Conceptual Design of Document NoSQL Database with Formal Concept Analysis

Viorica Varga, Katalin Tünde Jánosi-Rancz, Balázs Kálmán

Babeş-Bolyai University, Kogălniceanu 1, 400084 Cluj-Napoca, Romania
Sapientia Hungarian University of Transilvania, Corunca 1C, 540485 Târgu Mureş, Romania
Babeş-Bolyai University, Kogălniceanu 1, 400084 Cluj-Napoca, Romania
ivarga@cs.ubbcluj.ro, tsuto@ms.sapientia.ro, kbim1225@scs.ubbcluj.ro

Abstract: Early Big Data solutions were not based on database management system principles. As the popularity of these solutions have increased and are applied in more data management scenarios in the recent years, DBMS principles, are being recognized as important factors and are becoming important in newly developed solutions. Big Data collections do not enforce document structure, but just because a data store is schema-less, it does not mean the structure of the stored documents will not play an important role in the overall performance and flexibility of an application. In this paper we will explore a method for the conceptual modeling for document based databases, using Formal Concept Analysis (FCA). We have shown that FCA is a valuable visual analyzer for large-scale data, for example, offering a means of reading the possibility of nested scheme design from the built concept lattice. Results of experiments using our method have proven that decisions affecting the modeling of data can affect application performance and database capacity.

Keywords: conceptual design; NoSQL database; document store; Formal Concept Analysis

1 Introduction

Data is growing exponetially in the digital world, increasing in volume, variety (structured, un-structured or hybrid) and velocity (high speed of growth). This phenomenon is refered to as ‘Big Data’. This growing data collection is so large that it can not be effectively managed using conventional relational data management tools. To handle this problem, traditional RDBMS are complemented with rich set systems: NoSQL [1, 4, 20] data stores, NewSQL and Search-based systems.

NoSQL systems generally have some common features. The first is the ability to horizontally scale simple operations over many servers. They can replicate and partition data over many servers with a simple call level interface. These new systems accept a weaker concurrency model, than the ACID transactions of relational database systems. They are often flexible enough to accommodate semi-structured and sparse data sets [20]. NoSQL data stores vary in their data and query model. The most common categorization of these systems is by data model, distinguishing key-value stores, document stores, column-family stores, and graph databases [4]. Key-value stores data structure is composed of a unique key and an opaque value. Document based NoSQL systems also store key-value pairs, but the values are structured as documents. The document is a set of name-value pairs, usually in JSON (JavaScript Object Notation) [8] format or the binary representation BSON. Name-value pairs represent the properties of data objects. Values can be scalar or appear as lists, but may contain nested documents too. Column-family stores manage records with properties. A schema for a column-family declares property families, and new properties can be added to a property family ad hoc. Graph databases represent data in graph format; objects are stored in nodes and their relationships in the edges.

Most NoSQL data stores do not enforce any structural constraints on the data; they are usually referenced as schema-less data. But programmatically accessing this data, it is important to have some notion about its structure. Without knowing the general structure of the data, it is nearly impossible to perform any application development or data analysis [15]. Many NoSQL data stores provide a declarative query language. Developers need to know which attributes are present (or absent) in persisted objects in order to formulate queries. For OLAP-style data analysis, developers also need to know which structure to expect when parsing JSON documents. So, these tasks require some form of schema description. The structure of the stored documents plays an important role in the overall performance of the application.

In this paper we focus on designing document stores, which are based on a semi-structured data model, implemented as JSON, XML or BSON format. The design of any database follows a well-defined methodology for conceptual, logical, and physical data modeling. A prevalent model in the conceptual database design is the Entity-Relationship (E-R) model. The semi-structured data has a loose schema: a core of attributes is shared by all objects, but many individual variants are possible. A hierarchical structure of the data is a common design opportunity for embedding complex entities.

Formal Concept Analysis (FCA) [11] supports knowledge discovery and knowledge representation [14]. Current FCA methods have the capabilities for taking into account the presence and management of relational attributes or links in the data [18, 19].

In this paper a Formal Concept Analysis (FCA) approach for conceptual modeling of document based databases is proposed. A mapping from Entity-Relationship model to a schema for semi-structured data in the form of concept lattices are presented for relations of type One-to-One, One-to-Many and Many-to-Many. For different relationship types we obtain different concept lattices [22]. The possibility of nested scheme design can be read from the lattices.

We propose a Relational Concept Analysis (RCA) grounded approach to conceptual document based NoSQL database design, one which is a data model, for the systems that can store and manage Big Data.

In Section 2, we assume familiarity with the basic notions of FCA and RCA and after that, in Section 3, we discuss how relational modeling can be emulated in the case of document databases using RCA. Section 4 presents the experiments on DBLP [6] bibliography dataset. Finally, we finish our work with conclusions.

2 Preliminaries and Basic Notions in FCA and RCA

Our research is mainly based on the mathematical foundations of FCA and in this section we introduce the necessary formal background.

FCA is a data analysis method which enables the discovery of knowledge hidden within data, such as association rule mining, ontology engineering, machine learning. FCA actually provides support for processing large dynamic complex data.

In FCA, data are represented by a formal context, which will contain objects and attributes. From the context, formal concepts are generated by grouping objects which have the same set of attributes. Each formal context is transformed into a concept lattice, which forms the basis for further data analysis.

Definition 1. $(Formal\ context)$. A formal context $K = (G, M, I)$ consists of two sets G and M and a binary relation I between G and M. Elements of G are called objects while elements of M are called attributes of the context. The fact $(g, m) \in I$ is interpreted as "the object g has attribute m".

Example 1. Consider the set of objects $G = \{Bicycle, Chariot, Boat,\ Car,\ Airplane\}$. Consider the set of attributes $M = \{Has\ wings,\ Has\ engine,\ Has\ windows,\ Has\ wheels\}$ that are properties that vehicles may have or not. Table 1 gives an example of formal context (G, M, I), the X indicates that a certain object has a certain attribute.

Table 1
An example of formal context $K = (G, M, I)$

	Has wings	Has engine	Has windows	Has wheels
Bicycle				X
Chariot				X
Boat		X	X	
Car		X	X	X
Airplane	X	X	X	X

Definition 2. ($Formal\ concept$). A formal concept of a context (G, M, I) is a pair (A, B) with $A \subseteq G, B \subseteq M, A' = B$ and $B' = A$. A is called the extent of the concept (A, B) while B is called its intent. A' and B' define a Galois connection between the power sets of G and M. The set of all formal concepts of a context (G, M, I) is written $\mathfrak{B}(G, M, I)$. Concepts are partially ordered by $(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2$ $(\Leftrightarrow\ B_2 \subseteq\ B_1)$. (A_1, B_1) is called sub-concept and (A_2, B_2) a super-concept.

Example 2. From the previous example, it directly follows that the pair $(\{Boat, Car, Air-plane\}, \{Has\ engine, Has\ windows\})$ is a formal concept. A Galois connection implies that if one makes the sets of one type larger, they correspond to smaller sets of the other type, and vice versa. Using this concept, if $Has\ wings$ is added to the list of attributes, the set of vehicles reduces to $\{Airplane\}$.

FCA organizes the information through concept lattices, which fundamentally comprises a partial order, modeling the subconcept-superconcept hierarchy. Concept lattice is the common name for a specialized form of Hasse diagram that is used in conceptual data processing.

Definition 3. ($Concept\ lattice$). The set of all formal concepts from a context $K = (G, M, I)$ ordered with the relation $\leq$ form a complete lattice called concept lattice of (G, M, I) and denoted by $\mathfrak{B}(G, M, I)$.

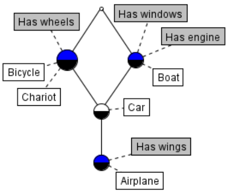


Figure 1
Concept lattice raised from Table 1

Figure 1 shows the concept lattice, associated with Table 1. A line diagram consists of circles, lines and the names of all objects and all attributes of the given context appearing as labels. Each node in the lattice corresponds to a formal concept while a line denotes an order relation between two concepts. An object g has an attribute m if and only if there is an upwards leading path from the circle named by g to the circle named by m.

Real-life data are often more complex than those given by a formal context. There are several extensions of FCA to handle complex data, such as *Conceptual scaling* [23] (where complex data are turned into binary contexts by using scales), *Pattern structures* [24] (a general approach to conceptual scaling by giving a direct method of knowledge discovery in complex data such as logical formulas, graphs, strings, tuples of numerical intervals), *Power Context Families* [25], *Relational Concept Analysis* [18] and *Logical Concept Analysis* (for analyzing arbitrary relations between objects), *Triadic Concept Analysis* [26] (to analyze three-dimensional data), *Fuzzy FCA* [23] and *Rough FCA* [16] (were developed to work with uncertain data and approximations). These papers have proven that FCA is feasible for Big Data. The approach in [16] makes FCA useful for analyzing extremely large data. Also, FCA contributes to the analysis and mining of social networks [27] such as affiliation and interaction networks and possibly more complex structures using this theory and some of its extensions. [17] gives an overview of several extensions of the main FCA model, and shows that FCA algorithms are efficient from both theoretical and practical points of view [25]. Its time complexity and performance proves its supremacy over concurrent methods and allows us to use it for Big Data problems.

In this paper, to cope with complex data, we use the following FCA extensions: conceptual scaling [23] and Relational Concept Analysis (RCA) [18, 19]. The process of deriving a one-valued context from a many-valued context is called *conceptual scaling*.

Definition 4. $(Many - valued\ context)$. A many-valued context (G, M, W, I) consists of sets G, M and W and a ternary relation I between those three sets, i.e. $I \subseteq G \times M \times W$, for which it holds that $(g, m, w) \in I\ and\ (g, m, v) \in I$ always imply $w = v$.

Elements of G are still called objects. Elements of M are called (many-valued) attributes. Elements of W are called attribute values. Accordingly, the fact $(g, m, w) \in I$ means "the attribute m takes value w for object g", simply written as $m(g) = w$.

Standard FCA is restricted to data sets that are either already represented as binary relations or that can be easily transformed into such a representation [14]. Relational Concept Analysis (RCA) [18, 19] extends standard FCA by taking relations between objects into account.

The objective of RCA is to build a set of lattices whose concepts are related by relational attributes, similar to UML associations.

In RCA, data are organized within a structure composed of a set of contexts and a set of binary relations.

Definition 5. $(Relational\ context\ family)$. A relational context family F is a pair (K, R), where K is a set of contexts $K_i = (G_i, M_i, I_i)$ - with objects G_i, properties M_i and a relationship I_i between these objects and properties; and R is a set of Object-Object relations $r_k \subseteq G_i \times G_j$ where G_i and G_j are the object sets of the formal contexts K_i and K_j. The structure (K, R) can be compared to a relational database schema, including both classes of individuals and classes of relations.

For example Table 2 shows two Object – Attribute contexts of some authors and their papers presented in different conferences. Table 3 represents the Object-Object context related to Table 2.

In RCA data tables are iteratively merged into one in the following way: in each step all formal concepts are computed of one data table and these concepts are used as additional attributes for the merged data table. After obtaining a final merged data table, all formal concepts are extracted.

The RCA methodology is the following: given the RCF Object-Attribute contexts and Object-Object contexts - the concept lattice is built for each Object-Attribute context at first, then relational scaling is applied to all Object-Object contexts and relational extension of each Object-Attribute context is built. Finally the concept lattice for each relational extension is constructed, thus a concept lattice family is obtained.

Table 2
RCF: Object-Attributes contexts

Author	Lecturer	PhD Student	Professor
Adam		X	
Tom			X
Jack			X
Lena		X	
Jim	X		
Sophia	X		
Lucia			X
Mike			X

Paper	ICFCA	ADBIS	VLDB	SIGMO	KDD
FESTA	X				
RK			X		
ELKA					X
OpenR					X
XKENA		X			

Table 3
RCF: Object-Object context

	FESTA	RK	ELKA	OpenR	XKENA
Adam	X		X		X
Tom	X				X
Jack		X			
Lena		X			
Jim					
Sophia	X		X		
Lucia				X	
Mike				X	X

3 Document Store Data Modeling using RCA

In this section we investigate the architectural challenges in document based NoSQL database design. The Entity-Relationship diagram is the most common tool for conceptual schema design. It is independent of the physical implementation of the database. It can be transformed to any other data model: relational, object oriented, hierarchical, semi-structured etc. In relational data model design, the tables obtained from the E-R diagram can be analyzed for different normal forms. If an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization.

Documents using the semi-structured data model may contain redundant information and may be prone to update anomalies. Such problems are caused by some functional dependencies. XML Functional Dependency (FD) and normal form XNF for XML documents were defined by Arenas and Libkin introducing the so-called tree tuple approach [2]. Yu and Jagadish [13] show, that these XML FD notions are insufficient and propose a Generalized Tree Tuple (GTT) based XML functional dependency and key notion, which includes particular redundancies involving set elements. Based on these concepts, they present the GTT-XNF normal form. We offer some FCA based tools for finding functional dependencies in XML documents and propose a correct XML scheme in [14, 5].

Our FCA based method gives a mapping of E-R diagrams to RCA, using a graphical representation of binary relationships having two cardinalities. We consider that carefully designed relationships using our method will produce a document in XNF normal form. The method is simpler than the normalization process of semi-structured data.

3.1 Conceptual Design of Semi-structured Data

We model the Entity-Relationship (E-R) schema as a Relational Context Family as follows: The E-R schema consists of entity sets, attributes, and relationships between entity sets. Since attributes are properties representative of real world objects, they are many-valued, hence we can represent every entity set of the E-R model as a *many-valued context*. Let $E_1, E_2, \dots, E_n$ be entity sets of an E-R diagram. Every entity set E_i, $i = 1, \dots, n$ will be modelled as a many-valued context. The objects of the many-valued context modelled for entity set E_i will be the entities from E_i. Let us denote by $A_{i_1}, A_{i_2}, \dots, A_{i_k}$ the attributes of entity set E_i, which will be the attributes of the many-valued context. Entity sets are connected by relations. The relationship between entities from E_i and E_j will be represented using their entity keys in a formal context.

Let R_{ij} be a relation between the entity sets E_i and E_j. The Object-Object context of R_{ij} is defined as the context having as object sets different key values of E_i and

E_j. The relationship between objects of E_i and objects of E_j is given in the incidence table.

The simplest form of a relationship is the binary relation. Relationships involving more than two entity sets are called n-ary relations. Binary relationships have two cardinality constraints [9] of the form (x; y), where x, y are natural numbers, x specifies the minimum cardinality and y the maximum participation constraint. Let us consider two entity sets E_1 and E_2 and a binary relation R between them with left cardinality constraint (x_1; y_1) and right cardinality constraint (x_2; y_2) denoted with: $E_1 \overset{(x_1,y_1)}{\longleftrightarrow} R \overset{(x_2,y_2)}{\longleftrightarrow} E_2$.

Based on their maximum cardinality constraints binary relationships are:

(i) One-to-One, if both roles have maximum cardinality 1

(ii) One-to-Many, if one role has maximum cardinality 1 and the other one has maximum cardinality N

(iii) Many-to-Many, if both roles have maximum cardinality N

The same E-R conceptual schema can be mapped into a different schema for semi-structured data. We consider two alternative ways of mapping E-R conceptual schemas into schema for semi-structured data: a relational-style (flat) design methodology and a nesting (or embedding) approach. In the flat schema model each entity is at the same level of the hierarchy and uses references to another entity as foreign keys in relational databases, the schema never nests. The nested schema embed entities as much as possible. M. Franceschet et al. in [10] proved that both validation of data and query processing are globally more efficient with nested schemas than with flat ones. Highly nested XML schemas reduce the number of expensive join operations.

To optimize application performance and reliability, a NoSQL schema must be driven by the application's intended purpose; it is about our data and how it is used. In the design process one has to decide whether flat or nested structuring optimizes one's schema. Both designs have advantages and disadvantages. The main advantage of the nested data model is that one can retrieve the complete class master information with one query. The main disadvantage of it is that there is no way of accessing the nested details as stand-alone entities.

In our method we will first create the Object-Object context of our data, build the concept lattice related to that context and read its background knowledge. From the graphical representation of the built concept lattice, one can discover and understand the conceptual relationships within a given set of data. Reading the built concept lattice we can decide if nesting is feasible or not based on Algorithm 1. We distinguish between the nodes of the Object-Object lattice the top node, the bottom node and intermediate nodes. The concept nodes of the Object-Object lattice are labeled with the keys of the entities.

Algorithm 1.

```
Input: a concept lattice
Output: nesting is possible or not
begin
if intermediate nodes are situated in one level then
   if intermediate nodes are labeled with utmost one key from $E_1$
     and utmost one key from $E_2$ then
        the relationship is One-to-One;
        if top node has labels and bottom node has labels then
             nesting is not possible;
        else
             nesting is possible;
        end;
   end;
   if intermediate nodes are labeled with more keys from $E_1$
     and utmost one key from $E_2$ then
        the relationship is Many-to-One;
        if top node has labels from $E_1$ or bottom node has labels from $E_1$ then
             nesting is not possible;
        else
             nesting is possible; // $E_1$ nested in $E_2$;
        end;
   end;
else
   the relationship is Many-to-Many;
   nesting is not possible;
end;
end;
```

Next, we will discuss in detail each of these three basic forms of relations: One-to-One, One-to-Many and Many-to-Many.

3.1.1 One-to-One Relationship Mapping

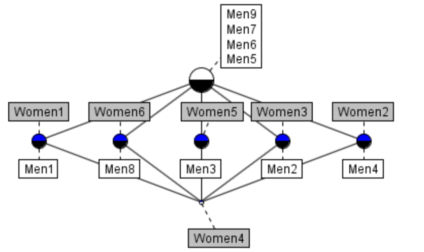
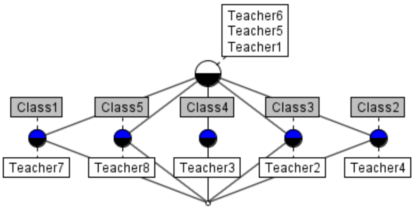
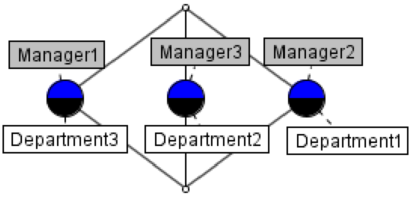
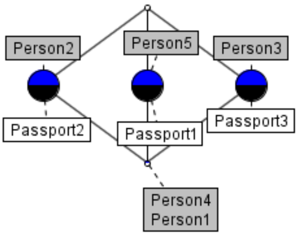
In general, the nested data model is the most advantageous, but when modeling One-to-One relationships, we have to carefully think through the structure of our data. Let $\boldsymbol{E_1}$ and $\boldsymbol{E_2}$ be two entity sets and R a One-to-One relationship between them. The question is how to nest them: $\boldsymbol{E_1}$ in $\boldsymbol{E_2}$ or $\boldsymbol{E_2}$ in $\boldsymbol{E_1}$? If we are using a nested data model and there are elements of $\boldsymbol{E_1}$ which are not related to any element of $\boldsymbol{E_2}$, or elements of $\boldsymbol{E_2}$ which are not related to any element of $\boldsymbol{E_1}$, then we may lose some elements of $\boldsymbol{E_1}$ or $\boldsymbol{E_2}$. Our method, using RCA helps one to decide if nesting is possible and provides the answer as how the entities should be nested.

In order to decide which data model to use we will firstly create the Object-Object context of our data and build the related concept lattice. The concept lattice will be the basis of further analysis. (We will present the Object-Object contexts only in case of One-to-Many relationships).

Depending on its cardinality the One-to-One relationship has four cases. Table 4 gives examples of these cases. We can observe, that for every concept (excepting the top and bottom of the lattice) there exists one element from $\boldsymbol{E_1}$ and one element from $\boldsymbol{E_2}$. The concept lattices vary only in bottom and top elements, but these are decisive in the nested data scheme. The elements of $\boldsymbol{E_1}$ and $\boldsymbol{E_2}$, which are not related to each other, will appear in the top or bottom of the lattice.

Table 4

Concept lattices of Object-Object context in case of One-to-One relationships

 Men9 Men7 Men6 Men5 Women1 Women6 Women5 Women3 Women2 Men1 Men8 Men3 Men2 Men4 Women4	 Teacher6 Teacher5 Teacher1 Class1 Class5 Class4 Class3 Class2 Teacher7 Teacher8 Teacher3 Teacher2 Teacher4
a) *Men* $\overset{(\mathbf{0,1})}{\leftrightarrow}$$\boldsymbol{spouse}$ $\overset{(\mathbf{0,1})}{\leftrightarrow}$ *Women* There are dangling entities in $\boldsymbol{Men}$ and $\boldsymbol{Women}$, the top and bottom of the lattice contains non empty intent/extent, thus nested design is not possible. The inclusion of $\boldsymbol{Women}$ in $\boldsymbol{Men}$ would lead to the loss of $\boldsymbol{Women}$ elements which are not related to elements of $\boldsymbol{Men}$ and vice-versa.	b) *Teacher* $\overset{(\mathbf{0,1})}{\leftrightarrow}$$\boldsymbol{responsible}$ $\overset{(\mathbf{1,1})}{\leftrightarrow}$ *Class* Every element of *Class* is related with one element of $Teacher$. The nested design is possible, including *Class* in $Teacher$, but not the inverse.
 Manager1 Manager3 Manager2 Department3 Department2 Department1	 Person2 Person5 Person3 Passport2 Passport1 Passport3 Person4 Person1
c) *Department* $\overset{(\mathbf{1,1})}{\leftrightarrow}$$\boldsymbol{managed}$ $\overset{(\mathbf{1,1})}{\leftrightarrow}$ *Manager* The intent of the top element and extent of the bottom element of the concept lattice are empty, thus both nesting is correct, we can include $\boldsymbol{Department}$ in $\boldsymbol{Manager}$ or $\boldsymbol{Manager}$ in $\boldsymbol{Department}$.	d) *Passport* $\overset{(\mathbf{1,1})}{\leftrightarrow}$$\boldsymbol{managed}$ $\overset{(\mathbf{0,1})}{\leftrightarrow}$ *Person* This case allows the existence of elements of $\boldsymbol{Person}$ not related to elements of $\boldsymbol{Passport}$, but every element of $\boldsymbol{Passport}$ is related to one element of $\boldsymbol{Person}$. The nested design is possible, $\boldsymbol{Passport}$ can be included in $\boldsymbol{Person}$, but not the inverse.

When one knows how to read a concept lattice, it can indeed provide valuable information. In [14] we have shown that an XML database can be translated into a power context family and that the functional dependencies of such a database correspond to FCA implications in a certain formal context. We are also able to visualize the structures of the different normal forms in a lattice. In Table 4 the possibility of nested scheme design can be read from the lattices. At this point, FCA proves to be a valuable tool for the design of such schemas. It gives an expressive graphical representation of the relationships between entities. This paper does not claim that these visualizations solve any computational problem or create new means for practical implementations. Instead, the visualizations are meant to serve as explanatory aids, to help us to decide which schema design to choose. NoSQL data modeling often requires a deeper understanding of data structures and algorithms than relational database modeling does. To the best of our knowledge, currently there are no applications which help in choosing between NoSQL data modeling techniques. Thus an FCA based visual aid is beneficial.

3.1.2 One-to-Many Relationship Mapping

In general, the N-side of a relationship is nested, if there is no need to access the embedded object outside the context of the parent object or one can use an array of references to the N-side objects if the N-side objects must stand alone, but we have to carefully think through the structure of our data.

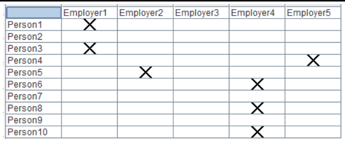
In order to decide which data model to use we will first create the Object-Object context of our documents and build the related concept lattice. Then we analyze the obtained lattice, reading the background knowledge from it, using Algorithm 1, to determine if nesting is possible.

Depending on its cardinality the One-to-Many relationship has four cases. We present three of them in Table 6. We denote $\boldsymbol{E_1}$ the entities of the left hand side of the One-to-Many relation and $\boldsymbol{E_2}$ the right hand side respectively, as a working example to generate the concept lattice, $\boldsymbol{E_1}$ representing the N side of the One-to-Many relationship. If we analyze the obtained lattices (Table 5) we can observe, that for every concept (excepting the top and bottom) there exists one element from $\boldsymbol{E_2}$ and N elements from $\boldsymbol{E_1}$. This illustrates the relationship between elements of $\boldsymbol{E_1}$ and elements of $\boldsymbol{E_2}$.

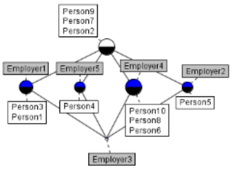
For the sake of simplicity, in the following examples we have presented very small concept lattices, but it has been shown in [3, 12] that readable lattices can be produced from real data sets with a straightforward process of creating sub-contexts.

Table 5

Object-Object contexts and their concept lattices in case of One-to-Many relationships

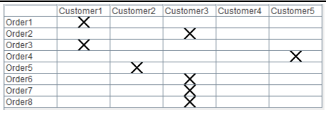


	Employer1	Employer2	Employer3	Employer4	Employer5
Person1	X				
Person2					
Person3	X				
Person4					X
Person5		X			
Person6				X	
Person7					
Person8				X	
Person9					
Person10				X	

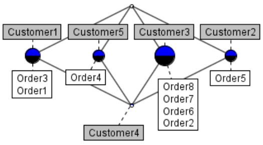


e) *Persons* $\overset{(\mathbf{0,1})}{\leftrightarrow}\boldsymbol{Employed}\overset{(\mathbf{0,}\boldsymbol{N})}{\leftrightarrow}$ *Employers*

There are elements of $\boldsymbol{Persons}$ which are not related to elements $\boldsymbol{Employers}$ being displayed at the top of the lattice, as well as elements of $\boldsymbol{Employers}$ which are not related to elements of Persons, appearing at the bottom of the lattice. Having elements in $\boldsymbol{Persons}$ representing the $\boldsymbol{N}$ side not related to a parent, hierarchical design is not possible. There can be persons who are not employed and employers who have no employed person.

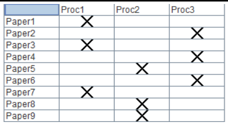


	Customer1	Customer2	Customer3	Customer4	Customer5
Order1	X				
Order2			X		
Order3	X				
Order4					X
Order5		X			
Order6			X		
Order7			X		
Order8			X		

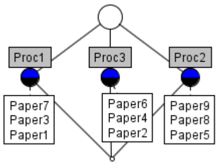


f) *Orders* $\overset{(\mathbf{1,1})}{\leftrightarrow}\boldsymbol{Ordered}\overset{(\mathbf{0,}\boldsymbol{N})}{\leftrightarrow}$ *Customers*

There are elements of $Customers$ which are not related to elements of $Orders$, appearing in the bottom of the lattice, but every element of $Orders$ is related with one element of $Customers$. Hierarchical design is possible, including $Orders$ in $Customers$.



	Proc1	Proc2	Proc3
Paper1	X		
Paper2			X
Paper3	X		
Paper4			X
Paper5		X	
Paper6			X
Paper7	X		
Paper8		X	
Paper9		X	



g) *Papers* $\overset{(\mathbf{1,1})}{\leftrightarrow}\boldsymbol{Appeared}\overset{(\mathbf{1,}\boldsymbol{N})}{\leftrightarrow}$ *Proceedings*

There are no dangling entities: every element of $\boldsymbol{Papers}$ is related to one element of $\boldsymbol{Proceedings}$, and also every element of $\boldsymbol{Proceedings}$ is related with elements of $\boldsymbol{Papers}$. Nested structure is possible.

3.1.3 Many-to-Many Relationship Mapping

Concepts of conceptual lattices which represent Many-to-Many relationships are on more hierarchical levels. The top and bottom of the concept lattice are labeled if there are dangling elements in the entity sets A and B.

Example 3. Consider the Many-to-Many relation between Students and Courses: Students $\overset{(0,N)}{\leftrightarrow}$Choose $\overset{(0,N)}{\leftrightarrow}$ Courses. One course can be chosen by 0 or more students and there can be students who choose 0 or more courses.

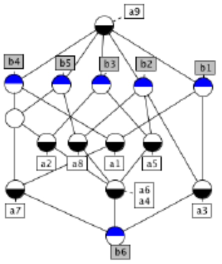


Figure 2

Concept lattice of Many-to-Many relationship

Nested design is not possible in this case.

In the case of Many-to-Many relationships, one can use bi-directional referencing if you are willing to pay the price of not having atomic updates or use of application-level joins, they are barely more expensive than server-side joins if you index correctly and use the projection specifier.

4 Experimental Evaluation

In the following we will use MongoDB for our discussion as it is one of the leading open-source NoSQL databases due to its simplicity, performance, scalability, and active user base. It has to be emphasized that our aproach is available in all semistructured datastores, not only in MongoDb.

We use the DBLP [6] bibliography dataset imported in MongoDB to test queries on flat versus nested structure of the documents. Our experiments were conducted on three computers, both with i7 processors and 4 GB RAM, running with a Windows 7 (64bit) OS.

3.2 Data Structure of Experimental Data

The structure of DBLP data is described in [7], where the flat representation of XML data is used. We import 1.559.572 conference proceedings and 1.232.729 journal articles from DBLP dataset. An *inproceeding* document is designed for the paper and a *proceeding* document for the volume. The journals are also stored in the proceeding collection with key value beginning with journal. We have 25593 proceedings and journal documents. Journal articles are also stored in the inproceeding collection.

The relationship between *papers* and *proceeding* is presented in Table 5 case g) from 3.1.2. Using Algorithm 1 we can see that nesting is possible.

We import the DBLP XML data in MongoDB in flat style first. In case of flat representation the *key* and *crossref* fields were used in order to map the one to many relationship between *proceeding* and the *papers* published in it, see Example 4. Then we constructed the nested representation in MongoDB using as input the flat MongoDB data. We used indexes for *crossref* field of *inproceeding* collection in flat style data to improve the retrieval of *papers* for one *proceeding*.

Example 4. Consider the flat structure: one conference *proceeding* and two *inproceeding* documents:

```
PROCEEDING:
{   "_id" : ObjectId("54d61a311ba3b50d0c1f5a20"),
    "key" : "conf/cla/2007",
    "editor" : ["Peter W. Eklund", "Jean Diatta", "Michel Liquiere"],
    "title" : "Proceedings of the Fifth International Conference on Concept Lattices
     and Their Applications, CLA 2007, Montpellier, France, October 24-26, 2007",
    "booktitle" : "CLA",
    "publisher" : "CEUR-WS.org",
    "volume" : "331",
    "year" : "2008",
    "series" : "CEUR Workshop Proceedings",
    "url" : "db/conf/cla/cla2007.html" }

INPROCEEDING:
{   "_id" : ObjectId("54d61a311ba3b50d0c1f59e4"),
    "author" : ["Radim Belohlvek", "Bernard De Baets", "Jan Outrata", "Vilm
                        Vychodil"],
    "title" : "Inducing Decision Trees via Concept Lattices.",
    "year" : "2007",
    "ee" : "http://ceur-ws.org/Vol-331/Belohlavek3.pdf",
    "crossref" : "conf/cla/2007",
    "url" : "db/conf/cla/cla2007.html#BelohlavekBOV07" }
{   "_id" : ObjectId("54d61a311ba3b50d0c1f59f4"),
    "author" : ["Laszlo Szathmary", "Amedeo Napoli", "Sergei O. Kuznetsov"],
    "title" : "ZART: A Multifunctional Itemset Mining Algorithm.",
    "year" : "2007",
    "ee" : "http://ceur-ws.org/Vol-331/Szathmary.pdf",
    "crossref" : "conf/cla/2007",
    "url" : "db/conf/cla/cla2007.html#SzathmaryNK07" }
```

Example 5. Consider the nested structure: two *inproceeding* documents nested in a conference *proceeding*:

```
NESTEDPROCEEDING:
{   "_id" : ObjectId("54d61a311ba3b50d0c1f5a20"),
    "key" : "conf/cla/2007",
    "editor" : ["Peter W. Eklund", "Jean Diatta", "Michel Liquiere"],
    "title" : "Proceedings of the Fifth International Conference on Concept Lattices
```

```
 and Their Applications, CLA 2007, Montpellier, France, October 24-26, 2007",
"booktitle" : "CLA",
"publisher" : "CEUR-WS.org",
"volume" : "331",
"year" : "2008",
"series" : "CEUR Workshop Proceedings",
"url" : "db/conf/cla/cla2007.html",
"inproceedings" :
  [ { "author" : ["Radim Belohlvek", "Bernard De Baets", "Jan Outrata", "Vilm
                  Vychodil"],
      "title" : "Inducing Decision Trees via Concept Lattices.",
      "year" : "2007",
      "ee" : "http://ceur-ws.org/Vol-331/Belohlavek3.pdf",
      "url" : "db/conf/cla/cla2007.html#BelohlavekBOV07" },
    { "author" : ["Laszlo Szathmary", "Amedeo Napoli", "Sergei O. Kuznetsov"],
      "title" : "ZART: A Multifunctional Itemset Mining Algorithm.",
      "year" : "2007",
      "ee" : "http://ceur-ws.org/Vol-331/Szathmary.pdf",
      "url" : "db/conf/cla/cla2007.html#SzathmaryNK07" }] }
```

3.2.1 Testing the Queries

We test ten queries for a single server MongoDB configuration and the same queries for three servers. In the case of three servers the queries were executed by one and three different users. The queries were executed with different parameters in different order more times. In the graphic representation we consider the average of the execution times.

The first five queries do not involve both *proceeding* and *inproceeding* type documents, only one of them. In the last five queries both - *proceeding* and *papers* needs to be accessed. In the case of nested design, the paper documents are embedded in the containing proceeding document. We create index on *crossref* field of *inproceeding* collection in case of flat representation to improve the 'join' operation, which has to be solved programmatically.

The queries were:

Query 1: Select the conference papers and journal articles for one author.

Query 2: Select the journal articles written by author1 and written by author2 in a given year or articles written by author1 and not written by author3 in a given year.

Query 3: Select the conference papers and journal articles in 3 given year.

Query 4: Select conference proceeding or journal information given a *booktitle* value. The *booktitle* is the same for a series of conferences which are held every year or a journal which appears more times in a year.

Query 5: The same as Query 4 complemented with a condition excluding a given year.

Query 6: Select conference or journal from proceedings collection given a *booktitle* value and a year value, together with papers from the selected proceeding.

Query 7: Given a paper title finds the paper and the proceeding where the paper was published.

Query 8: Given a part of a paper title finds the papers and the proceedings where the papers were published.

Query 9: Given a part of a proceeding title finds the corresponding proceedings and the papers published in these proceedings.

Query 10: Select journal articles of a given author in a given journal, in a given volume, for a given year and the corresponding journal.

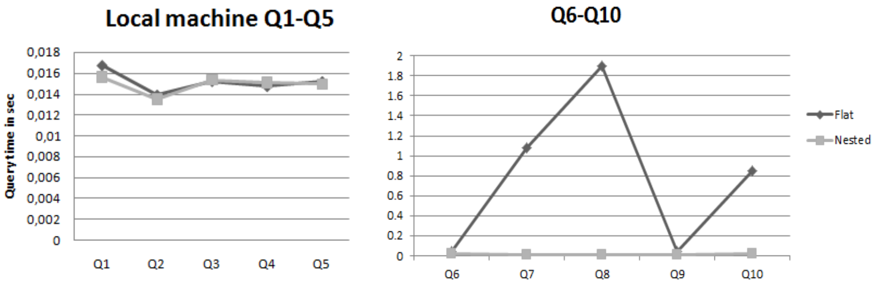


Figure 3
Execution time of queries on a local machine

The execution time for every query in nested design mode of data is between 0.01 and 0.02 sec in every case, using local machine or 3 servers. The execution time in flat structure of data for queries 6-10 is between 0.04 and 1.89 sec. Queries 1-5 are executed nearly at the same time in embedded and non-embedding cases.

In Figure 3 we can see the average execution time on a local machine for the ten queries in flat (Example 4) and nested (Example 5) structure of the data. The experiments for queries 6-10 show that the nested design mode of data is more suitable. The execution of the first five queries on one machine does not depend on the structure of the data.

Figure 4 presents the results of query execution on three servers with one user. In this case, the results are nearly the same, for queries 1-5 the structure of the data is not relevant, but for queries 6-10 the nested design of the data is more appropriate.

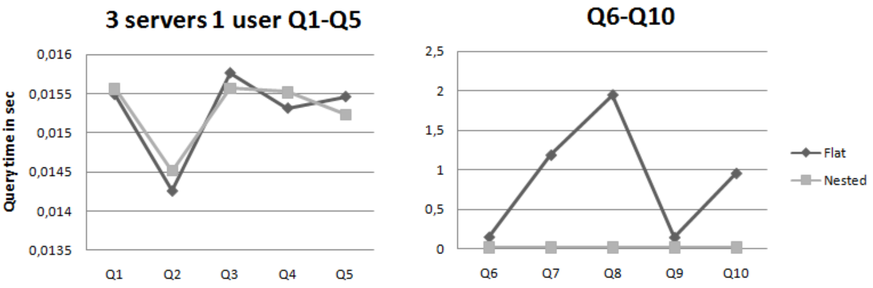


Figure 4

Execution time of queries on three servers one user

Figure 5 illustrates the execution time of the same queries on three servers by three different users concurrently. The results are nearly the same, nested structure of data is superior in this case as well.

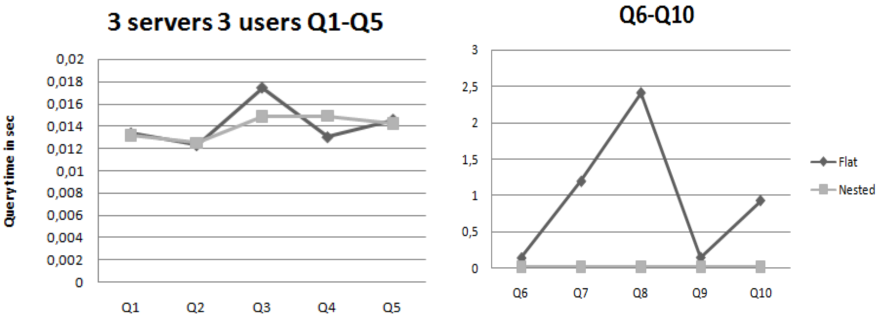


Figure 5

Execution time of queries on three servers with three users

Figure 6 presents the execution times for the three different architectures in the case of flat data structure. The differences are not relevant between the local machine and distributed architectures.

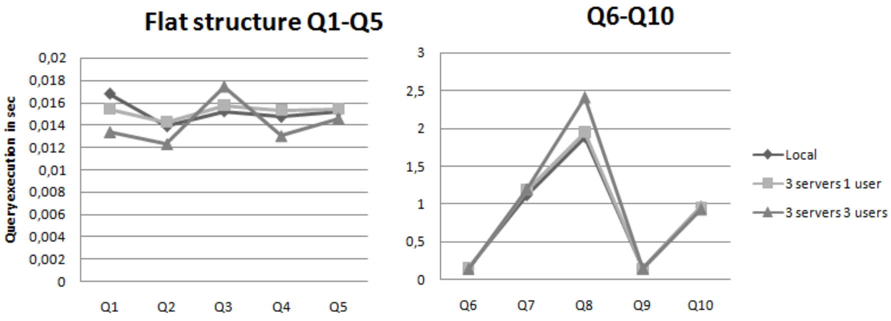


Figure 6
Execution time of queries for flat data structure

Figure 7 compares the query execution time for the nested structure of the data on different system architecture.

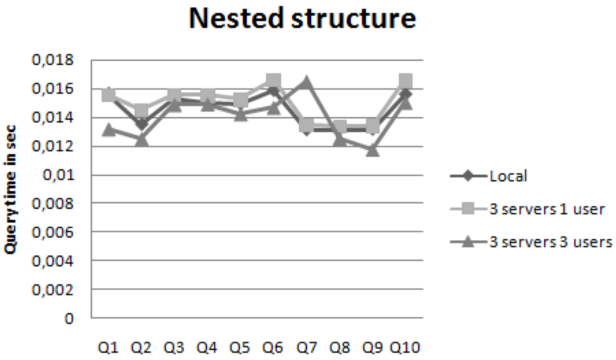


Figure 7
Execution time of queries for nested data structure

Our method of using RCA helps us to decide if nesting is feasible or not. We tested all queries in flat and nested versions and the results show that if we choose the design method, which is suggested by our algorithm, the execution time will be more efficient. Results of experiments using our method have proven that decisions affecting the modeling of data can affect application performance and database capacity.

Conclusions

In this paper we have proven that the more knowledge we have concerning a target domain, the better that certain tools can support the domain's analyses. Decisions that affect how we model data, can affect application performance and database capacity. We have covered the basics of data modeling for document based data stores. The three basic types of relations were discussed and illustrated with the help of examples.

We used FCA methods to visually analyze the schema of large-scale data. We proposed an RCA based approach to document based NoSQL database design. The possibility of nested scheme design can be read from the lattices. Results of experiments using our method have validated the feasibility of our approach.

References

[1] V. Abramova, J. Bernardino, NoSQL Databases: MongoDB vs Cassandra, Proceedings of the International C* Conference on Computer Science and Software Engineering, ACM, 2013, pp. 14-22

[2] M. Arenas, L. Libkin, A Normal Form for XML Documents. TODS 29(1), 2004, pp. 195-232

[3] S. Andrews, C. Orphanides, Analysis of Large Data Sets using Formal Concept Lattices, CLA 2010, pp. 104-115

[4] R. Cattell, Scalable SQL and NoSQL Data Stores. ACM SIGMOD Record 39.4, 2011, pp. 12-27

[5] K. T. Janosi Rancz, V. Varga, XML Schema Refinement Through Formal Concept Analysis, Studia Univ. “Babeş-Bolyai” Cluj-Napoca, Informatica, vol. LVII, No. 3, 2012, pp. 49-64

[6] M. Ley, DBLP Computer Science Bibliography. http://dblp.uni-trier.de/

[7] M. Ley, DBLP — Some Lessons Learned, Proc. VLDB Endowment, Vol. 2, Nr. 2, 2009, pp. 1493-1500

[8] Ecma International. The JSON Data Interchange Format, 2013, www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[9] R. Elmasri, S.B. Navathe: Fundamentals of Database Systems, Addison Wesley (2010)

[10] M. Franceschet, D. Gubiani, A. Montanari, C. Piazza: A Graph-Theoretic Approach to Map Conceptual Designs to XML Schemas, ACM Transactions on Database Systems, 2013, Vol. 38, pp. 6-44

[11] B. Ganter, R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer, 1999

[12] D. V Gnatyshak, D. I. Ignatov, S. O. Kuznetsov, L. Nourin, A One-Pass Triclustering Approach: Is There any Room for Big Data?, Proceedings of the 11th International Conference on CLA, 2014, pp. 231-242

[13] C. Yu, H. V. Jagadish, XML schema refinement through redundancy detection and normalization. VLDB J. 17(2), 2008, pp. 203-223

[14] K. T. Janosi-Rancz, V. Varga, T. Nagy, Detecting XML Functional Dependencies through Formal Concept Analysis, ADBIS 2010, Novi Sad, Serbia, LNCS 6295, pp. 595-598

[15] M. Klettke , U. Störl, S. Scherzinger: Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores, 16th Conference on Database Systems for Business, Technology, and Web (BTW), 2015

[16] B. Ganter, C. Meschke, A Formal Concept Analysis Approach to Rough Data Tables. Trans Rough Sets 2011, 14:37–61

[17] S. O. Kuznetsov, J. Poelmans, Knowledge Representation and Processing with Formal Concept Analysis, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, Vol. 3, Issue 3, pp. 200-215, 2013

[18] M. Rouane-Hacene, M. Huchard, A. Napoli, P. Valtchev, A Proposal for Combining Formal Concept Analysis and Description Logics for Mining Relational Data, ICFCA'07, France. Springer, LNAI 4390, pp. 51-65

[19] M. Rouane-Hacene, M. Huchard, A. Napoli, P. Valtchev, Relational Concept Analysis: Mining Concept Lattices from Multi-Relational Data, Annals of Mathematics and Artificial Intelligence 67, 1, 2013, pp. 81-108

[20] S. Tiwari, Professional NoSQL, O'Reilly, 2013

[21] S. Václav, H. Zdenek, A. Ajith, Understanding Social Networks Using Formal Concept Analysis, Proc. WI-IAT '08, Volume 03, pp. 390-393

[22] V. Varga, Ch. Sacarea, An FCA Driven Analysis of Mapping Conceptual Designs to XML Schemas, Studia Univ. "Babeş-Bolyai" Cluj-Napoca, Informatica, Vol. LIX, No. 1, 2014, pp. 46-57

[23] R. Belohlavek, Fuzzy Galois Connections. Math Logic Q 1999, 45:497–504

[24] B. Ganter, S. O. Kuznetsov, Pattern Structures and their Projections. Proc. of 9th ICCS'01. LNAI, 2120; July 30-August 3, 2001; Stanford University, CA. Berlin, Heidelberg: Springer 2001, 129-142

[25] D. Gnatyshak, D. I. Ignatov, S. O. Kuznetsov, L. Lourine: A One-pass Triclustering Approach: Is There any Room for Big Data?, CLA 2014, 231-242

[26] F. Lehmann, R. Wille, A Triadic Approach to Formal Concept Analysis. ICCS; August 14-18; Santa Cruz, CA. Berlin, Heidelberg: Springer; 1995, 32-43