entity-Relationship Unified View of Data Model-Toward a, ACM Trans. Database Syst., vol. 1, no. 1, pp. 936, 1976.

[36] G. C. Everest, Basic Data Structure Models Explained with a Common Example, in In Proc. Fifth Texas Conference on Computing Systems, 9176, pp. 1819.

[37] J. Han, E. Haihong, G. Le, and J. Du, Survey on NoSQL database, Proc. - 2011 6th Int. Conf. Pervasive Comput. Appl. ICPCA 2011, pp. 363366, 2011.

[38] T. A. Alhaj, M. M. Taha, and F. M. Alim, Synchronization Wireless Algorithm Based on Message Digest (SWAMD) For Mobile Device Database, 2013 Int. Conf. Comput. Electr. Electron. Eng. Synchronization, pp. 259262, 2013.

[39] K. Storm, How I stole roughly 100 BTC from an exchange and how I could have stolen more!, reddit, 2014. [Online]. Available: https://www.reddit.com/r/Bitcoin/comments/1wtbiu/how_i_s tole_roughly_100_btc_from_an_exchange_and. [Accessed: 02-Aug-2017].

[40] G. Khan, Why you should never, ever, ever use documet-store databases like MongoDB, reddit, 2015. [Online]. Available: https://www.reddit.com/r/programming/comments/3dvzsl/w hy_you_should_never_ever_ever_use_mongodb. [Accessed: 02-Aug-2017].

[41] M. L. Chouder, S. Rizzi, and R. Chalal, Enabling Self-Service BI on Document Stores, Work. Proceed- c ings EDBT/ICDT 2017 Jt. Conf. Venice, Italy, 2017.

[42] M. Mughees, DATA MIGRATION FROM STANDARD SQL TO NoSQL, 2013.

[43] T. Halpin, UML data models from an ORM perspective: Part 1 - 10, J. Concept. Model. 8, no. August, pp. 17, 1999.

[44] V. Abramova and J. Bernardino, NoSQL databases: MongoDB vs cassandra, Proc. Int. C* Conf. Comput. Sci. Softw. Eng. ACM 2013, pp. 1422, 2013.

[45] M. Gelbmann, DB-Engines Ranking of Document Stores, DB-Engines, 2017. [Online]. Available: https://db-engines.com/en/ranking/document+store. [Accessed: 21-Feb-2017].

[46] A. P. George Papamarkos, Lucas Zamboulis, XML Databases. School of Computer Science and Information Systems, Birkbeck College, University of London, 2013.

[47] A. Ron, A. Shulman-Peleg, and E. Bronshtein, No SQL, No Injection? Examining NoSQL Security, arXiv Prepr. arXiv1506.04082, 2015.

[48] D. L. Moody, S. Guttorm, T. Brasethvik, and A. S. lvberg, Evaluating the Quality of Process Models: Empirical Testing of a Quality Framework, in S. Spaccapietra, S.T. March, Y. Kambayashi (Eds.), Conceptual Modeling ER 2002, 21st International Conference on Conceptual Modeling, Tampere, Finland, October 711, Proceedings, Lecture Notes in Computer Science, Vol. 2503, Springer, 2002, pp. 380396.