Modeling and Querying Data in NoSQL Databases

Karamjit Kaur
Computer Sci. and Engg. Deptt.
Thapar University
Patiala, Punjab, India 147004
Email: karamjit.kaur@thapar.edu

Rinkle Rani
Computer Sci. and Engg. Deptt.
Thapar University
Patiala, Punjab, India 147004
Email: raggarwal@thapar.edu

***Abstract*—Relational databases are providing storage for several decades now. However for today's interactive web and mobile applications the importance of flexibility and scalability in data model can not be over-stated. The term NoSQL broadly covers all non-relational databases that provide schema-less and scalable model. NoSQL databases which are also termed as Internet-age databases are currently being used by Google, Amazon, Facebook and many other major organizations operating in the era of Web 2.0. Different classes of NoSQL databases namely key-value pair, document, column-oriented and graph databases enable programmers to model the data closer to the format as used in their application. In this paper, data modeling and query syntax of relational and some classes of NoSQL databases have been explained with the help of an case study of a news website like Slashdot.**

I. Introduction

After 1990's due to popularity of HTTP, the cost of posting and exchanging information became cheaper which led to the flooding of information on Internet. It was realized that traditional techniques of data storage will soon become stale and inefficient to handle such vast amount of unstructured and semi-structured data. Not all the information generated on Web is structured, rather interactive Web has produced more semi-structured or un-structured data. All the available rich information cannot be forcefully made to fit in the tabular format of relational databases. This problem was also faced by object-oriented databases under the name "Object-Relational Impedance Mismatch" problem [1]. This mismatch occurs when an object is molded to fit into relational structure. A large percentage of digital information floating around the world is in PDF, HTML and other types of formats which cannot be easily modeled, processed and analyzed. Natural text can not be easily captured in form of entities and relationships. The major problem is that legacy tools require data schema to be defined a priory to the data creation. In today's world, where Internet has become part of the common person's life, deciding on rigid pre-defined schema is unrealistic. Schema evolves as users start using the application and as information to be dealt with changes according to user's requirements. Structure of data also changes as information is gathered, stored and processed.

Above stated problems of storing semi-structured information, object-relational impedance mismatch and schema evolution originated because of the changes in usage pattern of databases since the the time the idea of relational databases were conceptualized in 1970s. Relational databases were not designed keeping in mind sparse information and scalability, where scalability is the ability of a system to handle growing amounts of data. Their data model is based on single machine architecture and was not designed to be distributed. Today all softwares are developed expecting a large user-base, which was not the case in 1970s. Scalability is one of the most discussed issue today, since web applications have got enormous popularity. Scalability can be achieved either vertically or horizontally. Vertical scalability, also known as (a.k.a.) scaling up is easier to achieve as compared to horizontal scalability a.k.a. scaling out. As the name suggests, scaling-up means adding up resources to a single node and scaling-out means adding more nodes to a system. Horizontal scaling provides more flexibility as commodity servers or cloud instances can be utilized. Traditional databases relies on vertical scaling where as recently evolved non-relational databases use horizontal scaling for achieving scalability [2].

Although relational databases have matured very well because of their prolonged existence and are still good for various use cases. Unfortunately for most of the today's software design, relational databases show their age and do not give good performance especially for large data sets and dynamic schemas. We live in a world, where domain model is constantly changing during development phase and even after deployment. These changes in requirements along with various other reasons described above, led to the development of non-relational databases known as NoSQL databases. Popularity of non-relational databases can be very well imagined by the fact that many universities have started teaching about these data stores as part of their curriculum [3].

NoSQL is a term most commonly used to cover all non-relational databases, it stands for Not only SQL. There is much disagreement on this name as it do not depict the real meaning of non-relational, non-ACID, schema-less databases, since SQL is not the obstacle as implied by the term NoSQL. The term "NoSQL" was introduced by Carlo Strozzi in 1998 as a name for his open source relational database that did not offer a SQL interface [4]. The term was re-introduced in October 2009 by Eric Evans for an event named *no:sql(east)* organized for the discussion of open source distributed databases [5]. The name was an attempt to describe the increased number of distributed non-relational databases that emerged during the second half of the 2000's. Increasing number of players dealing with WWW started recognizing the in-efficiency of relational databases to handle huge amount of diverse data generated by the introduction of Web 2.0 applications. Google was the first organization to lead this movement by introducing BigTable [6] in 2006 followed by Amazon's Dynamo [7] in 2007. Influenced by adoption of non-relational databases by these big firms most of the organizations started developing their own

NoSQL data stores customized according to their requirements. Most of today's popular NoSQL data stores have adopted ideas either from Google's BigTable or Amazon's Dynamo. Those inspired by BigTable are categorized as column-oriented or wide-table data stores and others which are descendants of Dynamo are termed as key-value based data stores. There are two other categories too, document and graph-based databases. These four classes of NoSQL databases deal with different types of data and hence are suitable for different use cases.

Column-oriented or Wide-table data stores are designed to address following three areas- huge number of columns, sparse nature of data and frequent changes in schema. Unlike relational databases where rows are stored contiguously, in column-oriented databases column values are stored contiguously. This change in storage design results in better performance for some operations like aggregations, support for ad-hoc and dynamic query etc. In row-oriented databases, all columns of those rows which satisfies the where clause of the query are retrieved, which causes unnecessary disk input-output if only few columns out of all returned columns were required. These databases are also best suited for analytical purposes as they deal with only few specific columns and for compression also as data-type and range of values are fixed for each column. For each column, row-oriented storage design deals with multiple data types and limitless range of values, thus making compression less efficient overall. Most of the columnar databases are also compatible with MapReduce framework, which speeds up processing of large amount of data by distributing the problem on large number of systems [8]. Popular open source column-oriented databases are Hypertable [9], HBase [10] and Cassandra [11]. Hypertable and HBase are derivatives of BigTable where as Cassandra takes its features from both BigTable and Dynamo.

Key-Value Databases have a very simple data model. Data is organized as an associative array of entries consisting of key-value pairs. Each key is unique and is used to retrieve the values associated with it. These databases can be visualized as relational databases having multiple rows and only two columns- key and value. Key-based lookups results in lesser query execution time, also since values can be anything like objects, hashes etc. resulting in flexible and schema-less model appropriate for today's unstructured data. Key-value databases are highly suitable for applications where schema is prone to evolution. Unlike keys, there is no limit on length of value to be stored. Most key-value stores favor high scalability over consistency and therefore most of them also omit rich ad-hoc querying and analytics features for example join and aggregate operations. Few popular key-value data stores are Riak [12], Voldemort [13] and Redis [14].

Document Databases stores documents as data. Document can be in any format like XML (eXtensible Markup Language) [15], YAML (Yet Another Markup Language) [16] and JSON (JavaScript Object Notation) [17] etc. Documents are grouped together in form of collections. These databases are flexible in nature as different documents can have different fields, also any number of fields can be added to the documents without wasting space by adding same empty fields to the other documents in a collection. Compared to relational databases, collections correspond to tables and documents to records. But there is one big difference, every record in a table have the same number of fields, while documents in a collection can have completely different fields. Documents are addressed in the database via a unique key that represents that document. Document-oriented databases are one of the categories of NoSQL databases that are appropriate for web-applications which involves storage of semi-structured data and execution of dynamic queries. Practical usability of these databases can be guessed by the fact that there are more than 15 document-oriented databases available of which widely used are MongoDB [18], CouchDB [19] and RavenDB [20]. MongoDB is the most popular document database. MongoDB is designed to be able to face new challenges such as horizontal scalability, high-availability and flexibility to handle semi-structured data. MongoDB has typical applications in content management systems, mobiles, gaming and archiving. In this paper, MongoDB has been chosen to represent the case-study under consideration.

Graph databases model the database as a network structure containing nodes and, edges relating nodes to represent relationship amongst them. Nodes may also contain properties that describes the real data contained within each object. Similarly, edges may also have their own properties. Relationship connects two nodes and may be directed where direction adds meaning to the relationship. Relationships are identified by their names and can be traversed in both the directions. Comparing with Entity-Relational Model(ER Model), a node corresponds to an entity, property of a node to an attribute and relationship between entities to relationship between nodes. In relational databases, queries requiring attributes from more than one table result in a join operation. Join is an slow operation which degrades the performance as table size increases. Graph databases are suitable for finding relationships within huge amounts of data at a faster rate, since processing of intensive joins are not performed instead it provides index-free adjacency. Also, due to user-driven data it has become difficult to know beforehand what type of attributes will be needed. Relational databases works on pre-defined schema and there is no provision for dynamic and ad-hoc data. Almost all the available graph databases have capability of storing semi-structured information. It does not require a pre-defined schema which leads to easier adaptation to schema evolution and ability to capture ad-hoc relationships. Social networking websites where relationships among data is as important as data itself are best candidates for graph-based storage.

More than 20 graph databases are available of which few are proprietary and others open-source, popular ones are Neo4j [21], Titan [22], OrientDB [23], AllegroGraph [24], InfiniteGraph [25] etc. For these graph databases various query languages are available, Cypher for Neo4j [26], Gremlin which works for various graph databases [27], SPARQL query language [28] is used to retrieve and manipulate data stored in Resource Description Framework (RDF) format [29] etc. In this paper, Neo4j have been used for the representation of the considered case study since it is open-source and one of the most popular graph database. Neo4j is also ACID compliant and written in Java, though there are bindings available with other languages like Ruby, Scala, Python too [21].

II. Data Modeling

A data model is a conceptual representation of the data structures that are required by a database. The data structures include the data objects, the associations between data objects, and the rules which govern operations on the objects. The goal of the data model is to make sure that the all data objects required by the database are completely and accurately represented. Because the data model uses easily understood notations and natural language, it can be reviewed and verified as correct by the end-users [30]. Data models are used by database designers to ease the interaction among designers, application programmers, and end users. A good data model bridges gap between database design components and real world data.

While implementing relational databases a lot of attention is paid on developing conceptual, logical and physical data model. New technologies such as XML and NoSQL databases have put doubts on the usefulness of data models. For example, storing data in databases as XML blobs and documents in case of document-oriented NoSQL databases try to eliminate the need for data modeling. But to describe any data structure, rules can best be described as a data model. Although, these new generation of databases are schema-less but the importance of data model will always remain to understand and demonstrate the storage of data. Unlike relational databases where modeling is decided by the structure of data, while modeling NoSQL databases the types of queries that will be executed on data are kept in mind. In other words, design theme of relational database are focused on *answers* whereas of NoSQL databases are focused on *questions*.

As explained in section I, four classes of NoSQL databases deals with different types of data sets. Each have their advantages and disadvantages in some particular context. Till now, relational database was the de-facto database. By default, database used to mean data stored in tables. But with the advent of these variety of non-relational storage solutions, programmers have got options to choose from. Moreover, within one application different classes of NoSQL databases can be used simultaneously which is popularly known as polyglot persistence. The term came from polyglot programming which implies usage of multiple programming languages within one application software. In next sections, data modeling of relational and two classes of non-relational databases namely document-oriented and graph-based databases have been discussed.

III. Case-study

In this section, example schema is explained in detail and is represented in relational model popularly known as Entity-Relationship (ER) model. We have considered database schema of new website like slashdot, where a user posts a story's snippet (a few lines description of a story) and gives a relevant title to the story. ER model of case-study explained below is shown in figure 1.

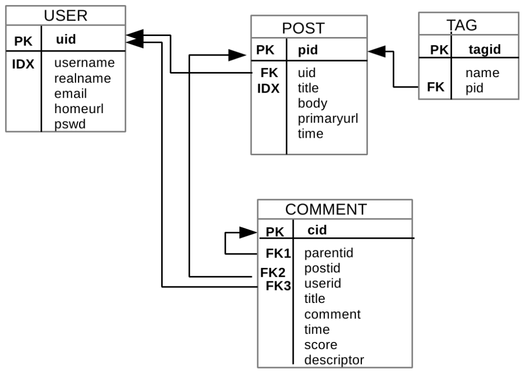

Fig. 1: ER Diagram of case-study

- Posted story's information is stored in the POST table. *Title* field stores the title of the story and *body* field stores the snippet of the story, along with embedded hyper-links. Each story is assigned a unique post id (*pid*) which also serves as the primary key of the POST table. As website can have hundreds of stories posted in a day, the title field of the POST table is indexed since it is being used to search for a story from the website's in built search option. The story may contain embedded hyper-links which points to the source of the story and to sources of further information related to the story. The main URL of the source of the story is stored in the *primaryurl* field. The *time* field in POST table stores the time stamp when the story is submitted. It can be used to show posting time of the story along with story in the homepage and can also be used to search and sort the stories. The user posting the story could be a logged in user, in which case his/her username will be shown as the submitter or the user can post the story anonymously without the requirement of logging in, in which case the story would be shown as submitted by 'anonymous'. The *uid* field of the POST table is used for storing user id of the user posting the story. Uid will be NULL in case of a user posting the story anonymously. The Uid of POST table is a foreign key pointing to the primary key of the same name in the USER table which stores the information about the users submitting stories to the website or posting comments on the submitted stories.
- Apart from uid, the USER table also stores the *username* of the user which is what shows as the submitted in case of story submission or a poster of a comment if that user comments on a story. Other information stored about a user constitutes his/her *realname*, *email*, *homeurl* and *password*. An index is also created on the *username* for fast search and retrieval.
- Apart from posting story and giving it a title, the user can also put some tags relevant to the story which help in classifying and searching relevant stories easier. As any number of a tags can be associated with a story, a separate TAG table with primary key *tagid* is used to store information about the tags. It simply stores the *name* of tag along with post-id of the story (*pid*) as foreign key linking to pid filed of POST.
- After submission of the story, it is moderated and if

it gets accepted then it is put up on the website's homepage. Once the story gets submitted and is shown on the homepage of website, other users can read and comment on it. A user can comment on the story or he/she can comment on previous comments of fellow users, thus giving rise to a tree like structure of comments. COMMENT table is used to store information about the comments posted on a story. As a comment needs to be linked to the post, to the parent comment (if the comment is in reply to a previous comment) and to the user who posted the comment. Fields *postid*, *parentid* and *userid* respectively are used which are all foreign keys pointing to POST, COMMENT and USER table. If a comment is directly in reply to the story (in contrast to a comment in reply to an earlier comment) then parentid is kept NULL, which otherwise would point to a comment-id in COMMENT table only. Apart from fields already discussed a primary key *cid* (comment id) is in the COMMENTS table. It also contains fields to store the *title* and body of comment along with *time*. A comment is further moderated and is given a *score* and a *descriptor* e.g. Funny, Insightful etc. Score is a value between 1 to 5 where 1 means a comment of lower value and 5 means comment of higher value. So a comment can be rated as Funny 5, Insightful 3 etc. As a popular story can get thousands of comments, these fields can be helpful in filtering out the comments according to the taste of the reader.

IV. Document-oriented Data Model

A document database is used to store, retrieve and manage semi-structured data. In this database, data is stored in the form of documents. Few document-based databases like CouchDB and RavenDB store data in JSON format, where as the most popular document database MongoDB stores data in BSON notation where BSON is Binary JSON (BSON) that enable binary serialization on data. Advantage of storing data in JSON or BSON representation is that it is easy to map object structures of most of the programming languages directly into this representation, no mapper/translators are required. We have chosen MongoDB as document-based database for implementing the case-study.

MongoDB is an open source NoSQL document database, initiated by 10gen Company. The word Mongo in its name comes from word humongous [18]. It does not support joins or ACID transactions and was specifically designed to handle growing data storage needs. It is written in C++ and provides interactive JavaScript shell for database management. Also supports a rich, ad-hoc query language of its own. In addition, there exist bindings for many programming languages like Java, C, C++, Erlang, Python, Ruby and more. MongoDB keeps all of the most recently used data in RAM. When indexes are created for queries, all data sets fits in RAM and queries are run from memory. There is no query cache in MongoDB. All queries are run directly from the indexes or data files. Data is physically written to the disk with in 100 milliseconds. To support massive data storage, it also comes with distributed file system GridFS.

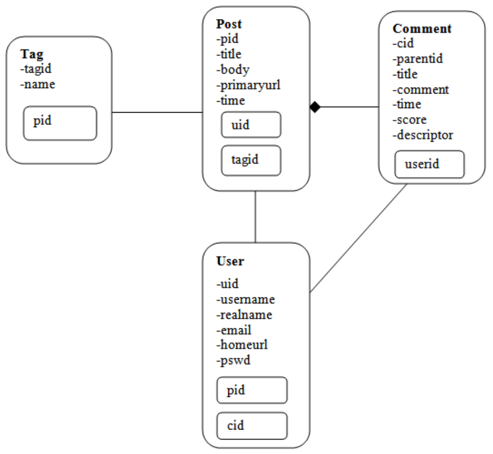

Fig. 2: Class Diagram of case-study

Class diagram can be used to represent the data model of MongoDB database. Documents are represented by classes. Embedded documents and referencing between documents is represented by relationships between the classes. Composition is used to represent the embedded documents and associations are used to represent the referencing. Document fields are represented by class attributes. Figure 2 shows the class diagram of the case-study. Post and Comment are embedded documents represented by composition. Comment exists only if the post corresponding to that comment exists that is comment is embedded into the post. Post, User and Tag are different documents. To relate these documents referencing is used. In post document, uid is used as reference field of user document and tagid is used as reference of tag document to indicate a relationship between documents. In user document, pid is used as reference of the post document and cid is used as reference of comment document. In comment document, userid is used as reference of the user document and in tag document pid is used as reference of the post document.

V. Graph-based Data Model

Graph databases models the whole database as a network structure crowded with relationships between nodes. Objects are stored in nodes and edges connecting nodes serve the roles of entities and relationships, if compared with relational database architecture. Nodes may also contain properties that describes the real data contained within each object, similarly edges may also have their own properties. In this paper, we have considered Neo4j for representing our case study, since it is the most popular graph database today as well as becuase it is open source. It is ACID compliant and written in Java, though there are bindings available with other languages too like ruby, scala, python etc. Neo4j and almost all available graph databases have capability of storing semi-structured information. It do not have pre-defined schema which leads to easier adaptation to schema evolution and ability to capture ad-hoc relationships. Hierarchical data can be best represented in graph databases. Graph databases enables storage of data in its natural format, if the data can be expressed better in graphical form than in tabular form. There by, reducing the problem of object-relational impedance mismatch.

Graph databases are good at storing relationships amongst data. A graph database can traverse thousands of edges at a fraction of the cost of the relational joins because relationships are direct links between nodes [23].Three basic components that build and annotates a graph are vertex, edge and property. Each vertex/node corresponds to entities in relational databases, where edge connects nodes to represent relationship between them. Edges are labeled and directed where label identifies the relationship name and direction adds meaning to the relationship.In today's era, we are surrounded with graphs like semantic web, natural sciences, social networks etc. Graph databases allows O(1) access to adjacent vertices because every element has a direct pointer to its adjacent element. Unlike relational databases, performance of graph databases do not degrade as size of database increases since the cost of local step remains the same because each vertex serves as mini-index of its adjacent elements [31]. For a connected graph, whole graph is a single atomic structure. Queries are performed using graph traversal techniques.

Neo4j is a powerful graph database written in Java [21]. It has capability of storing billions of nodes and relationships efficiently and also provides very fast querying and traversal. It is available under both open source and commercial license. In addition to Java, Neo4j has bindings available in other languages too like Python, Jython, Ruby and Clojure. It also provides transactional capabilities and traverses depths of 1000 levels and beyond at millisecond speed [21]. Graph databases contain deeply connected data, to query this data which if stored in relational schema will result in joins of high depth. Each join will result in cartesian product of all potential combinations of rows and then filters out those that do not match the condition specified in the where clause. For one million users, the cartesian product of 5 joins will result in huge number of rows. Filtering out all those records that do not match the query will be very expensive. However Neo4j visits only those nodes that are relevant to the traversal so it's performance is not affected much with increase in data. Though, traversal get slower with increase in the the depth because of the increased number of results that are returned.

Neo4j can be queried through its native Java APIs, SPARQL [28], Gremlin [27] or using Cypher [26] query language. Cypher query language is used the most, especially for Neo4j graph database. Neo4j can also be queried via its shell. Visual editor for graph databases are also available, namely Neoclipse [32] and Gephi [33]. Neo4j Web Administration is the primary user interface for Neo4j, which is available at http://127.0.0.1:7474/ after installation of Neo4j server. In this paper, Neoclipse have been used as editor and Cypher as the query language. Database backup facilities are also provided in Neo4j enterprise edition [34]. Importing data to Neo4j from MS Excel format is also possible using Py2neo [35] which parses csv (comma separated values) file and load it into Neo4j database.

Like SQL, Cypher is a declarative language. The syntax is easy to comprehend due to usage of clauses similar to SQL like start, match, where, return etc. Usage of wild characters like *,? enhances pattern matching. The command *order by* can be used similar to SQL. Almost all aggregate functions like count, min, max, avg, distinct etc. are available in this language. Presence of predicates like all, any, none, single, increases querying capabilities of the language. SQL literate developers can easily use Cypher query language [26].

Figure 3 represents the schema of case-study in Neoclipse editor of Neo4j graph database. Neoclipse is a tool for visualizing and altering Neo4j databases, including nodes, relationships and properties on both.In the Database graph view, it is possible to edit properties of both nodes and relationships. Further we can add new relationship types, relationships and nodes. There are also some ways to decorate the nodes and relationship representations. Neoclipse is a subproject of Neo4j which aims to be a tool that supports the development of Neo4j applications [32].

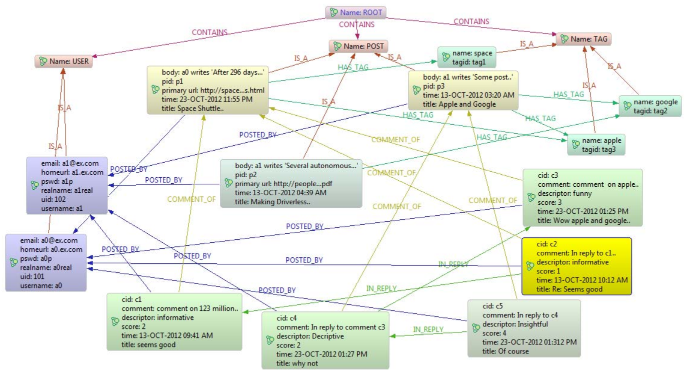
Name: ROOT
Name: USER
Name: POST
Name: TAG
CONTAINS
IS_A
HAS_TAG
POSTED_BY
COMMENT_OF
IN_REPLY
name: space tagid: tag1
name: google tagid: tag2
name: apple tagid: tag3
body: a0 writes 'After 296 days...' pid: p1 primary url: http://space...s.html time: 23-OCT-2012 11:55 PM title: Space Shuttle..
body: a1 writes 'Some post..' pid: p3 time: 13-OCT-2012 03:20 AM title: Apple and Google
body: a1 writes 'Several autonomous...' pid: p2 primary url: http://people...pdf time: 13-OCT-2012 04:39 AM title: Making Driverless..
email: a1@ex.com homeurl: a1.ex.com pswd: a1p realname: a1real uid: 102 username: a1
email: a0@ex.com homeurl: a0.ex.com pswd: a0p realname: a0real uid: 101 username: a0
cid: c3 comment: comment on apple.. descriptor: funny score: 3 time: 23-OCT-2012 01:25 PM title: Wow apple and google..
cid: c2 comment: In reply to c1.. descriptor: informative score: 1 time: 13-OCT-2012 10:12 AM title: Re: Seems good
cid: c1 comment: comment on 123 million.. descriptor: informative score: 2 time: 13-OCT-2012 09:41 AM title: seems good
cid: c4 comment: In reply to comment c3 descriptor: Decriptive score: 2 time: 23-OCT-2012 01:27 PM title: why not
cid: c5 comment: In reply to c4 descriptor: Insightful score: 4 time: 23-OCT-2012 01:312 PM title: Of course

Fig. 3: Graph represented in Neo4j database using Neoclipse

VI. Querying Formats

Developer community is well versed with the usage of Structured Query Language (SQL) for storing, retrieving and manipulating data in relational databases. As of now, there is no standard query language available for querying non-relational databases. Querying these new generation of databases are data-model specific. Each database comes with its own query language. For example, CQL (Cassandra Query Language) for Cassandra, MongoDB Query Language for MongoDB, Cypher Query Language for Neo4j etc. Efforts are in progress to design query languages which can be used by various databases. But designing a query language that can span multiple data-models is difficult because as explained in previous sections, each classes of NoSQL database is designed for some specific purpose. Hence, efforts to design common query languages for NoSQL databases have been localized to its classes. For example, SPARQL is a declarative query specification designed for most of the available graph databases [28]. Similarly, UnQL (Unstructured Query Language) was introduced in 2011 by Couchbase and SQLite team with the aim to create a standard for NoSQL database queries. It has SQL-like syntax for manipulating document databases and can be used across various document-oriented databases including CouchDB and MongoDB [36].

Most of the NoSQL databases allow RESTful interfaces to the data, many others offer query APIs. Few query tools/editors have also been developed for respective NoSQL databases. In this section, we have shown querying syntax of three databases, PostgreSQL for relational database, MongoDB query language for MongoDB and Cypher query language for Neo4j. Seven queries of varying complexities have been considered for the case-study explained in section III. Queries are chosen in such a way that maximum syntax of query languages are covered.

MongoDB has its own query language. The find () method selects documents from a collection that meets the ⟨query⟩ argument. ⟨Projection⟩ argument can also be passed to select the fields to be included in result set. The find () method returns a cursor to the results. This cursor can be assigned to variable. Query syntax looks like *db.collection.find(⟨query⟩,⟨projection⟩)*. The find() method is analogous to the SELECT statement, while the ⟨query⟩ argument corresponds to the WHERE statement, and the ⟨projection⟩ argument corresponds to the list of fields to select from the result set [37] . Unlike find(), findOne() method selects only one document from a collection.

Cypher is a declarative graph query language which is used to query Neo4j graph database. It allows for expressive and efficient querying and updating of the graph store without having to write traversals through the graph structure in code. Cypher is designed to be a humane query language, suitable for both developers and operations professionals who want to make ad-hoc queries on the database. Cypher is inspired by a number of different approaches and builds upon established practices for expressive querying. Most of the keywords like WHERE and ORDER BY are inspired by SQL [26]. Pattern matching borrows expression approaches from SPARQL. Being a declarative language, Cypher focuses on the clarity of expressing what to retrieve from a graph, not how to do it, in contrast to imperative languages like Java, and scripting languages and the JRuby Neo4j bindings.

TABLE I: *List the posts of each user.*

MongoDB: 1 is put against both _id and uid which implies that for each postid of POST collection, all uids should be returned.
Neo4j: 2,3,4 are node-ids similar to row-ids of relational databases. Multiple nodes are selected here (2,3 and 4) as starting point. A starting point is a relationship or a node where a pattern is anchored. In figure 3, root node has node-id 1, user post and tag has node-ids 2,3 and 4 respectively. Here, *g* is the post POSTED_BY *f* which IS_A node with name 'user'.

PostgreSQL	*select u.uid,p.pid from user u,post p where u.uid=p.uid;*
MongoDB	*db.post.find({},{_id:1,uid:1})*
Neo4j	*start n=node(2,3,4) match n⟨-[:IS_A]-f⟨-[:POSTED_BY]-g where n.name='user' return f.uid as userid, g.pid as posted by*

TABLE II: *Find the tag(s) which have been used in the posts of each user.*

MongoDB: 0 is placed against _id which implies that postids is not to be projected in the answer.
Neo4j: This query is same as above, just one more relationship HAS_TAG has been added.

PostgreSQL	*select u.uid,t.tagid from user u,post p,tag t where u.uid=p.uid and p.pid=t.pid;*
MongoDB	*db.post.find(,_id:0,uid:1,tagid:1)*
Neo4j	*start n=node(2,3,4) match n⟨-[:IS_A]-f⟨-[:POSTED_BY]-g-[:HAS_TAG]-⟩h where n.name='user' return f.uid as userid, h.tagid as tagid*

TABLE III: *Given a cid (say c2) of a comment find its related post, parent comment and tags associated with the post.*

MongoDB: To exactly match the value of field, syntax is "field"="value" as written for cid=c2 *"comments.cid":"c2"*.

PostgreSQL	*select c.cid, c.postid, c.parentid, t.tagid from comment c,post p ,tag t where c.postid=p.pid and p.pid=t.pid and c.cid='c2';*
MongoDB	*db.post.find("comments.cid":"c2","comments.cid":1, "comments.parentid":1,tagid:1);*
Neo4j	*start n=node(2,3,4) match n⟨-[:IS_A]-f⟨-[:COMMENTED _BY]-g-[:COMMENT_OF]-⟩h-[:HAS_TAG]-⟩k, g-[:IN_REPLY]-⟩p where n.name='user' and g.cid='c2' return g.cid, k.tagid, p.cid, h.pid*

TABLE IV: *Find the tag names which have used in the post under which user "a1" has commented.*

MongoDB: Document with username a1 is extracted from user collection and stored in variable u. Then comments corresponding to that user are found by matching u._id field with userid field of subdocument comments (comments.userid) in Post Collection and returned cursor is stored in the variable tag1. Then _id is matched in Tag collection. To exclude _id field, _id:0 is written in projection field and the fields which need to be included are written as fieldName: 1.

PostgreSQL	*select t.name from comment c,post p,user u,tag t where p.pid=c.postid and c.userid=u.uid and p.pid=t.pid and u.username='a1';*
MongoDB	*var u=db.user.findOne(username:"a1"); var tag=db.post.findOne("comments.userid":u._id); db.tag.find(_id:$in:tag.tagid,_id:0,name:1 OR db.tag.find(_id:tag1.tagid,_id:0,name:1))*
Neo4j	*start n=node(2,3,4) match n⟨-[:IS_A]-f⟨-[:COMMENTED _BY]-g-[:COMMENT_OF]-⟩h-[:HAS_ TAG]-⟩k where n.name='user' and f.username='a1' return k.name*

TABLE V: *Find the time of the post under which user "a1" has commented.*

PostgreSQL	*select p.time from comment c,post p,user u where p.pid=c.postid and c.userid=u.uid and u.username='a1';*
MongoDB	*var u=db.user.findOne(username:"a1"); db.post.find(uid:u._id,time:1,_id:0)*
Neo4j	*start n=node(2,3,4) match n⟨-[:IS_A]-f⟨ [:COMMENTED_BY]-g-[:COMMENT_OF]-⟩h where n.name='user' and f.username='a1' return h.time as time*

TABLE VI: *Find the total number of posts by a particular user (say 102).*

PostgreSQL	*select count(p.pid) from post p,user u where u.uid=p.uid and u.uid=102;*
MongoDB	*db.post.count(uid:102)*
Neo4j	*start n=node(2,3,4) match n⟨-[:IS_A]-f⟨-[:POSTED_BY]-g where n.name='user' and f.uid=102 return count(g.pid)*

VII. Conclusion

NoSQL solutions should not be thought of as replacement for RDBMS, instead as a complementary product for handling issues of scalability and complexity. Non-relational databases provide many enhancements over traditional relational databases such as increased scaling across commodity servers or cloud instances, non-adherence to rigid schema for inserting data and hence ease in capturing of different type of

TABLE VII: *Find out the username, uid, tags associated with posts of a comment and score associated with comments.*

PostgreSQL	*select c.cid, u.username,c.userid,c.score,t.tagid from comment c,user u, tag t where c.userid=u.uid and c.postid=t.pid;*
MongoDB	*db.post.find(,tagid:1,_id:0,"comments.score":1, "comments.userid":1,"comments.cid:1)*
Neo4j	*start n=node(2,3,4) match n⟨-[:IS_A]-f⟨-[:COMMENTED_BY]-g-[:COMMENT_OF]-⟩h-[:HAS_TAG]-⟩k where n.name='user' return f.uid,f.username,k.tagid,g.score*

data without much changes at schema level. These NoSQL databases may require additional storage, since data is denormalized, but results in overall improvements in performance, flexibility and scalability.

In this paper, we discussed about NoSQL databases and its four broad categories. ER Modeling is very popular among developers, but data modeling of non-relational databases is yet maturing. To the best of our knowledge, there is no publication that explained modeling and querying syntax of these non-relational databases. Due to limit on length, only two classes of NoSQL Databases: Document-oriented and Graph-based databases have been covered in this paper. A case-study have been explained and considered to illustrate the way of data modeling. Seven queries have been explained for the same case-study. PostgreSQL have been chosen for the implementation of relational database, MongoDB for document-oriented database and Neo4j for the implementation of graph databases.

References

[1] A. M. Keller, R. Jensen, and S. Agarwal, "Persistence software: Bridging object-oriented programming and relational databases," 1993.

[2] M. M. Michael, J. E. Moreira, D. Shiloach, and R. W. Wisniewski, "Scale-up x scale-out: A case study using nutch/lucene." in *IPDPS*. IEEE, 2007, pp. 1–8. [Online]. Available: http://www.cecs.uci.edu/ papers/ipdps07/pdfs/SMTPS-201-paper-1.pdf

[3] M. Grossniklaus and D. Maier, "The curriculum forecast for portland: cloudy with a chance of data," *SIGMOD Rec.*, vol. 41, no. 1, pp. 74–77, apr 2012. [Online]. Available: http://doi.acm.org/10.1145/2206869.2206885

[4] C. Strozzi. (2013) Nosql: A non-sql relational database management system. [Online]. Available: http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home-Page

[5] (2009, October) Nosql conference. [Online]. Available: https://nosqleast.com/2009/

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 15–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267308.1267323

[7] D. Hastorun, M. Jampani, G. Kakulapati, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazons highly available key-value store," in *In Proc. SOSP*, 2007, pp. 205–220.

[8] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[9] (2013) Hypertable homepage. [Online]. Available: http://hypertable.org

[10] (2013) Hbase homepage. [Online]. Available: http://hbase.apache.org/

[11] (2013) Cassandra homepage. [Online]. Available: http://cassandra.apache.org

[12] (2013) Basho technologies, inc. riak. [Online]. Available: http://wiki.basho.com/Riak.html

[13] Project voldemort: A distributed database. [Online]. Available: http://project-voldemort.com/

[14] J. Zawodny, "Redis: Lightweight key/value store that goes the extra mile," *Linux Magazine*, Aug. 2009.

[15] S. M. Tim Bray, Jean Paoli and E. Maler, "Extensible markup language (xml) 1.0 second edition w3c recommendation," *Technical Report RECxml-20001006, World Wide Web Consortium*, October 2000.

[16] (2013) Yaml homepage. [Online]. Available: http://www.yaml.org/

[17] Javascript object notation (JSON). [Online]. Available: http://www.json.org

[18] The mongodb's website. [Online]. Available: http://www.mongodb.org/

[19] (2013) The couchdb website. [Online]. Available: http://couchdb.apache.org/

[20] (2013) The ravendb website. [Online]. Available: http://ravendb.net/

[21] E. Eifrem. (2013) Neo4j homepage. [Online]. Available: http://www.neo4j.org/

[22] (2013) Titan distributed graph database. [Online]. Available: http://thinkaurelius.github.com/titan

[23] (2013) Orientdb wiki. [Online]. Available: http://code.google.com/p/orient/wiki/GraphDatabase

[24] (2013) Allegrograph database. [Online]. Available: http://www.franz.com/agraph/allegrograph/

[25] (2013) Infinitegraph database. [Online]. Available: http://objectivity.com/INFINITEGRAPH

[26] (2013) Cypher query language. [Online]. Available: http://docs.neo4j.org/chunked/stable/cypher-query-lang.html

[27] T. M. O'Brien, A. M. Ritz, B. J. Raphael, and D. H. Laidlaw, "Gremlin: An interactive visualization model for analyzing genomic rearrangements." *IEEE Trans. Vis. Comput. Graph.*, vol. 16, no. 6, pp. 918–926, 2010. [Online]. Available: http://dblp.uni-trier.de/db/journals/tvcg/tvcg16.html

[28] E. Prud'hommeaux and A. Seaborne, "SPARQL query language for rdf," *W3C Recommendation*, vol. 4, pp. 1–106, 2008. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/

[29] (2013) Rdf wikipedia entry. [Online]. Available: http://en.wikipedia.org/wiki/Resource_Description_Framework

[30] J. Mamenko, "Introduction to data modeling and msaccess," *Lecture Notes on Information Resources*, 2004. [Online]. Available: http://gama.vtu.lt/biblioteka/Information_Resources/i_part_of_information_resources.pdf

[31] M. A. Rodriguez and P. Neubauer, "The graph traversal pattern," *CoRR*, vol. abs/1004.1001, 2010.

[32] P. Neubauer. (2013) Neoclipse - a graph tool for neo4j. [Online]. Available: https://github.com/neo4j/neoclipse#neoclipse—a-graph-tool-for-neo4j

[33] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks." in *ICWSM*, E. Adar, M. Hurst, T. Finin, N. S. Glance, N. Nicolov, and B. L. Tseng, Eds. The AAAI Press, 2009. [Online]. Available: http://dblp.uni-trier.de/db/conf/icwsm/icwsm2009.html#BastianHJ09

[34] (2013) Neo4j manual on backup. [Online]. Available: http://docs.neo4j.org/chunked/milestone/operations-backup.html

[35] N. Small. (2013) Py2neo homepage. [Online]. Available: http://py2neo.org

[36] (2013) Unql home page. [Online]. Available: http://unql.sqlite.org/index.html/wiki?name=UnQL

[37] Getting started with the mongo shell. [Online]. Available: http://docs.mongodb.org/manual/tutorial/getting-started-with-the-mongo-shell/