Uniqueness Constraints on Property Graphs

Philipp Skavantzos, Kaiqi Zhao, and Sebastian Link[0000−0002−1816−2863]

School of Computer Science
The University of Auckland, Auckland 1010, New Zealand
pska752@aucklanduni.ac.nz|[kaiqi.zhao|s.link]@auckland.ac.nz

Abstract. Graph database are increasingly popular for data management and analytics. As with every data model, managing the integrity of entities is fundamental for data governance but also important for the efficiency of update and query operations. In response to shortcomings of uniqueness and existence constraints in graph databases, we propose a new principled class of constraints that separates uniqueness from existence dimensions, and fully supports multiple labels and composite properties. We illustrate benefits of the constraints on real-world examples by use of the node integrity they enforce for better update and query performance. We establish axiomatic and algorithmic characterizations for reasoning about any set of constraints in our new class. We also give examples of small node samples that satisfy the same constraints as the original data set, and are useful for the elicitation of business rules, and the identification of data quality problems. Finally, we briefly discuss the role of our constraints in the design for data quality, and propose extensions to managing node integrity within graph database systems.

Keywords: Existence constraint; Property graph; Reasoning; Uniqueness constraint

1 Introduction

Uniqueness constraints (UCs) enable the efficient and flexible handling of entity integrity in database management systems [3]. Given a collection of entities, a UC is a set of attributes whose values uniquely identify an entity in the collection. Keys form a special class of UCs where no entity in the database is permitted to have missing values on any attribute of the key. In contrast, UCs only require unique combinations of values for those entities that have no missing value on any attribute of the UC. Both keys and UCs are fundamental 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 redundancies to those that minimize them, and iii) users to access information effectively and efficiently. Knowledge about UCs enables us to i) uniquely reference entities 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 [5] is very expressive to serve its target application of entity resolution. The notion subsumes keys from XML and conditional constraints [5]. This expressiveness has its price, for example, implication is NP-complete, satisfiability and validation are both coNP-complete to decide [5]. 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. In this context, Neo4j keys have been investigated recently [8]. These keys require both existence and uniqueness constraints for all vertices.

In this article we develop a flexible and expressive notion of UCs for the property graph model. As existence and uniqueness of properties are often not both achievable for all vertices, UCs offer a more flexible mechanism to manage node integrity than Neo4j keys do. 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 nodes for which a set of properties exist.

In this article we make the following contributions:

- For the flexible management of node integrity in property graphs, we introduce the class of multi-label embedded uniqueness constraints (eUCs). These include various previous notions as special cases.
- We provide real-world examples and use cases that illustrate the benefit of eUCs for graph data management, including updates, indexing, and query optimization.
- We characterize the implication problem of eUCs both axiomatically and algorithmically. Our characterization makes it simple to obtain a set of eUCs that causes a minimal overhead for managing node integrity.
- We outline applications of eUCs in information systems engineering, such as the elicitation of business rules, the identification of data quality problems, and the design for data quality.
- Finally, we give four recommendations for the future use of uniqueness constraints within graph database management systems.

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 eUCs over property graphs and their illustration on real-world examples is given in Section 4. Use cases of eUCs for efficient updating and querying are discussed in Section 5. In Section 6, we establish axiomatic and algorithmic characterizations of the implication problem. Applications of eUCs to information systems engineering are illustrated in Section 7. Recommendations for the definition and use of uniqueness constraints by graph database management systems are given in Section 8, before we conclude and briefly comment on future work in Section 9.

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

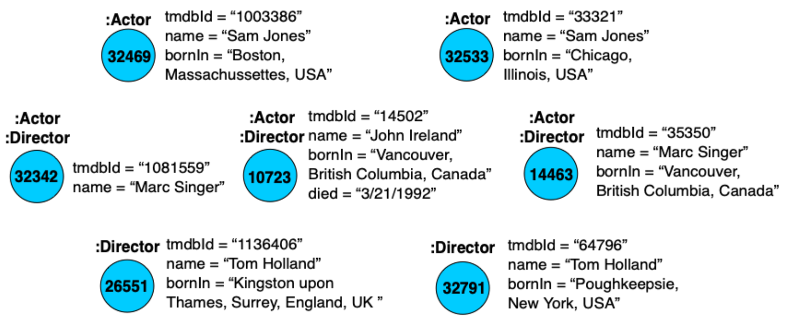


Fig. 1: Real-world vertices, labels, and property-value pairs in *Open Movie Database*

2 Application Scenario

As a 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. Figure 1 shows a few vertices of G_0. For example, the left-most vertex has node identifier *32342*, carries the labels of *Actor* and *Director*, and the properties *tmdbID="1081559"* and *name="Marc Singer"*.

Without looking at the data, we may expect that actors can be identified by their name, and similar for directors. However, the nodes *32469* and *32533* reveal different actors with the same name (*Sam Jones*), while the nodes *26551* and *32791* reveal different directors with the same name (*Tom Holland*). More surprisingly, there appear to be different people that are both actors and directors (with ids *32342* and *14463*), but still share the same name (*Marc Singer*). Hence, even though all nodes with label *Actor* or label *Director* have the property *name*, the three keys $\{Actor\}$:$\{name\}$, $\{Director\}$:$\{name\}$, and $\{Actor,Director\}$:$\{name\}$ are all violated.

Due to the existence requirement of keys, adding a property to the key does not help. For example, the key $\{Actor,Director\}$:$\{name,bornIn\}$ is violated since the vertex *32342* has no property on *bornIn*. In contrast, the UC $\{Actor,Director\}$:$\{name,bornIn\}$ is satisfied by G_0. That is, there are no different people that are actors and directors, for which properties *name* and *bornIn* exist, and who have matching values on both of those properties. Interestingly, Neo4j only supports UCs with singleton labels and singleton properties, so the UC above cannot be expressed by Neo4j.

However, the property graph G_0 satisfies an even sharper UC. Indeed, there are not even two different people that are both actors and directors, for whom *name* and *bornIn* exist, and who have matching values on just *name*. We call these UCs *embedded* (eUCs), and write $L : (E, U)$ to say that there are no two different vertices with all labels in L, all property attributes in E, and matching values on all the property attributes in U. Interestingly, neither of the single-label eUCs $\{Actor\}$:($\{name,bornIn\}$,$\{name\}$) and $\{Director\}$:($\{name,bornIn\}$,$\{name\}$) is satisfied by G_0: The first eUC is violated due to the two actors with name *Sam Jones*, and the second eUC is violated due to the two directors with name *Tom Holland*.

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

Most interestingly, the validity of {*Actor,Director*}:({*name,bornIn*},{*name*}) suggest that the offending nodes may represent node integrity issues. Indeed, it turns out that the two nodes *32342* and *14463* refer to the same individual *Marc Singer*. Most likely, the issue is that *32342* was entered into the dataset when the property *bornIn* was unavailable, and when it became available the node was not updated but a new node (*14463*) was introduced. This suggests that, ideally, the key {*Actor,Director*}:{*name*} should hold. However, without modifying the database, the best we can do is to specify the eUC {*Actor,Director*}:({*name,bornIn*},{*name*}). This is supported by the invalid eUCs {*Actor*}:({*name,bornIn*},{*name*}) and {*Director*}:({*name,bornIn*},{*name*}), and the fact that neither {*Actor*}:{*name*} nor {*Director*}:{*name*} constitute meaningful keys due to different actors with the same name, and similar for directors.

Our application scenario suggests that eUCs provide a robust notion of constraints that can handle node integrity in the presence of dirty data.

3 Literature Review

We discuss related work on keys and uniqueness constraints from relational databases with missing data, and previous related work on keys for graphs.

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 [7]. 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 E is a candidate key when the SQL constraint UNIQUE(E) holds and every attribute in E is defined to be NOT NULL. Recently, eUCs (E, U) have been proposed for relational databases with missing data [10]. Here, the uniqueness of values is only stipulated on a subset $U \subseteq E$ for all records for which values in all attributes in E exist. Indeed, eUCs (E, U) capture SQL UNIQUE as the special case where $E = U$. The benefit of eUCs over SQL UNIQUE in terms of physical data access for more efficient updating and querying has been demonstrated [11].

State-of-the-art graph database management systems provide limited support for node integrity. Similar to candidate keys, Neo4j keys require both existence and uniqueness for all the properties of the key [8]. In [4] 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 [5]. In [6] 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 [9] 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.

UCs provide a more flexible mechanism for managing node integrity than keys do. However, their current use is limited to single labels and single properties in Neo4j, while composite indices on single labels need to be specified manually. We therefore propose multi-label embedded uniqueness constraints $L : (E, U)$, which subsume Neo4j's UCs as the special case where both $L = \{\ell\}$ and $E = \{e\} = U$ are singleton labels and property attributes, respectively, and Neo4j's composite indices as the special case where $L = \{\ell\}$ is a singleton label and $E = U$. As a running example, the multi-label eUC $\{$*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 eUCs $\{$*Actor*$\}$:($\{$*name,bornIn*$\}$,$\{$*name*$\}$) nor $\{$*Director*$\}$:($\{$*name,bornIn*$\}$,$\{$*name*$\}$), as illustrated by G_0. We will formalize our ideas for multi-label eUCs and illustrate their benefits for the effective and efficient management of node integrity in property graphs.

4 Multi-label Embedded Uniqueness Constraints

We recall basics of the property graph model to prepare our formal definition of multi-label embedded uniqueness constraints. Subsequently, we illustrate the new concept of multi-label eUCs on a standard real-world property graph.

4.1 Property Graph Model

We provide the basic definitions for the property graph model [2]. For this we assume that the following sets are pairwise disjoint: $\mathcal{O}$ denotes a set of objects, $\mathcal{L}$ denotes a finite set of labels, $\mathcal{K}$ denotes a set of property attributes, and $\mathcal{N}$ denotes a set of values.

A *property graph* is a quintuple $G = (V, Ed, \eta, \lambda, \nu)$ where $V \subseteq \mathcal{O}$ is a finite set of objects, called *vertices*, $Ed \subseteq \mathcal{O}$ is a finite set of objects, called *edges*, $\eta : E \to V \times V$ is a function assigning to each edge an ordered pair of vertices, $\lambda : V \cup Ed \to \mathcal{P}(\mathcal{L})$ is a function assigning to each object a finite set of labels, and $\nu : (V \cup Ed) \times \mathcal{K} \to \mathcal{N}$ is a partial function assigning values for properties to objects, such that the set of domain values where ν is defined is finite. An example of a property graph is given in Figure 1.

4.2 Introducing Embedded Uniqueness Constraints

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

Before that, we define the subset $V_{\mathfrak{L}} \subseteq V$ of vertices in a property graph that carry at least all the labels of the given set $\mathfrak{L}$ of labels, as follows: $V_{\mathfrak{L}} = \{v \in V \mid \mathfrak{L} \subseteq \lambda(v)\}$.

Definition 1. *For a finite set $\mathcal{L}$ of labels and a finite set $\mathcal{K}$ of property attributes, an* embedded uniqueness constraint *(or* eUC*) over $\mathcal{L}$ and $\mathcal{K}$ is an expression $L : (E, U)$ where $L \subseteq \mathcal{L}$ and $U \subseteq E \subseteq \mathcal{K}$. For a singleton L we call $L : (E, U)$ a* single-labelled *eUC, and otherwise* multi-labelled*. For a given property graph $G = (V, E, \eta, \lambda, \nu)$ over $\mathcal{O}$, $\mathcal{L}$, $\mathcal{K}$, and $\mathcal{N}$ we say that G* satisfies *the eUC $L : (E, U)$ over $\mathcal{L}$ and $\mathcal{K}$, denoted by $\models_G L : (E, U)$, iff there are no vertices $v_1, v_2 \in V_{\mathfrak{L}}$ such that $v_1 \neq v_2$, for all $A \in E$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, and for all $A \in U$, $\nu(v_1, A) = \nu(v_2, A)$.* $\square$

Neo4j UCs are the special case of eUCs $L : (E, U)$ where $L = \{\ell\}$ and $E = U = \{u\}$, that is, $\{\ell\} : (\{u\}, \{u\})$. Neo4j's composite indices are covered as the special case where $L = \{\ell\}$ and $E = U$, that is, $\{\ell\} : (U, U)$.

4.3 Real-World eUCs for Managing Node Integrity and Node De-duplication

As real-world examples, Table 1 lists all minimal eUCs $\{Actor, Director\} : (E, U)$ that hold on G_0. Minimal means removal of any attribute from E or U leads to an eUC that is violated by G_0. The eUCs of Table 1 are ranked by their *coverage*, that is, the ratio among all vertices with labels *Actor* and *Director* for which the properties in E exist. The coverage indicates on what ratio of tuples the UC applies. Whenever $E = U$, the eUC denotes a composite UC. The cases where E contains attributes that are not in U are marked in bold font, and indicate eUCs 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* exist. 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 (*14463*) was inserted for the same individual rather than updating the existing node (*32342*). This led to duplication. Indeed, if G'_0 results from G_0 by removing the old node (*32342*), then G'_0 satisfies the key $\{$*Actor,Director*$\}$:$\{$*name*$\}$. The example shows that eUCs 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 eUCs are unary UCs.

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**

Table 1: Minimal eUCs with label set $\{$Actor,Director$\}$ satisfied by graph G_0

5 Use Cases

We showcase the use of eUCs on the most common data 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 a property graph which would violate some constraint will be rejected (or at least raises some red flag). This is no different for eUCs, but there are important differences to node keys which we will highlight now as well.

Unlike node keys, we can create nodes that miss properties from the eUC, or remove some eUC 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 eUC, while this is impossible for the node key.

Indeed, eUCs only check uniqueness of the required properties on those nodes for which the required properties exist. Consider the following update operation to the unique node where *tmdbID='1081559'*, see Figure 1:

```
MATCH (n :Actor :Director {tmdbID: '1081559'})
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 eUC {*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 property attributes, 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 index_Act FOR (n:Actor) ON (n.bornIn, n.name)
```

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

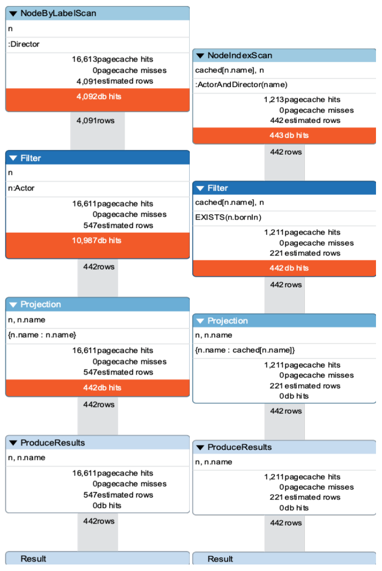
For eUCs there is no support in terms of index structures available. In SQL, eUCs can be enforced using so-called filtered indices, and this should be made available in graph data management systems as well. For example, specifying an eUC such as {*Actor,Director*}:({*bornIn,name*},{*name*}) should automatically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

```
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 speed ups of update operations such as those presented in the last subsection, but also result in speed ups 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 long on realistically-sized property graphs. However, if we create an index as above, the query optimizer 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 2 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 index, followed by ten runs with the index, and their average run time was 8.3ms without and 2.3ms with index. Hence, we can clearly recognize the benefit of indexing.



```
▼ NodeByLabelScan
n
:Director
16,613pagecache hits
0pagecache misses
4,091estimated rows
4,092db hits
4,091rows
▼ Filter
n
n:Actor
16,611pagecache hits
0pagecache misses
547estimated rows
10,987db hits
442rows
▼ Projection
n, n.name
{n.name : n.name}
16,611pagecache hits
0pagecache misses
547estimated rows
442db hits
442rows
▼ ProduceResults
n, n.name
16,611pagecache hits
0pagecache misses
547estimated rows
0db hits
442rows
Result
```

```
▼ NodeIndexScan
cached[n.name], n
:ActorAndDirector(name)
1,213pagecache hits
0pagecache misses
442 estimated rows
443 db hits
442 rows
▼ Filter
cached[n.name], n
EXISTS(n.bornIn)
1,211pagecache hits
0pagecache misses
221 estimated rows
442 db hits
442 rows
▼ Projection
n, n.name
{n.name : cached[n.name]}
1,211pagecache hits
0pagecache misses
221 estimated rows
0db hits
442 rows
▼ ProduceResults
n, n.name
1,211pagecache hits
0pagecache misses
221 estimated rows
0db hits
442 rows
Result
```

Fig. 2: Performance of Query (1) without (left) and with index

6 Reasoning

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

Given a set $\mathcal{L}$ of labels and a set $\mathcal{K}$ of property attributes, let $\Sigma \cup \{\varphi\}$ denote a set of eUCs over $\mathcal{L}$ and $\mathcal{K}$. The *implication problem* for eUCs 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 property graph G that satisfies all eUCs in Σ also satisfies φ.

The ability to efficiently decide whether some eUC φ is implied by Σ is fundamental for node integrity management on property graphs. If φ is implied by by a meaningful set Σ of eUCs, 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.

6.1 Axiomatic Characterization

We will establish an axiomatization for eUCs. The set $\Sigma^* = \{\varphi \mid \Sigma \models \varphi\}$ denotes the *semantic closure of* Σ, that is, the set of all eUCs 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 eUCs 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 2 shows an inference rule for the implication of eUCs, which we will show to be sound and complete. The extension rule serves three different purposes, covering redundant extensions of the i) label set L to LL', ii) embedding E to EE', and unique property set U to UU'. Note that we implicitly assume here that $L : (E, U)$ and $LL' : (EE', UU')$ are both well-formed, that is, $U \subseteq E$ and $UU' \subseteq EE'$.

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

Table 2: Axiomatization $\mathfrak{E} = \{\mathcal{E}\}$ of multi-label eUCs with the extension rule $\mathcal{E}$

$$\boxed{\dfrac{L : (E, U)}{LL' : (EE', UU')}}$$

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 eUCs over property graphs.

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

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 a property graph G that violates the eUC $LL' : (EE', UU')$. We need to show that G also violates the eUC $L : (E, U)$. Since G violates $L : (E, U)$, there are vertices $v_1, v_2 \in V_{LL'}$ such that $v_1 \neq v_2$ and for all $A \in EE'$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, and for all $A \in UU'$, $\nu(v_1, A) = \nu(v_2, A)$. Hence, $v_1, v_2 \in V_{LL'} \subseteq V_L$ such that $v_1 \neq v_2$ and for all $A \in E$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, and for all $A \in U$, $\nu(v_1, A) = \nu(v_2, A)$. Consequently, G would also violate the eUC $L : (E, U)$. This means $\mathfrak{E}$ is sound.

We show the completeness of $\mathfrak{E}$ by contra-position. Let $\Sigma \cup \{L : (E, U)\}$ denote a set of eUCs over $\mathcal{L}$ and $\mathcal{K}$ such that $L : (E, U) \notin \Sigma^+$. We will show that Σ does not imply $L : (E, U)$ by defining a property graph G that satisfies all eUCs in Σ and violates $L : (E, U)$.

$\dfrac{\{l\} : (U,U)}{\{l\} : (UU',UU')}$	$\dfrac{\{l\} : U}{\{l\} : UU'}$

Table 3: Simplifying $\mathfrak{E}$ for Composite Indices in Neo4j

Let us define the property graph $G = (V, Ed, \eta, \lambda, \nu)$ as follows: $V = \{v_1, v_2\}$, $Ed = \emptyset$, and therefore there is nothing to define for η, $\lambda(v_1) = L = \lambda(v_2)$, for all $A \in U$ we define $\nu(v_1, A) = 0 = \nu(v_2, A)$, for all $A \in E$ we define $\nu(v_1, A) = 0$ and $\nu(v_2, A) = 1$, and for all $A \in \mathcal{K} - E$, $\nu(v_1, A)$ and $\nu(v_2, A)$ remain undefined. It follows that G violates $L : (E, U)$ since there are $v_1, v_2 \in V_L$ such that $v_1 \neq v_2$, for all $A \in E$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, and for all $A \in U$, $\nu(v_1, A) = \nu(v_2, A)$. It remains to show that G satisfies every $L' : (E', U') \in \Sigma$. Assume, to the contrary, that G violates some $L' : (E', U') \in \Sigma$. Consequently, $v_1, v_2 \in V_{L'}$ such that $v_1 \neq v_2$, for all $A \in E'$, $\nu(v_1, A)$ and $\nu(v_2, A)$ are defined, and for all $A \in U'$, $\nu(v_1, A) = \nu(v_2, A)$. This is only possible when $L' \subseteq L$, $E' \subseteq E$, and $U' \subseteq U$ hold. This, however, would mean that $L : (E, U) \in \Sigma^+$ by a single application of the extension rule $\mathcal{E}$ to $L' : (E', U') \in \Sigma$. This would contradict our premise that $L : (E, U) \notin \Sigma^+$. Consequently, our assumption that G violates some eUC in Σ must have been wrong. We conclude that G satisfies every eUC in Σ and violates $L : (E, U)$. We conclude that Σ does not imply $L : (E, U)$, which establishes the completeness of $\mathfrak{E}$. □

Suppose Σ consists of the single multi-label UC $\{$*Actor,Director*$\}$:($\{$*bornIn,name*$\}$,$\{$*bornIn,name*$\}$), and φ denotes the multi-label eUC $\{$*Actor,Director*$\}$:($\{$*bornIn,name*$\}$, $\{$*name*$\}$). The property graph in Figure 3 witnesses that Σ does not imply φ, since the two vertices are both labeled *Actor* and *Director*, both have properties *bornIn* and *name* with matching values on *name* and non-matching values on *bornIn*.

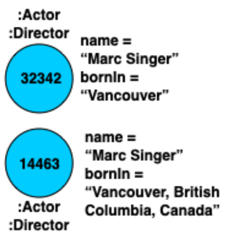


Fig. 3: Witness graph for $\varphi \notin \Sigma^*$

Strictly speaking, UCs in Neo4j 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, only those constraints in the class are implied that are explicitly specified. Of course, the Neo4j UCs still imply constraints from other classes, such as composite UCs or eUCs. Table 3 shows how the axiomatization for eUCs becomes simpler for the special case of composite indices, as used by Neo4j. In particular, the separation of E and U is no longer necessary since $E = U$ in this case. For that reason the inference rule on the left-hand side of Table 3 can be written in a more concise form as the inference rule on the right-hand side.

It should be noted that eUCs enjoy a more natural axiomatization than Neo4j's keys from [8]. In particular, the axiomatization for Neo4j keys is binary, while that for eUCs is unary. This means that any eUC that is implied by a given set Σ of eUCs is always implied by a single eUC $\sigma \in \Sigma$. In particular, the implied eUC can be inferred from σ by a single application of the extension rule $\mathcal{E}$.

Algorithm 1 Implication of eUCs

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

6.2 Algorithmic Characterization

The axiomatization of eUCs 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 eUC set $\Sigma \cup \{L : (E, U)\}$ over $\mathcal{L}$ and $\mathcal{K}$, $\Sigma \models L : (E, U)$ if and only if there is some $L' : (E', U') \in \Sigma$ such that $L' \subseteq L$, $E' \subseteq E$ and $U' \subseteq U$.*

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

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

Theorem 2 gives linear time decidability for eUC implication.

Corollary 1. *Algorithm 1 decides eUC implication in linear input time.*

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

Suppose $\Sigma = \{\sigma\}$ consists of the single multi-label UC {*Actor,Director*}:({*bornIn, name*},{*bornIn,name*}), and φ denotes the multi-label eUC {*Actor,Director*}:({*bornIn, name*},{*name*}). Here, φ is not implied by Σ. Indeed, while σ and φ share the same labels and embeddings, the property *bornIn* of σ is not a property of φ. For a different example, let $\Sigma' = \{\sigma'\}$ consists of the eUC {*Actor,Director*}:({*poster,name*},{*name*}), and φ' denotes the eUC {*Actor,Director*}:({*bornIn,poster,name*},{*bornIn,name*}). Then φ' is indeed implied Σ' due to the soundness of $\mathcal{E}$.

7 Applications in Information Systems Engineering

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

7.1 Business Rule Elicitation and Detecting Data Quality Problems

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

Figure 1 shows two different actors with the name *Sam Jones*, and two different directors with the name *Tom Holland*. The multi-label UC $\{Actor{,}Director\}$:$\{name\}$ is violated due to different nodes with *Marc Singer*. Figure 4 reveals that both nodes refer to the same person. It is simply that this director and actor was recorded with different identifiers and *tmdbIDs*. For one of these nodes, the properties *bornIn*, *born*, and *poster* do not exist.

The validity of the eUC $\{Actor{,}Director\}$:($\{bornIn{,}name\}$,$\{name\}$) 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 node samples. In the perfect case, we find a small number of nodes that satisfy the same eUCs as the original dataset 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.

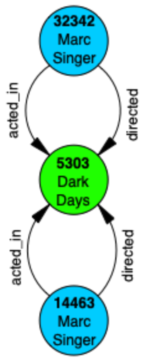
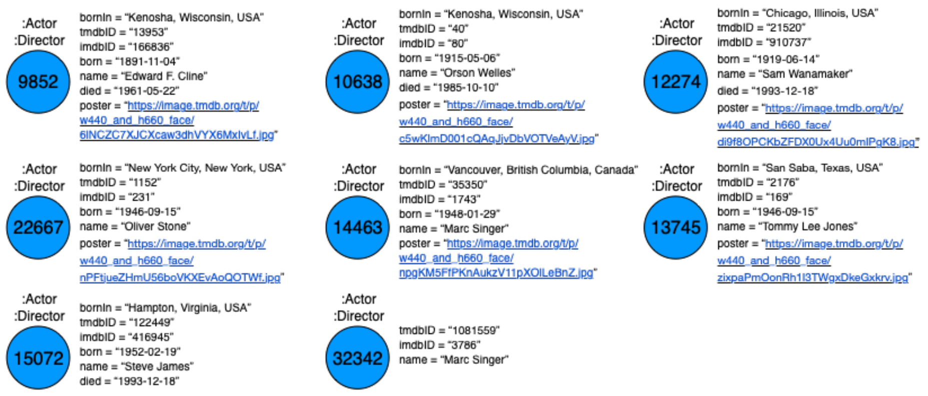


Fig. 4: Duplication of Marc Singer



```
bornIn = "Kenosha, Wisconsin, USA"
tmdbID = "13953"
imdbID = "166836"
born = "1891-11-04"
name = "Edward F. Cline"
died = "1961-05-22"
poster = "https://image.tmdb.org/t/p/
w440_and_h660_face/
6INCZC7XJCXcaw3dhVYX6MxlvLf.jpg"

bornIn = "Kenosha, Wisconsin, USA"
tmdbID = "40"
imdbID = "80"
born = "1915-05-06"
name = "Orson Welles"
died = "1985-10-10"
poster = "https://image.tmdb.org/t/p/
w440_and_h660_face/
c5wKImD001cQAqJjvDbVOTVeAyV.jpg"

bornIn = "Chicago, Illinois, USA"
tmdbID = "21520"
imdbID = "910737"
born = "1919-06-14"
name = "Sam Wanamaker"
died = "1993-12-18"
poster = "https://image.tmdb.org/t/p/
w440_and_h660_face/
di9f8OPCKbZFDX0Ux4Uu0mIPgK8.jpg"

bornIn = "New York City, New York, USA"
tmdbID = "1152"
imdbID = "231"
born = "1946-09-15"
name = "Oliver Stone"
poster = "https://image.tmdb.org/t/p/
w440_and_h660_face/
nPFtjueZHmU56boVKXEvAoQOTWf.jpg"

bornIn = "Vancouver, British Columbia, Canada"
tmdbID = "35350"
imdbID = "1743"
born = "1948-01-29"
name = "Marc Singer"
poster = "https://image.tmdb.org/t/p/
w440_and_h660_face/
npgKM5FfPKnAukzV11pXOlLeBnZ.jpg"

bornIn = "San Saba, Texas, USA"
tmdbID = "2176"
imdbID = "169"
born = "1946-09-15"
name = "Tommy Lee Jones"
poster = "https://image.tmdb.org/t/p/
w440_and_h660_face/
zixpaPmOonRh1I3TWgxDkeGxkrv.jpg"

bornIn = "Hampton, Virginia, USA"
tmdbID = "122449"
imdbID = "416945"
born = "1952-02-19"
name = "Steve James"
died = "1993-12-18"

tmdbID = "1081559"
imdbID = "3786"
name = "Marc Singer"
```

Fig. 5: Perfect node sample for the node set with labels *Actor* and *Director*

Figure 5 shows a perfect node sample that consists of only eight nodes that satisfy exactly the same eUCs with label set $\{Actor{,}Director\}$ as the original dataset with 487 nodes does. Note that the minimal eUCs from Table 1 cover the set of eUCs with label set $\{$*Actor,Director*$\}$ that hold on G_0. In this sense, the eUCs from Table 1 and the node sample from Figure 5 represent the same information from different points of views. The eUC set is the abstract description of the eUCs that hold, while the perfect sample is a user-friendly view that violates all those eUCs that do not hold.

The UC $\{$*Actor,Director*$\}$:$\{$*bornIn,born*$\}$ that holds on G_0 raises the question why $\{$*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, $\{$*Actor*$\}$:$\{$*bornIn,born*$\}$ or $\{$*Director*$\}$:$\{$*bornIn,born*$\}$ may already be 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 eUC $\{$*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 in the dataset along the way.

7.2 Towards Data quality-driven Schema Design

The design for data quality is an unresolved problem in information systems engineering [1]: "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 from [12] as a first step towards a comprehensive methodology for data-quality driven schema design.

Our notion of eUCs facilitates an extension of this framework to property graphs. Consider the property graph on the left of Figure 6, where nodes represent parents that pay a benefit, which is determined by the number of their known children. Based on the embedded functional dependency (eFD) $\{$*parent,child,benefit*$\}$: *parent* $\rightarrow$ *benefit* on nodes with label *LiableParent*, both occurrences of *benefit* $= 240$ are redundant. The eFD 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 eFD facilitates a transformation into the graph on the right of Figure 6. 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 eFD

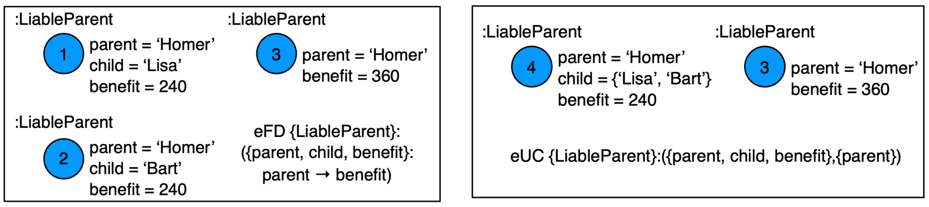


Fig. 6: Removing node redundancy by transforming eFDs into eUCs

into the eUC $\{$*LiableParent*$\}$:($\{$*parent,child, benefit*$\}$*:*$\{$*parent*$\}$) ensuring that no data redundancy can occur on benefit, whenever the property *child* exists.

8 Recommendations for Graph Database Systems

Finally, we offer some recommendations to enhance the capability of graph database systems in managing the integrity of nodes with uniqueness constraints.

Support Composite UCs. Currently, Neo4j only offers 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 entity, creating problems for data analytics and management. The support of composite UCs is not only natural but also beneficial.

Support Multi-label UCs. Currently, Neo4j only supports constraints on single labels. The use of multi-labels is beneficial for at least two reasons. (1) Property graph databases often claim support for multi-labels. While true in terms of data modeling and querying, there are opportunities to improve updates. (2) Constraints on nodes with a single label also apply to nodes with additional labels. However, additional constraints naturally apply to multi-label nodes. 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$\}$.

Support Embedded UCs. Foremost, eUCs separate completeness and uniqueness requirements to clarify the role of each property. This separation translates into more targeted management of node integrity, more efficient updates and query operations, business rule elicitation, de-duplication of nodes, and graph model design.

Support Filtered Composite Indices. Currently, 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 eUCs $L : (E, U)$ which require filtering by existence constraints on E, and the specification of properties in U, based on the node set with labels in L. For our example eUC $\{$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)
```

9 Conclusion and Future Work

We have introduced the class of multi-label embedded uniqueness constraints (eUCs) for the flexible management of node integrity in property graphs. Our class separated existence from uniqueness concerns, resulting in additional benefits for the efficient updating and querying of properties in graphs. Our axiomatic and algorithmic characterization of their associated implication problem shows that eUCs can be reasoned about efficiently, further justifying our recommendations for their use in future graph database management systems.

For future work, we plan to investigate other computational problems associated with eUCs. These include their discovery from property graphs, the computation of perfect samples to facilitate the elicitation of meaningful eUCs and the identification of node integrity faults, and their extreme behavior. The latter refers to the combinatorial problem of characterizing which families of non-redundant eUCs attain maximum cardinality. A solution to this problem will provide us with worst-case bounds for their management, and a better understanding of the search space for the discovery problem.

References

1. Batini, C., Maurino, A.: Design for data quality. In: Liu, L., Özsu, M.T. (eds.) Encyclopedia of Database Systems, Second Edition (2018)
2. Bonifati, A., Fletcher, G.H.L., Voigt, H., Yakovets, N.: Querying Graphs. Synthesis Lectures on Data Management, Morgan & Claypool Publishers (2018)
3. Codd, E.F.: A relational model of data for large shared data banks. Commun. ACM **13**(6), 377–387 (1970)
4. Fan, W., Fan, Z., Tian, C., Dong, X.L.: Keys for graphs. PVLDB **8**(12), 1590–1601 (2015)
5. Fan, W., Lu, P.: Dependencies for graphs. ACM Trans. Database Syst. **44**(2), 5:1–5:40 (2019)
6. Lausen, G.: Relational databases in RDF: keys and foreign keys. In: SWDB-ODBIS. pp. 43–56 (2007)
7. Link, S.: Old keys that open new doors. In: Foundations of Information and Knowledge Systems - 10th International Symposium, FoIKS 2018, Budapest, Hungary, May 14-18, 2018, Proceedings. pp. 3–13 (2018)
8. Link, S.: Neo4j keys. In: Conceptual Modeling - 39th International Conference, ER 2020, Vienna, Austria, November 3-6, 2020, Proceedings. pp. 19–33 (2020)
9. Pokorný, J., Valenta, M., Kovacic, J.: Integrity constraints in graph databases. In: The 8th International Conference on Ambient Systems, Networks and Technologies (ANT 2017) / The 7th International Conference on Sustainable Energy Information Technology (SEIT 2017), 16-19 May 2017, Madeira, Portugal. Procedia Computer Science, vol. 109, pp. 975–981. Elsevier (2017)
10. Wei, Z., Leck, U., Link, S.: Discovery and ranking of embedded uniqueness constraints. PVLDB **12**(13), 2339–2352 (2019)
11. Wei, Z., Leck, U., Link, S.: Entity integrity, referential integrity, and query optimization with embedded uniqueness constraints. In: 35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019. pp. 1694–1697 (2019)
12. Wei, Z., Link, S.: Embedded functional dependencies and data-completeness tailored database design. Proc. VLDB Endow. **12**(11), 1458–1470 (2019)