Uniqueness Constraints for Object Stores

PHILIPP SKAVANTZOS, The University of Auckland
UWE LECK, The University of Flensburg
KAIQI ZHAO and SEBASTIAN LINK, The University of Auckland

Object stores offer an increasingly popular choice for data management and analytics. As with every data model, managing the integrity of objects is fundamental for data quality but also important for the efficiency of update and query operations. In response to shortcomings of unique and existence constraints in object stores, we propose a new principled class of constraints that separates uniqueness from existence dimensions of data quality, and fully supports multiple labels and composite properties. We illustrate benefits of the constraints on real-world examples of property graphs where node integrity is enforced for better update and query performance. The benefits are quantified experimentally in terms of perfectly scaling the access to data through indices that result from the constraints. We establish axiomatic and algorithmic characterizations for the underlying implication problem. In addition, we fully characterize which non-redundant families of constraints attain maximum cardinality for any given finite sets of labels and properties. We exemplify further use cases of the constraints: elicitation of business rules, identification of data quality problems, and design for data quality. Finally, we propose extensions to managing the integrity of objects in object stores such as graph databases.

CCS Concepts: • **Information systems** → **Integrity checking**; *Inconsistent data*; *Graph-based database models*; *Database performance evaluation;*

Additional Key Words and Phrases: Completeness, computational problem, data management, data quality, data semantics, existence, graph database, uniqueness, object store

ACM Reference format:
Philipp Skavantzos, Uwe Leck, Kaiqi Zhao, and Sebastian Link. 2023. Uniqueness Constraints for Object Stores. *ACM J. Data Inform. Quality* 15, 2, Article 13 (June 2023), 29 pages.
https://doi.org/10.1145/3581758

1 INTRODUCTION

Uniqueness constraints (**UCs**) enable the efficient and flexible handling of object integrity in database management systems [10]. Given a collection of objects, a UC is a set of properties whose values uniquely identify any object in the collection. Keys form a special class of UCs where no object in the data store is permitted to have missing values on any property of the key. In contrast,

We would like to thank the anonymous reviewers for several suggestions that have improved the positioning of our contributions, including the use of object stores as context for our definition of completeness-tailored uniqueness constraints.
Authors' addresses: P. Skavantzos, K. Zhao, and S. Link, School of Computer Science, 38 Princes Street, Auckland, Auckland, 1010, New Zealand; emails: {philipp.skavantzos, kaiqi.zhao, s.link}@auckland.ac.nz; U. Leck, The University of Flensburg, Flensburg, Schleswig-Holstein, 24943, Germany; email: Uwe.Leck@uni-flensburg.de.

1936-1955/2023/06-ART13 $15.00
https://doi.org/10.1145/3581758

UCs only require unique combinations of values for those objects that have no missing property on the UC. Both keys and UCs are essential for understanding the structure and semantics of data. As such they are fundamental to the entire information systems cycle, from requirements engineering and conceptual modeling to logical and physical design. Indeed, uniqueness constraints enable (i) analysts to identify objects such as entities and relationships, (ii) designers to transform schemata that exhibit data redundancy to those that minimize them, and (iii) users to access information effectively and efficiently. Knowledge about UCs enables us to (i) uniquely reference objects across data repositories, (ii) minimize data redundancy at schema design time to process updates efficiently at run time, (iii) provide better selectivity estimates in cost-based query optimization, (iv) provide a query optimizer with new access paths that can lead to substantial speedups in query processing, (v) allow the database administrator to improve the efficiency of data access via physical design techniques such as data partitioning or the creation of indexes and materialized views, and (vi) provide new insights into application data.

Due to their importance, keys and UCs have been studied for most data models, including graphs. For instance, a notion for keys proposed in academia [19] is very expressive to serve its target application of entity resolution. The notion subsumes keys from XML and conditional constraints [19]. This expressiveness has its price, for example, implication is *NP*-complete, satisfiability and validation are both *coNP*-complete to decide [19]. Indeed, the investigation of graph dependencies has been stipulated as an important and challenging area of data quality research [18]. Among a recent surge of graph databases, Neo4j is the most popular one.[1] It employs an expressive property graph model. Objects such as vertices and edges may have properties, which are pairs of an attribute and a value, reflecting the NoSQL nature of graph databases. A recent community effort has brought forward an expressive proposal for keys on property graphs, applicable to vertices, edges, and properties [2]. Around the same time, the concept of *completeness-tailored uniqueness constraints* was proposed for property graphs [42], following an earlier proposal of Neo4j keys in [32]. Noticeably, the keys of [32] and the *mandatory PG-keys* of [2] combine both existence and uniqueness constraints, while the uniqueness constraints of [42] and *exclusive PG-keys* of [2] separate them. The separation between existence and uniqueness constraints had already been made explicit in the context of SQL [46–48].

In this article, we extend our proposal of completeness-tailored uniqueness constraints from graph data models [42] to general object stores. Our proposal forms a special case of exclusive PG-keys [2], can express important requirements on the integrity of objects, and can be reasoned about efficiently. As the existence and uniqueness of properties are often not both achievable for all objects, UCs offer a more flexible mechanism to manage the integrity of objects than keys do. For example, while Neo4j does support UCs, their use is restricted to single labels and single properties. In response, our notion of UCs can take advantage of multiple labels and properties. In contrast to Neo4j UCs, our notion of UCs can separate existence from uniqueness requirements. This allows us to minimize the subset of properties that can uniquely identify all objects for which a set of properties exists. In addition to these extensions over keys, we establish that our notion of uniqueness constraints also enjoys desirable computational characteristics.

An overview of the computational features for our uniqueness constraints is illustrated in Figure 1. The central computational problems we address are those of deciding validation and implication. In contrast to previous proposals for keys on property graphs, we target a class that is highly efficient. On the one hand, such class enables systems to minimize update overheads and the maintenance of business rules (namely, uniqueness constraints that model requirements of the underlying domain). On the other hand, such a classes also help systems detect inconsistencies

[1] https://db-engines.com/en/ranking_trend/graph+dbms.

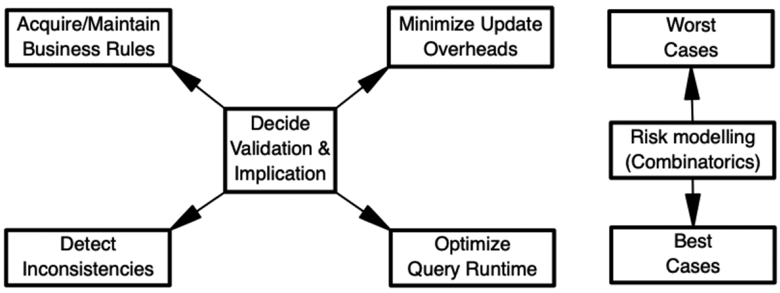

Acquire/Maintain Business Rules
Minimize Update Overheads
Decide Validation & Implication
Detect Inconsistencies
Optimize Query Runtime
Worst Cases
Risk modelling (Combinatorics)
Best Cases

Fig. 1. Overview of computational features.

in the data (as violations of the constraints) and optimize the runtime of queries. In particular, indices arising from the specification of uniqueness constraints help with these tasks and achieve scalability for validation. For the purpose of gaining a thorough understanding and modeling extreme cases, one can apply techniques from combinatorics to determine the maximum possible number of non-redundant uniqueness constraints that is attainable. Indeed, such information illustrates worst-case scenarios in terms of maintaining these constraints and processing updates but also best-case scenarios in terms of detecting inconsistencies and optimizing queries. Overall, we make the following contributions:

(1) For the flexible management of object integrity in object stores, we introduce the class of **completeness-tailored uniqueness constraints** (**ctUCs**). These form a special case of exclusive PG-keys but also include various previous notions from property graphs as special cases. In particular, the formal semantics of ctUCs allows us to separate existence and uniqueness concerns for the efficient management of integrity in object stores.
(2) We provide real-world examples and use cases that illustrate the benefit of ctUCs for indexing, updating, and optimizing queries.
(3) We include experiments that quantify the impact of indices for ctUCs on both query and update operations in object stores, such as graph database systems. In particular, we demonstrate on the example of Neo4j property graphs that the validation of ctUCs achieves perfect scalability under updates by requiring a number of accesses to the ctUC-index that remains constant while the instance size grows.
(4) We characterize the implication problem of ctUCs both axiomatically and algorithmically. Our characterization makes it simple to obtain a set of ctUCs that causes a minimal overhead for managing object integrity.
(5) We characterize which non-redundant families of ctUCs attain maximum cardinality for any given finite sets of labels and properties. These results provide insight into the opportunities available for query and update optimization, based on ctUCs.
(6) We outline applications of ctUCs in data engineering, such as the elicitation of business rules, the identification of data quality problems, and the design for data quality.
(7) Finally, we give four recommendations for the future use of uniqueness constraints within object stores.

In what follows, we motivate our work with an application scenario in Section 2. Section 3 includes a concise review of relevant work. The formal semantics for ctUCs over object stores, their illustration on real-world property graphs, and their validation on property graphs with queries are given in Section 4. Use cases of ctUCs for efficient updating and querying are discussed in Section 5. Experiments on real-world data quantify the benefit of ctUCs for these use cases in Section 6. In

Section 7, we establish axiomatic and algorithmic characterizations of the implication problem. We characterize non-redundant families of ctUCs that attain maximum cardinality for any given finite set of labels and properties in Section 8. Applications of ctUCs to data engineering are illustrated in Section 9. Recommendations for the definition and use of uniqueness constraints by object stores are given in Section 10, before we conclude and briefly comment on future work in Section 11.

2 GENERAL IDEA AND APPLICATION SCENARIO

Before giving a concrete application scenario, we start with some general comments highlighting the intricate interplay between existence and uniqueness requirements.

Constraints provide declarative means for enforcing data quality requirements of an application domain. Traditionally, keys and uniqueness constraints enforce requirements on the uniqueness of data. Already in relational databases, however, uniqueness requirements are intrinsically linked to existence requirements. In fact, this is the reason why keys and uniqueness constraints co-exist and enforce different requirements. While keys require objects to be complete and unique on the properties stipulated (such as primary keys on SQL tables), uniqueness constraints need only be unique on those objects that are complete on the properties stipulated (such as SQL `UNIQUE` constraints).

More specifically for object stores, a uniqueness constraint $L : P$ may require that there are no two different objects which carry a given set L of labels and a given set P of properties and have matching values on all the properties in P. In other words, every L-object that exhibits all properties in P is uniquely identified by the values on all the properties in P. In this sense, the utility of P is overloaded by not only requiring existence but also uniqueness on all properties in P. More flexibly, one may summarize the existence requirements of an application for those objects that carry all the labels in L by the property set P. This means that the application only considers those L-objects for which all the properties in P are defined (that is, every property in P has been assigned a value). In addition, one may ask what uniqueness requirements the application has for data on its target objects. Just like the SQL `UNIQUE`(U) constraint, a uniqueness constraint on U shall only be invalidated whenever there are two different objects with matching values on all the properties in U. This follows the principle that constraints should not be invalidated by missing data, in this case some missing property. With this principle in mind, we have arrived at an expression $L : PU : U$ which denies the existence of any two different objects that carry all labels in L, all properties in P and U, and have matching values on all properties in U. Without loss of generality, the expression $L : PU : U$ can be simplified to $L : P : U$ by requiring that $U \subseteq P$. Hence, we call these UCs *completeness-tailored* (ctUCs), and write $L : P : U$ to say that there are no two different objects that carry all labels in L, all properties in P, and have matching values on all properties in $U \subseteq P$. We argue that ctUCs provide a flexible mechanism to combine existence and uniqueness constraints for object stores. The article will argue that the definition of ctUCs also enjoys computational properties that allow object stores to manage the integrity of objects effectively and efficiently.

Next, we provide a concrete application scenario that illustrates our general ideas. As running example, we use the *Open Movie Database*.[2] This dataset is referenced by Neo4j in its user guide. It contains 28,863 nodes and 166,261 edges. We denote the dataset by G_0. While the dataset represents a property graph, we are not so much interested in its graph structure but more from the view points of an object store. Here, objects may be nodes and edges, but the use of specific labels usually focuses the attention on either nodes or edges. Hence, edge relationships between two vertices, as illustrated in Figure 2, are not in our focus. Instead, we will focus in our running example on

[2]http://www.omdbapi.com.

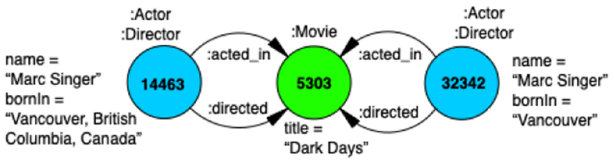


Fig. 2. Property graph with labeled vertices and edges.

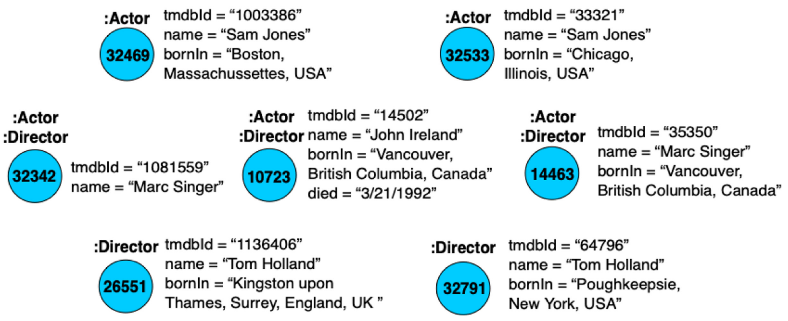


Fig. 3. Real-world vertices, labels, and property-value pairs in *Open Movie Database*.

specific objects, namely nodes that carry labels such as *Actor* or *Director*. In G_0, such nodes have properties like *identity*, *url*, *imdbID*, *name*, *born*, *bornIn*, *died*, *and poster*. Figure 3 shows a few vertices of G_0. The left-most vertex has node identifier *32,342*, carries labels *Actor* and *Director*, and properties *tmdbID="1,081,559"* and *name="Marc Singer"*.

Consider an application that targets some data analysis for actors and directors based on the countries they were born in. Naturally, the application has existence requirements for the properties *name* and *bornIn*. For the purpose of this work, we are interested how nodes about these individuals, that also meet these existence requirements, may be identified uniquely. Starting off, none of the three keys

{Actor}:{name,bornIn}, *{Director}:{name,bornIn}*, *{Actor,Director}:{name,bornIn}*

hold since there are nodes that carry both labels *Actor* and *Director* and that miss property *bornIn*. However, each of the three uniqueness constraints *{Actor}:{name,bornIn}*, *{Director}:{name,bornIn}*, *{Actor,Director}:{name,bornIn}* are satisfied. Considering uniqueness constraints only, we can argue that *{Actor}:{name,bornIn}* and *{Director}:{name, bornIn}* are both minimal. In fact, none of the UCs *{Actor}:{name}*, *{Actor}:{bornIn}*, *{Director}:{name}* nor *{Director}:{bornIn}* hold. In addition, since every individual that is an actor and director can be identified by *{name,bornIn}*, the uniqueness constraint *{Actor,Director}:{name,bornIn}* is already implied by *{Actor}:{name,bornIn}* (and also implied by *{Director}:{name,bornIn}*). Furthermore, neither *{Actor,Director}:{name}* nor *{Actor,Director}:{bornIn}* hold. In fact, there are different people that are both actors and directors (with ids *32,342* and *14,463*), but still, share the same name (*Marc Singer*). Interestingly, Neo4j only supports UCs with singleton labels and singleton properties, so several of the UCs above cannot be expressed by Neo4j.

Once we widen our scope to include ctUCs, things change and a more detailed analysis becomes possible. For actor or director nodes we may consider ctUCs

— *{Actor}*:*{name,bornIn}*:*{name}*
— *{Actor}*:*{name,bornIn}*:*{bornIn}*
— *{Director}*:*{name,bornIn}*:*{name}* or
— *{Director}*:*{name,bornIn}*:*{bornIn}*,

none of which is satisfied by G_0. In particular, *{Actor}*:*{name,bornIn}*:*{name}* is violated due to the two actors with name *Sam Jones*, and *{Director}*:*{name,bornIn}*:*{name}* is violated due to the two directors with name *Tom Holland*. However, for individuals that are both actors and directors, the multi-label ctUC *{Actor,Director}*:*{name,bornIn}*:*{name}* holds. Hence, under the given existence requirements, individuals that are both actors and director can be identified uniquely by their name.

When considering ctUCs, the following can all be regarded as minimal:

- *{Actor}*:*{name,bornIn}*:*{name, bornIn}*
- *{Director}*:*{name,bornIn}*:*{name, bornIn}* and
- *{Actor,Director}*:*{name, bornIn}*:*{name}*.

The scenario illustrates not only the power of ctUCs in expressing existence and uniqueness requirements, but also how intricate the constraints may interact on object stores with multiple labels and composite properties. This motivates our investigation of ctUCs that follows.

3 LITERATURE REVIEW

We will discuss related work on keys and uniqueness constraints from relational databases, other data models including those with missing data, and previous work on constraints for graphs. We also discuss extensions of our work over the conference version.

3.1 Relational Model

Keys are fundamental for all data models, being the primary mechanism to enforce Codd's principle of entity integrity. In relational databases, the concepts of keys and candidate keys are well-established. Essentially, a key is a set of attribute names whose combinations of values uniquely identify every tuple in every relation over the schema. A key is called a candidate key when it is minimal in the sense that none of its proper subsets also forms a key. Sometimes, the literature refers to keys as superkeys and to candidate keys as either keys or minimal keys. Candidate keys give rise to indices that speed up access to data for any operations on the data, but also help with referential integrity since foreign keys also reference candidate keys. Various computational problems arise for keys, such as the computation of all candidate keys implied by a set of constraints [35], the computation of all candidate keys that hold on a given relation [20, 25] (discovery problem), the computation of a perfect sample relation (Armstrong relation) that exhibits precisely those keys implied by a given set of keys [5, 13, 14, 20, 27], or the extremal problem to determine all non-redundant families of candidate keys that attain maximum cardinality given a schema with an arbitrary number of attributes [12]. The fundamental goal of database normalization is to organize data in relations such that updates incur minimal overheads to keep data consistent. Since this becomes possible when data redundancy is avoided, complex data dependencies that cause data redundancy are transformed into key constraints that prohibit data redundancy, in a way that preserves the original data dependencies. Essentially, Fagin's Domain Key Normal Form [17] captures this idea: a database schema is well-designed when all constraints can be expressed as keys (and domain constraints), generalizing proposals such as Boyce–Codd [11, 24], Fourth [16], and Fifth Normal Form [45]. Similarly, keys and their generalization to cardinality constraints play an important role in normal forms that tolerate data redundancy, such as Third Normal Form [6] or Bounded Cardinality Normal Form [33]. All of these problems are not limited to database relations, but apply more generally to all data models.

3.2 Keys for Data with Missing Values

While the concept of keys over relational databases with no missing data is unchallenged, various complementary classes of keys and uniqueness constraints exist when some data is missing [31].

Candidate keys require that values on every key attribute exist for every record and that the combination of values on the key attributes is unique for every record. In contrast, UCs only require a unique combination of values for those records for which these values exist for every attribute of the UC. Hence, a set of attributes P is a candidate key when the SQL constraint `UNIQUE`(P) holds and every attribute in P is defined to be `NOT NULL`. Recently, embedded UCs (eUCs) $P : U$ have been proposed for relational databases with missing data [46]. Indeed, eUCs decouple existence requirements, denoted by P, from uniqueness requirements, denoted by U. That is, the eUC $P : U$ can only be invalidated when there are two records with no missing data on every attribute in PU, and matching values on all the attributes in U. Hence, without loss of generality, we have an expression $P : U$ where $U \subseteq P$. Indeed, eUCs $P : U$ capture SQL `UNIQUE` as the special case where $P = U$. The benefit of eUCs over SQL `UNIQUE` in terms of physical data access for more efficient entity and referential integrity management, updating and querying has been demonstrated [47]. In addition, the concept of embedded UCs has been generalized to **embedded functional dependencies** (**eFDs**) and a schema design approach has been developed that can tailor relational schema designs such as Boyce–Codd and Third Normal Form to completeness requirements for a set P of attributes, as driven by an application. Therefore, stipulating which subset $U \subseteq P$ of properties actually already ensure uniqueness on P-complete records, enables the tailoring of UCs and FDs to different completeness requirements [46, 49].

In key sets, different elements of the set can separate different pairs of records in a data set [22, 43]. A key set is satisfied by a given relation, if for every pair of different records in the relation there is some element in the key set on which the two records have no missing value and have non-matching values on some attribute of the element. When every missing value in an incomplete relation is replaced by an actual domain value (or a special marker that no value exists), we obtain a possible world of the incomplete relation. Possible keys of an incomplete relation are keys that hold in some of its possible worlds, while certain keys are keys that hold in all of its possible worlds [29]. Recently, the intermediary concept of strongly possible keys [1] has been introduced that limits possible worlds to those resulting from replacements of missing data by values that already occur in the incomplete relation.

3.3 Other Data Models and Object Stores

Keys and uniqueness constraints have also been studied in various different data models, including Web models such as XML [23], JSON [37] and RDF [30], models where data is nested using records, lists, sets, or union operators [38, 41], description logics [44], temporal models [26, 50], object-relational [28] and object-oriented data models [7], as well as models for uncertain data such as probabilistic databases [9] or possibilistic databases [3].

All of these data models introduce special structures to what are more generally known as object stores. The latter simply assumes the existence of some globally unique identifier and then assign a value to properties that the object is known to have. Other data models then simply add further requirements to the structure. However, the intention for this article is to investigate uniqueness constraints on object stores which only rely on identifiers and property-value pairs. This simple structure is essentially the First Order Normal Form for Relational Databases, as advocated for Litwin et al. [34].

3.4 Graph Models

An important popular class of object stores are graph databases, which also provide limited support for object integrity, in particular nodes. Similar to candidate keys, Neo4j keys require both existence and uniqueness for all properties of the key [32]. As a consequence, reasoning about these keys is not as intuitive as one may expect. For example, the underlying implication problem re-

Table 1. Specifying the ctUC $\{L_1, \ldots, L_m\} : \{P_1, \ldots, P_n\} : \{U_1, \ldots, U_k\}$ as an Exclusive PG-key

FOR $x : L_1 \ldots : L_m$	WHERE $x.P_1$ IS NOT NULL AND ... AND $x.P_n$ IS NOT NULL
EXCLUSIVE	$x.U_1, \ldots, x.U_k$.

quires a binary rather than a unary axiomatization [32]. Hence, it is not clear what the concepts of superkeys or minimal keys mean. In [19] the authors propose a class of keys for graphs to perform entity matching. The associated implication problem is *NP*-complete, and those of satisfiability and validation are *coNP*-complete [19]. In [30] different ways are discussed for mapping relational databases into an RDF graph, with an emphasis on how to represent the original key and foreign key constraints in the resulting RDF graph. Finally, in [40] the authors put forward some proposals for extending the capabilities of Neo4j in specifying integrity constraints. The authors provide a simple prototype implementation and experiments. More generally, the investigation of graph dependencies has been stipulated as an important and challenging area of data quality research [18].

Recently, a community effort has resulted in a proposal for keys on property graphs [2]. The proposal is comprehensive as it sets out an expressive framework for specifying keys on nodes, edges, and properties. For example, the ctUC $\{L_1, \ldots, L_m\} : \{P_1, \ldots, P_n\} : \{U_1, \ldots, U_k\}$ can be specified as the exclusive PG-key in Table 1. While the framework provides expressiveness and flexibility in specifying keys, it does not contain any technical results yet, such as the ability to reason about keys efficiently or use them for query/update optimization. Based on our results and applications, our proposal of ctUCs may therefore be seen as a sub-class of general keys on property graphs that enjoys good computational properties. We also note that simplicity is another important feature of constraints, since practitioners may not utilize proposals that are complicated.

Throughout the article, we will assume that missing data is synonymous with missing properties. That is, we do not distinguish between the two cases where a property is missing, and a property exists but the null marker has been assigned to it. The former case may be regarded as the classical null marker interpretation that a property is not applicable to an object, while the latter conforms to the interpretation that a property value exists but is currently unknown. SQL only permits one uniform interpretation of all null marker occurrences, which is *no information*. Permitting additional interpretations retains more information but also leads to higher complexity for database operations. If any object store ever permitted the assignment of a null marker to a property, our proposal of completeness-tailored uniqueness constraints would remain robust since we would still continue to require matches of values to be matches of non-null values only. This is very much in line with the recommendation of the PG-keys in handling null markers [2]. In contrast, for most classes of keys we would regard them as invalid whenever missing properties or null marker assignments to properties of the key would occur. The reason is that keys are meant to identify every object uniquely. That being said, we regard it as future work to investigate the concepts of certain keys and key sets for object stores, which do permit null marker occurrences on properties or missing properties, as long as every object can still be identified uniquely [22, 29, 43].

UCs provide a more flexible mechanism for managing object integrity than keys do. However, their current use is limited to single labels and single properties of nodes in Neo4j, while composite indices on single labels need to be specified manually. We, therefore, propose multi-label completeness-tailored uniqueness constraints $L : P : U$, which subsume Neo4j's UCs as the special case where both $L = \{\ell\}$ and $P = \{p\} = U$ are singleton labels and properties, respectively, and Neo4j's composite indices as the special case where $L = \{\ell\}$ is a singleton label and $P = U$. As a running example, the multi-label ctUC *{Actor,Director}*:*{name,bornIn}*:*{name}* is satisfied by the *Open Movie* property graph G_0, frequently used as a showcase by Neo4j. The graph does not satisfy the single-label ctUCs *{Actor}*:*{name,bornIn}*:*{name}* nor *{Director}*:*{name,bornIn}*:*{name}*, as illustrated by G_0. We will formalize our ideas for multi-label ctUCs and illustrate their benefits for the effective and efficient management of object integrity in object stores, including property graphs and Neo4j.

3.5 Extension Over Conference Article

The current article extends the conference version [42] in multiple directions. Most importantly, the current article treats completeness-tailored uniqueness constraints within the context of object stores, generalizing the setting from its special case of graph databases considered in [42]. As one consequence, our class of ctUCs now handles arbitrary objects, such as combinations of nodes and vertices from property graphs. In the conference version, objects were limited to nodes of property graphs [42]. Apart from including all proof details in the current article, motivation, and literature review have been extended, and the conference version did not include any experiments (Section 6) nor combinatorial results (Section 8).

4 COMPLETENESS-TAILORED UNIQUENESS CONSTRAINTS

We recall basics of object stores and property graphs to prepare our formal definition of completeness-tailored uniqueness constraints. Subsequently, we illustrate the new concept with examples from a well-known real-world property graph. Finally, we comment on the validation problem of completeness-tailored uniqueness constraints.

4.1 Object Stores

We provide a basic definition of an object store and illustrate graph databases as a primary showcase of object stores.

We assume that the following sets are pairwise disjoint: $\mathcal{O}$ denotes a set of objects, $\mathcal{P}$ denotes a set of properties, $\mathcal{V}$ denotes a set of values. As a special type of property, we also allow labels, identified by the word `label` $\in \mathcal{P}$, which serve as a means to classify objects into different categories by putting a label on them. Hence, we single out the subset $\mathcal{L} \subseteq \mathcal{V}$ as label values that can be assigned to the special property `label` $\in \mathcal{P}$ and to no other property in $\mathcal{P} - \{$`label`$\}$. For the remainder, we will always assume that `label` $\in \mathcal{P}$, and that $\mathcal{L} \subseteq \mathcal{V}$ denotes the set of label values.

Definition 1. We define an *object store* as the triple $\mathfrak{O} = (\mathcal{O}, \mathcal{P}, \mathcal{V})$. An *instance* $\mathfrak{o}$ of object store $\mathfrak{O} = (\mathcal{O}, \mathcal{P}, \mathcal{V})$ is defined as a partial function $\mathfrak{o} : \mathcal{O} \times \mathcal{P} \nrightarrow \mathcal{V}$ that assigns a value to each of finitely many pairs of objects and their properties.

Since we are using property graphs [8] as a running example, we highlight how they appear as special cases of object stores and their instances. Indeed, the objects of a property graph simply exhibit a graph structure composed of vertices and edges between them.

A *property graph* is a quintuple $G = (V, E, \eta, \lambda, \nu)$ where $V \subseteq \mathcal{O}$ is a finite set of objects, called *vertices*, $E \subseteq \mathcal{O}$ is a finite set of objects, called *edges*, $\eta : E \rightarrow V \times V$ is a function assigning to each edge an ordered pair of vertices, $\lambda : V \cup E \rightarrow 2^{\mathcal{L}}$ is a function assigning to each object a finite set of labels, and $\nu : (V \cup E) \times \mathcal{P} \nrightarrow \mathcal{V}$ is a partial function assigning values for properties to objects, such that the set of domain values where ν is defined is finite. Figure 2 shows a property graph with three vertices and four edges (the vertex identifiers are shown inside the vertices), vertex and edge labels with the symbol ":" as prefix, and property-value pairs attached to the vertices.

4.2 Introducing Completeness-tailored Uniqueness Constraints

We now formally define the syntax and semantics of completeness-tailored uniqueness constraints. These will cover the uniqueness constraints, as used by Neo4j, as a special case.

Definition 2. A ***completeness-tailored uniqueness constraint (ctUC)*** over object store $\mathfrak{O} = (\mathcal{O}, \mathcal{P}, \mathcal{V})$ is an expression $L : P : U$ where $L = \{L_1, \dots, L_m\} \subseteq \mathcal{L}$, $P = \{P_1, \dots, P_n\} \subseteq \mathcal{P}$ and $U = \{P_{i_1}, \dots, P_{i_k}\} \subseteq P$. The ctUC $L : P : U$ over object store $\mathfrak{O}$ is satisfied by the instance $\mathfrak{o}$ over $\mathfrak{O}$ if and only if

$$\begin{gathered}\forall x, y \in \mathcal{O}((\mathfrak{o}(x, \mathtt{label}, L_1) \wedge \cdots \wedge \mathfrak{o}(x, \mathtt{label}, L_m) \wedge \mathfrak{o}(x, P_1, v_1) \wedge \cdots \wedge \mathfrak{o}(x, P_n, v_n) \wedge \\ \mathfrak{o}(y, \mathtt{label}, L_1) \wedge \cdots \wedge \mathfrak{o}(y, \mathtt{label}, L_m) \wedge \mathfrak{o}(y, P_1, w_1) \wedge \cdots \wedge \mathfrak{o}(y, P_n, w_n) \wedge \\ v_{i_1} = w_{i_1} \wedge \cdots \wedge v_{i_k} = w_{i_k}) \Rightarrow x = y).\end{gathered}$$

In other words, the instance satisfies the completeness-tailored uniqueness constraint whenever there are no two different objects x and y that each carry all labels in L, all properties in P and have matching values on all the properties in U. For a singleton L we call $L : P : U$ a *single-labeled* ctUC, and otherwise *multi-labeled.*

As mentioned before, it is without loss of generality that we define $U \subseteq P$. Indeed, we may lift this requirement and give the definition as follows. For $L = \{L_1, \dots, L_m\} \subseteq \mathcal{L}$, $P = \{P_1, \dots, P_n\} \subseteq \mathcal{P}$ and $U = \{U_1, \dots, U_k\} \subseteq \mathcal{P}$ (not necessarily $U \subseteq P$), the ctUC $L : P : U$ over object store $\mathfrak{O}$ is satisfied by the instance $\mathfrak{o}$ over $\mathfrak{O}$ if and only if

$$\begin{gathered}\forall x, y \in \mathcal{O}((\mathfrak{o}(x, \mathtt{label}, L_1) \wedge \cdots \wedge \mathfrak{o}(x, \mathtt{label}, L_m) \wedge \\ \mathfrak{o}(x, P_1, v_1) \wedge \cdots \wedge \mathfrak{o}(x, P_n, v_n) \wedge \mathfrak{o}(x, U_1, u_1) \wedge \cdots \wedge \mathfrak{o}(x, U_k, u_k) \wedge \\ \mathfrak{o}(y, \mathtt{label}, L_1) \wedge \cdots \wedge \mathfrak{o}(y, \mathtt{label}, L_m) \wedge \\ \mathfrak{o}(y, P_1, w_1) \wedge \cdots \wedge \mathfrak{o}(y, P_n, w_n) \wedge \mathfrak{o}(y, U_1, u'_1) \wedge \cdots \wedge \mathfrak{o}(x, U_k, u'_k) \wedge \\ u_1 = u'_1 \wedge \cdots \wedge u_k = u'_k) \Rightarrow x = y).\end{gathered}$$

Then this definition is equivalent to that of $L : PU : U$ in Definition 2. Hence, without loss of generality, we have $U \subseteq P$.

Again, we illustrate the definition of completeness-tailored uniqueness constraints on our running example of property graphs. For that purpose, we define the subset $O_L \subseteq V \cup E$ of vertices and edges in a property graph that carry all the labels of the given set $L \subseteq \mathcal{L}$ of labels, as follows: $O_L = \{o \in O \mid L \subseteq \lambda(o)\}$.

A *completeness-tailored uniqueness constraint* (or ctUC) over $\mathcal{O}, \mathcal{P}$ and $\mathcal{V}$ is an expression $L : P : U$ where $L \subseteq \mathcal{L}$ and $U \subseteq P \subseteq \mathcal{P}$. For a given property graph $G = (V, E, \eta, \lambda, \nu)$ over $\mathcal{O}, \mathcal{P}, \mathcal{V}$, we say that G *satisfies* the ctUC $L : P : U$ over $\mathcal{O}, \mathcal{P}$ and $\mathcal{V}$, denoted by $\models_G L : P : U$, iff there are no objects $x, y \in O_L$ such that $x \neq y$, for all $P_i \in P$, $\nu(x, P_i)$ and $\nu(y, P_i)$ are defined, and for all $P_i \in U$, $\nu(x, P_i) = \nu(y, P_i)$.

Note that our definition of ctUCs on property graphs mixes vertices and edges. While this covers a general case, it may be desirable to focus on either case individually, such as managing node or edge integrity separately. For that purpose, one may simply restrict $\mathcal{O}_L$ to V_L or E_L with V and E denoting the sets of vertices and edges, respectively. For our running example, we will restrict our attention to vertices. For example, Neo4j UCs are the special case of ctUCs $L : P : U$ on vertices and where $L = \{\ell\}$ and $P = U = \{u\}$, that is, $\{\ell\} : \{u\} : \{u\}$. Similarly, Neo4j's composite indices are covered as the special case on vertices, where $L = \{\ell\}$ and $P = U$, that is, $\{\ell\} : U : U$.

4.3 Real-world ctUCs for Managing Node Integrity and Node De-duplication

As real-world examples, Table 2 lists all minimal ctUCs $\{Actor, Director\} : P : U$ that hold on G_0. Here, minimal means removal of any property from P or U leads to a ctUC that is violated by G_0. The ctUCs of Table 2 are ranked by their *coverage*, that is, the ratio among all vertices with labels *Actor* and *Director* for which the properties in P exist. The coverage indicates on what ratio of

Table 2. Minimal ctUCs with Label Set {Actor, Director} Satisfied by Graph G_0

E	*U*	*Coverage*
identity	identity	1
tmdbId	tmdbID	1
url	url	1
imdbId	imdbId	0.995893
bornIn, name	**name**	**0.907598**
born, name	**name**	**0.905544**
bornIn, born	*bornIn, born*	*0.891170*
poster, name	**name**	**0.856263**
poster	poster	0.856263
bornIn, died	*bornIn, died*	*0.211499*
born, died	**born**	**0.211499**
died, name	**name**	**0.211499**
poster, died	**died**	**0.197125**

tuples the UC applies. Whenever $P = U$, the ctUC denotes a composite UC. The cases where P contains properties that are not in P are marked in bold font, and indicate ctUCs that cannot be expressed as composite UCs. Interestingly, on nodes for people that are both an actor and a director, the *name* uniquely identifies the person as long as one of the properties *bornIn*, *born*, or *poster* exists. Indeed, the UC *{Actor,Director}*:*{name}* is violated. However, the violation only results from the incorrect duplication of nodes for the same individual (*Marc Singer*). In fact, when the values for properties *bornIn*, *born*, and *poster* became available, a new node (*14,463*) was inserted for the same individual rather than updating the existing node (*32,342*). This led to duplication. Indeed, if G_0' results from G_0 by removing the old node (*32,342*), then G_0' satisfies the key *{Actor,Director}*:*{name}*. The example shows that ctUCs such as *{Actor,Director}*:*{name,bornIn}*:*{name}* can manage node integrity when dirty data is present (for example when duplication is unknown or unresolvable), or clean dirty data (de-duplicate) and promote UCs such as *{Actor,Director}*:*{name}*. Finally, the two composite UCs are marked in italic, and the remaining ctUCs are unary UCs.

4.4 Validation of Completeness-tailored Uniqueness Constraints

Validation is the problem where, given a constraint σ and an instance $\mathfrak{o}$ over an object store $\mathfrak{O}$, one decides whether $\mathfrak{o}$ satisfies σ (which is sometimes stated as σ holds on $\mathfrak{o}$). The validation problem is fundamental for any constraint as an efficient solution minimizes the overhead for maintaining consistency of the instance under updates.

PG-keys [2] are deliberately defined loosely since they simply use some appropriate query language for the purpose of defining the scope and descriptor of keys. Among other desiderata for PG-keys, the authors state “It should be relatively straightforward to validate a key, i.e., check whether it holds in a given property graph. Its complexity should be comparable to the complexity of executing a query in the available querying apparatus”. Indeed, the validation of PG-keys is recast as the problem of query evaluation.

Since ctUCs constitute a special case of exclusive PG-keys, it is not a surprise that they can be validated using queries. In fact, given the ctUC $L : P : U$ with $L = \{L_1, \ldots, L_m\}$, $P = \{P_1, \ldots, P_n\}$, and $U = \{U_1, \ldots, U_k\}$, a given property graph (instance) satisfies $L : P : U$ if and only if the following Cypher query returns an empty result:

```
MATCH    $n_1 : L_1 : \cdots : L_m, n_2 : L_1 : \cdots : L_m$
WHERE    $n_1 <> n_2$ AND
         EXISTS($n_1.P_1$) $\cdots$ AND EXISTS($n_1.U_k$) AND
         EXISTS($n_2.P_1$) $\cdots$ AND EXISTS($n_2.U_k$) AND
         $n_1.U_1 = n_2.U_2$ AND $\cdots$ AND $n_1.U_k = n_2.U_k$
RETURN   $n_1, n_2$.
```

We believe that any form of uniqueness constraint for use in practice must be able to promote efficient index structures that enable the underlying data system to scale validation in a strong sense. Indeed, as the next section shows, property graph systems already have the means to perfectly scale ctUC validation under updates since the number of accesses to the ctUC-index required remains constant while the instance size grows.

5 USE CASES

We showcase the use of ctUCs on the most common data processing tasks: updates and queries.

5.1 Updates

A major use case of constraints is to enforce them during updates. This means that every update operation that results in an instance that would violate some constraint will be rejected (or at least raises some red flag). This is no different for ctUCs. However, for the special case of property graphs, we now highlight some important differences to keys on nodes.

Unlike keys on nodes, we can create nodes that miss properties from the ctUC, or remove some ctUC properties from nodes. Given the key *{Actor,Director}*:*{bornIn,name}*, the operation

```
CREATE (n:Actor:Director {name:‘‘Billy Jean’’})
```

would not create a new node since a value for the property *bornIn* is not provided. In contrast, the UC *{Actor,Director}*:*{bornIn,name}*,*{bornIn,name}* would permit the creation of this node. Similarly, we may remove the property *bornIn* or *name* from any existing node without violating the ctUC, while this is impossible for the key on nodes.

Indeed, ctUCs only check uniqueness of the required properties on those objects for which these properties exist. Consider the following update operation to the node with unique property-value pair *tmdbID=“1,081,559”*, see Figure 3:

```
MATCH (n :Actor :Director {tmdbID: ‘‘1,081,559’’})
SET n.bornIn = ‘‘Vancouver’’
RETURN n.name, n.bornIn.
```

This operation creates the property *bornIn=“Vancouver”* for this node, without violating the composite UC *{Actor,Director}*:*{bornIn,name}*:*{bornIn,name}* since the only other *:Actor:Director* node that has the same value (*Marc Singer*) on property *name* has a different value on *bornIn* (Vancouver, British Columbia, Canada). In contrast, the same operation will violate ctUC *{Actor, Director}*:*{bornIn,name}*:*{name}*.

5.2 Indexing for Efficient Updates and Queries

Currently, Neo4j offers two kinds of support for the creation of indices based on uniqueness constraints. Whenever a unary UC with a single label is specified, such as $\{l\} : \{u\}$, Neo4j will automatically add an index on those properties, so such an index cannot be added separately. Currently there is no support for composite uniqueness constraints in Neo4j (not even for single labels). However, composite indices on single labels can be specified manually. For example,

```
CREATE INDEX iAct FOR (n:Actor) ON (n.bornIn, n.name)
```

creates an index to enforce *{Actor}:{bornIn,name}*.

For ctUCs there is no support in terms of index structures available. In SQL, ctUCs can be enforced using so-called filtered indices, and this should be made available in graph data management systems as well. For example, specifying a ctUC such as *{Actor,Director}*:*{bornIn,name}*:*{name}* should automatically create a filtered composite index like:

```
CREATE INDEX index_Actor_Director
FOR (n:Actor:Director)
ON (n.name) WHERE exists(n.bornIn).
```

Note that the index will only be created on nodes *n* for which the property *name* exists, so `exists(n.name)` has been omitted from the `WHERE` clause. The use of such a filtered index would result in faster update operations such as those presented in the last subsection, but also result in faster evaluation of queries. For instance, executing the following query without an index

```
MATCH (n:Actor:Director)                                  (1)
WHERE EXISTS(n.bornIn) AND EXISTS(n.name)
RETURN n.name,
```

will do a NodeByLabelScan for the node label *Director*, followed by a filter operation on *Actor*. This query would still take a long on realistically-sized property graphs. However, if we create an index as above, the query optimizer will use the index and perform an efficient `NodeIndexScan` search. Illustrating the benefit of these indices, we worked around the index creation by assigning a new label to all nodes that are labeled by *Actor* and *Director* for which the properties *bornIn* and *name* exist, and then created an index on property *name* for nodes with the new label. This is impractical and users must not be burdened with it. As Figure 4 shows, executing Query (1) results in a total of 15,521 database hits, while utilizing the index will result in a total of 885 hits. Neo4j evaluates the performance of queries by *database hits* as a more accurate measure than query response time. We ran the query ten times without the index, followed by ten runs with the index, and their average run time was 8.3 ms without and 2.3 ms with the index. Hence, we can recognize the benefit of indexing.

6 EXPERIMENTS

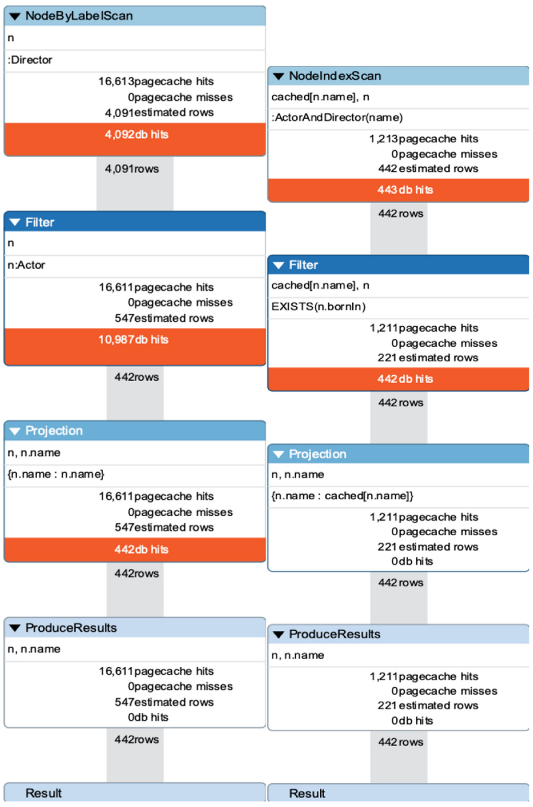
We will now report on some experiments we have conducted to quantify the impact of indices on query and update performance. These indices are based on ctUCs that govern the property graphs.

6.1 Queries

Previously, we analyzed the effect of an index on the query performance over static data. Now we analyze the impact of data size, more precisely the number of vertices.

To change the size of vertex set $V_{\{\text{Actor,Director}\}}$, i.e., the number of vertices carrying the labels *Actor* and *Director*, we created new vertices using the Cypher command

```
UNWIND range(1,k) AS vertices
CREATE (n:Actor:Director
name:apoc.text.random(20, ‘‘A-Z’’),
bornIn: apoc.text.random(20, ‘‘A-Z’’),
tmdbId: vertices),
```



```
NodeByLabelScan                      NodeIndexScan
n                                    cached[n.name], n
:Director                            :ActorAndDirector(name)
16,613 pagecache hits                1,213 pagecache hits
0 pagecache misses                   0 pagecache misses
4,091 estimated rows                 442 estimated rows
4,092 db hits                        443 db hits
4,091 rows                           442 rows

Filter                               Filter
n                                    cached[n.name], n
n:Actor                              EXISTS(n.bornIn)
16,611 pagecache hits                1,211 pagecache hits
0 pagecache misses                   0 pagecache misses
547 estimated rows                   221 estimated rows
10,987 db hits                       442 db hits
442 rows                             442 rows

Projection                           Projection
n, n.name                            n, n.name
{n.name : n.name}                    {n.name : cached[n.name]}
16,611 pagecache hits                1,211 pagecache hits
0 pagecache misses                   0 pagecache misses
547 estimated rows                   221 estimated rows
442 db hits                          0 db hits
442 rows                             442 rows

ProduceResults                       ProduceResults
n, n.name                            n, n.name
16,611 pagecache hits                1,211 pagecache hits
0 pagecache misses                   0 pagecache misses
547 estimated rows                   221 estimated rows
0 db hits                            0 db hits
442 rows                             442 rows

Result                               Result
```

Fig. 4. Performance of Query (1) without and with index.

with k being replaced by a respective multiple of 442 to scale the vertex set for the previously discussed queries. With this query, we create vertices carrying both *Actor* and *Director* labels with the values for *name* and *bornIn* set as a randomly generated string of length 20. This helps us on the one hand to generate names and birthplaces and on the other hand to obtain vertices where the values for *bornIn* and especially for *name* are unique with very high likelihood. The value for *tmdbId* for these vertices goes through all integer values from 1 to k which is just used to create the vertices (similar to a for loop).

Figure 5 shows the performance for our query, using scale factors between 2 and 50. The black line (Series 1) shows the database hits for the query without the index, and the gray line (Series 2) for the query with the index. Clearly, larger datasets result in bigger savings when the index is used.

6.2 Updates

One of the main use cases of integrity constraints is to facilitate data integrity when performing updates. When we perform an update operation the graph database management system needs to check whether that update would violate one of these constraints on the property graph. If this is the case then such an update would be rejected. Similarly to the query, we discussed before we like to look at the performance benefits for updates when introducing a filtered index structure. For this purpose, we used the same dataset as before and compared the performance of the update operation

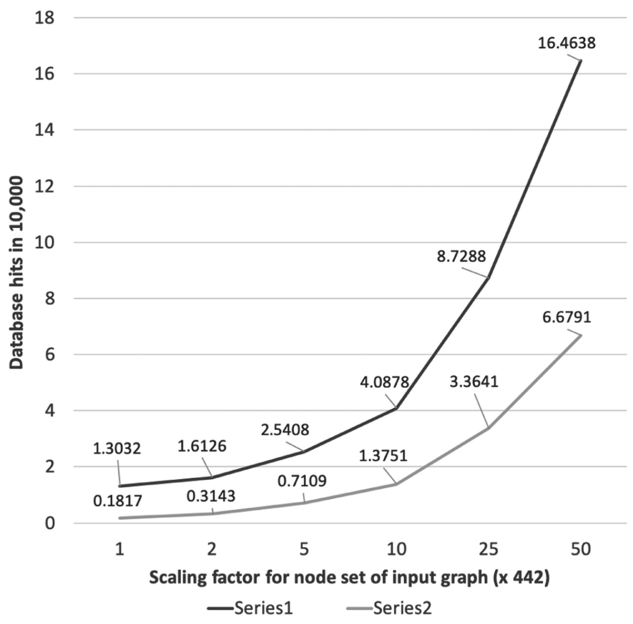


Fig. 5. Comparing query performance with and without index based on ctUC.

```
MATCH (n:Actor:Director) WHERE
EXISTS(n.name) AND EXISTS(n.bornIn)
AND (n.name = ‘‘Larry David’’)
SET n.languages = ‘‘English’’,
```

where we do not employ the filtered index structure to the respective update operation

```
MATCH (n:ActorAndDirector) WHERE
EXISTS(n.name) AND EXISTS(n.bornIn)
AND (n.name = ‘‘Larry David’’)
SET n.languages = ‘‘English’’,
```

where we make use of the index structure by the artificial label *ActorAndDirector*. The update performance is illustrated in Figure 6. Note that the database hits for the update operation remain constant at 4, even when increasing the amount of vertices. This shows again how such an index structure enables direct access to the respective object and emphasizes the benefits for update operations which are one of the most crucial and frequent tasks in databases.

7 AXIOMATIC AND ALGORITHMIC SOLUTIONS TO THE IMPLICATION PROBLEM

We formally define the implication problem associated with ctUCs over object stores, illustrate it on examples over property graphs, establish axiomatic and algorithmic solutions.

Let $\Sigma \cup \{\varphi\}$ denote a set of ctUCs over object store $\mathfrak{O} = (\mathcal{O}, \mathcal{P}, \mathcal{V})$. The *implication problem* for ctUCs is to decide, given $\Sigma \cup \{\varphi\}$, whether Σ implies φ. In fact, Σ *implies* φ, denoted by $\Sigma \models \varphi$, if and only if every instance $\mathfrak{o}$ over object store $\mathcal{O}$ that satisfies all ctUCs in Σ also satisfies φ.

The ability to efficiently decide whether some ctUC φ is implied by Σ is fundamental for managing the integrity of objects over object stores. If φ is implied by by a meaningful set Σ of ctUCs, then we do not need to specify φ because it is specified already implicitly. However, if φ is not implied, then failure to specify it explicitly will result in integrity faults that cannot be detected.

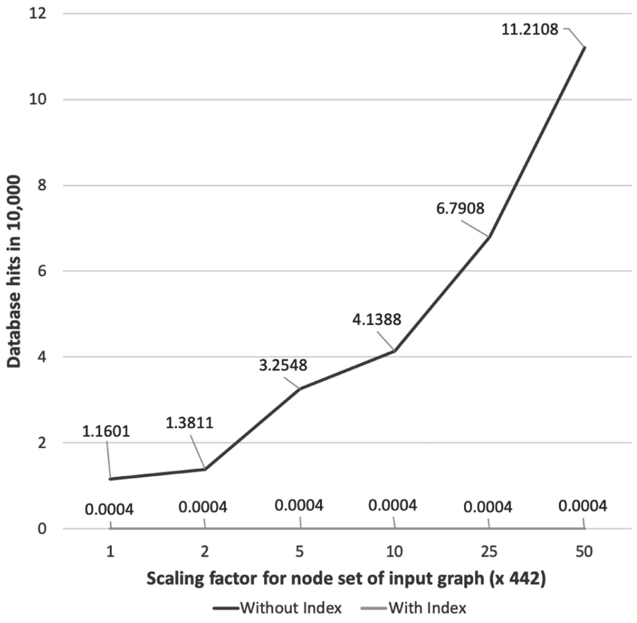


Fig. 6. Perfect scalability of ctUC-validation using ctUC-index, and ctUC-validation without index.

Table 3. Axiomatization $\mathfrak{E} = \{\mathcal{E}\}$ of multi-label ctUCs with the extension rule $\mathcal{E}$

$$\boxed{\frac{L : P : U}{LL' : PP' : UU'}}$$

7.1 Axiomatic Characterization

We will establish an axiomatization for ctUCs. The set $\Sigma^* = \{\varphi \mid \Sigma \models \varphi\}$ denotes the *semantic closure of* Σ, that is, the set of all ctUCs implied by Σ. The semantic closure does not tell us whether we can compute it, nevermind how. It is a core reasoning task to investigate whether/how a semantic notion can be characterized syntactically. In fact, we determine the semantic closure Σ^* of a set Σ of ctUCs by applying *inference rules* of the form $\frac{\text{premise}}{\text{conclusion}}$. For a set $\mathfrak{R}$ of inference rules let $\Sigma \vdash_{\mathfrak{R}} \varphi$ denote the *inference* of φ from Σ by $\mathfrak{R}$. That is, there is some sequence $\sigma_1, \ldots, \sigma_n$ such that $\sigma_n = \varphi$ and every σ_i is an element of Σ or is the conclusion that results from an application of an inference rule in $\mathfrak{R}$ to some premises in $\{\sigma_1, \ldots, \sigma_{i-1}\}$. Let $\Sigma^+_{\mathfrak{R}} = \{\varphi \mid \Sigma \vdash_{\mathfrak{R}} \varphi\}$ be the *syntactic closure* of Σ under inferences by $\mathfrak{R}$. $\mathfrak{R}$ is *sound* (*complete*) if for every set Σ of constraints we have $\Sigma^+_{\mathfrak{R}} \subseteq \Sigma^*$ ($\Sigma^* \subseteq \Sigma^+_{\mathfrak{R}}$). The (finite) set $\mathfrak{R}$ is a (finite) *axiomatization* if $\mathfrak{R}$ is both sound and complete. Table 3 shows an inference rule for the implication of ctUCs, which we will show to be sound and complete. In line with conventions, we write LL' for the set union $L \cup L'$ of label sets, and similarly PP' and UU' for the set unions $P \cup P'$ and $U \cup U'$ of property sets, respectively. The extension rule serves three different purposes, covering redundant extensions of the (i) label set L to LL', (ii) property set P to PP', and unique property set U to UU'. We implicitly assume here that $L : P : U$ and $LL' : PP' : UU'$ are both well-formed, that is, $U \subseteq P$ and $UU' \subseteq PP'$.

We illustrate the use of the inference rules on our running example. There are three distinct ways in which the extension rule $\mathcal{E}$ can be applied. Firstly, we may use it to extend the labels. For instance, the ctUC *{Actor}*:*{died,name}*:*{name}* holds on G_0. Consequently, we can apply $\mathcal{E}$ to infer

Table 4. Simplifying $\mathfrak{E}$ for Composite Indices in Neo4j

$\dfrac{\{l\} : U : U}{\{l\} : UU' : UU'}$	$\dfrac{\{l\} : U}{\{l\} : UU'}$

the ctUC *{Actor,Director}:{died,name}:{name}*. Secondly, we may use $\mathcal{E}$ to extend the property set P. For instance, the ctUC *{Actor, Director}:{poster, name}:{name}* holds on G_0. Hence, an application of the extension rule $\mathcal{E}$ allows us to infer the ctUC *{Actor, Director}:{bornIn,poster,name}:{name}*. Finally, we may use $\mathcal{E}$ to extend the unique property set U. For instance, an application of $\mathcal{E}$ to the ctUC *{Actor, Director}:{bornIn,poster,name}:{name}* results in the ctUC *{Actor, Director}:{bornIn,poster,name}:{poster,name}*. Since we prove in the following theorem that the extension rule $\mathcal{E}$ is sound, all inferred ctUCs are also satisfied by G_0.

We will now show that $\mathfrak{E} = \{\mathcal{E}\}$ forms indeed a sound and complete set of inference rules for the implication of multi-label ctUCs over object stores.

Theorem 1. *The set $\mathfrak{E} = \{\mathcal{E}\}$ forms a finite axiomatization for the implication of multi-label completeness-tailored uniqueness constraints over object stores.*

Proof. For the soundness of $\mathfrak{E}$ we use contra-position to prove that the extension rule $\mathcal{E}$ is sound. For that purpose assume there is an instance $\mathfrak{o}$ that violates the ctUC $LL' : PP' : UU'$. We need to show that $\mathfrak{o}$ also violates the ctUC $L : P : U$. Since $\mathfrak{o}$ violates $LL' : PP' : UU'$, there are objects $x, y \in O$ with $x \neq y$ such that

— $\mathfrak{o}(x, \texttt{label}, \ell)$ and $\mathfrak{o}(y, \texttt{label}, \ell)$ for all $\ell \in LL'$,
— $\mathfrak{o}(x, p_i, v_i)$ and $\mathfrak{o}(y, p_i, w_i)$ for all $p_i \in PP'$
— $v_j = w_j$ for all $p_j \in UU'$.

Consequently, $\mathfrak{o}$ would also violate the ctUC $L : P : U$. This means $\mathfrak{E}$ is sound.

We show the completeness of $\mathfrak{E}$ by contra-position. Let $\Sigma \cup \{L : P : U\}$ denote a set of ctUCs over object store $\mathfrak{O} = (\mathcal{O}, \mathcal{P}, \mathcal{V})$ such that $L : P : U \notin \Sigma^+$. We will show that Σ does not imply $L : P : U$ by defining an instance $\mathfrak{o}$ over $\mathfrak{O}$ that satisfies all ctUCs in Σ and violates $L : P : U$.

Let us define the instance $\mathfrak{o} : \mathcal{O} \times \mathcal{P} \nrightarrow \mathcal{V}$ as follows: for all $\ell \in L$, define $\mathfrak{o}(x, \texttt{label}, \ell)$ and $\mathfrak{o}(y, \texttt{label}, \ell)$, for all $p \in U$ we define $\mathfrak{o}(x, p, 0)$ and $\mathfrak{o}(y, p, 0)$, and for all $p \in P - U$ we define $\mathfrak{o}(x, p, 0)$ and $\mathfrak{o}(y, p, 1)$, while $\mathfrak{o}$ remains undefined elsewhere.

It follows that $\mathfrak{o}$ violates $L : P : U$ since there are $x, y \in \mathcal{O}$ such that $x \neq y$, for all $\ell \in L$, $\mathfrak{o}(x, \texttt{label}, \ell)$ and $\mathfrak{o}(y, \texttt{label}, \ell)$, for all $p_i \in P$, $\mathfrak{o}(x, p_i, v_i)$ and $\mathfrak{o}(y, p_i, w_i)$, and for all $p_j \in U$, $v_j = w_j$.

It remains to show that $\mathfrak{o}$ satisfies every $L' : P' : U' \in \Sigma$. Assume, to the contrary, that $\mathfrak{o}$ violates some $L' : P' : U' \in \Sigma$. Consequently, there are some $x, y \in \mathcal{O}$ such that $x \neq y$, for all $\ell \in L'$, $\mathfrak{o}(x, \texttt{label}, \ell)$ and $\mathfrak{o}(y, \texttt{label}, \ell)$, for all $p_i \in P'$, $\mathfrak{o}(x, p_i, v_i)$ and $\mathfrak{o}(y, p_i, w_i)$, and for all $p_j \in U'$, $v_j = w_j$. By definition of $\mathfrak{o}$, this is only possible when $L' \subseteq L$, $P' \subseteq P$, and $U' \subseteq U$ hold. This, in turn, would mean that $L : P : U \in \Sigma^+$ by a single application of the extension rule $\mathcal{E}$ to $L' : P' : U' \in \Sigma$. This would contradict our premise that $L : P : U \notin \Sigma^+$. Consequently, our assumption that $\mathfrak{o}$ violates some ctUC in Σ must have been wrong. We conclude that $\mathfrak{o}$ satisfies every ctUC in Σ and violates $L : P : U$. Hence, Σ does not imply $L : P : U$, which establishes the completeness of $\mathfrak{E}$. □

Suppose Σ consists of *{Actor,Director}:{bornIn,name}: {bornIn,name}*, and φ denotes the multi-label ctUC *{Actor,Director}:{bornIn,name}:{name}*. The instance in Figure 7 witnesses that Σ does not

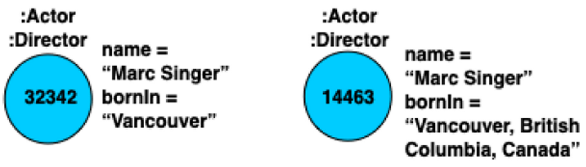


Fig. 7. Witness graph for $\varphi \notin \Sigma^*$.

ALGORITHM 1: Implication of ctUCs

```
Require: Set $\Sigma \cup \{L : P : U\}$
Ensure: TRUE, if $\Sigma \models L : P : U$, and FALSE, otherwise
1: for all $L' : P' : U' \in \Sigma$ do
2:     if $L' \subseteq L$ and $P' \subseteq P$ and $U' \subseteq U$ then    ▷ We found some ctUC that implies $L : P : U$
3:         return TRUE
4: return FALSE
```

imply φ, since the two objects (eg. vertices) are both labeled *Actor* and *Director*, both have properties *bornIn* and *name* with matching values on *name* and non-matching values on *bornIn.*

For the special case of property graphs and Neo4j, UCs are only defined on single labels and only for a single property, that is, $\{l\} : \{u\}$. In this form, no inference rule is required to achieve completeness. In other words, all implied constraints must be specified explicitly. Of course, the Neo4j UCs still imply constraints from other classes, such as composite UCs or ctUCs. Table 4 shows how the axiomatization for ctUCs becomes simpler for the special case of composite indices, as used by Neo4j. In particular, the separation of P and U is no longer necessary since $P = U$ in this case. For that reason the inference rule on the left-hand side of Table 4 can be written in a more concise form as the inference rule on the right-hand side.

It should be noted that ctUCs enjoy a more natural axiomatization than Neo4j's keys from [32]. In particular, the axiomatization for Neo4j keys is binary, while that for ctUCs is unary. This means that any ctUC that is implied by a given set Σ of ctUCs is always implied by a single ctUC $\sigma \in \Sigma$. In particular, the implied ctUC can be inferred from σ by a single application of the extension rule $\mathcal{E}$.

7.2 Algorithmic Characterization

The axiomatization of ctUCs enables us to establish an algorithm that decides implication efficiently. In fact, we can directly prove the following characterization for the implication problem, from which we will derive such an algorithm.

THEOREM 2. *For every ctUC set $\Sigma \cup \{L : P : U\}$ over object store $\mathfrak{O} = (\mathcal{O}, \mathcal{P}, \mathcal{V})$, $\Sigma \models L : P : U$ if and only if there is some $L' : P' : U' \in \Sigma$ such that $L' \subseteq L$, $E' \subseteq E$ and $U' \subseteq U$.*

PROOF.
Sufficiency (⇐). If there is some $L' : P' : U' \in \Sigma$ such that $L' \subseteq L$, $P' \subseteq P$ and $U' \subseteq U$ hold, then an application of the *extension rule* $\mathcal{E}$ shows that $L : P : U \in \Sigma^+_{\mathfrak{E}}$. Soundness of $\mathfrak{E}$ means $L : P : U$ is implied by Σ.

Necessity (⇒). Suppose for all $L' : P' : U' \in \Sigma$ where $L' \subseteq L$, $P' \subseteq P$, we have $U' \nsubseteq U$. That is, there is some property $p \in U' - U$. This constitutes the main case in the proof of Theorem 1. Hence, the instance created in that case satisfies Σ and violates $L : P : U$. Consequently, $L : P : U$ is not implied by Σ. □

As a consequence, we get linear time decidability.

Corollary 3. *Algorithm 1 decides ctUC implication in linear input time.*

Proof. The soundness of Algorithm 1 is a simple application of Theorem 2. Hence, we scan the input $\Sigma \cup \{L : P : U\}$ once to determine if there is a ctUC $L' : P' : U' \in \Sigma$ such that $L' \subseteq L, P' \subseteq P$ and $U' \subseteq U$. Consequently, Algorithm 1 runs in time $\mathcal{O}(|\Sigma \cup \{L : P : U\}|)$. □

Suppose $\Sigma = \{\sigma\}$ consists of the single multi-label UC

{Actor,Director}:*{bornIn, name}*:*{bornIn,name}*,

and φ denotes *{Actor,Director}*:*{bornIn, name}*:*{name}*. Here, φ is not implied by Σ. Indeed, while σ and φ share the same labels and property sets, the unique property *bornIn* of σ is not a unique property of φ. For a different example, let $\Sigma' = \{\sigma'\}$ consists of the ctUC

{Actor,Director}:*{poster,name}*:*{name}*,

and φ' denotes the ctUC *{Actor,Director}*:*{bornIn,poster,name}*:*{bornIn,name}*. Then φ' is indeed implied by Σ' due to the soundness of $\mathcal{E}$.

8 EXTREMAL COMBINATORICS

As uniqueness constraints offer tremendous benefits for managing object integrity and efficient access to objects, it is a practical question to ask how many uniqueness constraints may arise.

Similar to the case of relation schemata where we only need to specify minimal uniqueness constraints, we will only need to specify minimal completeness-tailored uniqueness constraints over object stores. The reason is that any non-minimal ctUC is already implied by a minimal one, and therefore redundant. More formally, given a family $\mathcal{F}$ of ctUCs over $\mathfrak{O} = (\mathcal{O}, \mathcal{P}, \mathcal{V})$, we say that $L : P : U \in \mathcal{F}$ is *redundant* if and only if $\mathcal{F} - \{L : P : U\} \models L : P : U$. We say that $\mathcal{F}$ is *non-redundant* if and only if there is no redundant $L : P : U \in \mathcal{F}$. Due to Theorem 2, $\mathcal{F}$ is non-redundant if and only if there is no $L : P : U \in \mathcal{F}$ such that there is some $L' : P' : U' \in \mathcal{F} - \{L : P : U\}$ where $L' \subseteq L$, $P' \subseteq P$ and $U' \subseteq U$. Hence, we have arrived at some interesting combinatorial questions for non-redundant families of ctUCs.

Firstly, given some finite set P_0 of properties and some finite set L_0 of labels, what is the maximum cardinality $|\mathcal{F}|$ that a non-redundant family

$$\mathcal{F} = \{L : P : U \mid L \subseteq L_0 \wedge U \subseteq P \subseteq P_0\},$$

of ctUCs can possibly attain? Secondly, what families do attain the maximum cardinality?

Answers do not only facilitate our understanding of the constraints, but also provide insight into extreme situations that may occur in object stores. For example, answers tell us how many indices may arise from the constraints. In other words, we understand the maximum number of opportunities for speeding up query and update operations, but, at the same time, also the maximum effort required to maintain indices.

In this section, we will first provide a concise summary of the notions necessary for developing answers to our questions. Subsequently, we will provide full solutions to the two questions above. Finally, we will illustrate the solutions on our running example.

8.1 Prerequisites

Throughout, L_0 and P_0 are disjoint finite and nonempty sets. Furthermore, let $\ell := |L_0|$ and $p := |P_0|$. The power set of a set X is denoted by 2^X. As an example, let us consider $L_0 = \{a(ctor)\}$ with $\ell = 1$, and $P_0 = \{b(orn), n(ame)\}$ with $p = 2$.

A *partially ordered set* (or *poset*) $(S, \leq)$ is a set S together with a reflexive, antisymmetric and transitive relation $\leq$ on S. An *antichain* in $(S, \leq)$ is a subset of S which consists of pairwise unrelated elements. For distinct elements $x, y \in S$, we say that y *covers* x and write $x \prec y$ if $x \leq y$ and $x \leq z \leq y \Longrightarrow z \in \{x, y\}$. A finite poset $(S, \leq)$ is *ranked* if there is a *rank function* $\mathrm{rk} : S \to \mathbb{N} \cup \{0\}$ such that $\mathrm{rk}(x) = 0$ for some minimal element $x \in S$ and $\mathrm{rk}(z) = \mathrm{rk}(y) + 1$ whenever $y \prec z$. The ith *level* of a ranked poset $(S, \leq)$ with rank function rk is the set $S_i := \{x \in S : \mathrm{rk}(x) = i\}$. Note that every level of a ranked poset is an antichain.

For our example, we may consider the set

$$\begin{aligned} S' = \{&(0,0,0),(0,0,1),(0,0,2),(0,1,0),(0,1,1),(0,1,2),(0,2,0),(0,2,1),(0,2,2),\\ &(1,0,0),(1,0,1),(1,0,2),(1,1,0),(1,1,1),(1,1,2),(1,2,0),(1,2,1),(1,2,2)\}, \end{aligned}$$

with the point-wise partial order and the rank of an element being the sum. Then

- $S'_0 = \{(0,0,0)\}$,
- $S'_1 = \{(1,0,0),(0,1,0),(0,0,1)\}$,
- $S'_2 = \{(0,0,2),(0,1,1),(0,2,0),(1,0,1),(1,1,0)\}$,
- $S'_3 = \{(0,1,2),(0,2,1),(1,0,2),(1,1,1),(1,2,0)\}$,
- $S'_4 = \{(0,2,2),(1,1,2),(1,2,1)\}$ and
- $S'_5 = \{(1,2,2)\}$.

For positive integers $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_n$, the chain product $C(\ell_1, \ell_2, \ldots, \ell_n)$ is the Cartesian product of the chains $0 \leq 1 \leq \cdots \leq \ell_i$ of lengths ℓ_i ($i = 1, 2, \ldots, n$), i.e., the set of all vectors $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ with $x_i \in \{0, 1, \ldots, \ell_i\}$ for $i = 1, 2, \ldots, n$, partially ordered by $\mathbf{x} \leq \mathbf{y} \Longleftrightarrow x_i \leq y_i$ for $i = 0, 1, \ldots, n$. Clearly, $C(\ell_1, \ell_2, \ldots, \ell_n)$ is ranked with $\mathrm{rk}(\mathbf{x}) = \sum_{i=1}^{n} x_i$.

For our example, we may consider the chain product $C(1, 2, 2)$ of the chains $0 < 1$, $0 < 1 < 2$ and $0 < 1 < 2$, which is the poset S' above. Note how the rank of S' is simply the sum of ranks for its member chains.

It is well-known that chain products $C(\ell_1, \ell_2, \ldots, \ell_n)$ are *symmetric chain orders* which was first shown by de Bruijn et al. [39]. That is, they can be partitioned into (disjoint) chains of the form $\{a_0 \prec a_1 \prec \cdots \prec a_j\}$ with $\mathrm{rk}(a_0) + \mathrm{rk}(a_j) = s := \sum_{i=1}^{n} \ell_i$ and $j \in \{0, 1, \ldots, s\}$. Note that these chains are *saturated*, meaning that each element (except the last) is covered by its successor. By the symmetry condition $\mathrm{rk}(a_0) + \mathrm{rk}(a_j) = s$, every chain in such a partition intersects the middle level(s) $S_{\lfloor s/2 \rfloor}, S_{\lceil s/2 \rceil}$ in exactly one element. Hence, the number of chains in the partition is $|S_{\lfloor s/2 \rfloor}| = |S_{\lceil s/2 \rceil}|$. On the other hand, any antichain in S can intersect any chain in at most one element. It follows that $S_{\lfloor s/2 \rfloor}$ and $S_{\lceil s/2 \rceil}$ are antichains of maximum cardinality in S, i.e., that chain products have the *Sperner Property* (see also Theorem 5.1.4 in Engel's book [15]).

For our example, we may partition $(S', \leq)$ into the following chains

- $C_1 = (0,0,2) \leq (1,0,2)$
- $C_2 = (1,0,0) \leq (1,1,0) \leq (1,2,0) \leq (1,2,1)$
- $C_3 = (0,0,0) \leq (0,1,0) \leq (0,2,0) \leq (0,2,1) \leq (0,2,2) \leq (1,2,2)$
- $C_4 = (0,0,1) \leq (0,1,1) \leq (0,1,2) \leq (1,1,2)$
- $C_5 = (1,0,1) \leq (1,1,1)$.

In C_3, for example, we have $a_0 = (0,0,0)$ and $a_5 = (1,2,2)$ with $\mathrm{rk}(a_0) + \mathrm{rk}(a_5) = 0 + 5 = 5$. Similarly for C_5, we have $a_0 = (1,0,1)$ and $a_1 = (1,1,1)$ with $\mathrm{rk}(a_0) + \mathrm{rk}(a_1) = 2 + 3 = 5$. Since $s = \ell_1 + \ell_2 + \ell_3 = 1 + 2 + 2$ in our example, the antichains S'_2 and S'_3 have maximum cardinality $s = 5$.

A ranked poset has the *strict Sperner Property* (or is *strictly Sperner*) if the levels of maximum cardinality are the only antichains of maximum cardinality. Griggs [21] proved that the chain product $C(\ell_1, \ell_2, \ldots, \ell_n)$ (with $\ell_1 \leq \cdots \leq \ell_n$) is strictly Sperner if and only if either $n = 1$ or $\sum_{i=1}^{n-1} \ell_i \geq \ell_n$.

In particular, let us consider the product $S = C(\underbrace{1, \dots, 1}_{\ell}, \underbrace{2, \dots, 2}_{p})$ of ℓ chains of length one and p chains of length two. The cardinality $|S_i|$ of its ith level S_i is the coefficient of x^i in the expansion of $(1 + x)^\ell (1 + x + x^2)^p$. Hence, the unique level(s) of maximum cardinality is/are the middle level(s) $S_{\lfloor \ell/2 \rfloor + p}$, $S_{\lceil \ell/2 \rceil + p}$. In what follows, we use $W(\ell, p)$ to denote the coefficient of $x^{\lfloor \ell/2 \rfloor + p}$ in the expansion of $(1 + x)^\ell (1 + x + x^2)^p$, i.e., $W(\ell, p) = |S_{\lfloor \ell/2 \rfloor + p}| = |S_{\lceil \ell/2 \rceil + p}|$. By Griggs' result, if $\ell + p > 2$ then there are no antichains of the maximum cardinality $W(\ell, p)$ other than $S_{\lfloor \ell/2 \rfloor + p}$, $S_{\lceil \ell/2 \rceil + p}$. If $\ell = p = 1$, then it is easy to see that in addition to S_1 and S_2 there is exactly one antichain of maximum size, namely $\{(1, 0), (0, 2)\}$. Definitions not given here can be found in [15].

For our example, we have $\ell = 1$ and $p = 2$, so $S' = C(1, 2, 2)$ is the product of one chain $0 < 1$ and two chains $0 < 1 < 2$. The cardinality $|S_i|$ of the ith level S_i is the coefficient of x^i in $(1 + x) \cdot (1 + x + x^2)^2 = x^5 + 3x^4 + 5x^3 + 5x^2 + 3x + 1$. The levels S_2 and S_3 have therefore both maximum cardinality 5.

Now, the intriguing connection between our ctUCs $L : P : U$ over $L_0 = \{a\}$ and $P_0 = \{b, n\}$ with $\ell = 1$ and $p = 2$ is that every label forms a chain $0 < 1$, resulting from $\emptyset < \{a\}$ for L, and every property forms a chain $0 < 1 < 2$, resulting from $(\emptyset, \emptyset) < (\{b\}, \emptyset) < (\{b\}, \{b\})$ and $(\emptyset, \emptyset) < (\{n\}, \emptyset) < (\{n\}, \{n\})$ for $P : U$ (note in particular that it is not possible for a property to be in U without being in P).

8.2 Main Result

The following main result summarizes full answers to our two questions and formalizes the ideas from the previous section. In particular, the proof establishes the isomorphism between our families of ctUCs and the Cartesian product of their chains.

Before we state the main result, we illustrate the isomorphism on our example. Indeed, the different levels associated with our poset S' were as follows. For each of these, we include a translation into the corresponding ctUC $L : P : U$ over $L_0 = \{a\}$ and $P_0 = \{b, n\}$:

- $S'_0 = \{(0, 0, 0)\}$ translates into $\mathcal{F}'_0 = \{\emptyset : \emptyset : \emptyset\}$
- $S'_1 = \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$ translates into $\mathcal{F}'_1 = \{\{a\} : \emptyset : \emptyset, \emptyset : \{b\} : \emptyset, \emptyset : \{n\} : \emptyset\}$
- $S'_2 = \{(0, 0, 2), (0, 1, 1), (0, 2, 0), (1, 0, 1), (1, 1, 0)\}$ translates into
  $\mathcal{F}'_2 = \{\emptyset : \{n\} : \{n\}, \emptyset : \{b, n\} : \emptyset, \emptyset : \{b\} : \{b\}, \{a\} : \{n\} : \emptyset, \{a\} : \{b\} : \emptyset\}$
- $S'_3 = \{(0, 1, 2), (0, 2, 1), (1, 0, 2), (1, 1, 1), (1, 2, 0)\}$ translates into
- $\mathcal{F}'_3 = \{\emptyset : \{b, n\} : \{n\}, \emptyset : \{b, n\} : \{b\}, \{a\} : \{n\} : \{n\}, \{a\} : \{b, n\} : \emptyset, \{a\} : \{b\} : \{b\}\}$
- $S'_4 = \{(0, 2, 2), (1, 1, 2), (1, 2, 1)\}$ translates into
  $\mathcal{F}'_4 = \{\emptyset : \{b, n\} : \{b, n\}, \{a\} : \{b, n\} : \{n\}, \{a\} : \{b, n\} : \{b\}\}$
- $S'_5 = \{(1, 2, 2)\}$ translates into $\mathcal{F}'_5 = \{\{a\} : \{b, n\} : \{b, n\}\}$.

Indeed, $\mathcal{F}'_2$ and $\mathcal{F}'_3$ attain the maximum cardinality of 5. We will now formalize our main result.

Theorem 4. *Let* $\mathcal{F} \subseteq 2^{L_0} \times 2^{P_0} \times 2^{P_0}$ *such that*

(1) $U \subseteq P$ *holds for all* $L : P : U \in \mathcal{F}$ *and*
(2) *there are no distinct* $L : P : U, L' : P' : U' \in \mathcal{F}$ *with* $L' \subseteq L, P' \subseteq P, U' \subseteq U$.

Then $|\mathcal{F}| \leq W(\ell, p)$, *where equality is attained if and only if*

$$\mathcal{F} = \{L : P : U \in 2^{L_0} \times 2^{P_0} \times 2^{P_0} \ : \ |L| + |P| + |U| = \lfloor \ell/2 \rfloor + p\}$$

or

$$\mathcal{F} = \{L : P : U \in 2^{L_0} \times 2^{P_0} \times 2^{P_0} \ : \ |L| + |P| + |U| = \lceil \ell/2 \rceil + p\}$$

or

$$\ell = p = 1 \ \textit{and}\ \mathcal{F} = \{\, L_0 : \emptyset : \emptyset,\ \emptyset : P_0 : P_0 \,\}.$$

Proof. Let $\mathcal{F}$ be the set of all $L : P : U \in 2^{L_0} \times 2^{P_0} \times 2^{P_0}$ that satisfy condition (1), i.e., $U \subseteq P$. Consider the partial order on $\mathcal{F}$ defined by

$$L' : P' : U' \leq L : P : U \ :\Longleftrightarrow\ L' \subseteq L \wedge P' \subseteq P \wedge U' \subseteq U.$$

Clearly, $(\mathcal{F}, \leq)$ is ranked, where the rank of an element $L : P : U$ equals $|L| + |P| + |U|$. We have to show that any antichain $\mathcal{A}$ in $\mathcal{F}$ has size at most $W(\ell, p)$ and that equality is attained if and only if $\mathcal{A}$ is (one of) the middle level(s) of $\mathcal{F}_{\lfloor \ell/2 \rfloor + p}$ and $\mathcal{F}_{\lceil \ell/2 \rceil + p}$-th level or $\ell = p = 1$ and $\mathcal{A} = \{L_0 : \emptyset : \emptyset, \emptyset : P_0 : P_0\}$. According to Section 8.1, it is sufficient to show that $(\mathcal{F}, \leq)$ is isomorphic to the Cartesian product of ℓ chains of length one and p chains of length two, i.e., to

$$(0 < 1)^{\ell} \times (0 < 1 < 2)^{p} = C(\underbrace{1, \ldots, 1}_{\ell}, \underbrace{2, \ldots, 2}_{p}).$$

With $L_0 = \{a_1, a_2, \ldots, a_\ell\}$ and $P_0 = \{b_1, b_2, \ldots, b_p\}$, the corresponding isomorphism $\varphi : \mathcal{F} \to \{0,1\}^{\ell} \times \{0,1,2\}^{p}$ is given by

$$\varphi(L : P : U) = (x_1, x_2, \ldots, x_\ell, y_1, y_2, \ldots, y_p),$$

where

$$x_i = \begin{cases} 0 & \text{if } a_i \notin L, \\ 1 & \text{if } a_i \in L \end{cases} \qquad \text{and} \qquad y_j = \begin{cases} 0 & \text{if } b_j \notin P, \\ 1 & \text{if } b_j \in P \setminus U, \\ 2 & \text{if } b_j \in U. \end{cases}$$

Indeed, it is straightforward to verify that φ is a bijection from $\mathcal{F}$ to $\{0,1\}^{\ell} \times \{0,1,2\}^{p}$ and that

$$L' \subseteq L \wedge P' \subseteq P \wedge U' \subseteq U,$$

holds if and only if

$$\varphi(L' : P' : U') \leq \varphi(L : P : U),$$

holds.

In particular, note that we have $\varphi^{-1}(\{(1,0),(0,2)\}) = \{L_0 : \emptyset : \emptyset, \emptyset : P_0 : P_0\}$ in the case $\ell = p = 1$. □

8.3 Illustration

For illustrative purposes let us consider another example. This time we fix the label set L_0 that contains the two labels *a(ctor)* and *d(irector)*, as well as the set P_0 of properties consisting of *b(orn)* and *n(ame)*.

Following Theorem 4, there is a unique non-redundant family $\mathcal{F}$ of ctUCs whose maximum cardinality is the coefficient of x^3 in the polynomial

$$\begin{aligned} W(2,2) &= (1+x)^2 \cdot (1 + x + x^2)^2 \\ &= x^6 + 4x^5 + 8x^4 + 10x^3 + 8x^2 + 4x + 1. \end{aligned}$$

Hence, $\mathcal{F}$ consists of the 10 ctUCs $L : P : U$ such that $|L| + |P| + |U| = 3$. These are

- $\emptyset : \{b,n\} : \{n\}$, $\emptyset : \{b,n\} : \{b\}$,
- $\{a\} : \{b\} : \{b\}$, $\{a\} : \{n\} : \{n\}$, $\{a\} : \{b,n\} : \emptyset$,
- $\{d\} : \{b\} : \{b\}$, $\{d\} : \{n\} : \{n\}$, $\{d\} : \{b,n\} : \emptyset$,
- $\{a,d\} : \{b\} : \emptyset$, and $\{a,d\} : \{n\} : \emptyset$.

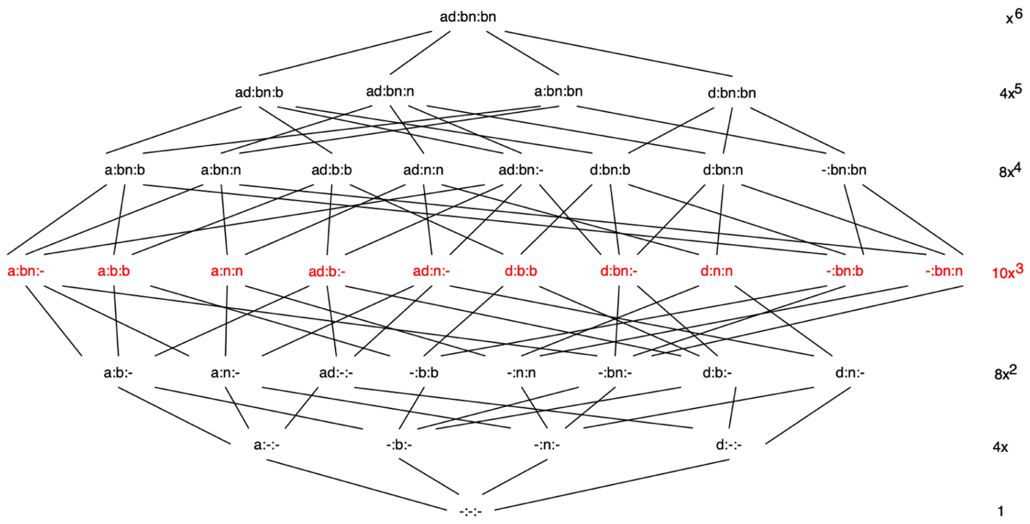


Fig. 8. Lattice of ctUCs over $L_0 = \{a, d\}$ and $P_0 = \{b, n\}$ with the maximum-sized, non-redundant family of ctUCs marked in bold. The right-hand side shows the atoms of the polynomial $W(2, 2)$ together with its coefficients.

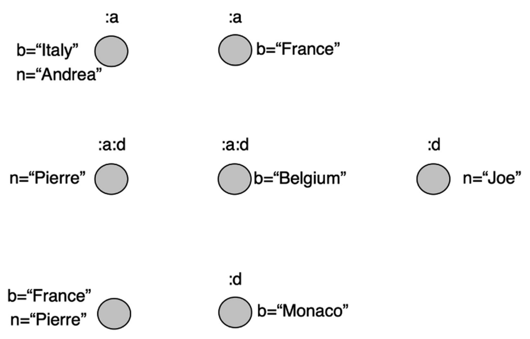


Fig. 9. Objects of instance satisfying a ctUC if and only if it is implied by the family $\mathcal{F}$.

Figure 8 shows the lattice representing all ctUCs over L_0 and P_0 with an edge between two ctUCs $L : P : U$ and $L' : P' : U'$ whenever $L' : P' : U' < L : P : U$ and there is no other $L'' : P'' : U''$ such that $L' : P' : U' < L'' : P'' : U''$ and $L'' : P'' : U'' < L : P : U$. The ctUCs marked in red form the maximum-sized non-redundant family $\mathcal{F}$ of ten ctUCs in this context. Note that the coefficient a_i of x^i in the polynomial $W(2, 2)$ denote the cardinality of the ith level $\mathcal{F}_i$ consisting of ctUCs $L : P : U$ such that $|L| + |P| + |U| = i$.

Finally, Figure 9 shows a set of objects in some object store over $L_0 = \{a, d\}$, $P_0 = \{b, n\}$, and $V_0 = \{a, d, \text{Italy}, \text{Andrea}, \text{France}, \text{Pierre}, \text{Belgium}, \text{Joe}, \text{and Monaco}\}$ that satisfies a ctUC if and only if the ctUC is implied by $\mathcal{F}$.

9 APPLICATIONS TO DATA AND INFORMATION QUALITY

We highlight how ctUCs can be used to identify in parallel (i) meaningful business semantics, and (ii) data quality problems. In addition, we give an example of how ctUCs can be used for the design of graph data models.

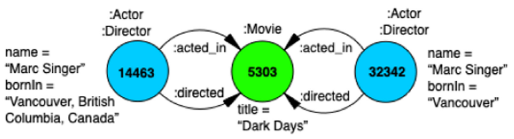


Fig. 10. Duplication of Marc Singer.

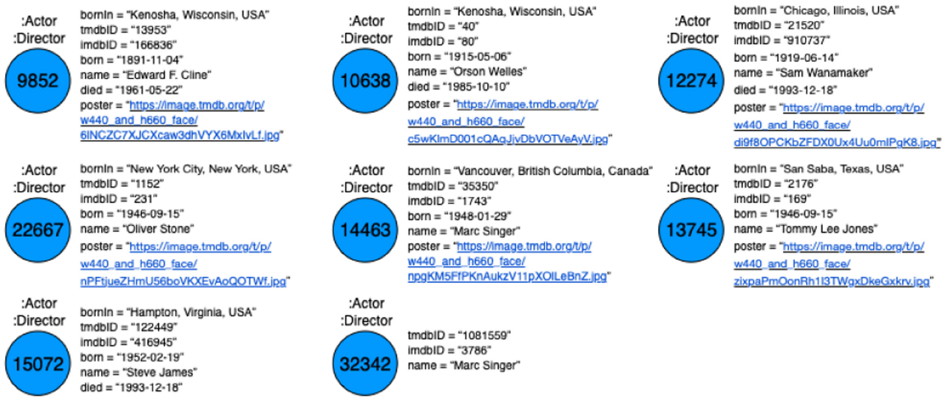


Fig. 11. Perfect node sample for the node set with labels *Actor* and *Director*.

9.1 Business Rule Elicitation and Detecting Data Quality Problems

As mentioned, a closer look at the ctUC *{Actor,Director}*:*{bornIn,name}*:*{name}* suggests the following question: Are there really different people who were actors and directors and have the same name?

Figure 3 shows two actors with the name *Sam Jones*, and two directors with the name *Tom Holland*. The multi-label UC *{Actor,Director}*:*{name}* is violated due to different nodes with *Marc Singer*. Figure 10 reveals that both nodes refer to the same person. This director and actor were recorded with different identifiers and *tmdbIDs*. For one of these nodes, the properties *bornIn*, *born*, and *poster* do not exist.

The validity of *{Actor,Director}*:*{bornIn,name}*:*{name}* made us question why the UC *{Actor,Director}*:*{name}* does not hold. A closer look showed us that the only violation resulted from duplicating the person Marc Singer. This dirty data makes us realize that no two different people that are actors and directors share the same name. Hence, the UC *{Actor,Director}*:*{name}* is a business rule.

Realizing that constraints express meaningful business semantics and any violations constitute dirty data can be facilitated by mining constraints from data, and letting domain experts inspect object samples. In the perfect case, we find a small number of objects that satisfy the same ctUCs as the original instance of objects does. This supports domain experts in their task of spotting dirty data and help data stewards realize, as a consequence, which constraints constitute business rules.

Figure 11 shows a perfect node sample that consists of only eight nodes that satisfy exactly the same ctUCs with label set *{Actor,Director}* as the original dataset with 487 nodes does. Note that the minimal ctUCs from Table 2 cover the set of ctUCs with label set *{Actor,Director}* that hold on G_0. In this sense, the ctUCs from Table 2 and the node sample from Figure 11 represent the same information from different points of views. The ctUC set is the abstract description of the ctUCs that hold, while the perfect sample is a user-friendly view that violates all those ctUCs that do not hold.

The UC *{Actor,Director}*:*{bornIn,born}* that holds on G_0 raises the question why the UCs *{Actor, Director}*:*{bornIn}* and *{Actor,Director}*:*{born}* are not business rules. For the former, *Edward F. Cline* and *Orson Welles* were both born in Kenosha, Wisconsin, USA. For the latter, *Oliver Stone* and *Tommy Lee Jones* were both born on September 15 in 1946. However, is *{Actor,Director}*:*{bornIn,born}* indeed a minimal UC? In fact, it may be true that *{Actor}*:*{bornIn,born}* or *{Director}*:*{bornIn,born}* are already UCs. The twins Maurice and Robin Gibb (the Bee Gees) were also actors, so *{Actor}*: *{bornIn,born}* does not hold. According to G_0, the directors Remy Belvaux and Andre Bonzel were both born in Naumur on 11/10/1966, so *{Director}*:*{bornIn,born}* does not appear to be a business rule either. However, Andre Bonzel was actually born on 13 May 1961 in Paris, so the information in the dataset is wrong. The dataset further says that both directors Danny Pang and Alan Mak were born on 1 January 1965 in Hong Kong. However, Danny Pang was actually born on 11 November 1965, so we have found more data quality problems. Finally, the twins Christoph and Wolfgang Lauenstein were both directors, so the UC *{Director}*:*{bornIn,born}* is not a business rule. However, the ctUC *{Director}*:*{poster,bornIn,born}*:*{bornIn,born}* holds on G_0 with a coverage of 0.43. We just saw that *{Actor,Director}*:*{bornIn,born}* is indeed a minimal UC on G_0, and have found some dirty data along the way.

9.2 Towards Data Quality-driven Schema Design

The design for data quality is an unresolved problem in data engineering [4]: "The problem of measuring and improving the quality of data has been dealt with in the literature as an activity to be performed a posteriori, instead of during the design of the data. As a consequence, the design for data quality is still a largely unexplored area. In particular, while the data quality elicitation and the translation of data quality requirements into conceptual schemas have been investigated; there is a lack of comprehensive methodologies covering all the phases of design of information systems/databases/data warehouse systems." We understand our data-completeness tailored framework [48, 49] as a first step towards a comprehensive methodology for data-quality driven schema design.

Our notion of ctUCs facilitates an extension of this framework to object stores. Consider, as an example of an object store instance, the property graph on the left of Figure 12, where nodes represent parents that pay a benefit, which is determined by the number of their known children. Based on the **completeness-tailored functional dependency (ctFD)** $\{parent,child,benefit\} : parent \to benefit$ on nodes with label *LiableParent*, both occurrences of $benefit = 240$ are redundant. The ctFD is satisfied since for all *LiableParent* nodes with properties *parent*, *child*, and *benefit*, matching values on *parent* imply matching values on *benefit*. The occurrence of *benefit = 360* does not cause inconsistency since the property *child* is missing on the node. Hence, the ctFD facilitates a transformation into the graph on the right of Figure 12. The data redundancy is removed by forming a collection of all the existing children, which can be understood as a value itself. This is achieved by transforming the ctFD into the ctUC *{LiableParent}*:*{parent,child, benefit}*:*{parent}* ensuring that no data redundancy can occur on *benefit*, whenever the property *child* exists.

10 RECOMMENDATIONS FOR OBJECT STORES

Finally, we offer some recommendations to enhance the capability of object stores in managing the integrity of objects with uniqueness constraints.

10.1 Support Composite Uniqueness Constraints

Typically, object stores such as Neo4j only offer support for UCs on single properties. This is beneficial to natural identifiers, such as employee id or national health identifiers, or surrogate identifiers. The latter invites multiple identifiers for the same object, creating problems for data analytics and

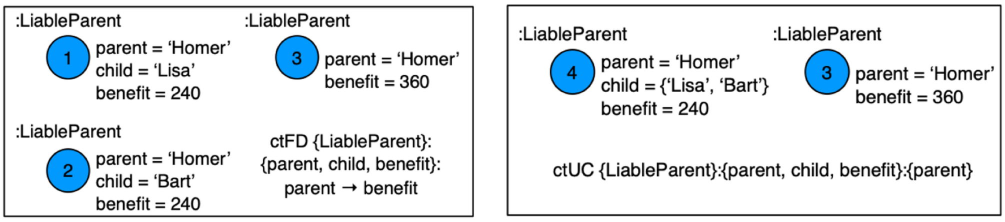


Fig. 12. Removing object redundancy by transforming ctFDs into ctUCs.

management. For example, in the standard Neo4j property graph, two different tmdbIDs have been assigned to the same person *Marc Singer*. This could be avoided by introducing a composite UC, for example on *born* and *bornIn*. Supporting composite UCs is natural and beneficial.

10.2 Support Multi-label Uniqueness Constraints

Currently, object stores, such as Neo4j, only support constraints on single labels. The use of multi-labels is beneficial for at least two reasons. (1) Object stores often claim support for multi-labels. While true in terms of data modeling and querying, there are opportunities to improve updates. (2) Constraints on objects with a single label also apply to nodes with additional labels. However, additional constraints naturally apply to multi-label objects. For example, G_0 does not satisfy single-label UCs *Actor*:*{bornIn,born}* nor *Director*:*{bornIn,born}*, but does satisfy the multi-label UC *{Actor,Director}*:*{bornIn,born}*.

10.3 Support Completeness-tailored Uniqueness Constraints

Foremost, ctUCs separate existence and uniqueness requirements to clarify the role of each property. This separation translates into more targeted management of object integrity, more efficient updates and query operations, business rule elicitation, de-duplication of objects, and data repository design. For instance, the multi-label UC *{Actor,Director}*:*{name,bornIn}*:*{name,bornIn}* is satisfied, but even the stronger ctUC *{Actor,Director}*:*{name,bornIn}*:*{name}* holds. Inspection of the instance based on the ctUC has led to the de-duplication of objects and realization that *{Actor,Director}*:*{name}* is a meaningful business rule.

10.4 Support Filtered Composite Indices

Object stores, such as Neo4j, only supports composite indices on single labels. An extension to multiple labels is illustrated by

```
CREATE INDEX FOR (n:Actor:Director) ON (n.bornIn,n.name),
```

which supports the UC *{Actor,Director}*:*{name,bornIn}*. This is insufficient for ctUCs $L : P : U$ which require filtering by existence constraints on P, and the specification of unique properties in U, based on objects with labels in L. For our example ctUC *{Actor,Director}*:*{name,bornIn}*:*{name}*, the following is a filtered composite index.

```
CREATE INDEX FOR (n:Actor:Director) ON (n.name) WHERE exists(n.bornIn) and
                              exists(n.name).
```

11 CONCLUSION AND FUTURE WORK

We have introduced the class of **completeness-tailored uniqueness constraints** (**ctUCs**) for the flexible management of object integrity in object stores. Our class separates existence from uniqueness concerns, resulting in additional benefits for the efficient updating and querying of properties in object stores. As a special case of the general proposal for PG-keys [2], ctUCs stand out through their ability to express important application semantics and efficacy of their associated computational problems. Firstly, ctUCs facilitate efficient index structures that achieve perfect scalability for validation. Secondly, our axiomatic and algorithmic characterization of their associated implication problem shows that ctUCs can be reasoned about efficiently, paving the way for their utilization in applications such as object integrity management, data cleaning, and the design of object stores. Thirdly, we have also characterized which non-redundant families of ctUCs attain maximum cardinality, giving us an understanding of worst-case integrity management and all opportunities to optimize query and update operations. The ability for these insights further justify our recommendations for their use in future object store systems.

For future work, we plan to investigate other computational problems associated with ctUCs. These include their discovery from instances, which is the problem of computing all minimal ctUCs that are satisfied by a given instance of an object store. Note that the discovery problem for keys over the special object store of property graphs has recently been stated as one of the open problems for data profiling [36]. Another problem is the computation of perfect object samples to facilitate the elicitation of meaningful ctUCs and the identification of integrity faults. In other words, given a set of ctUCs, compute a set of objects that satisfies the given ctUCs but violates all ctUCs not implied by the given ones. This problem may be rephrased with respect to a given instance of objects. In this case, the goal would be to compute a set of objects from the instance which satisfy precisely the ctUCs that hold on the entire instance. Such a sample can be understood as a semantic sample or summary of the given instance, which is also an area of data profiling that should be supported for object stores [36].

REFERENCES

[1] Munqath Alattar and Attila Sali. 2020. Strongly possible keys for SQL. *Journal on Data Semantics* 9, 2–3 (2020), 85–99.

[2] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Keith W. Hare, Jan Hidders, Victor E. Lee, Bei Li, Leonid Libkin, Wim Martens, Filip Murlak, Josh Perryman, Ognjen Savkovic, Michael Schmidt, Juan F. Sequeda, Slawek Staworko, and Dominik Tomaszuk. 2021. PG-Keys: Keys for property graphs. In *Proceedings of the SIGMOD'21: International Conference on Management of Data.* 2423–2436.

[3] Nishita Balamuralikrishna, Yingnan Jiang, Henning Koehler, Uwe Leck, Sebastian Link, and Henri Prade. 2019. Possibilistic keys. *Fuzzy Sets and Systems* 376 (2019), 1–36.

[4] Carlo Batini and Andrea Maurino. 2018. Design for data quality. In *Proceedings of the Encyclopedia of Database Systems, Second Edition*, Ling Liu and M. Tamer Özsu (Eds.).

[5] Joachim Biskup. 2012. Some remarks on relational database schemes having few minimal keys. In *Proceedings of the Conceptual Modelling and Its Theoretical Foundations - Essays Dedicated to Bernhard Thalheim on the Occasion of His 60th Birthday.* Antje Düsterhöft, Meike Klettke, and Klaus-Dieter Schewe (Eds.), Lecture Notes in Computer Science, Vol. 7260, Springer. 19–28.

[6] Joachim Biskup, Umeshwar Dayal, and Philip A. Bernstein. 1979. Synthesizing independent database schemas. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data.* 143–151.

[7] Joachim Biskup and Torsten Polle. 2003. Adding inclusion dependencies to an object-oriented data model with uniqueness constraints. *Acta Informatica* 39, 6–7 (2003), 391–449.

[8] Angela Bonifati, George H. L. Fletcher, Hannes Voigt, and Nikolay Yakovets. 2018. *Querying Graphs.* Morgan & Claypool Publishers.

[9] Pieta Brown and Sebastian Link. 2017. Probabilistic keys. *IEEE Transactions on Knowledge and Data Engineering* 29, 3 (2017), 670–682.

[10] Edgar F. Codd. 1970. A relational model of data for large shared data banks. *Communications of the ACM* 13, 6 (1970), 377–387.

[11] E. F. Codd. 1971. Further normalization of the data base relational model. *Research Report / RJ / IBM / San Jose, California* RJ909 (1971), 1–33.

[12] János Demetrovics. 1978. On the number of candidate keys. *Information Processing Letters* 7, 6 (1978), 266–269.

[13] János Demetrovics, Zoltán Füredi, and Gyula O. H. Katona. 1985. Minimum matrix representation of closure operations. *Discrete Applied Mathematics* 11, 2 (1985), 115–128.

[14] János Demetrovics, Gyula O. H. Katona, Dezsö Miklós, Oleg Seleznjev, and Bernhard Thalheim. 1998. Asymptotic properties of keys and functional dependencies in random databases. *Theoretical Computer Science* 190, 2 (1998), 151–166.

[15] Konrad Engel. 1997. *Sperner Theory*. Cambridge University Press.

[16] Ronald Fagin. 1977. Multivalued dependencies and a new normal form for relational databases. *ACM Transactions on Database Systems* 2, 3 (1977), 262–278.

[17] Ronald Fagin. 1981. A normal form for relational databases that is based on domains and keys. *ACM Transactions on Database Systems* 6, 3 (1981), 387–415.

[18] Wenfei Fan. 2019. Dependencies for graphs: Challenges and opportunities. *ACM Journal of Data and Information Quality* 11, 2 (2019), 5:1–5:12.

[19] Wenfei Fan, Zhe Fan, Chao Tian, and Xin Luna Dong. 2015. Keys for graphs. *PVLDB* 8, 12 (2015), 1590–1601.

[20] Georg Gottlob. 2004. Hypergraph transversals. In *Proceedings of theInternational Symposium on Foundations of Information and Knowledge Systems.* Dietmar Seipel and Jose Maria Turull Torres (Eds.), Lecture Notes in Computer Science, Vol. 2942. Springer, 1–5.

[21] Jerrold R. Griggs. 1984. Maximum antichains in the product of chains. *Order* 1 (1984), 21–28.

[22] Miika Hannula and Sebastian Link. 2018. Automated reasoning about key sets. In *Proceedings of theInternational Joint Conference on Automated Reasoning.* Didier Galmiche, Stephan Schulz, and Roberto Sebastiani (Eds.), Vol. 10900. Springer, 47–63.

[23] Sven Hartmann and Sebastian Link. 2009. Efficient reasoning about a robust XML key fragment. *ACM Transactions on Database Systems* 34, 2 (2009), 10:1–10:33.

[24] I. J. Heath. 1971. Unacceptable file operations in a relational data base. In *Proceedings of the 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control.* 19–33.

[25] Arvid Heise, Jorge-Arnulfo Quiané-Ruiz, Ziawasch Abedjan, Anja Jentzsch, and Felix Naumann. 2013. Scalable discovery of unique column combinations. *Proceedings of the VLDB Endowment* 7, 4 (2013), 301–312.

[26] Christian S. Jensen, Richard T. Snodgrass, and Michael D. Soo. 1996. Extending existing dependency theory to temporal databases. *IEEE TKDE* 8, 4 (1996), 563–582.

[27] Gyula O. H. Katona and Krisztián Tichler. 2006. Some contributions to the minimum representation problem of key systems. In *Proceedings of the International Symposium on Foundations of Information and Knowledge Systems.* 240–257.

[28] Vitaliy L. Khizder and Grant E. Weddell. 2003. Reasoning about uniqueness constraints in object relational databases. *IEEE Transactions on Knowledge and Data Engineering* 15, 5 (2003), 1295–1306.

[29] Henning Köhler, Uwe Leck, Sebastian Link, and Xiaofang Zhou. 2016. Possible and certain keys for SQL. *Proceedings of the VLDB Endowment* 25, 4 (2016), 571–596.

[30] Georg Lausen. 2007. Relational databases in RDF: Keys and foreign keys. In *Proceedings of the SWDB-ODBIS.* 43–56.

[31] Sebastian Link. 2018. Old keys that open new doors. In *Proceedings of the International Symposium on Foundations of Information and Knowledge Systems.* 3–13.

[32] Sebastian Link. 2020. Neo4j keys. In *Proceedings of the International Conference on Conceptual Modeling.* 19–33.

[33] Sebastian Link and Ziheng Wei. 2021. Logical schema design that quantifies update inefficiency and join efficiency. In *Proceedings of the 2021 International Conference on Management of Data.* 1169–1181.

[34] Witold Litwin, Mohammad A. Ketabchi, and Ravi Krishnamurthy. 1991. First order normal form for relational databases and multidatabases. *ACM SIGMOD Record* 20, 4 (1991), 74–76.

[35] Claudio L. Lucchesi and Sylvia L. Osborn. 1978. Candidate keys for relations. *Journal of Computer and System Sciences* 17, 2 (1978), 270–279.

[36] Sofía Maiolo, Lorena Etcheverry, and Adriana Marotta. 2020. Data profiling in property graph databases. *ACM Journal of Data and Information Quality* 12, 4 (2020), 20:1–20:27.

[37] Wai Yin Mok. 2016. Utilizing nested normal form to design redundancy free JSON schemas. *International Journal of Recent Contributions from Engineering, Science & IT* 4, 4 (2016), 21–25.

[38] Wai Yin Mok, Yiu-Kai Ng, and David W. Embley. 1996. A normal form for precisely characterizing redundancy in nested relations. *ACM Transactions on Database Systems* 21, 1 (1996), 77–106.

[39] N. G. de Bruijn, C. A. v. E. Tengbergen, and D. Kruyswijk. 1951. On the set of divisors of a number. *Nieuw Arch. Wiskunde* 23, 2 (1951), 191–193.

[40] Jaroslav Pokorný, Michal Valenta, and Jirí Kovacic. 2017. Integrity constraints in graph databases. In *Proceedings of the 8th International Conference on Ambient Systems, Networks and Technologies and the 7th International Conference on Sustainable Energy Information Technology (Procedia Computer Science)*, Vol. 109. Elsevier, 975–981.

[41] Attila Sali. 2004. Minimal keys in higher-order datamodels. In *Proceedings of the International Symposium on Foundations of Information and Knowledge Systems.* 242–251.

[42] Philipp Skavantzos, Kaiqi Zhao, and Sebastian Link. 2021. Uniqueness constraints on property graphs. In *Proceedings of the International Conference on Advanced Information Systems Engineering.* 280–295.

[43] Bernhard Thalheim. 1989. On semantic issues connected with keys in relational databases permitting null values. *Elektronische Informationsverarbeitung und Kybernetik* 25, 1/2 (1989), 11–20.

[44] David Toman and Grant E. Weddell. 2008. On keys and functional dependencies as first-class citizens in description logics. *Journal of Automated Reasoning* 40, 2–3 (2008), 117–132.

[45] Millist W. Vincent. 1997. A corrected 5NF definition for relational database design. *Theoretical Computer Science* 185, 2 (1997), 379–391.

[46] Ziheng Wei, Uwe Leck, and Sebastian Link. 2019. Discovery and ranking of embedded uniqueness constraints. *PVLDB* 12, 13 (2019), 2339–2352.

[47] Ziheng Wei, Uwe Leck, and Sebastian Link. 2019. Entity integrity, referential integrity, and query optimization with embedded uniqueness constraints. In *Proceedings of the 35th IEEE International Conference on Data Engineering*. 1694–1697.

[48] Ziheng Wei and Sebastian Link. 2019. Embedded functional dependencies and data-completeness tailored database design. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1458–1470.

[49] Ziheng Wei and Sebastian Link. 2021. Embedded functional dependencies and data-completeness tailored database design. *ACM Transactions on Database Systems* 46, 2 (2021), 7:1–7:46.

[50] Jef Wijsen. 1999. Temporal FDs on complex objects. *ACM Transactions on Database Systems* 24, 1 (1999), 127–176.

Received 9 March 2022; revised 5 November 2022; accepted 13 January 2023