PG-FD: Mapping Functional Dependencies to the Future Property Graph Schema Standard ⋆

Maude Manouvrier, Khalid Belhajjame

PG-FD: Mapping Functional Dependencies to the Future Property Graph Schema Standard*

Maude Manouvrier[0000−0001−8878−058X] and Khalid Belhajjame[0000−0001−6938−0820]

Université Paris-Dauphine, PSL Research University CNRS UMR [7243] LAMSADE
first_name.last_name@dauphine.fr

Abstract. There has been a notable effort, in the past five years, to develop and promote GQL as a standard for querying graph data, akin to the role SQL plays in relational data querying. Although this goal is still a work in progress, the graph database community has been advancing by not only defining the GQL specification but also introducing additional specifications such as PG-Key and later PG-Schema for specifying graph schema and dependencies. In this regard, a number of proposals have been made in the literature for expressing FD-like dependencies in graph data. Our first contribution is a survey of such proposals, highlighting important ones and their differences. Our second contribution is a solution for translating dependencies defined within such proposals into dependencies that conform to PG-Schema. Our solution is accompanied by a working prototype for translating graph dependencies into PG-Schema compliant dependencies.

Keywords: Property graphs · Graph dependencies · Unified formalism · Graph schema standard

1 Introduction

Graph databases have gained momentum in recent years and are increasingly used to model data and knowledge, as suggested by the popularity and adoption of graph database systems such as Neo4J[1], TigerGraph[2] and Amazon Neptune[3]. They are used in many different areas like social networks, recommendation systems, biomedical research, and fraud detection – see in [7, 36]. The growing demand for graph database systems has highlighted a significant issue: the presence of heterogeneous data models and query languages. This heterogeneity poses challenges for achieving interoperability among different systems and creates difficulties for users and developers who need to navigate diverse data

* This work received support from the National Research Agency under the France 2030 program, with reference to ANR-22-PESN-0007.
This is a preprint of an article published in the proceedings of the 24^{th} European Conference on Advances in Databases and Information Systems (ADBIS'24).

[1] https://neo4j.com

[2] https://www.tigergraph.com

[3] https://aws.amazon.com/neptune

models and understand the nuances of various query languages: openCypher (Neo4j) [21], GSQL (TigerGraph) [11], Gremlin (Apache TinkerPop) [26] and PGQL (Oracle) [25] for property graphs and, SPARQL [1] for RDF graphs, to cite a few – see in [8] for a review.

The above issues have been recognized by academics and industry alike, and some working groups have started working on standardizing query languages for property graph databases. A notable effort in this respect is GQL (Graph Query Language) [20], where the objective is to achieve a success similar to that of SQL by establishing a new standard for querying graphs. GQL has been officially published as an ISO/IEC standard in April 2024[4].

Within the GQL standardization efforts, several initiatives are currently underway at the time of writing. One of these initiatives is the formalism to define keys for property graphs, called PG-Keys [4], and more recently the specification of language for property graphs schema, PG-Schema [5].

Several approaches are particularly interested in integrity constraints and dependencies for graphs defining, for example, Graph Functional Dependencies (GFD) [19], some of them being temporal [3] or probabilistic [37], Graph Entity Dependencies (GED) [16, 17], and recently normal forms for property graphs [30] – see in [8, 32] for a review.

With the emergence of PG-Keys [4] and its extension PG-Schema [5], that should lead to future property graph schema standard, we argue that it is essential to understand how existing integrity constraints and dependencies defined for graphs can be expressed using such future standard. This article contributes in the following two ways: (1) it provides a survey of existing data models and types of constraints utilized for graph-shaped data and (2) it presents a mapping for translating a notable subset of the identified constraint types into the future property graph schema standard PG-Schema.

The paper is organized as follows. Section 2 contains a brief guide of concepts and notation. Section 3 includes a concise review of related work on Graph Dependencies. Section 4 presents the mapping for translating such dependencies into PG-Schema. Section 5 presents our proof-of-concept prototype. Finally, we conclude and briefly comment on future work in Section 6.

2 Preliminaries

Using the same notations and concepts of [8, 17, 28], let a $\mathcal{O}$ be a set of objects, $\mathcal{L}$ be a finite set of labels, $\mathcal{K}$ be a set of property keys, and $\mathcal{N}$ be a set of values. A *Labeled Property Graph* is a graph $G = (V, E, \eta, \lambda, v)$ where:

- $V \subseteq \mathcal{O}$ is a final set of vertices and $E \subseteq \mathcal{O}$ is a final set of edges;
- $\eta : E \rightarrow V \times V$ is a function assigning an ordered pair of vertices to each edge;
- $\lambda : (V \cup E) \rightarrow \mathcal{P}(\mathcal{L})$ is a function assigning to each object a finite set of labels (i.e., $\mathcal{P}(L)$ denotes the set of finite subsets of set $\mathcal{L}$);

[4] https://www.iso.org/standard/76120.html and https://www.gqlstandards.org/home

– $v \rightarrow (V \cup E) \times \mathcal{K} \rightarrow \mathcal{N}$ is a partial function assigning values for properties to objects, such that the object sets V and E are disjoint (i.e., $V \cap E = \emptyset$) and the set of domain values, where v is defined, is finite.

The left-hand side of Figure 1 shows a property graph example representing the current article and its authors, where $V = \{1, 2, 3, 6\}$, $E = \{4, 5, 7\}$, $\lambda(1) = Article$, $\lambda(2) = \lambda(3) = Author$, $\lambda(4) = \lambda(5) = AuthorOf$, $\lambda(6) = Conf$, $\lambda(7) = SubmittedTo$, $\eta(4) = (1, 2)$, $\eta(5) = (1, 3)$, $\eta(7) = (1, 6)$, $v(1, Title) = PG\_FD$, $v(6, Acronym) = ADBIS$, and $v(6, Year) = 2024$, for example.

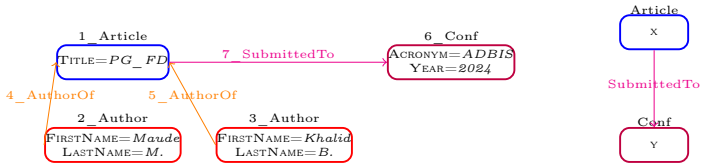


Fig. 1: A Property Graph example (left) and a Graph Pattern example (right)

A *Graph Pattern* is a directed graph $Q[\bar{x}] = (V_Q, E_Q, L_Q)$, where (1) V_Q (E_Q, respectively) is a finite set of pattern nodes (edges, respectively); (2) L_Q is a function that assigns a label to each node $u \in V_Q$ (edge $e \in E_Q$, respectively); and (3) $\bar{x}$ is a list of distinct variables, each denoting a node in V_Q.

An example of a graph pattern is represented in the right-hand side of Figure 1, where $\bar{x}$ contains two variables: a variable x of label *Article* and a variable y of label $Conf$. Both variables are connected with an edge of label $SubmittedTo$.

A *Functional Dependency* (FD) has been recognized as integrity constraints in databases [9]. In relational databases, a functional dependency $X \rightarrow Y$ is defined on a relational schema R with attribute sets X and Y, where R specifies the "scope" of the FD, meaning that, for each instance r of R and for any tuples t_1 and $t_2 \in r$, if t_1 and t_2 have the same value for X, then t_1 and t_2 must have the same value on Y. A review of functional dependencies in databases can be found in [2, 33, 32]. In relational databases, the scope of a *relational Functional Dependency* (RFD) spans the entire relation R on which it has been defined, while applied to graph databases, a *Graph FD* (GFD) is defined on each subgraph matching a graph pattern Q [8].

3 Graph dependencies

In this section, we analyze and compare the main types of dependencies proposed in the contact of graph data over the last few years, taking into consideration the kind of graph data model the dependency operates on, the method by which the dependency's scope is delineated, and the actual definition of the dependency itself.

Most of the dependencies proposed in the context of graphs are drawn from dependencies proposed in the context of relational databases, in particular functional dependencies. In relational databases, however, the scope of the functional

dependency is well delimited intrinsically by the relation to which the attributes participating in the FD belong. This is not the case with graph data. As such, in the context of graph databases, in addition to the dependency itself, information about the scope of the dependency delimiting the sub-graphs in which the dependency is valid, is necessary.

Table 1: Acronyms and references of related work

Acronym	Definition	References
gFD	Graph-tailored functional dependency	[30]
GKs	Graph Keys	[13]
CFDs	Conditional Functional Dependencies	[14]
GPAR	Graph-Pattern Association Rule	[18]
GFD	Graph Functional Dependency	[14]
TGFD	Temporal Graph Functional Dependency	[3]
GED	Graph Entity Dependency	[16, 17]
GDC	Graph Denial Constraint	[17]
GAR	Graph Association Rules	[15]
GD	Graph Dependency	[38]
GDD	Graph Differential Dependency	[22]
GPD	Graph Probabilistic Dependency	[37]

Table 1 presents the references of graph dependencies approaches, and Figure 2 recalls the relationships between all theses approaches. A part of the related work define graph dependencies based on graph pattern (See definition in Section 2). These approaches are presented in Section 3.1. We particularly focus on the ones of W. Fan's research Team [12] who has defined the GED approach [17] that has been widely extended (see red node in Figure 2). In Section 3.2, we present Graph-tailored Functional Dependencies (gFD) of [30] that targets normalization of graph data. Section 3.3 finally presents how constraints are defined in PG-Schema [5], the unifying language for property graph, which serves as a recommendation for GQL standard.

3.1 Pattern-based graph dependencies

Keys are a special case of FDs. Keys for graphs (GKs), identifying entities in RDF graph, have been defined in [13]. In this work, keys are interpreted by means of graph pattern matching via sub-graph isomorphism. Analogous to GKs, *Graph-Pattern Association Rules* GPARs [18] also capture graph patterns to define constraints on graph data, by studying association rules between entities [32].

Subsuming Functional Dependencies (FDs) and Conditional[5] Functional Dependencies (CFDs) of [14], *Graph Functional Dependencies* (GFD) have been defined in [19]. A GFD consists of a graph pattern, identifying entities on which the dependency is defined, and an extension of CFDs specifying the dependencies

[5] i.e. FDs that conditionally hold in a part of the relation [32]

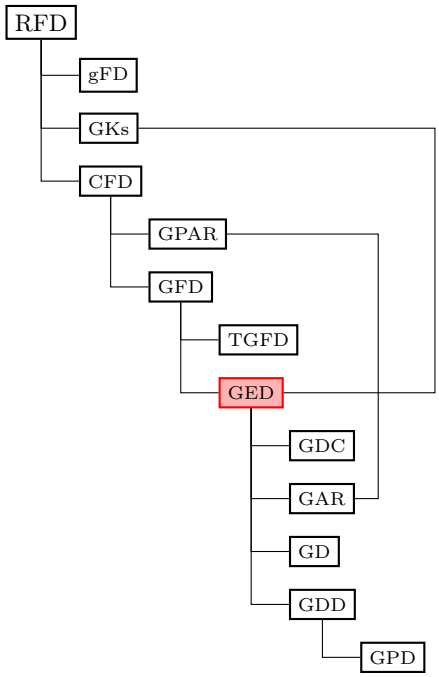


Fig. 2: The relationships among graph dependencies related work

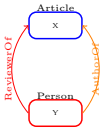
of the attribute values of the entities. *Temporal Graph Functional Dependencies* (TGFDs) of [3] extend GFD to temporal graph (i.e. sequence of graph snapshots).

Later, *Graph Entity Dependencies* (GEDs) [16, 17] have been defined to extend GFDs by supporting literal equality of entity id over two nodes in the graph pattern [3]. GED subsumes GFD of [19] and GKs of [13] – see in [12] for a complete comparison of GKs, GFD and GED.

As defined in [8, 32], a GED φ is a pair $Q[\bar{x}](X \rightarrow Y)$ where: $Q[\bar{x}]$ is a graph pattern (See definition in Section 2) and, $X \rightarrow Y$ is a FD to be applied to entities identified by Q, such that X and Y are two (possibly empty) sets of literals of $\bar{x}$. A literal of $\bar{x}$ can be:

- a constant literal $x.A = c$ where $A \in \mathcal{K}$ is an attribute of $x \in \bar{x}$ and $c \in \mathcal{N}$ is a constant, meaning $v(x, A) = c$;
- a boolean constant $False$, meaning that pattern $Q[\bar{x}]$ is "illegal";
- a variable literal $x.A = y.B$ where $A, B \in \mathcal{K}$ are attributes of the respective entities $x, y \in \bar{x}$, meaning $v(x, A) = v(y, B)$;
- an id literal $id(x) = id(y)$, $x, y \in \bar{x}$ and $id()$ denoting the vertex or edge identities. In this case, pattern $Q[\bar{x}]$ is composed of two similar sub-patterns.

Y can be interpreted as the disjunction of its literals, in an extension of GED called GED$^{\vee}$s – see in [17] for more details.



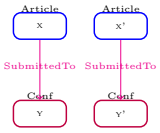
$\mathcal{Q}_1[x, y]$

$\varphi_1 = \mathcal{Q}_1[x, y](\emptyset \rightarrow False)$

```
FOR (x:Article)
 COUNT 0 OF (y:Person)-[:AuthorOf]->(x)<-[:ReviewerOf]-(y:Person)
```

Fig. 3: An "illegal" pattern GED and its translation into PG-Schema

The top of Figure 3 represents an example of a GED, where X is empty and $Y = False$, meaning that the pattern $Q[x, y]$ is "illegal". Indeed, an author of an article cannot be a reviewer of one of its articles. The top of Figure 4 represents an other GED example. It means that if two *Article* entities x and x' have the same title and are submitted to the same conference then, both articles are the same. An article is therefore identified by its title and the id of the conference where it has been submitted.



$\mathcal{Q}_2[x, x', y, y']$

$\varphi_2 = \mathcal{Q}_2[x, x', y, y'](X_2 \rightarrow Y_2)$
with $X_2 = \{x.title = x'.title,\ y.id = y'.id\}$
and $Y_2 = \{x.id = x'.id\}$

```
FOR (x:Article) IDENTIFIER x.title, y.id WITHIN (x)-[:SubmittedTo]->(y:Conf)
```

Fig. 4: A vertex identity GED and its translation into PG-Schema

GEDs have been extended to *Graph Differential Dependencies* (GDDs) in [22], by incorporating distance and matching function. *Graph Probabilistic Dependencies* (GPDs) [37] extend GDD with probabilistic and statistical approaches in order to relax the dependency constraints. *Graph Association Rules* (GARs) extend GFDs and GEDs with the existential semantics for attributes and edges, and by allowing ML classifiers as predicates [15]. *Graph Denial Constraints* (GDCs) [17] extends GEDs, replacing the equality relationship in the definition of GEDs with built-in predicates $=$, $\neq$,$<$,$>$,$\leq$ and $\geq$ [32]. Recent approaches have been proposed in [23, 39] to discover GEDs in property graphs.

As explained in [30], the aforementioned approaches define graph dependencies that compare values of properties or constants for all pairs of entities identified by a graph pattern. The next section presents other graph dependencies having a different expressiveness requiring the existence of some graph objects, i.e. nodes, edges or properties.

3.2 Existence-based functional dependency

Several approaches limit the graph objects on which the graph dependencies hold by existence conditions.

Recently, the *graph-tailored functional dependency* (gFD), defined in [30], restricts the set of vertices on which the graph dependency holds to the nodes for which all properties in a property set P exist, P containing at least all the properties that occur in the dependency. More formally, a gFD is defined as an expression $L : P : X \rightarrow Y$ where $L \subseteq \mathcal{L}$ and $X, Y \subseteq P \subseteq \mathcal{K}$. $L : P : X \rightarrow Y$ is satisfied iff there are no vertices $v_1, v_2 \in V$ such that $v_1 \neq v_2$, for all $A \in P$, $v(v_1, A)$ and $v(v_2, A)$ are defined, for all $A \in X$, $v(v_1, A) = v(v_2, A)$ and for some $A \in Y$, $v(v_1, A) \neq v(v_2, A)$. Basically, a gFD indicates that the combination of values on some node properties uniquely determine the values on some other node properties.

For example, in the property graph of Figure 1, we can define the gFD stipulating that $Conf$ nodes, with properties $Acronym$ and $ConfName$, have matching values on $ConfName$ whenever they have matching values on $Acronym$.:

$$Conf : \{Acronym, ConfName\} : ConfName \rightarrow Acronym$$

The same authors also defined, in [29, 31], *graph-tailored uniqueness constraint* (gUC), which is an expression $L : P : X$, which is satisfied iff there are no vertices $v_1, v_2 \in V$ such that $v_1 \neq v_2$, for all $A \in P$, $v(v_1, A)$ and $v(v_2, A)$ are defined, for all $A \in X$, $v(v_1, A) = v(v_2, A)$. For example, we have:

$$Conf : \{Acronym, ConfName, Year\} : ConfName, Year$$

As explained in [30], every gUC $L : P : X$ implies the gFD $L : P : X \rightarrow P$. According to the authors, $Neo4j$ UCs of [24] are gUCs [30], which are a sub-class of PG-Keys [4], that we explain in the next section.

The Graph Dependency (GD) of [38] extend GED [6] of [16, 17] with existence constraint. A GD φ is a pair $Q[\bar{x}](X \rightarrow Y)$ where $Q[\bar{x}]$ is a topological pattern in the knowledge graph, and X and Y are extended from the GED definition (see Section 3.1) :

- X is extended to the existence of a graph object (node or edge) in the knowledge graph with certain properties value or labels which can be constant or related to $\bar{x}$. More formally, X is expressed as an existing condition, such as $\exists o \in V$ or $\exists o \in E$, associated with predicates such as $o.label = \ell$ with $\ell \in \mathcal{L}$, or $o.A = c$ with c a constant and $A \in \mathcal{K}$, or $o.A = x.A$ with $x \in \bar{x}$.
- Y is extended to support connection between nodes defined in X and nodes in $\bar{x}$. Y is expressed in ASCII art notation of Cypher [21], also adopted by GQL [20]: `(x)->(y)` means that there exists an edge between nodes x and y and `(x)-[e]->(y)` states that edge between nodes x and y is e.

For example, using the Graph Pattern $Q[x, y]$ of the right-hand side of Figure 1, we can define a constraint indicating that all $Article$ nodes should be connected to a $Conf$ one with the following GD: $Q[x, y]$, $X \rightarrow Y$, with X is $\exists$ edge $e \in E$, $\lambda(e) = SubmittedTo$ and Y is `(x)-[e]->(y)`.

[6] Authors of [38] said they adapt GFD of [19]. However, they used graph database and a syntax inspired from *Cypher* language [21] and used node id. Therefore, they rather extend GED than GFD.

3.3 PG-Schema

PG-Schema [5] specifies a schema language for property graphs that should lead to future schema standards [20]. A constraint in PG-Schema, that extends the work on keys for property graphs [4], called PG-Keys, is of the form:

$$\texttt{FOR}\ p(x) < qualifier > q(x, \bar{y})$$

where both $p(x)$ and $q(x, \bar{y})$ are queries, respectively called the scope and the descriptor, and $< qualifier >$ consists of combinations of `EXCLUSIVE`, `MANDATORY`, `SINGLETON`, `IDENTIFIER` and `COUNT LB..UB OF`, with LB an integer representing a lower bound and UB, optional, an integer representing an upper bound.

- `FOR` $p(x)$ `MANDATORY` $q(x, \bar{y})$ means that for every output x of $p(x)$ there should be at least one tuple $\bar{y} = (y_1, y_2, .., y_n)$ that satisfies $q(x, \bar{y})$. As explained in [29], `MANDATORY` combines existence and uniqueness constraints.
- `FOR` $p(x)$ `SINGLETON` $q(x, \bar{y})$ means that for every output x of $p(x)$ there should be at most one tuple $\bar{y} = (y_1, y_2, .., y_n)$ that satisfies $q(x, \bar{y})$.
- `FOR` $p(x)$ `EXCLUSIVE` $q(x, \bar{y})$ means that no $\bar{y}$ should be shared by two different values of x. As explained in [29], `EXCLUSIVE` separates existence and uniqueness constraints.
- `FOR` $p(x)$ `IDENTIFIER` $q(x, \bar{y})$ is equivalent to: `FOR` $p(x)$ `EXCLUSIVE MANDATORY SINGLETON` $q(x, \bar{y})$.
- `FOR` $p(x)$ `COUNT LB..UB OF` $q(x, \bar{y})$ expresses that the number of distinct results returned by $q(x, \bar{y})$ must be between the lower bound and the optional upper bound. $p(x)$ `COUNT 0 OF` $q(x, \bar{y})$ means that $q(x, \bar{y})$ returns an empty set.

For example, to express the constraint indicating that "an article should at least have one author" using PG-Schema, we can write:

```
FOR x:Article MANDATORY e,y WITHIN (x)-[e:AuthorOf]->(y:Author)
```

Keyword `WITHIN` in PG-Schema specifies the pattern to be matched in the property graph.

As explained in [5], PG-Schema allows to handle key and foreign key constraints defined in property graph and can also express express SQL-style CHECK constraints and SQL-style `CHECK` constraints or denial constraints.

4 Expressing graph dependency through PG-Schema

We show, in this section, how graph dependencies can be translated using the PG-Schema language of [5] for property graph. In Subsection 4.1, we focus on *Graph Entity Dependencies* (GEDs) of [17], which is described in Section 3.1. Subsection 4.2 is dedicated to gFD and gUC of [30], described in Section 3.2. Subsection 4.3 refers to GD [38], presented in Section 3.2. Finally, Subsection 4.4 presents how relational FD can also be translated using PG-Schema, relational databases that can be converted into graphs (see in [10] for example). A proof of the soundness of our proposed translation is presented in Subsection4.5.

4.1 From GED to PG-Schema

In GED [17], $\mathcal{Q}[\bar{x}]$ represents a Graph Pattern where $\bar{x}$ is a list of distinct variables, each one denoting a vertex. Let x be the first variable of $\bar{x}$ and L its label ($\lambda(x) = L$). Variable x corresponds to the node on which the dependency is defined.

The following rules (numbered from 1 to 4) represent the translation into PG-Schema of GEDs according to the different forms of FD $X \rightarrow Y$. `Q`, in PG-Schema constraint, represents the pattern $\mathcal{Q}[\bar{x}]$ using the ASCII art: `(var1:NodeLabel)-[var2:EdgeLabel]->(var3:NodeLabel)` with $var1$, $var2$ and $var3$ variables that can be optional.

1. When X and Y only consist of constant literals:

```
FOR (x:L) WHERE X MANDATORY Y WITHIN Q
```

2. When X is $\emptyset$ and Y consists of variable literals:

```
FOR (x:L) MANDATORY Y WITHIN Q
```

3. When X consists of variable literals and Y consists of id literals:

```
FOR (x:L) IDENTIFIER left side of variable literals X WITHIN Q'
```

with `Q'` a sub-pattern of `Q` – An example of this form of FD is given in Figure 4, where pattern `Q` consists of two article vertices linked to two conference vertices. `Q'` represents one article vertex and its related conference vertex.

4. When X is $\emptyset$ and Y is $False$:

```
FOR (x:L) COUNT 0 OF Q
```

An example of this form of FD is given in Figure 3 detailed in Section 4.1.

For example, the bottom of Figure 3 represents the translation of GED $\mathcal{Q}_1$ (above in the figure) into PG-Schema. It means that, for a node x of label $Article$, there does not exists any node y of label $Person$ participating in both $AuthorOf$ and $ReviewerOf$ relations. The bottom of Figure 4 represents the translation of GED $\mathcal{Q}_2$, meaning that an article is identified by its title and the id of the conference where it has been submitted.

4.2 From gFDs to PG-Schema

As explained in [29, 30], a gUC $\{L_1, ..., L_m\} : \{P_1, ..., P_n\} : \{U_1, ..., U_k\}$, with $\forall i$, $L_i \in \mathcal{L}$, $P_i \in \mathcal{K}$ and $\{U_1, ..., U_k\} \subset \{P_1, ..., P_n\}$ can be specified using PG-Schema by the following constraint:

```
FOR x:L1 ... :Lm WHERE x.P1 IS NOT NULL ... AND x.Pn IS NOT NULL
 EXCLUSIVE x.U1, ..., x.Uk
```

For example, the $gUC, Conf : \{Acronym, ConfName, Year\} : ConfName, Year$ can be translated to:

```
FOR x:Conf WHERE x.Acronym IS NOT NULL AND x.ConfName IS NOT NULL
                                       AND x.YEAR IS NOT NULL
 EXCLUSIVE x.ConfName, x.Year
```

A gFD [30], defined by $\{L_1, ..., L_m\} : \{P_1, ..., P_n\} : \{X_1, ..., X_k\} \rightarrow \{Y_1, ..., Y_j\}$ where $\{L_1, ..., L_m\} \subseteq \mathcal{L}$ and $\{X_1, ..., X_k\}, \{Y_1, ..., Y_j\} \subseteq \{P_1, ..., P_n\} \subseteq \mathcal{K}$, can therefore be translated using the `EXCLUSIVE MANDATORY` of PG-Schema:

```
FOR x.Y1, ..., x.Yk WITHIN x:L1 ... :Lm
  WHERE x.X1 IS NOT NULL AND ... AND x.Xk IS NOT NULL
   EXCLUSIVE MANDATORY x.X1, ..., x.Xj
```

For example, the gFD $Conf : \{Acronym, ConfName\} : ConfName \rightarrow Acronym$ can be expressed into PG-Schema by:

```
FOR x.Acronym WITHIN x:Conf
  WHERE x.ConfName IS NOT NULL AND x.Acronym IS NOT NULL
    EXCLUSIVE MANDATORY x.ConfName
```

4.3 From GD to PG-Schema

The following two rules represent the translation into PG-Schema of GD, proposed in [38] and presented in Section 3.2.

1. When X is $\exists e \in E$, $\lambda(e) = L$ and Y is `(x)-[e]->(y)`:

```
FOR (x:x.label) MANDATORY e,y WITHIN (x)-[e:L]->(y:y.label)
```

2. When X is $\exists x \in V$, $\lambda(x) = L$ and Y is `(x)->(y)`:

```
FOR (x:L) MANDATORY e,y WITHIN (x)-[e]->(y:y.label)
```

For instance, the example of GD presented in Section 3.2, using the Graph Pattern $Q[x, y]$ of Figure 1 (right-hand side), and $X \rightarrow Y$, with X defined by $\exists$ edge $e \in E$, $\lambda(e) = SubmittedTo$ and Y is defined by `(x)-[e]->(y)`, can be translated using PG-Schema by:

```
FOR (x:Article) MANDATORY e,y WITHIN (x)-[e:SubmittedTo]->(y:Conf)
```

4.4 From relational databases to PG-Schema

Relational databases can be converted to property graph [10] or into RDF store [27] that can be mapped into property graph [6].

Therefore, a functional dependency (FD) $X \rightarrow Y$ is defined on a relation schema $R(\mathcal{K}_R)$, with $\mathcal{K}_R$ the attribute set containing sets $X = \{X_1, X_2, .., X_n\}$ and $Y = \{Y_1, Y_2, .., Y_m\}$, can also be translated into PG-Schema by:

```
FOR x.Y1, ... Y.Ym WITHIN (x:R) EXCLUSIVE MANDATORY x.X1, ..., X.Xn
```

meaning that the same value of X cannot be shared by two values of Y.

For example, the FD $address \rightarrow region$, with $address$ and $region$ being two attributes of a schema R, means that each address corresponds to an explicit region. When relation R is mapped to a vertex R, the aforementioned FD is translated to:

```
FOR x.region WITHIN (x:R) EXCLUSIVE MANDATORY x.address
```

4.5 Proof of the soundness of the translation

In this section, we outline the sketch of the proof that can be used to demonstrate the soundness of the translation operations presented thus far. Let $\mathcal{M}$ be a graph data model, which could encompass various existing proposals ranging from relational data models to triple-based graph models (RDF) and property graph models. Consider a data dependency *dep* expressed within the data model $\mathcal{M}$. Note that PG-Schema considers graph data expressed using property graphs, which is an expressive data model that encompasses all the data models used by existing proposals. Consequently, an instance $\mathcal{I}$ of a data model $\mathcal{M}$ can be translated to a property-graph instance $\mathcal{I}^{\mathcal{PG}}$ without any loss of information.

As defined in [6], the properties of a mapping are: computability, semantics preservation and information preservation. Our translation is computable. Indeed, it is easy to see that the mapping rules we define can be implemented as an algorithm. An implementation is presented in the next section. Our translation is also information preserving because, due to our mapping rules, it does not lose any information about the graph dependency being translated. Indeed, we can define inverse rules to recover the original dependency from the PG-Schema constraint resulting from the translation process.

To establish the soundness of our proposed translation for transforming a dependency *dep* expressed in a model $\mathcal{M}$ into a dependency dep^{PS} compliant with PG-Schema, and then prove its semantics preservation, it suffices to prove that:

$$\forall \mathcal{I} \in instances(\mathcal{M})\ I \models dep \implies I^{PG} \models dep^{PS} \quad (1)$$
$$I \not\models dep \implies I^{PG} \not\models dep^{PS} \quad (2)$$

In what follows, we show the case where the source model $\mathcal{M}$ is the relational model. Other cases can be proven in a similar manner, [16] showing for example that relational FDs are special cases of GEDs, when tuples in a relation are represented as nodes in a graph.

Consider the following FD: $X \rightarrow Y$ with $X = \{X_1, X_2, .., X_n\}$ and $Y = \{Y_1, Y_2, .., Y_m\}$. According to the solution presented in the previous section, the corresponding PG-Schema dependency is expressed as follows:
`FOR x.Y1, ... Y.Ym WITHIN (x:R) EXCLUSIVE MANDATORY x.X1, ..., X.Xn`.

Consider an instance $\mathcal{I}$ of the relational model where FD holds, signifying that any two tuples t_1 and t_2 sharing identical attribute values in X also share identical values in Y. By contradiction, we can demonstrate that the PG-Schema dependency mentioned earlier also applies to the instance $\mathcal{I}^{\mathcal{PG}}$, thereby showing (1).

Now, suppose FD does not hold in a given relational model $\mathcal{I}$, indicating the existence of two tuples t_1 and t_2 sharing the same values for attributes in X but differing in at least one attribute Y_i in Y. Upon translation of $\mathcal{I}$ into $\mathcal{I}^{\mathcal{PG}}$, two vertices representing t_1 and t_2 emerge, each associated with attributes or properties (edges) specifying values for attributes in X and Y. In this scenario, the PG-Schema dependency defined previously does not hold, as the two vertices in question are linked to the same values (or vertices) in X but not to the same attribute (or vertex) in Y. Hence, this shows (2).

5 Proof-of-concept prototype implementation

To prove that the PG-schema mapping we propose in this article is computable, we have developed a Python prototype called *PG-FD*. In order to manage graph pattern, for GED of [17] and GD of [38], our prototype uses the `network`[7] Python module, which is a powerful library for manipulating property graphs in Python. To manage functional dependencies, for gFDs of [30] or relational schema for example, *PG-FD* uses the Python module functional_dependencies[8], that defines the classe FD to represent a functional dependency.

To translate GED (resp. GD) into PG-Schema, user should call function `GED2PGS(GP,X,Y)` (resp. `GD2PGS(GP,X,Y)`), with `GP` the graph pattern, and `X` and `Y` the right and left part of the graph dependency (coded by list of strings). To define the graph pattern, user can either manually define it using the `MultiDiGraph`[9] class of the `network` module or using *Cypher/GQL* ASCII art style, which will be translated into `MultiDiGraph` by our prototype. Our prototype could be extended to allow the user to create a graph in a graph database (e.g. *Neo4j*) or using *PG-Types* of PG-Schema, that describes the shape of data and the types of graph components such as nodes and edges. To translate a gFD of [30] into PG-Schema, user should call function `gFD2PGS(L,P,fd)` with `L` a set of node labels, `P` a set of properties and `fd` a functional dependency defined using the `functional_dependencies` module. This module is also used to translate a relational FD, using function `Rel2PGS(fd,R)`, with `R` the relation name.

The source code of our prototype can be downloaded from GitHub[10]. The prototype can be used to replay the examples of this article, some GED examples defined in [16], as well as to define new graph dependencies to translate.

6 Conclusions

In this article, we have surveyed the different solutions for defining graph functional dependencies in the literature. We have also defined mapping rules allowing to translate graph dependencies, focusing on prominent ones, into the

[7] https://networkx.org/
[8] https://oer.gitlab.io/cs/functional-dependencies/
[9] https://networkx.org/documentation/stable/reference/classes/multidigraph.html
[10] https://github.com/MaudeManouvrier/PG-FD

PG-Schema [5], which serves as a recommendation or upcoming versions of GQL for specifying property graph schema. A proof of the soundless of the proposed translation and a proof-of-concept implementation prototype are presented in this article.

In our ongoing work, we are extending our solution to cater for other graph-based dependencies, for example the one in [34, 35] which introduces a representation similar to functional dependencies to translate SQL constraints to SHACL (Shapes Constraint Language).

Acknowledgment We would like to thank Ibtissem Khedim for facilitating the initiation of this project during her master's research internship.

References

1. SPARQL 1.1 (2013), https://www.w3.org/TR/rdf-sparql-query/
2. Abedjan, Z., Golab, L., Naumann, F., et al.: Data profiling. Springer Nature (2022). https://doi.org/10.1007/978-3-031-01865-7
3. Alipourlangouri, M.: Temporal dependencies for graphs. In: ACM SIGMOD Conf. (2021). https://doi.org/10.1145/3448016.3450586
4. Angles, R., Bonifati, A., Dumbrava, S., et al.: PG-Keys: Keys for Property Graphs. In: ACM SIGMOD Conf. (2021). https://doi.org/10.1145/3448016.345756
5. Angles, R., Bonifati, A., Dumbrava, S., et al.: PG-Schema: Schemas for property graphs. PACMMOD **1**(2), 1–25 (2023). https://doi.org/10.1145/3589778
6. Angles, R., Thakkar, H., Tomaszuk, D.: Mapping RDF databases to property graph databases. IEEE Access **8**, 86091–86110 (2020). https://doi.org/10.1109/ACCESS.2020.2993117
7. Boncz, P.: The (sorry) state of graph database systems (2022), Keynote in EDBT
8. Bonifati, A., Fletcher, G., Voigt, H., et al.: Querying graphs. Springer (Vol. 10, No. 3, pp. 1-184) (2018). https://doi.org/10.1007/978-3-031-01864-0
9. Codd, E.F.: Further normalization of the data base relational model. Data base systems **6**, 33–64 (1972)
10. De Virgilio, R., Maccioni, A., Torlone, R.: Converting relational to graph databases. In: GRADES (2013). https://doi.org/10.1145/2484425.2484426
11. Deutsch, A., Xu, Y., Wu, M., et al.: Aggregation support for modern graph analytics in TigerGraph. In: ACM SIGMOD Conf. (2020). https://doi.org/10.1145/3318464.3386144
12. Fan, W.: Dependencies for graphs: Challenges and opportunities. ACM J. Data and Inf. Quality **11**(2), 1–12 (2019). https://doi.org/10.1145/3310230
13. Fan, W., Fan, Z., Tian, C., et al.: Keys for graphs. VLDB Endowment **8**(12), 1590–1601 (2015). https://doi.org/10.14778/2824032.282405
14. Fan, W., Geerts, F., Jia, X., et al.: Conditional functional dependencies for capturing data inconsistencies. ACM TODS **33**(2), 1–48 (2008). https://doi.org/10.1145/1366102.136610
15. Fan, W., Jin, R., Liu, M., et al.: Capturing associations in graphs. VLDB Endowment **13**(12) (2020). https://doi.org/10.14778/3407790.3407795
16. Fan, W., Lu, P.: Dependencies for graphs. In: ACM SIGMOD-SIGACT-SIGAI Symp. (2017). https://doi.org/10.1145/3034786.3056114
17. Fan, W., Lu, P.: Dependencies for graphs. ACM TODS **44**(2), 1–40 (2019). https://doi.org/10.1145/3287285

18. Fan, W., Wang, X., Wu, Y., et al.: Association rules with graph patterns. VLDB Endowment **8**(12), 1502–1513 (2015). https://doi.org/10.14778/2824032.2824048
19. Fan, W., Wu, Y., Xu, J.: Functional dependencies for graphs. In: ACM SIGMOD Conf. (2016). https://doi.org/10.1145/2882903.2915232
20. Francis, N., Gheerbrant, A., Guagliardo, P., et al.: A Researcher's Digest of GQL. In: ICDT Conf. vol. 255 (2023). https://doi.org/10.4230/LIPIcs.ICDT.2023.1
21. Francis, N., Green, A., Guagliardo, P., et al.: Cypher: An evolving query language for property graphs. In: ACM SIGMOD Conf. (2018). https://doi.org/10.1145/3183713.3190657
22. Kwashie, S., Liu, L., Liu, J., et al.: Certus: An effective entity resolution approach with graph differential dependencies (GDDs). VLDB Endowment **12**(6), 653–666 (2019). https://doi.org/10.14778/3311880.3311883
23. Liu, D., Kwashie, S., Zhang, Y., Zhou, G., et al.: An Efficient Approach for Discovering Graph Entity Dependencies (GEDs). CoRR **abs/2301.06264** (2023)
24. Pokornỳ, J., Valenta, M., Kovačič, J.: Integrity constraints in graph databases. Procedia Computer Science **109**, 975–981 (2017)
25. van Rest, O., Hong, S., Kim, J., et al.: PGQL: a property graph query language. In: GRADES (2016). https://doi.org/10.1145/2960414.2960421
26. Rodriguez, M.A.: The gremlin graph traversal machine and language (invited talk). In: DBPL (2015). https://doi.org/10.1145/2815072.2815073
27. Sequeda, J.F., Arenas, M., Miranker, D.P.: On directly mapping relational databases to RDF and OWL. In: WWW Conf. (2012). https://doi.org/10.1145/2187836.2187924
28. Shimomura, L.C., Fletcher, G., Yakovets, N.: GGDs: Graph Generating Dependencies. In: ACM CIKM (2020). https://doi.org/10.1145/3340531.3412149
29. Skavantzos, P., Leck, U., Zhao, K., et al.: Uniqueness constraints for object stores. ACM J. of Data and Inf. Quality (2023). https://doi.org/10.1145/3581758
30. Skavantzos, P., Link, S.: Normalizing Property Graphs. VLDB Endowment **16**(11), 3031–3043 (2023). https://doi.org/10.14778/3611479.3611506
31. Skavantzos, P., Zhao, K., Link, S.: Uniqueness constraints on property graphs. In: CAiSE Conf. pp. 280–295. Springer (2021)
32. Song, S., Chen, L.: Integrity Constraints on Rich Data Types. Springer Nature (2023)
33. Song, S., Gao, F., Huang, R., et al.: Data dependencies extended for variety and veracity: A family tree. IEEE TKDE **34**(10), 4717–4736 (2020). https://doi.org/10.1109/TKDE.2020.3046443
34. Thapa, R.B., Giese, M.: A source-to-target constraint rewriting for direct mapping. In: ICWS Conf. Springer (2021). https://doi.org/10.1007/978-3-030-88361-4_2
35. Thapa, R.B., Giese, M.: Mapping Relational Database Constraints to SHACL. In: ISWC Conf. Springer (2022). https://doi.org/10.1007/978-3-031-19433-7_13
36. Tian, Y.: The world of graph databases from an industry perspective. ACM SIGMOD Record **51**(4), 60–67 (2023). https://doi.org/10.1145/3582302.3582320
37. Zada, M.S.H., Yuan, B., Anjum, A., et al.: Large-scale Data Integration Using Graph Probabilistic Dependencies (GPDs). In: IEEE/ACM BDCAT Conf. (2020). https://doi.org/10.1109/BDCAT50828.2020.00028
38. Zheng, X., Dasgupta, S., Gupta, A.: P2KG: Declarative Construction and Quality Evaluation of Knowledge Graph from Polystores. In: ADBIS Conf. Springer (2023). https://doi.org/10.1007/978-3-031-42941-5_26
39. Zhou, G., Kwashie, S., Zhang, Y., et al.: FASTAGEDS: Fast Approximate Graph Entity Dependency Discovery. In: WISE Conf. Springer (2023). https://doi.org/10.1007/978-981-99-7254-8_35