Received 13 April 2016
Accepted 5 May 2016

On proposing and evaluating a NoSQL document database logical approach

Cláudio Lima
Federal University of Santa Catarina, Florianopolis, Brazil, and
Ronaldo Santos Mello
Informatics and Statistics Department,
Federal University of Santa Catarina, Florianopolis, Brazil

Abstract
Purpose – NoSQL databases do not require a default schema associated with the data. Even that, they are categorized by data models. A model associated with the data can promote better strategies for persistence and manipulation of data in the target database. Based on this motivation, the purpose of this paper is to present an approach for logical design of NoSQL document databases that consists a process that converts a conceptual modeling into efficient logical representations for a NoSQL document database. The authors also evaluate their approach and demonstrate that the generated NoSQL logical structures reduce the amount of data items accessed by queries.
Design/methodology/approach – This paper presents an approach for logical design of NoSQL document database schemas based on a conceptual schema. The authors generate compact and redundancy-free schemas and define appropriate representations in a NoSQL document logical model. The estimated volume of data and workload information can be considered to generate optimized NoSQL document structures.
Findings – This approach was evaluated through a case study with an experimental evaluation in the e-commerce application domain. The results demonstrate that the authors' workload-based conversion process improves query performance on NoSQL documents by reducing the number of database accesses.
Originality/value – Unlike related work, the reported approach covers all typical conceptual constructs, details a conversion process between conceptual schemas and logical representations for NoSQL document database category and, additionally, considers the estimated database workload to perform optimizations in the logical structure. An experimental evaluation shows that the proposed approach is promising.

Keywords Web databases, Extended entity-relationship, NoSQL data modeling, NoSQL document database, NoSQL logical design, Workload-driven approach

Paper type Research paper

1. Introduction
Applications from several domains, such as Web data management, social networks, sensor networks, e-commerce and educational evaluation, generate a massive amount of data every day. It brings several challenges for data management in the cloud, including how to handle and store these data. NoSQL databases (DBs) are designed to manage large volumes of data, commonly referred to as *Big Data*, and a large number of

International Journal of Web Information Systems
Vol. 12 No. 4, 2016
pp. 398-417
© Emerald Group Publishing Limited
1744-0084
DOI 10.1108/IJWIS-04-2016-0018

read-and-write operations, a common feature in modern Web applications (Cattell, 2010).

NoSQL DBs support complex data types, semi-structured or unstructured data, and although they do not require a default schema associated with the data, they are categorized by data models (key-value, document, columnar and graph-based) (McMurtry *et al.*, 2013), demonstrating that their data show some degree of structuring. In fact, data organization in NoSQL DBs requires significant design decisions because it affects important quality requirements such as scalability and performance (Bugiotti *et al.*, 2014). In addition, the importance of a model associated with the data is related to the definition of better strategies for persistence and manipulation of such data in the target DB.

In this context of data modeling, conceptual schemas and ontologies are crucial to define data semantics, providing access to them with higher accuracy. A traditional DB design is a process consisting of three data modeling phases (Batini *et al.*, 1992; Elmasri and Navathe, 2011): *conceptual*, *logical* and *physical* design. At the conceptual modeling phase, a schema with information about a domain is represented in a high-level abstraction model. In the sequence, in the logical modeling phase, the conceptual schema is transformed into a schema with lower abstraction but suitable to the target DB data model. This logical design phase, specifically for NoSQL document DBs, is the scope of this paper.

In database literature, support methodologies for the logical design of NoSQL DBs is still a topic little explored, despite its importance (Atzeni *et al.*, 2013). This paper aims to contribute to this issue by proposing an approach for the logical design of NoSQL document DBs. This approach consists a process that converts conceptual modeling for suitable and efficient logical representations for a NoSQL document DB. We chose document-oriented DBs because they are an appropriate category for Web applications or applications that deal with Big Data, once they provide semi-structured data storage and dynamic queries execution, as well as horizontal scalability and high availability (Kaur and Rani, 2013).

Our conversion approach for generating NoSQL document logical schemas from conceptual schemas can consider the expected workload of the target application. Workload information is provided by the designer in terms of the amount of data instances estimated for the NoSQL DB, as well as the main operations that will be performed over these data. This information is used to determine an optimized logical structuring for the NoSQL DB schema, contributing, in general, to a better access performance for the application. We also evaluate our approach through a case study with an experimental evaluation in the e-commerce domain. An existing data set was redesigned by our approach to compare the number of accesses generated by queries over the redesigned schema as well as over the schemas generated by our approach. We demonstrate that our method can improve query performance on NoSQL documents by reducing the number of access to the NoSQL DB. It highlights that our workload-aware design process is promising.

The remainder of this paper is organized as follows. Section 2 presents an analysis of related work. Section 3 gives an overview of our approach for converting conceptual schemas into NoSQL document logical schemas. Section 4 presents a case study to evaluate our logical design approach, and Section 5 is dedicated to the conclusion.

2. Related work

This section presents a feature comparison of the (few) related works that focus on NoSQL DBs modeling. A different related work comparison, including proposals that deal with other non-relational schemas (XML and object-oriented), is presented in Lima and Mello (2015) to identify the modeling levels (conceptual, logical and physical) attended by each of them, being complementary to this comparison.

The work of Bugiotti *et al.* (2014) presents an approach to the NoSQL DBs design that explores the commonalities of some NoSQL DB categories. The proposal introduces a data model called *NoAM* (*NoSQL Abstract Model*) for the logical level, and demonstrates how data modeled in NoAM can be implemented in some NoSQL DBs. NoAM is based on the concept of *aggregates* for NoSQL data modeling, which is a term of domain-driven design (DDD) area (Evans, 2003).

The work of Jovanovic and Benson (2013), uses *IDEF1X* (*Integration DEFinition for Information Modeling*), a data modeling language for the development of semantic data models, in the conceptual modeling phase to represent the application domain, and also to represent an aggregate-based NoSQL logical model obtained through a conversion process between these models. This proposal gives support to the analysis of different modeling strategies like schema partitioning into smaller and independent aggregates in the SOA context (*service-oriented architecture*).

The approach of Chebotko *et al.* (2015) presents a data modeling methodology, directed by main application queries, to the NoSQL columnar DB Cassandra. It introduces principles of modeling, rules and standard mappings to guide the logical data modeling to the Cassandra DB. Visual diagrams for logical and physical data model for Cassandra DB are defined, which are called *Chebotko diagrams*.

Table I presents a feature comparison of these approaches for NoSQL DBs modeling. Our approach is also considered in this comparison (last column of Table I). The first column lists the features of the considered approaches, which are detailed below. As shown in Table I, the evaluation of the approaches regarding these features is provided by setting one of the four possible choices:

Table I.
Feature comparison of related approaches

	Approach			
Feature	Bugiotti *et al.* (2014)	Jovanovic and Benson (2013)	Chebotko *et al.* (2015)	Lima and Mello (2015)
1	✓✓✓✓	✓✓✓	✓✓✓	✓✓✓✓
2	✓✓	✓✓✓	✓✓✓	✓✓✓✓
3	✓	✓✓	✓✓✓	✓✓✓✓
4	✓	✓	✓✓	✓✓✓✓
5	✓✓✓✓	✓	✓	✓✓✓✓
6	✓	✓	✓✓✓	✓✓✓✓
7	✓✓✓✓	✓	✓	✓✓✓✓
8	✓	✓	✓✓✓	✓✓✓✓
9	✓✓✓	✓✓✓	✓	✓✓

Notes: ✓✓✓✓ Complete/Detailed; ✓✓✓ Incomplete/Partial; ✓✓ Superficial/Specific; ✓ Poor/Very specific/Non-existent

(1) complete/detailed;
(2) incomplete/partial;
(3) superficial/specific; and
(4) poor/very specific/non-existent.

The proposed features as well as the consideration of each one of them by the related work are presented in the following text.

2.1 Conceptual model considered for DB design

Our approach (identified as Lima and Mello) uses the EER model, and the Bugiotti *et al.* (2014) approach uses the UML class diagram. We consider that both models are suitable to a DB conceptual design. Jovanovic and Benson (2013) use IDEF1X and Chebotko *et al.* (2015) use the ER model. These conceptual models have lower modeling capabilities compared to the EER model or UML class diagram.

2.2 Full use of the concepts of the conceptual model

Different from the related work, our proposal covers all typical conceptual constructs of the EER model. Bugiotti *et al.* (2014) consider few concepts of the UML class diagram and, for this reason, we evaluate it as superficial. Jovanovic and Benson (2013) and Chebotko *et al.* (2015) deal with a bigger set of concepts, being evaluated as incomplete.

2.3 Conversion rules detailing (conceptual to logical levels)

Our approach details the conversion rules between conceptual constructors and logical representations for a NoSQL document DBs category. Chebotko *et al.* (2015) present a set of conversion rules on their website and, for this reason, we evaluate it as incomplete. Jovanovic and Benson (2013) present a smaller set of conversion rules, being evaluated as superficial. Bugiotti *et al.* (2014) do not present the conversion rules.

2.4 Conversion process detailing (conceptual to logical levels)

Different from the related work, our approach details the conversion process between conceptual schemas and logical representations for a NoSQL document DBs category. Chebotko *et al.* (2015) present some information about the conversion process on their website and, for this reason, we evaluate as superficial. Bugiotti *et al.* (2014) and Jovanovic and Benson (2013) do not present a conversion process.

2.5 Experimental evaluation

Our approach presents experiments with an analysis of the results. Jovanovic and Benson (2013) and Chebotko *et al.* (2015) do not present experiments, although the latter state that the approach is widely adopted in production environments. Bugiotti *et al.* (2014) present experiments with an analysis of the results and, for this reason, we evaluate it as complete.

2.6 Automation level of the conversion process

Only our approach defines a completely automatic conversion process. Chebotko *et al.* (2015) automate their approach. However, user interaction is expected to generate the logical schema. For this reason, their process is considered semi-automatic and the approach is evaluated as incomplete. Bugiotti *et al.* (2014) and Jovanovic and Benson (2013) do not present a conversion process.

2.7 Avoidance of data redundancy at the logical level
Our process generates a redundancy-free schema. Jovanovic and Benson (2013) and Chebotko *et al.* (2015) do not report about this issue. Bugiotti *et al.* (2014) state that the approach generates a redundancy-free schema and, for this reason, we evaluate it as complete.

2.8 Workload consideration
Only our approach considers the estimated DB workload to perform optimizations in the logical structure. The Chebotko *et al.* (2015) approach is driven by queries and, for this reason, we evaluate it as partial. Bugiotti *et al.* (2014) and Jovanovic and Benson (2013) do not consider the estimated DB workload to perform optimizations in the logical structure.

2.9 The considered logical model (NoSQL DB categories)
The Bugiotti *et al.* (2014) and Jovanovic and Benson (2013) proposals claim that three categories of NoSQL DBs are supported:

(1) key-value;

(2) document; and

(3) columnar.

The Chebotko *et al.* (2015) approach is the most specific one because it covers only a specific NoSQL columnar product. Our approach focuses on the NoSQL document DBs and, for this reason, we evaluate it as (category) specific.

In short, our main contributions with respect to related work are that our proposal covers all typical conceptual constructs, details the conversion algorithms between conceptual schemas and logical representations for a NoSQL document DBs category and additionally considers the estimated DB workload to perform optimizations in the logical structure.

Our conversion process is presented in the following.

3. The conversion process
Our approach provides the conversion of conceptual schemas into NoSQL document logical schemas. It starts with a conceptual schema and workload information given by the application designer, as shown in Figure 1. The workload information is estimated over a conceptual schema, being also used as input for the *logical design phase* to generate appropriate logical structures. The mapping of the conceptual schema to a

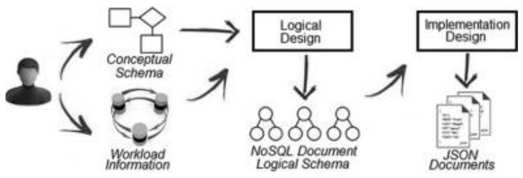

Figure 1.
An overview of the proposed approach

NoSQL document logical schema is governed by a set of rules that converts each conceptual constructor to an equivalent representation in the NoSQL document logical model.

Our logical model is an abstract model to represent NoSQL document implementation models. In the *implementation design phase*, a NoSQL document logical schema is translated to the common implementation model for NoSQL documents, i.e. the JSON[1] specification. Even though the *implementation design phase* is considered by our approach, this paper focuses on generating NoSQL document structures from a conceptual schema in the *logical design phase*.

Our input conceptual schema is defined by the extended entity-relationship (EER) model (Batini *et al.*, 1992; Elmasri and Navathe, 2011), a classical and suitable model for representing data concerning an application domain. We adopt EER because it contains the essential constructs for conceptual modeling. Workload information and the logical model defined by our approach are briefly presented in the following.

Workload information corresponds to the data load expected for a NoSQL-based application. This information allows our conversion process to choose an optimized NoSQL document structure to represent a conceptual schema. According to Batini *et al.* (1992), we may concentrate on the 20 per cent of the most frequent operations that will be performed by the application. This assumption is rooted on the so-called 20-80 rule, which says that 20 per cent of the operations produce 80 per cent of the application load. Our workload analysis identifies the concepts frequently accessed by transactions and is based on the workload modeling methodology defined in Batini *et al.* (1992), Schroeder and Mello (2008) and Schroeder *et al.* (2011). More details about the workload modeling methodology can be found in Lima and Mello (2015).

We propose a NoSQL document logical model to represent the document data model. Our conversion approach generates NoSQL schemas defined by this logical model in the *logical design* phase. The NoSQL document logical model is an abstract representation for NoSQL document schemas and consists an adaptation of the *aggregate* approach (Evans, 2003). In this context, an aggregate represents a collection of related objects, in a nested way, which can be treated as a unit. Such a notion is suitable to NoSQL documents, given that they are hierarchical data structures that consist of nested data collections and scalar values (Sadalage and Fowler, 2013). A NoSQL document logical schema is composed of *collections*, *blocks* and *attributes*. A schema has one or more collections, and each collection has a root block. All updates to a collection pass through its respective root block, ensuring the business rules. The root block is the only block accessible out of the collection. More details about the NoSQL document logical model can be found in Lima and Mello (2015).

Our conversion process is based on the conversion rules for mapping EER constructs into equivalent NoSQL document constructs in the logical model. Figure 2 presents the overall process, which comprises two main steps:

(1) conversion of hierarchy types; and

(2) conversion of relationship types.

As shown in Figure 2, hierarchy types are converted first, followed by the conversion of the relationships. The blocks generated by the *hierarchy conversion* step are maintained by the *relationship conversion* step. At the end of the process, a list of collections is returned. Load information can be considered during the conversion process to generate well-structured NoSQL document logical schemas.

The next sections summarize the hierarchy and relationship conversion. More details about them can be found in Lima and Mello (2015).

3.1 Hierarchy types conversion

A generalization hierarchy in the EER model defines a subset relationship between a generic entity, namely, superclass, and one or more specialized entities, namely, subclasses. The *disjointeness* and *completeness* constraints that are set to the subclasses establish four possible constraints on generalization types:

(1) total and disjoint (*t*, *d*);

(2) partial and disjoint (*p*, *d*);

(3) total and overlapping (*t*, *o*); and

(4) partial and overlapping (*p*, *o*) (Batini *et al.*, 1992).

Categories or *union types* of the EER model can be considered restricted cases of multiple inheritance (Batini *et al.*, 1992). Thus, their conversion strategies are similar to the strategies for processing generalization types. For sake of paper space, we omit these strategies.

In this section, we present alternative rules to convert a generalization hierarchy from an EER schema to a NoSQL document logical schema. We also introduce the process that selects the suitable rule to be applied on each occurrence of an EER generalization type.

3.1.1 Conversion rules. Three alternatives are provided to convert generalization types inspired by the relational logical design methodology (Batini *et al.*, 1992). The difference among these alternatives is given by the different size of the NoSQL document schema that each one generates and the constraints on generalization types they are able to support, as shown in Figure 3.

The conversion strategy defined by *Rule 1* generates only one block from a generalization hierarchy. The block represents the superclass and its attributes, as well as the attributes of its subclasses. Subclasses' attributes are defined as optional in the content model of the superclass block. On applying this rule, we assume that the subclasses' attributes will act as discriminating attributes to identify an instance of a subclass in the NoSQL documents. The subclasses previously converted (*marked subclasses*) become an optional inner block of the block generated by this rule.

The alternative defined by *Rule 2* generates only NoSQL document blocks for the subclasses, and the superclass attributes are reproduced into each subclass block. In the

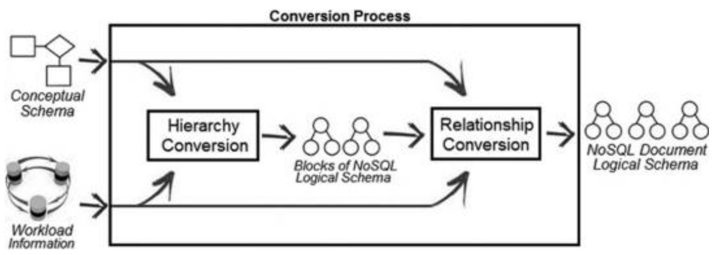

Figure 2.
The EER-NoSQL conversion process

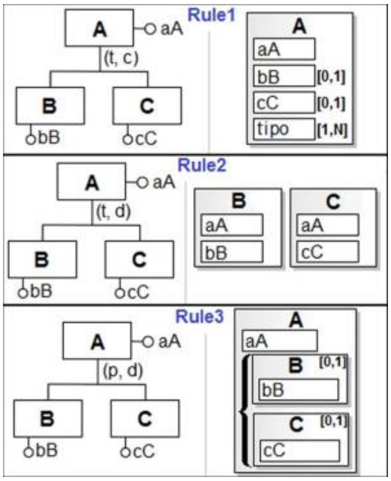

Figure 3.
Three alternatives to convert generalization types

alternative defined by *Rule 3*, the superclass and subclasses are explicitly represented by blocks. Hierarchical relationships are established among the superclass and subclasses blocks to represent the relationship. The generalization constraints are represented by the minimum and maximum occurrences of the subclasses' blocks in the superclass block.

3.1.2 Conversion process for hierarchy types. In our overall conversion process, a function called *convertHierarchies* is responsible to choose the appropriate rule for converting each generalization type of a conceptual schema. During the conversion process, load information can be considered to generate well-structured NoSQL document logical schemas. A generalization type is converted by analyzing the constraints of the generalization hierarchy and, optionally, the workload data. For sake of understanding, the conversion process is called *conventional* conversion when we do not consider workload information, and *optimized* conversion when we consider workload information.

The function establishes a conversion order in which the entities involved in a generalization hierarchy must be treated. A *bottom-up* conversion is accomplished when there is a multiple-level hierarchy, i.e. the entities are converted from the bottom to the top of the hierarchy. Besides, when there is a multiple-inheritance case, the superclass with the highest *General Access Frequency* (GAF) has high priority in the optimized conversion. It means that the superclass that is most frequently accessed becomes the parent block of a block that represents the subclass with more than one superclass. In this case, the remaining superclasses are related by reference attributes, as defined in Rule 3.

Once the conversion order of the generalization types is established, we apply the conversion rules for generalization types (Rule 1, 2 or 3) and verify the preconditions of each one, so that the rules that generate the smallest NoSQL document logical fragment are verified first. For the conventional conversion, we verify the presence of relationships among subclasses and superclasses for the application of the Rules 1 and 2, respectively. Rule 1 assumes that the explicit distinction between subclasses is irrelevant for most instances of the superclass. The existence of relationships involving subclasses is the main impediment for the application of this rule. Rule 2 is not considered for cases where the superclasses' relationships must be converted into relationships with each one of the subclasses. Otherwise, for the optimized conversion, before applying Rule 1 and Rule 2, we verify whether the GAF of the entities that will be omitted is lower than the *Minimal Access Frequency* (MAF). MAF is a value that represents the minimal frequency for accesses involving operations, and values below it are considered as insignificant frequencies. If the GAF is higher than MAF, it means that these entities participate in frequent operations and the distinction between superclass and subclasses must be preserved. If Rules 1 and 2 do not apply, the last option to convert a generalization type is Rule 3.

3.2 Relationship types conversion

A relationship type is a common conceptual construct that establishes a correspondence among two or more entities (Batini *et al.*, 1992). The cardinality of a relationship type is the main constraint that is considered in the conversion to a NoSQL document logical structure. Our rules for converting EER relationship types also proceed from the logical design of relational data models. In this section, we present these rules and their constraints, as well as the process that selects the suitable rule to be applied on each occurrence of an EER relationship type.

3.2.1 Conversion rules. We define three conversion rules that deal with specific constraints for relationship types, as shown in Figure 4. *Rule 4* is applied only to 1:1 relationships, *Rule 5* regards 1:N relationships and *Rule 6* is applied to relationships with cardinality N:N, n-ary ones with $n > 2$ or in cases where 1:1 and 1:N relationships cannot be treated by Rules 4 and 5.

Rule 4 generates only one block to represent the relationship type and its related entities. *Rule 5* generates blocks for each related entity, being one of them a nested block of the other one, and the relationship attributes are appended to the nested block. Finally, *Rule 6* generates independent blocks for each entity of a relationship type, and reference relationships are established among the generated blocks.

3.2.2 Conversion process for relationship types. In our overall conversion process, a function called *convertRelationships* controls the execution of the conversion rules for relationship types of an EER schema. It orders the relationship types, for conventional conversion, using the concept of *fully functional closures* (Mok and Embley, 2006), which determines the list of entities that can be reached from a starting entity through relationship pathways determined by participation (1,1) of entities in the relationships. After identifying the *fully functional closures* of the entities of the conceptual schema, a

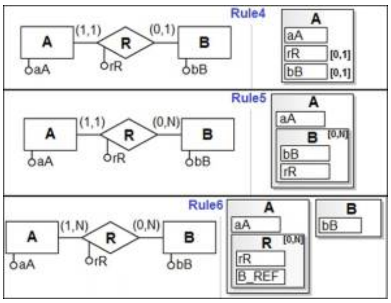

Figure 4.
Three alternatives to convert relationship types

list of entities (*EL*) is generated. These entities must be ordered in *EL* so that the participating entities on more *fully functional closures* appear first in the list. Then, the remaining entities are added to the end of *EL*, so that the entities that have a higher number of relationships appear first.

In case of the optimized conversion, the function orders the relationship types so that relationships with the highest GAF appear first. This order is established to give priority to the relationships that represent the largest impact on the application workload. Then, if there is more than one nesting possibility for an entity type giving all the relationship types on which it participates, we process this entity by considering the relationship with the highest GAF first. This order ensures that relationship types involving associative entities are converted after the internal relationship types of these associative entities.

For converting a relationship type, we first determine which entity will be the one on the top of the hierarchy in the NoSQL document logical schema. If the relationship type is binary and there is an entity with participation (1,1) in the relationship, the top entity is the other entity of the relationship type. For other cases, on considering the optimized conversion, the top entity is the entity type with the highest GAF in the relationship, i.e. we assume that the relationship type is more frequently accessed through this entity for the considered operations.

In the next section, we evaluate our approach with a case study in the e-commerce domain.

4. Case study

This section presents a case study to evaluate our conversion approach with an experiment in the e-commerce domain. Our intention here is to exemplify the application of our process and show its positive effects, in terms of processing time, on considering the application workload. In fact, we show here that our method can improve query performance on NoSQL documents by reducing the number of accesses to the NoSQL document DB. The process application, the experiment settings and the experiment results are presented in the following text.

4.1 Process application

The EER schema considered in this case study is presented in Figure 5. We perform a *reverse engineering* from a real e-commerce application data set to represent the main business processes related to e-commerce, that are related to transactions of customer's orders. We apply our conversion process twice over the conceptual schema. In the first time, we do not consider workload information, and the generated logical schema was called *conventional* schema. In the second time, we consider workload information, and the generated logical schema was called *optimized* schema.

For the generation of the *conventional* schema, presented in Figure 6, *Rule 2* was applied to Person's generalization hierarchy and *Rule 3* was applied to the Payment's category hierarchy. On considering relationship types, after identifying the *fully functional closures* of the entities of the conceptual schema, the list of entities obtained was *EL* = {*Customer, Category, Order, Product, Carrier, Item, Supplier, CreditCard, Payment, Bill, Person*}, and the relationship types *request*, *owner*, *delivery*, *composite*, *reference*, *catalog*, *furnishing* and *commitment* were converted by the rules 5, 6, 5, 5, 6, 6,

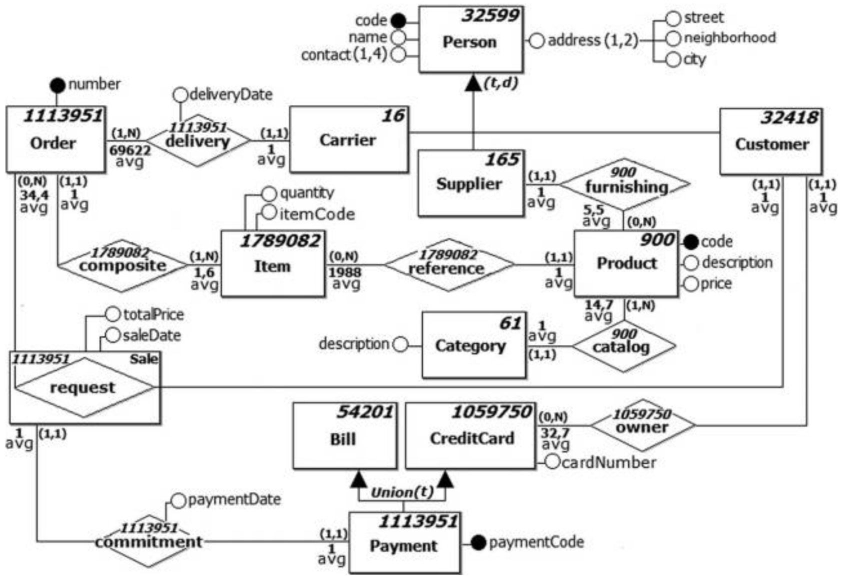

Figure 5.
The EER schema for an e-commerce application

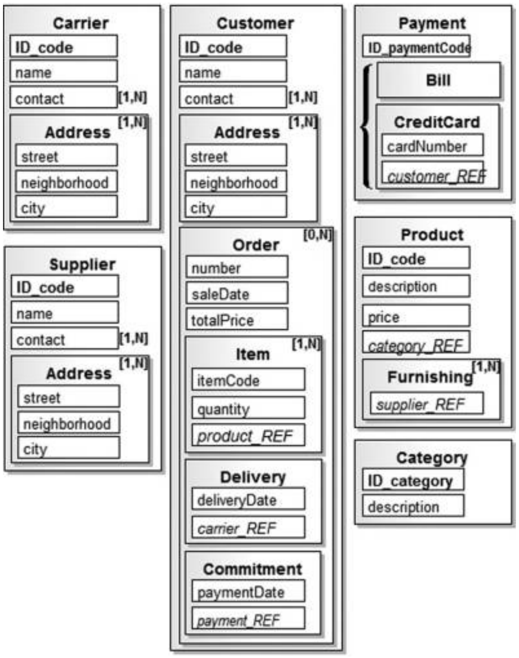

Figure 6.
The *conventional* NoSQL document logical schema

6 and 6, respectively. Note that all information of the considered EER schema was represented at the produced NoSQL document logical schema, called *conventional* schema (Figure 6).

To generate the NoSQL document logical schema that considers workload information, called *optimized* schema, we applied the workload modeling methodology defined in Batini *et al.* (1992), Schroeder and Mello (2008) and Schroeder *et al.* (2011). The EER schema, the volume of data, a set of operations and their average frequencies were given as input for the conversion process. The volume of data was included in the conceptual schema and is also shown in Figure 5. The number of instances from the original e-commerce application data set was used to measure the volume of data in our case study, i.e. the average number of instances of the entities and relationships, as well as the average cardinality of the entities in each relationship type.

We also obtained the main operations that comprise the application workload. They are shown in Table II and were provided by an expert user application considering the concepts (entity and relationship types) defined in the conceptual schema. Operation O3, for example, has an average frequency (*f(O)*) of 450 times a day. The entity and relationship types *Customer*, *request*, *Order*, *delivery* and *Carrier* are accessed, in this sequence, by O3. The fourth column of Table II presents the operation load in terms of access frequency for each concept of the conceptual schema.

The GAF of each concept involved in the operations on the conceptual schema was also measured. They are shown in Table III. To obtain the MAF measure, we considered that the set of operations produces 80 per cent of the load. Thus, the total data volume generated on the conceptual schema (1,340,990 daily accesses) represents 80 per cent of the total of accesses, which can be performed by the application over the conceptual types. We assume that 0.8 per cent is also given by an expert user application and denotes the MAF in percentage value. Such a percentage is applied over the total volume and we obtain 13,410 accesses as the MAF value.

Based on these input information, for the generation of the *optimized* schema (Figure 7), *Rule 2* was applied to the Person's generalization hierarchy and *Rule 1* was applied to the Payment's category hierarchy. The function *convertRelationships* ordered the relationship types so that relationships with the highest GAF appear first. On considering Table III, the list obtained is R = {*composite, reference, request, delivery, commitment, catalog, furnishing, owner*}, and the applied conversion rules were 5, 6, 5, 6, 4, 6, 6 and 6, respectively.

It is important to notice that the generation of the *conventional* and *optimized* schemas have the same goal, which is to generate compact and redundancy-free schemas and define appropriate representations in the NoSQL document logical model. The main difference between the conversion processes is that the *optimized schema* is generated based on the consideration of workload information to select appropriate conversion rules.

4.2 Experiment settings

The produced logical schemas were evaluated in terms of the performance of the operations over compliant NoSQL documents. The performance of the operations was measured over the access volume generated by each operation over each logical schema using the same workload methodology applied to measure the access volume of the

			Access frequency			
O	*f(O)*	Concept	Conceptual schema	Conventional logical schema	Optimized logical schema	Real application logical schema
O1	1,500	Order	1,500	1,500	1,500	1,500
		request	51,600	–	–	–
		Customer	51,600	51,600	51,600	1,670,926,500
		composite	82,560	–	–	–
		Item	82,560	82,560	82,560	2,673,482,400
		reference	82,560	–	–	–
		Product	82,560	74,304,000	74,304,000	2,406,134,160,000
		Subtotal:	434,940	74,439,660	74,439,660	2,410,478,570,400
O2	900	Order	900	900	900	900
		request	900	–	–	–
		Customer	900	900	900	29,176,200
		commitment	900	900	–	–
		Payment	900	1,002,555,900	900	1,002,555,900
		Subtotal:	4,500	1,002,558,600	2,700	1,031,733,000
O3	450	Customer	450	450	450	450
		request	15,480	–	–	–
		Order	15,480	15,480	15,480	501,277,950
		delivery	15,480	15,480	15,480	–
		Carrier	15,480	247,680	247,680	8,020,447,200
		Subtotal:	62,370	279,090	279,090	8,521,725,600
O4	300	Customer	300	300	300	300
		request	10,320	–	–	–
		Order	10,320	10,320	10,320	334,185,300
		commitment	10,320	10,320	–	–
		Payment	10,320	11,495,974,320	10,320	372,266,049,120,300
		Subtotal:	41,580	11,495,995,260	20,940	372,266,383,305,900
O5	100	Product	100	100	100	100
		reference	198,800	–	–	–
		Item	198,800	178,908,200	178,908,200	178,908,200
		composite	198,800	–	–	–
		Order	198,800	178,908,200	178,908,200	178,908,200
		Subtotal:	795,300	357,816,500	357,816,500	357,816,500
O6	100	Supplier	100	100	100	100
		furnishing	550	90,000	90,000	–
		Product	550	90,000	90,000	90,000
		catalog	550	–	90,000	–
		Category	550	5,490,000	5,490,000	5,490,000
		Subtotal:	2,300	5,670,100	5,760,100	5,580,100
Total			1,340,990	12,936,759,210	438,318,990	374,686,778,731,500

Table II.
Operations on schemas

operations over the conceptual schema. Proposed queries were executed over JSON documents stored in the NoSQL document-oriented DB MongoDB[2] to show the access volume provided by the operations on each schema.

For evaluating the performance of the schemas produced by our conversion approach (*conventional* and *optimized*), we also consider a logical schema artificially

Concept	GAF	Concept	GAF
Item	281,360	Composite	281,360
Order	277,100	Reference	281,360
Product	83,210	Request	78,300
Carrier	15,480	Delivery	15,480
Payment	11,220	Commitment	11,220
Customer	3,150	Catalog	550
Category	550	Furnishing	550
Supplier	100	Owner	0
CreditCard	0		
Bill	0		

Table III. GAF of concepts of Figure 5

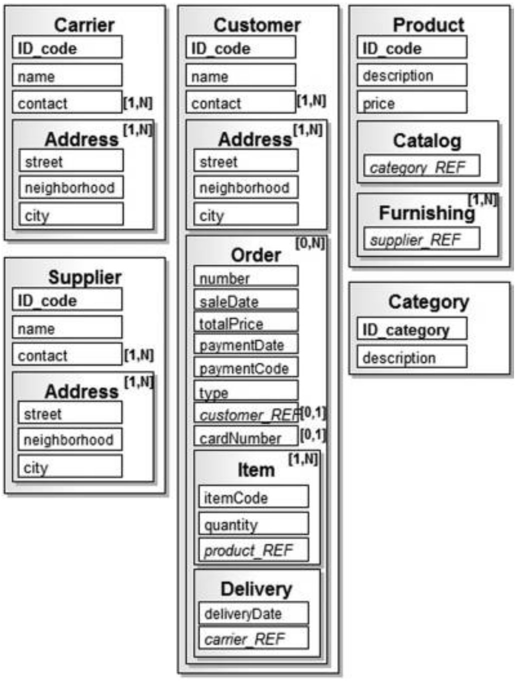

Figure 7. The *optimized* NoSQL document logical schema

produced by industry recommendations (Jboss Teiid, 2015; MongoDB, 2015). It was generated by the mapping of a relational physical schema to a NoSQL document schema in MongoDB. The physical schema comes from a real application data set compatible with the EER conceptual schema of Figure 5. For identification purposes, the produced logical schema was called *real application* logical schema and is shown at Figure 8.

It is relevant to notice that the accomplished conversion that produced the *real application* schema did not consider the EER conceptual schema of Figure 5. Instead, the logical schema was generated from the relational physical schema by applying mappings from industry recommendations, as stated before. This generated NoSQL

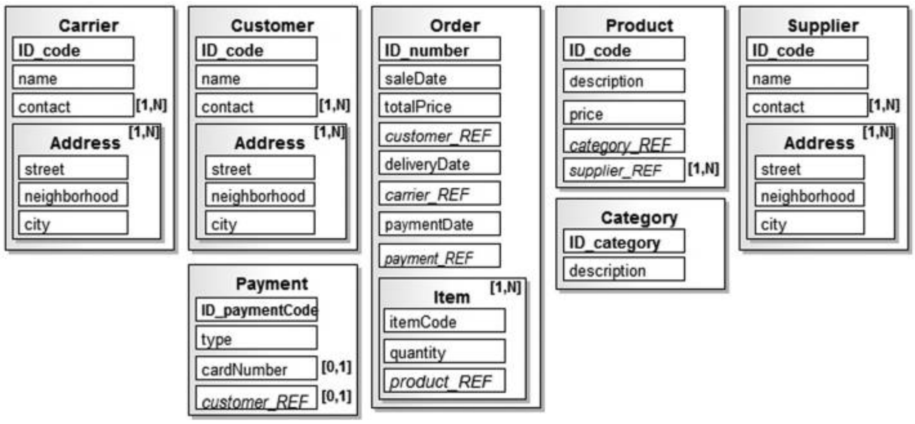

Figure 8.
The *real application* NoSQL document logical schema

logical schema was then represented in the NoSQL document logical model defined by Lima and Mello (2015). As the *conventional* and *optimized* schemas, the *real application* logical schema tries to minimize data redundancy through the generation of the more compact logical schema fragments whenever possible.

We also emphasize that the goal of the *real application* logical schema is to provide a comparison with the produced schemas by our proposed conversion process. It was necessary because related work to logical design of NoSQL (document) DBs presented in Section 2 (Bugiotti *et al.*, 2014; Chebotko *et al.*, 2015; Jovanovic and Benson, 2013) did not detail conversion processes between a conceptual and a logical schema, which makes difficult the development and implementation of their processes to compare with our approach. On the other hand, the industry recommendation mappings considered for the generation of the real application logical schema are available on the manufacturers' websites.

The access volume generated by the operations over the three presented logical schemas is measured considering the estimated access volume for the EER conceptual schema concepts (Figure 5) and the estimated frequency for each operation according to the application workload. The same methodology used to produce the data values in the fourth column of Table II is applied again, considering the execution of these operations on the defined structures of the three NoSQL document logical schemas.

The three last columns of Table II show the access volume generated for each schema and for each involved concept (entity or relationship) on the considered operations. For example, consider the NoSQL document logical schema of Figure 6 and the operation O4 in Table II. As the estimated execution of O4 is 300 times a day, the accesses amount over the *Customer* block is considered 300 for the conventional logical schema. As the *commitment* concept is represented as a nested block of *Order*'s block which, in turn, is a *Customer*'s nested block, it is necessary to execute a *structure join*, and we must multiply the number of daily access of the *Customer* block by the average cardinality associated with the *Order* and *commitment* concepts, resulting in 10,320 (300 × 34.4 × 1) accesses over the *commitment* block.

However, to obtain payments related with customers, it is necessary compare the *Payment*'s block identifier with the reference attribute *payment_REF* from all *commitment* blocks. It means that to obtain the payments associated with 300 customers accessed per day, it is necessary compare the value of 10,320 reference attributes *payment_REF* with 1,113,951 instances of *Payment*. Thus, this *value join* generates 11,495,974,320 (10,320 × 1,113,951) accesses per day over the *Payment*'s block of the conventional logical schema.

The same operation O4 generates 10,320 daily accesses over *Payment's* block on the optimized logical schema due to the fact that *Payment*' block is modeled as an *Order*'s nested block. In this case, as the average cardinality to *Order* on the *request* relationship is 34.4, to obtain payments related to customers it is necessary to execute *structure joins* that return, in average, 34.4 *Order's* nested block. On considering the 300 times per day, 10,320 accesses are recorded.

As stated before, to evaluate the effects of the query processing on these logical schemas, the operations were performed on compliant NoSQL documents generated and stored in the NoSQL document-oriented DB MongoDB. We develop a Java application to produce JSON documents defined by each collection for all schemas. For each schema, we generate a set of documents with the same volume of data presented in Figure 5.

The tests were carried out in a processor Core i7 2.40 GHz with 8 GB of memory, 1 TB of disk and Windows 8.1 Pro. All processed queries over MongoDB were executed on the same environment conditions using the default settings of *MongoDB Server 3.0.3*. The *MongoDB-shell* query specifications were defined according to the structure of the schemas and the sequence of accesses for the query operations, as presented in Table II. We use the trial version of *NoSQL Manager for MongoDB Professional* tool to execute the queries.

Table IV presents the response time in seconds for the execution of each one of the six operations on JSON documents conformed to the *conventional*, *optimized* and *real application* schemas. For each schema, it is shown the spent time in a single run[3] (*one* execution) of a query, as well as the daily spent system time (*accumulated* execution) to run a query considering its daily frequency. For example, for *conventional* schema, a single query *O1* shows the response time of 0.573 seconds. On considering the frequency of 1,500 times a day, the daily occupation system time to perform this accumulated operation is 859.500 seconds (0.573 × 1,500).

The next section discusses the experiment results.

4.3 Result analysis

Initially, we compare *Total Access Frequency* (TAF) of 1,340,990 estimated for EER conceptual schema (fourth column of Table II) with the values of the last three columns of Table II and verify that the TAF for NoSQL document logical schemas increases

		Conventional		*Optimized*		*Real application*	
O	*f(O)*	One	Accumulated	One	Accumulated	One	Accumulated
O1	1500	0,573	859,500	0,569	853,000	1,170	1.755,500
O2	900	1,018	915,900	0,750	675,300	0,510	459,300
O3	450	0,884	397,800	0,601	270,600	1,283	577,200
O4	300	0,776	232,800	0,143	42,900	1,189	356,600
O5	100	2,097	209,667	2,080	208,033	1,825	182,500
O6	100	0,471	47,100	0,524	52,400	0,518	51,800
Total		5,818	2.662,767	4,667	2.102,233	6,495	3.382,900

Table IV. Operation processing total time in seconds

substantially for all kinds of schema. This increase is due to the large number of comparisons caused by *value joins*. Value joins are necessary to retrieve conceptual relationships represented by reference relationships on NoSQL document logical schemas. These logical schemas, presented in Figures 6, 7 and 8, establish a total of 6, 5 and 7 reference relationships, respectively. The presented TAFs for each one of these schemas are different and the biggest difference occurs between the FAT of *optimized* schema (438,318,990 daily accesses) and the FAT of *real application* schema (374,686,778,731,500 daily accesses).

Although a small difference in the number of reference relationships exists, the nesting structures presented by these schemas produce an access volume differentiated for the considered operations. The *optimized* schema presents a nesting for the *Payment* block different than the *conventional* logical schema. It occurs because, in the *optimized* schema, *Rule 1* was applied to convert the union type involving the subclass *Payment*. This rule was considered because the GAF of the superclasses is lower than the assumed MAF. Besides, for the relationship *commitment*, the associative entity *Sale* was nested on the *Order* block content, as the GAF of the *Order* entity (277,100) is higher than the *Payment* entity (11,220). Due to this, the process that converts relationship types had chosen *Order* as a block to represent the relationship type, and *Rule 4* was processed. This nesting structure allows that only 900 accesses are necessary to achieve the instances of payments of 900 orders through the operation O2. For the same operation, *conventional* and *real application* schemas generate 1,002,555,900 accesses to retrieve payments. This elevated number of accesses is due to the reference relationship established between *Order* and *Payment* to represent the *commitment* relationship. Giving also the nesting of the *Payment* block, the operation O4 generates a smaller number of accesses in the *optimized* schema (20,940) with respect to other schemas (11,495,995,260 accesses for *conventional* schema and 372,266,383,305,900 accesses for *real application* schema).

In practice, the impact of these different logical structures is evaluated by measuring the query processing time on the three schemas at MongoDB NoSQL document DB. The results, as shown in Table IV, reveal that the *optimized* schema had spent times close to the ones spent for the *conventional* schema for some operations. However, the cost to perform O4 on *conventional* and *real application* schemas are notoriously higher because it is necessary to retrieve *Payment* instances in the *Order* block through *value joins* by a reference relationship. This result demonstrates the positive effect of avoiding the value joins. The daily total accumulated execution of operations performed on NoSQL documents demonstrates that the *optimized* schema produces a better response time. It raises the relevance of considering the workload information.

The queries response time presented in Table IV also confirms the difference in the processing time generated by queries executed over the *optimized* schema in comparison to the others logical schemas. Note that the sum of the single execution of all queries for *conventional* schema (5,818 seconds) has a value in seconds close to the sum of the single execution of all queries for *optimized* schema (4,667 seconds). As for the *real application* logical schema, the total value of single executions is higher than the other schemas, totalizing 6,495 seconds. The sum of accumulated executions also is superior for *real application* schema, totalizing 3.382,900 seconds. Note that the proportion between the total accesses of each operation presented in Table II and the accumulated execution of an operation presented in Table IV is not always equivalent. We argue that these differences may be caused by variations in the size of the considered documents and specific features of the used instances for each document in each query execution. These and other variations are not analyzed in this paper because the purpose of the experimental evaluation on MongoDB was to confirm the difference of the access volume generated by the three schemas according to Table II.

Although the considered operations in this case study do not represent a large volume of DB daily access, it could be seen by the performance comparison that the estimated DB workload information guides the conversion process to generate optimized NoSQL document schemas. The spent time of operations over the *optimized* schema is considerably less than the produced time by the other schemas. Note that the gain generated by the application of the optimized conversion is dependent of possible optimizations to be applied on the conceptual schema and provided workload information. The existence of several options for the nesting of concepts of the schema and the indicative that certain nestings have greater impact on operations provide information that allow to perform optimizations on a NoSQL document schema.

Finally, is important to observe that the conventional schema also provides a good logical design strategy for NoSQL document DBs, and it is an option to be adopted in cases where the estimate workload information is not available, often by the difficulty of an application in predict such a workload.

5. Conclusion

Indeed, NoSQL DBs are suitable solutions for *Big Data* management, specially in environments as the Web and the cloud. An associated data model allows the definition of better strategies for persistence and manipulation of data in the target DB. In this context, the aggregate-based logical representation, a related work tendency, provides support to scalability and consistency, as they are a natural unit for *sharding* and atomic manipulation of data in distributed environments.

This paper raises from this motivation and presents an approach for logical design of NoSQL document DB schemas based on a conceptual schema. We generate compact and redundancy-free schemas and define appropriate representations in the NoSQL document logical model. NoSQL document DBs are a suitable category for Web and cloud applications that provide dynamic queries execution, horizontal scalability and high availability. In our proposal, the estimated volume of data and workload information can be considered to generate optimized NoSQL document structures in terms of the main application operations and their frequency.

We evaluate our approach through a case study with an experimental evaluation in the e-commerce application domain. The results had shown that our workload-based conversion process improves query performance on NoSQL documents by reducing the number of DB accesses.

As future work, we intend to evaluate the application of our process over a larger volume of data, in a distributed environment, over another *Big Data* application domain, as well as to consider the NoSQL DB physical design, including the definition of indexes.

We also consider the comparison of our approach with a baseline to evaluate application performance for the logical schemas generated by each one of them. It depends on the availability of detailed conversion algorithms by related work.

Finally, we consider the possibility of extending the approach to reduce the number of reference relationships in a NoSQL document logical schema by analyzing the access type (query or update) of the estimated DB operations. This new approach would allow data redundancy in some parts of the logical schema if the involved concepts are very frequently accessed by query operations. .

Notes

1. A lightweight data-interchange format (json.org).
2. A document-oriented database (mongodb.org).
3. A single run refers to the average of five runs of each query operation on each schema.

References

Atzeni, P., Jensen, C.S., Osri, G., Ram, S., Tanca, L. and Torlone, R. (2013), "The relational model is dead, SQL is dead, and I don't feel so good myself", *SIGMOD Record*, Vol. 42 No. 2, pp. 64-68.

Batini, C., Ceri, S. and Navathe, S.B. (1992), *Conceptual Database Design: An Entity-Relationship Approach*, Benjamin/Cummings, San Francisco, CA.

Bugiotti, F., Cabibbo, L., Atzeni, P. and Torlone, R. (2014), "Database design for NoSQL systems", ER 2014, pp. 223-231.

Cattell, R. (2010), "Scalable SQL and NoSQL data stores", *SIGMOD Record*, Vol. 39 No. 4, pp. 12-27.

Chebotko, A., Kashlev, A. and Lu, S. (2015), "A big data modeling methodology for apache Cassandra", *International Congress On Big Data*, IEEE, pp. 238-245.

Elmasri, R. and Navathe, S.B. (2011), *Fundamentals of Database Systems*, Addison-Wesley, Boston, MA.

Evans, E. (2003), *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Addison-Wesley Professional, Boston, MA.

Jboss Teiid (2015), "MongoDB translator", available at: https://docs.jboss.org/author/display/teiid812final/MongoDB+Translator (accessed 3 September 2015).

Jovanovic, V. and Benson, S. (2013), "Aggregate data modeling style", *SAIS* 2013, pp. 70-75.

Kaur, K. and Rani, R. (2013), "Modeling and querying data in NoSQL databases", *International Conference on Big Data*, IEEE, Silicon Valley, CA, pp. 1-7.

Lima, C. and Mello, R.S. (2015), "A workload-driven logical design approach for NoSQL document databases", *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services (iiWAS '15)*, ACM, New York, NY.

McMurtry, D., Oakley, A., Sharp, J., Subramanian, M. and Zhang, H. (2013), "Data access for highly-scalable solutions: using SQL, NoSQL, and polyglot persistence", Microsoft, available at: www.microsoft.com/en-us/download/details.aspx?id=40327 (accessed 5 June 2014).

Mok, W.Y. and Embley, D.W. (2006), "Generating compact redundancy-free xml documents from conceptual-model hypergraphs", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 18 No. 8, pp. 1082-1096.

MongoDB (2015), "RDBMS to MongoDB migration guide", available at: www.mongodb.com/collateral/rdbms-mongodb-migration-guide (accessed 3 September 2015).

Sadalage, P.J. and Fowler, M.J. (2013), *NoSQL Distilled*, Addison-Wesley, Upper Saddle River, NJ.

Schroeder, R. and Mello, R.S. (2008), "Improving query performance on XML documents: a workload-driven design approach", *DocEng*, Vol. 2008, pp. 177-186.

Schroeder, R., Duarte, D. and Mello, R.S. (2011), "A workload-aware approach for optimizing the XML schema design trade-off", *iiWAS* 2011, ACM, New York, NY, pp. 12-19.

About the authors
Cláudio Lima is an IT Analyst at the Federal University of Santa Catarina (UFSC), Brazil. He received his BS degree in 2008 from the Federal University of Santa Catarina (UFSC) and MSc in 2016 from the Federal University of Santa Catarina (UFSC), both in Computer Science. He received a specialization degree in 2012 from the University of Southern Santa Catarina (UNISUL) in Software Projects Engineering. His main research interests are data modeling, Web data management and cloud databases. He is a member of UFSC Research Group on Databases and author of papers in national and international conferences. Cláudio Lima is the corresponding author and can be contacted at: claudio.lima@ufsc.br

Ronaldo Santos Mello is a PhD Professor at the Federal University of Santa Catarina (UFSC), Brazil. He held a post-doc grant as a Visiting Professor in the University of Utah, Salt Lake City, USA, in 2009. His research interests are concentrated in the database area, with a current focus on data integration, Web and cloud data management, non-conventional data modeling and data integrity constraints. Dr Mello is the head of UFSC Research Group on Databases, having publications in national and international conferences and journals. He also acts as an external reviewer of national and international conferences and journals.