Entity/Relationship Graphs

Principled Design, Modeling, and Data Integrity Management of Graph Databases

Author names surpressed due to double-blind reviewing

ABSTRACT

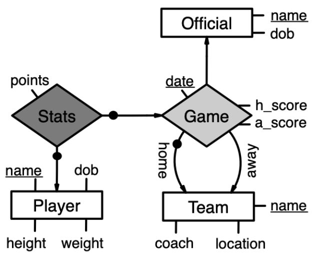
Chen's Entity/Relationship (E/R) framework is a lingua franca for well-designed databases. We define E/R graphs as property graphs that are instances of E/R diagrams. As the latter are a subclass of PG-Schema, E/R modeling constitutes a methodology for designing graph databases that guarantee data integrity, the absence of data redundancy and update anomalies. In addition, E/R graphs provide the first graph semantics for E/R diagrams. Further to the unification of conceptual and graph data modeling, referential integrity for E/R graphs can be managed by directed edges, called E/R links, between nodes. As a consequence, redundancy and sources of potential inconsistency can be eliminated, minimizing update maintenance. This is achieved by E/R keys that use properties and E/R links to enforce entity integrity, in contrast to property keys that rely exclusively on properties to enforce entity integrity. We use the TPC-H benchmark as running example and for extensive experiments that quantify the effort for i) managing entity integrity using property keys or E/R keys, ii) managing referential integrity using property redundancy or E/R links, iii) query evaluation. In summary, E/R diagrams form a principled core of PG-Schema for well-designed property graphs, while E/R keys constitute an efficient core of PG-Key for data integrity management.

ACM Reference Format:
Author names surpressed due to double-blind reviewing. 2025. Entity/Relationship Graphs: Principled Design, Modeling, and Data Integrity Management of Graph Databases. In *Proceedings of the 2025 International Conference on Management of Data (SIGMOD '25), June 22–27, 2025, Berlin, Germany.* ACM, New York, NY, USA, 14 pages.

1 INTRODUCTION

This work brings together classical Entity/Relationship (E/R) modeling [11] and modern graph databases [10], for their mutual benefit.

Graph databases are an exciting development in data management [27]. The property graph model [10] is emerging as a strong candidate for a standard, and proposals for property graph schemata (PG-Schema [2]) and keys (PG-Keys [3]) have been crafted. However, the uptake of graph databases remains inhibited by the lack of a principled design methodology. Hence, our first research question asks (Q1) What constitutes a principled design methodology for property graphs? This question is fundamental for efficient support of internal and external database workloads. Internally, well-designed schemata unlock efficient data integrity management, queries and updates [2, 23, 29]. Externally, such schemata enable data integration, exchange, and transformation [9, 20].



(a) E/R design

- TEAM={<u>name</u>, *coach*, *location*}
- OFFICIAL={<u>name</u>, *dob*}
- PLAYER={<u>name</u>, *dob*, *height*, *weight*}
- GAME={<u>hname</u>, *aname*, *oname*, <u>date</u>, *h_score*, *a_score*} with foreign keys
  - [*hname*] ⊆ TEAM[*name*]
  - [*aname*] ⊆ TEAM[*name*]
  - [*oname*] ⊆ OFFICIAL[*name*]
- STATS={<u>pname</u>, <u>hname</u>, <u>date</u>, *points*} with foreign keys
  - [*pname*] ⊆ PLAYER[*name*]
  - [*hname, date*] ⊆ GAME[*hname, date*]

(b) Matching relation schemata

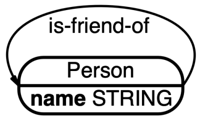
Figure 1: E/R diagram and relational database schema

Chen's Entity/Relationship (E/R) model [11] constitutes a best breed of conceptual data models. The model captures entities and their relationships in an easy-to-understand framework powerful enough to derive a formal data model. E/R models represent complex requirements for the target database visually [12]. Indeed, the graphical depiction of an E/R diagram is invaluable for effective communication between experts with different expertise. E/R modeling is a methodology for generating E/R diagrams that are well-designed as they guarantee data integrity and do not exhibit data redundancy or update anomalies in any instances [22, 23, 29].

Our first contribution is to show that E/R diagrams are both property graphs and a subclass of PG-Schema [2]. Hence, the E/R framework is available as a principled design methodology for property graphs that are instances of E/R diagrams.

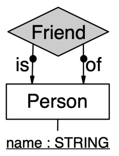
Fig. 1a depicts an E/R diagram for a basketball app, recording games between home and away teams that are refereed by an official, and the points players score. Entity types are visualized as rectangles with their attributes, relationship types are visualized as diamonds with their attributes and directed edges (called E/R links), pointing to each of its components, that is, those object types (entity or other relationship types) for which they define a relationship. The E/R key of an object type consists of its attributes that are underlined and its components with a dot on their E/R link. For example, the E/R key of GAME is {*home*:TEAM,*date*}. Fig. 1b shows matching relation schemata, with attributes of the key underlined.

The example illustrates how E/R diagrams can express relationships of every arity and order. In fact, ORDER, PLAYER, TEAM are entity types, which have order 0 (white color) as they do not link to any type. GAME is a ternary relationship type between the three entity types *home*:TEAM, *away*:TEAM, OFFICIAL, and is of order 1 (light grey) since it only links to entity types. STATS is a binary relationship type between PLAYER (order 0) and GAME (order 1). Hence, STATS has order 2 (dark grey). The example further illustrates how



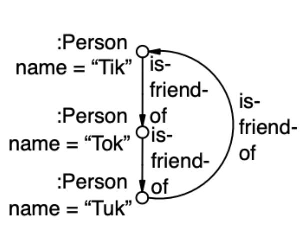
```
CREATE GRAPH TYPE Friends
STRICT {
(pType: Person {name STRING}),
(:pType) —[is-friend-of]—> (:pType),
FOR (x:pType) IDENTIFIER x.name
}
```

(a) PG-schema can be cyclic

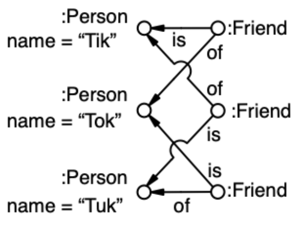


```
CREATE GRAPH TYPE Friends
STRICT {(pType: Person {name STRING}),
(fType: Friend),(:fType) —[is]—> (:pType),
(:fType) — [of] —> (:pType),
FOR (x:pType) IDENTIFIER x.name,
FOR (x:fType) IDENTIFIER y,z WITHIN
(x) —[is]—> (y:pType), (x) —[of]—> (z:pType)}
```

(b) E/R diagrams are acyclic



(c) Instance of PG-schema



(d) Instance of E/R diagram

Figure 2: Schemata and Instances for Friends Application

the E/R methodology leads to principled designs, which cause no data redundancy and update anomalies in any of their instances.

By design, E/R diagrams are directed acyclic graphs, a feature that guarantees effective and efficient maintenance of data integrity. Indeed, directed cycles may cause semantic issues and update inefficiency. As example, consider an app that records friend relationships. Its PG-schema in Fig. 2a defines a type Person with attribute *name* of domain STRING, with a directed edge labeled *is-friend-of* expressing that a person is a friend of a person, plus the PG-key expressing that the name identifies a person. With this schema, we can only insert people if there is some directed cycle of friends, see Fig. 2c. Semantically, this restriction may be unintended. Also, directed cycles make automated reasoning infeasible [24], already for keys and foreign keys [16]. In contrast, the E/R diagram with its matching PG-schema in Fig. 2b break the directed cycle: Friend is a binary relationship type of order 1 between entity types *is*:Person and *of*:Person. Indeed, the E/R diagram and PG-schema from Fig. 2b do not require instances with directed cycles. However, Fig. 2d illustrates how the friend cycle of Fig. 2c can be represented as a valid E/R instance. Moreover, inserts of a person are done independently of other people. Every friend relationship simply requires people who are friends, a referential integrity requirement.

Traditional database design translates conceptual to logical models [23, 29]. In contrast, our goal is to use E/R diagrams directly for graph databases. We are unaware of database systems that manage E/R diagrams and their instances directly. Hence, we ask (Q2) What can property graphs contribute to E/R modeling?

As central contribution we propose the concept of *E/R graphs*. Informally, E/R graphs are directed acyclic property graphs that are instances of E/R diagrams. Formally, they define homomorphisms on the E/R diagram, but the informal definition enables us to explain the far-reaching benefits of E/R graphs for now. Firstly, they provide E/R models with a graph semantics, meaning that E/R instances are directed acyclic property graphs. Hence, graph database systems like Neo4j can manage E/R instances directly. This addresses (Q2). Secondly, as E/R graphs are instances of E/R diagrams, they also constitute instances of well-designed property graphs. Hence, the E/R framework is a methodology for principled property graph design, which addresses (Q1). Together, this unifies E/R, logical and graph modeling.

As we propose E/R diagrams as a design core of PG-Schema, we also need to manage data integrity directly in property graphs. Hence, our third research question is (Q3) How can entity and referential integrity be managed capably for property graphs?

As third contribution, we show that E/R keys form an efficient core of PG-Key to manage entity and referential integrity. As exemplified by the E/R diagrams and their PG-schema in Fig. 2, E/R keys can be expressed by PG-keys. We propose three semantics of E/R graphs to facilitate efficient integrity maintenance and processing of workloads on them. Firstly, relational semantics duplicates the key properties of a component on the relationship type targeted, just like foreign key attributes duplicate key attributes, such as *h_name* on Game duplicates *name* on Team in Fig. 1b. We no longer need E/R links since referential integrity can be managed by comparing values between the duplicated properties. In this case, E/R keys reduce to property keys which only utilize properties of their object type. This semantics reflects the current bottleneck in referential integrity management, as updates need propagation to guarantee consistency. As a core novelty, we observe that E/R links serve as an alternative for implementing referential integrity. In contrast to comparing values across duplicated properties in E/R graphs, E/R links can simply use the identifiers of nodes they link to. That is as simple and natural as referential integrity can be, supports a core strength of property graphs in utilizing edges, and eliminates data redundancy from the graph. Hence, we call this graph semantics. Interestingly, even native graph database systems only support property keys but not E/R keys [25]. As a solution within current technology and minimizing the duplication of properties, we propose mixed semantics which uses E/R links whenever the component they link to does not feature in the E/R key, while we duplicate all key properties of a component otherwise. However, as E/R keys constitute the best solution for data integrity, we do promote their future support as an efficient core of PG-Key [3].

Our **contributions** and their impact are: (1) We propose E/R graphs as the first graph semantics for E/R models. Hence, graph databases can natively support E/R diagrams and their instances, E/R graphs. (2) We identify the E/R framework as a methodology for the principled design of property graphs. In particular, E/R diagrams form the core of PG-Schema that guarantees data integrity, the absence of data redundancy and update anomalies for all their E/R graph instances. (3) We establish E/R keys as an efficient core of PG-Key for managing the integrity of well-designed property graphs, facilitated by relational, mixed, and graph semantics. We also show that E/R keys can be reasoned about in linear time while reasoning about PG-Keys is undecidable. (4) Using TPC-H, we illustrate how well-designed relational databases can be implemented as graph databases using our different semantics, how well these facilitate update maintenance, and how well they process queries. This turns an inhibitor to graph database uptake into a future driver.

Our work advocates how two strong lines of database communities and their research can be brought together to advance best practice for data modeling and data integrity maintenance.

Organization. Sec. 2 summarizes the property graph model and discusses related work, while Sec. 3 provides concepts of E/R models we need for our work. Sec. 4 shows that E/R diagrams are property graphs and PG-schemata, defines E/R graphs, and shows how they unify conceptual, logical, and graph modeling, demonstrates the undecidability of PG-Keys, and the linear-time decidability of E/R keys. Sec. 5 proposes the three semantics for E/R graphs. Experiments are discussed in Sec. 6, quantifying how well data integrity maintenance and query processing are supported by E/R graphs. We conclude and comment on future work in Sec. 7. **An extended paper and all artifacts for our experiments are available in our GitHub repository.**

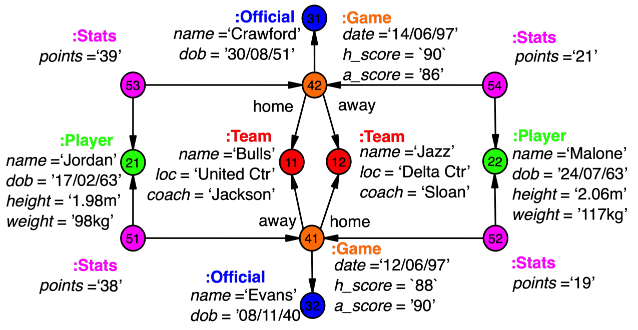


Figure 3: Property graph for basketball application

2 PRELIMINARIES AND RELATED WORK

We recall basics of well-designed relational databases, the property graph model, discuss proposals and related work for modeling property graphs, in particular with techniques from E/R modeling.

Database schema design proposes conditions on schemata that ensure good characteristics of their instances, including the absence of data redundancy, sources of inconsistency, udpate inefficiency, and to guarantee data integrity [22, 23]. We recall the normal form proposal for functional and inclusion dependencies, with the aim to transfer this proposal to property graphs.

A relation schema is a finite, non-empty set R of attributes A that have some domain$dom(A)$ of possible values. A database schema is a finite, non-empty set $\mathcal{S}$ of relation schemata. A tuple t or record over R maps every attribute $A \in R$ to a value $t(A) \in dom(A)$ of its domain. A relation over R is a finite set of tuples over R. A relational database (or instance) $\mathcal{I}(\mathcal{S})$ over $\mathcal{S}$ maps every relation schema $R \in \mathcal{S}$ to a relation $\mathcal{I}(R)$. A functional dependency (FD) is an expression $R : X \longrightarrow Y$ with attribute sets $X, Y \subseteq R$. A relation $\mathcal{I}(R)$ satisfies the FD $R : X \longrightarrow Y$ if every pair of tuples $t, t' \in \mathcal{I}(R)$ with values matching on all attributes in X ($t(X) = t'(X)$) has also values matching on all attributes in Y ($t(Y) = t'(Y)$). An FD $R : X \longrightarrow Y$ is trivial whenever $Y \subseteq X$ holds. An inclusion dependency (IND) over $\mathcal{S}$ is an expression $R[X] \subseteq S[Y]$ where $R, S \in \mathcal{S}$, X and Y denote attribute sequences of equal length over R and S, respectively, such that for attribute pairs $A \in R$ and $B \in S$ in the same position of X and Y, $dom(A) = dom(B)$. The IND $R[X] \subseteq S[Y]$ is satisfied by an instance $\mathcal{I}(\mathcal{S})$ iff for every tuple $t \in \mathcal{I}(R)$ there is some tuple $t' \in \mathcal{I}(S)$ such that $t(X) = t'(Y)$. The IND $R[X] \subseteq S[Y]$ is trivial whenever $S = R$ and $Y = X$. The implication problem for FDs and INDs is to decide whether for every given database schema $\mathcal{S}$ with set $\Sigma \cup \{\varphi\}$ of FDs and INDs over $\mathcal{S}$, Σ implies φ, that is, if every database instance over $\mathcal{S}$ that satisfies all elements of Σ also satisfies φ. We use Σ^+ to denote the set of FDs and INDs implied by Σ.

For $(\mathcal{S}, \Sigma)$, call an FD $R : X \longrightarrow R \in \Sigma^+$ a super-key dependency and $R : X$ a super-key, as every pair of distinct tuples in any relation over R must not have matching values on all attributes in X. A super-key $R : X$ is a key if there is no proper subset Y of X such that $R : Y$ is also a super-key. An IND $R[X] \subseteq S[Y]$ is a foreign key if $S : Y$ is a key. The IND-graph of $(\mathcal{S}, \Sigma)$ is a directed graph with vertex set $\mathcal{S}$ and directed edge from R to S if there is some non-trivial IND $R[X] \subseteq S[Y] \in \Sigma$. $(\mathcal{S}, \Sigma)$ is *acyclic* if there is no directed cycle in its IND-graph, and cyclic otherwise. Essentially, acyclicity prevents the case that tuples in a relation r necessitate the existence of other tuples in r, which may lead to semantic issues and update problems. $(\mathcal{S}, \Sigma)$ is *key-based* if every IND $R[X] \subseteq S[Y]$ is a foreign key.

$(\mathcal{S}, \Sigma)$ is in Boyce-Codd Normal Form (BCNF) iff for every $R \in \mathcal{S}$, for every FD $R : X \longrightarrow Y \in \Sigma$, the FD is trivial or $R : X$ is a super-key. $(\mathcal{S}, \Sigma)$ is in Inclusion Dependency Normal Form (IDNF) iff $(\mathcal{S}, \Sigma)$ is in BCNF, acyclic and key-based.

Relational database schemata in IDNF are well-designed due to strong technical arguments [22, 23], realizing the good characteristics mentioned previously. While the implication problem for the general class of FDs and INDs is undecidable [24], the conditions of IDNF ensure no interaction between FDs and acyclic INDs, which means that their implication problem becomes decidable, NP-complete [13] and fixed-parameter tractable in the arity of INDs [19]. Translating E/R diagrams into relation schemata ensures IDNF, and every schema in IDNF can be translated into an E/R diagram [22, 23]. In Fig. 1, the E/R diagram and relational database schema translate into one another, and the latter is in IDNF.

The **property graph model** [10] uses disjoint sets: $\mathcal{O}$ for objects, $\mathcal{L}$ for labels, $\mathcal{K}$ for properties, and $\mathcal{N}$ for values. A *property graph* is a quintuple $G = (V, E, \eta, \lambda, \nu)$ where $V \subseteq \mathcal{O}$ is a finite set of *vertices*, $E \subseteq \mathcal{O}$ is a finite set of *edges*, $\eta : E \longrightarrow V \times V$ is a function mapping edges to ordered pairs of vertices, $\lambda : V \cup E \longrightarrow \mathcal{P}(\mathcal{L})$ is a function assigning objects finite sets of labels, and $\nu : (V \cup E) \times \mathcal{K} \longrightarrow \mathcal{N}$ is a partial function assigning values for properties to objects. The set of domain values where ν is defined is finite. If $\nu(o, A)$ is defined, we write $\nu(o, A) =\downarrow$, and $\uparrow$ otherwise. Fig. 3 shows a property graph for our basketball application.

PG-Schema [2] is a proposal by academics and practitioners to help standardizing schema support for graph databases. PG-Schema aims at meeting principled design requirements of property graph management, and its authors "deliberately target minimal data modeling capabilities and as a reference point ... take the most basic variant of Entity-Relationship (ER) diagrams ... as the ultimate lower bound in the expressiveness of conceptual modelling languages" [2]. We will formally show that every E/R diagram can be specified as a PG-Schema. They are a great lower bound for PG-Schema as they capture well-designed property graphs. Fig. 2 shows examples of PG-Schema definitions.

PG-Key [2, 3] is a powerful framework for defining key constraints, designed by industry and academia. A PG-Key is an expression `FOR` $p(x)$ `qualifier` $q(x,\overline{y})$ with queries $p(x)$ and $q(x,\overline{y})$. The *scope* $p(x)$ defines all possible target objects that need to be identified, and the *descriptor* $q(x,\overline{y})$ selects the key value that identifies them. Objects can be nodes, relationships or properties. A `qualifier` combines keywords `EXCLUSIVE` where no two targets can share the same key value, `MANDATORY` where each target has at least one key value, `SINGLETON` where each target as at most one key value, and `ID` as shorthand for `EXCLUSIVE`, `MANDATORY` and `SINGLETON`. As example for Fig. 1a, we express that "every game between a home and away team that is refereed by some official is identified by the date and home team" as PG-key: `FOR` g `WITHIN` (g:Game) − [home] → (t:Team), (g:Game) − [away] → (:Team), (g:Game) − [ ] → (:Official) `ID` g.date, t .

We show that E/R keys form a core of PG-Key that handles integrity maintenance for well-designed property graphs efficiently, PG-Keys can express E/R keys, but while implication for PG-Keys is undecidable, that of E/R keys is decidable in linear time.

Conceptual languages, such as UML and E/R models, have been translated into graph databases [14, 15, 26, 30]. Designing general property graphs, not limited to acyclicity, has been studied by mapping entities to nodes and relationships to edges [7]. Mapping relationships to edges limits usability. Firstly, edges restrict relationships to binary ones, and having to translate non-binary relationships to edges creates overheads, and is only possible by maintaining expensive join constraints that preserve equivalence [29]. Secondly, mapping relationships to edges means it is no longer possible to reference the relationships. As example, call a sequence of n foreign keys $R_1[X_1] \subseteq R_2[X_2], R_2[X_2'] \subseteq R_3[X_3], \ldots, R_n[X_n'] \subseteq R_{n+1}[X_{n+1}]$ with distinct relation schemata $R_1, \ldots, R_n$ a chain of order n. For example, Fig. 1b has the following foreign key chain of order 2: Stats[*h_name*, *date*] ⊆ Game[*h_name*, *date*] and *Game*[*h_name*] ⊆ Team[*name*]. Chains of foreign keys beyond order one cannot be expressed when relationships are modeled as edges. Indeed, there is no need for restricting expressiveness to binary relationship types of order 1. Hence, we use E/R models that can express objects types of arbitrary arity and order. That means we will also map entities to nodes but we will map relationships to nodes instead of edges, reserving edges (E/R links) for referential integrity management. In Fig. 1a, relationship type Game is a component of relationship type Stats, using an E/R link to define complex relationship types from simpler ones. This is particularly useful for data integration when objects are linked by edges. Our ideas follow best modeling practice [8, 21, 29] including XML [1].

Other work. The value of acyclicity is long known [8], and recent efforts merge graph approaches on the basis of acyclicity [18]. The range of graph data models is large [4, 10]. There is extensive work on graph constraints [10, 17] and early work on graph normalization [28]. Recent work on using E/R models for graph summarization [6] and PG-schema inference [5] from semi-structured data is restricted to binary relationship types of order one.

Summary. While broad and deep, no work on graph modeling has used E/R diagrams directly as models for well-designed property graphs, proposed directed acyclic property graphs as a graph semantics for E/R models, nor investigated how to manage entity and referential integrity by property graph models.

3 FUNDAMENTALS OF E/R MODELING

We summarize the syntax and semantics of Entity/Relationship modeling [29], emphasizing their ability to express object types of arbitrary arity and order.

Entity types, Entities and Entity Sets. Entities form the atomic units of a data model, and entities of the same type are modeled by the same entity type, that is, entity types consist of a finite set of attributes (properties). In addition, any entity of any type needs to be uniquely identifiable by a subset of these attributes. Formally, an *entity type* E consists of a finite, non-empty set $attr(E)$ of attributes and a non-empty subset $id(E) \subseteq attr(E)$, called the *key* of E and every element $A \in id(E)$ is called a key attribute of E. Every attribute A has a domain $dom(A)$, which is a non-empty set of possible values we can assign to any entity on that attribute.

An *entity* e of of type E assigns a value $e(A) \in dom(A)$ to every attribute of E. That is, $e : attr(E) \rightarrow \bigcup_{A \in attr(E)} dom(A)$ such that for all $A \in attr(E)$, $e(A) \in dom(A)$. The collection $ent(E)$ of entities over entity type E is the Cartesian product over the domains assigned to attributes of E. Formally, $ent(E) = \prod_{A \in attr(E)} dom(A)$.

An *entity set* of type E is a finite set $E^t \subseteq ent(E)$ where different entities have different values on some key attribute. That is, for all $e, e' \in E^t$ where $e \neq e'$ there is some $A \in id(E)$ such that $e(A) \neq e'(A)$. Hence, values on key attributes identify entities uniquely.

The TPC-H schema has two entity types: Region=({*regionkey*, *name*, *comment*}, {*regionkey*}) and Part=({*partkey*, *name*, *mfgr*, *brand*, *type*, *size*, *container*, *retailprice*, *comment*},{*partkey*}).

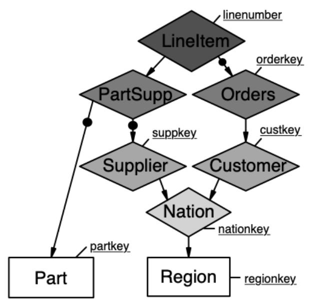
Relationship types, Relationships, and Relationship Sets. Relationships are (complex) aggregations between (less complex) relationships and entities. Based on problems illustrated in the introduction, no directed cycles can exist between object types.

For every positive integer k, a *relationship type* R *of order* k consists of (i) a finite set $comp(R)$ of relationship types, called *components* of R, such that if $k = 0$, then $comp(R) = \emptyset$, and if $k > 0$, then every $R' \in comp(R)$ is of order smaller than k and there is some $R' \in comp(R)$ of order $k - 1$; (ii) a finite set $attr(R)$ of attributes such that every $A \in attr(R)$ has a domain $dom(A)$; and (iii) a non-empty subset $id(R) \subseteq comp(R) \cup attr(R)$, called the *key* of R such that the components in $id(R) \cap comp(R)$ are called *key components* and the attributes in $id(R) \cap attr(R)$ are called *key attributes* of R.

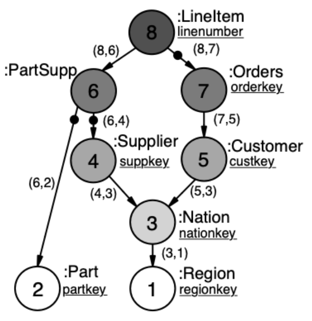
Hence, entity types are relationship types of order 0 which have a non-empty set of attributes but no component. Relationship types of order $k > 0$ do not need to have any attribute but they will have some relationship type of order $k - 1$ as a component.

If the order of relationship type R is 0, the relationships of R are the entities of R, that is, $rel(R) := ent(R)$. Otherwise, the order $k > 0$ of R is a positive integer, and the set $rel(R)$ comprises all relationships r of order k where every $R' \in comp(R)$ is mapped to a relationship $r(R') \in rel(R')$ of the order of R'; and every $A \in attr(R)$ is mapped to a domain value $r(A) \in dom(A)$. In summary, we may also write $rel(R) = \prod_{R' \in comp(R)} rel(R') \times \prod_{A \in attr(R)} dom(A)$. Collections of relationships over a relationship type R of order k are well defined, as all components of R have an order strictly less than k and relationships of order 0 are entities.

Relationship sets must satisfy the unique key property, that is, a *relationship set* over relationship type R is a finite set $R^t \subseteq rel(R)$ such that for all $r, r' \in R^t$ such that $r \neq r'$, $r(id(R)) \neq r'(id(R))$.



(a) E/R Diagram



(b) Its property graph

Figure 4: E/R diagram and property graph of TPC-H, limited to key properties and components

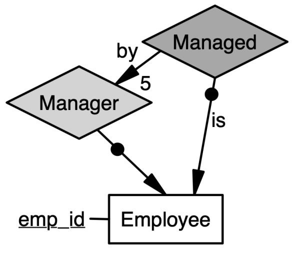
When roles of components need highlighting we permit role labels l as in $l : R' \in comp(R)$, such as a *home* : Team playing an *away* : Team, with labels *home* and *away* that distinguish teams.

Key attributes motivate the use of a foreign key semantics, in which for relationships r of order $k > 0$, we only use $rel(R) := \prod_{R' \in id(R) \cap comp(R)} rel(R') \times \prod_{A \in id(R) \cap attr(R)} dom(A)$ to define relationships and relationship sets of order k. Indeed, this semantics minimizes the redundant duplication of attributes from components. Similar to the use of *object type* we use the term *object* (*object set*) as reference to relationship (*relationship set*) or entities (*entity set*) without specifying which.

For the TPC-H schema, we have the following relationship types and their orders (we ignore domains of attributes, and only list key attributes): Nation=({Region}, {*nationkey*, ...}, {*nationkey*}) of order 1, Supplier=({Nation}, {*suppkey*, ...}, {*suppkey*}) of order 2, Customer=({Nation}, {*custkey*, ... }, {*custkey*}) of order 2, PartSupp=({Part, Supplier}, {...}, {Part, Supplier}) of order 3, Orders=({Customer}, {*orderkey*, ...}, {*orderkey*}) of order 3, Lineitem=({PartSupp, Orders}, {*linenumber*, ...}, {Orders, *linenumber*}) of order 4. Interestingly, the keys of these relationship types are diverse. Some contain only attributes, such as Nation, Supplier, Customer, and Orders, some contain only components, such as PartSupp, and some contain components and attributes, such as Lineitem. This diversity is representative of best practice.

E/R Schemata and E/R Instances. Referential integrity is deeply embodied in E/R Schemata. An *Entity/Relationship Schema* (E/R Schema) is a non-empty, finite set $\mathcal{S}$ of object types such that for all relationship types $R \in \mathcal{S}$, all (labeled) components $R'(l : R') \in comp(R)$ of R, are also part of the schema, that is, $R'(l : R') \in \mathcal{S}$. Relationship types need all their components.
The set $\mathcal{S}$ of object types we previously defined for the TPC-H schema form indeed an E/R Schema. For instance, for Lineitem $\in \mathcal{S}$ and PartSupp $\in comp$(Lineitem) we do have PartSupp $\in \mathcal{S}$.

These ideas on the schema level transcend naturally to the instance level. An *Entity/Relationship instance* (E/R instance) $\mathcal{I}$ over E/R Schema $\mathcal{S}$ assigns to each entity type $E \in \mathcal{S}$, an entity set $\mathcal{I}(E)$ and to each relationship type $R \in \mathcal{S}$, a relationship set $\mathcal{I}(R)$ such that for each relationship $r \in \mathcal{I}(R)$ and for each object type $O \in comp(R)$, $r(O)$ is an object in $\mathcal{I}(O)$. Indeed, higher-order objects cannot exist without the lower-order objects they depend on.



(a) E/R Diagram

- Employee={$\underline{emp\_id}$}
- Manager={$\underline{emp\_id}$}
  and foreign key
  – [emp_id] ⊆ Employee[emp_id]
- Managed={m_id,$\underline{emp\_id}$}
  with foreign keys
  – [m_id] ⊆ Manager[emp_id]
  – [emp_id] ⊆ Employee[emp_id]

(b) Relation Schemata

Figure 5: Example schema Manage

E/R Diagrams are intuitive visualizations that facilitate communication about the target database. We recall a standard definition of E/R diagrams [29]. An *E/R Diagram* of E/R Schema $\mathcal{S}$ is a directed acyclic graph $\mathcal{G} = (V_G, E_G)$ such that the set V_G of vertices contains the object types from the schema, $V_G := \mathcal{S}$, and the set E_G of directed edges, called *E/R links*, represents the component relationships between object types, $E_G := \{(R, R') \mid R' \in comp(R)\} \cup \{(R, R')$ with edge-label $l \mid l : R' \in comp(R)\}$. In particular, the order of object type $O \in \mathcal{S}$ is the maximum length across all directed paths from vertex $O \in V_G$ to any leaf in V_G. We have explained how E/R diagrams are visualized in the introduction. Fig. 1a shows the E/R diagram for our basketball example. Fig. 4a shows the E/R diagram of the TPC-H schema where all attributes that do not belong to any key have been omitted. Orders of object types are represented by different shades of gray, starting with white for order 0 up to the darkest gray for order 4.

Other E/R concepts, such as weak entity types, specializations, and cardinality constraints can be captured by the E/R model presented. We consider the example Manage in Fig. 5 where every manager is an employee and employees have managers. Specializations are sub-classes, which can be represented as unary relationship types $R = (\{R'\}, attr(R), \{R'\})$ whose key only consists of the singleton component $\{R'\}$. In our example, every manager is an employee, and as employees are identified by their *emp_id*, so are managers. Likewise, Manager is an example of a *weak entity type* whose objects cannot be identified without other object types. In our example, Managers can only be identified by their component Employee. Cardinality constraints stipulate conditions on the number of relationships in which relationships of lower order can participate. In our example, employees can have at most one manager, which we may declare by *card*(Managed, *is* : Employee) ≤ 1. However, this is equivalent to the E/R key *id*(Managed)={*is*:Employee}. As *id*:Manager is not a key component of Managed, there is no upper bound on how many employees a manager can manage. If this number should be restricted, say to five, we may stipulate *card*(Managed, *by* : Manager) ≤ 5. For a detailed treatment of E/R concepts and constraints, we refer to a standard textbook [29].

4 E/R GRAPHS, E/R KEYS, AND E/R LINKS

We present our main concepts: E/R graphs as well-designed property graphs and as graph semantics for E/R models, E/R keys as core fragment of PG-Key for handling object integrity in well-designed property graphs, and E/R links as alternative to foreign keys for managing referential integrity. In Sec. 4.1, we show that E/R diagrams are both directed acyclic property graphs and a sub-class of PG-Schema. Hence, we call them E/R graph models. Sec. 4.2 defines E/R graphs as instances of E/R graph models, specifying a homomorphism from graph to graph model. This also provides a graph semantics for E/R diagrams. We discuss how E/R graphs unify conceptual, logical and graph modeling in Sec. 4.3. Sec. 4.4 and 4.5 identify E/R keys as a core of PG-Key that efficiently supports graph data modeling and integrity maintenance. While implication of PG-Keys is shown to be undecidable, that of E/R keys can be decided in time linear. Finally, Sec. 4.6 discusses how E/R links can maintain referential integrity in property graphs.

4.1 E/R Diagrams as E/R Graph Models

Essentially, E/R diagrams are non-attributed, acyclic property graphs. For an E/R diagram $\mathcal{D} = (V, E)$, the set O_D of object ids for $\mathcal{D}$ introduces a unique id i_o for each vertex in V (that is, each object type $O \in \mathcal{S}$), and a unique id $i_{(o,o')}$ for each edge in E. That is, $O_D = O_V \cup O_E$ where O_V (O_E) consists of unique ids for each element of V (E, respectively).

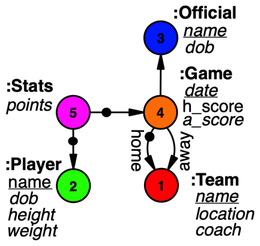
The set $\mathcal{L}_D$ of labels for $\mathcal{D}$ comprises the names O of object types in V, edge labels in $\mathcal{D}$ (resulting from labels l in $l : O' \in comp(O)$), and a distinguished label $\bullet$ for dots on edges. The set $\mathcal{K}_D$ of properties for $\mathcal{D}$ consists of the attributes in $attr(O)$ of all object types O in V. The set $\mathcal{N}_D$ of values for $\mathcal{D}$ is the set all domains $dom(A)$ for any attribute A of any object type in $\mathcal{D}$. In keeping values of non-key attributes optional, we distinguish between $dom(A)$ and $dom_\perp(A)$. The latter contains the distinguished null marker symbol $\perp$. As a principle of entity integrity and in line with primary keys, values for key attributes are mandatory, so the attribute is `NOT NULL`.

For O_D, $\mathcal{L}_D$, $\mathcal{K}_D$, and $\mathcal{N}_D$, the *E/R graph model* $\mathcal{G}_D = (V_D, E_D, \eta_D, \lambda_D, \nu_D)$ of E/R diagram $\mathcal{D} = (V, E)$ is defined by:

(1) $V_D := O_V$ and $E_D := O_E$,
(2) $\eta_D : E_D \to V_D \times V_D$ is defined by $\eta_D(i_{o,o'}) = (i_o, i_{o'})$ where i_o is the unique identifier for node $O \in V$ and $i_{o'}$ is the unique identifier for node $O' \in V$ for every $(O, O') \in E$,
(3) $\lambda_D : V_D \cup E_D \to \mathcal{P}(\mathcal{L})$ is defined by $\lambda_D(i_o) = \{O\}$ for $O \in V$ with unique identifier $i_o \in O_V$, $l \in \lambda_D(i_{(o,o')})$ for $(O, O') \in E$ with edge label l and unique identifier $i_{(o,o')} \in O_E$, and $\bullet \in \lambda_D(i_{(o,o')})$ for $(O, O') \in E$ with key component $O' \in comp(O) \cap id(O)$ and unique identifier $i_{(o,o')} \in O_E$, and
(4) $\nu_D : (V_D \cup E_D) \times \mathcal{K}_D \to \mathcal{N}_D$ is defined by $\nu(i_o, A) = dom_\perp(A)$ for all $A \in attr(O) - id(O)$ and $O \in V$, and $\nu(i_o, A) = dom(A)$ for all key attributes $A \in attr(O) \cap id(O)$ and $O \in V$.

Key components are denoted as in E/R diagrams, and key attributes become key properties with the null-marker free domain $dom(A)$. We underline key properties $\underline{A}$ instead of writing $A = dom(A)$, and write A for non-key attribute A instead of $A = dom_\perp(A)$. Examples include id(LINEITEM) = {ORDERS, *linenumber*}, id(PARTSUPP) = {SUPPLIER, PART}, and id(ORDERS) = {*orderkey*}. Hence, domain constraints of E/R diagrams are encoded as property-value pairs. Indeed, E/R graph models are property graphs.

Our definition converts E/R diagrams $\mathcal{D}$ into E/R graph models $\mathcal{G}_D$. As E/R diagrams $\mathcal{D}$ can be obtained from their E/R graph model $\mathcal{G}_D$, E/R schemata, diagrams, and graph models are isomorphic.



(a) E/R Diagram

- $V_D := \{i_1, i_2, i_3, i_4, i_5\}$,
- $E_D := \{i_{(4,1,\text{home})}, i_{(4,1,\text{away})}, i_{(4,3)}, i_{(5,2)}, i_{(5,4)}\}$,
- $\eta_D : E_D \to V_D \times V_D$:
  $i_{(4,1,\text{home})} \mapsto (i_4, i_1)$, $i_{(4,1,\text{away})} \mapsto (i_4, i_1)$,
  $i_{(4,3)} \mapsto (i_4, i_3)$, $i_{(5,2)} \mapsto (i_5, i_2)$, $i_{(5,4)} \mapsto (i_5, i_4)$
- $\lambda_D : V_D \cup E_D \to \mathcal{P}(\mathcal{L})$ (selection):
  $i_4 \mapsto \{\text{GAME}\}$, $i_{(4,1,\text{home})} \mapsto \{\bullet, \text{home}\}$,
  $i_{(4,1,\text{away})} \mapsto \{\text{away}\}$, $i_{(5,2)} \mapsto \{\bullet\}$, and
- $\nu_D : (V_D \cup E_D) \times \mathcal{K}_D \to \mathcal{N}_D$ (selection):
  $(i_1, \text{name}) \mapsto$ `String`, $(i_1, \text{location}) \mapsto$ $\texttt{String}_\perp$,
  $(i_2, \text{name}) \mapsto$ `String`, $(i_2, \text{dob}) \mapsto$ $\texttt{Date}_\perp$,
  $(i_4, \text{date}) \mapsto$ `Date`, $(i_5, \text{points}) \mapsto$ `Int`.

(b) Formal Definition

Figure 6: E/R diagram from Fig. 1a as property graph

Fig. 6 shows a visualization and formal definition of the E/R diagram from Fig. 1a as an E/R graph model.

E/R diagrams form also a sub-class of PG-Schema. We show this by converting object types $O = (comp(O), attr(O), id(O))$ with

- $comp(O) = \{\ell_1{:}O_1, \ldots, \ell_k{:}O_k\}$
- $attr(O) = \{A_1, \ldots, A_l\}$, and
- $id(O) = \{O_{i_1}, \ldots, O_{i_m}, A_{j_1}, \ldots, A_{j_n}\}$

into PG-Schema type definitions consisting of a node type (O-$Type$: $O\{A_1\ dom(A_1), \ldots, A_l\ dom(A_l)\}$), for $i = 1, \ldots, k$ we add the E/R link as edge type (:O-$Type$) $-$ [: ℓ_i] $\to$ (:O_i-$Type$), and PG-key: `FOR` (x : O-$Type$) `ID` $y_{i_1}, \ldots, y_{i_m}, x.A_{j_1}, \ldots, x.A_{j_n}$ `WITHIN` $(x) - [{:}\ell_{i_1}] \to (y_{i_1}{:}O_{i_1}\text{-}Type), \ldots, (x) - [{:}\ell_{i_m}] \to (y_{i_m}{:}O_{i_m}\text{-}Type)$. The translation of entity types is the special case of $comp(O) = \emptyset$. Fig. 2b shows an E/R diagram and matching PG-Schema definition.

4.2 E/R graphs as Homomorphic Instances

We will provide a technical definition of E/R graphs as instances of E/R graph models. Indeed, we show that E/R graphs are property graphs which map homomorphically to E/R graph models. Hence, they are natural instances of E/R graph models.

As illustration, the E/R graph from Fig. 3 forms an instance of the E/R graph model in Fig. 6. The color-coding visualizes the homomorphism that maps each node of a given color from the E/R graph to the node of the same color in the E/R graph model. We now formalize the features of such homomorphisms.

We use $\mathcal{D} = (V, E)$ to denote an E/R diagram for E/R schema $\mathcal{S}$, O_G as set of object identifiers, $\mathcal{L}_G \subseteq \mathcal{L}_D - \{\bullet\}$ as set of labels, $\mathcal{K}_G \subseteq \mathcal{K}_D$ as set of properties, $\mathcal{N}_G \subseteq \bigcup_{A \in attr(O), O \in \mathcal{S}} dom(A)$ as set of values, and $\mathcal{G}_D$ to mark an E/R graph model of $\mathcal{D}$.

A property graph $G = (V_G, E_G, \eta_G, \lambda_G, \nu_G)$ over O_G, $\mathcal{L}_G$, $\mathcal{K}_G$, and $\mathcal{N}_G$ is called an *E/R graph* for $\mathcal{G}_D$ if and only if there is some function $h : O_G \to O_D$ that satisfies the following conditions:

(1) $h(V_G) \subseteq V_D$ and $h(E_G) \subseteq E_D$,
(2) for all $o \in E_G$, if $\eta_G(o) = (o_1, o_2)$, then $h(\eta_G(o)) = (h(o_1), h(o_2)) = \eta_D(h(o))$ (edge-preserving),
(3) for all $o \in V_G$, $\lambda_G(o) = \lambda_D(h(o))$ (node label-preserving),
(4) for all $o \in E_G$, $l \neq \bullet$, $\lambda_G(o) = \{l\}$ if and only if $l \in \lambda_D(h(o))$ (role label-preserving),
(5) for all $o, o' \in E_G$, if $\eta_G(o) = (u, v), \eta_G(o') = (u, w) \in E_G$ and $\lambda_G(v) = \lambda_G(w)$ and $\lambda_G(o) = \lambda_G(o')$, then $v = w$ (single-valued components, that is, equal target labels and role labels mean equal target nodes),

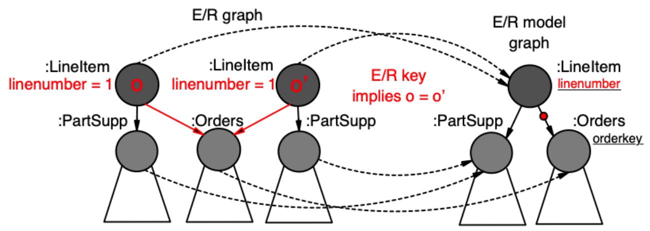


Figure 7: Illustration of E/R key on TPC-H Example

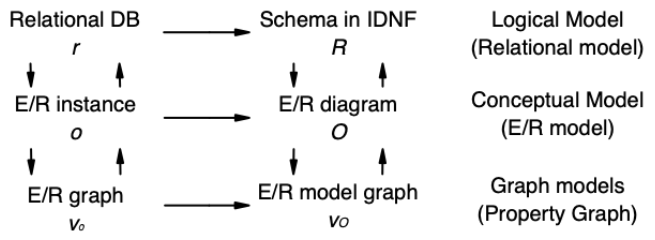

Relational DB
r
Schema in IDNF
R
Logical Model (Relational model)
E/R instance
o
E/R diagram
O
Conceptual Model (E/R model)
E/R graph
v_o
E/R model graph
v_O
Graph models (Property Graph)

Figure 8: Unifying conceptual, logical and graph modeling

(6) for all $v \in V_G, p \in \mathcal{K}_G$, if $\nu_G(v,p) = val$, then $val \in \nu_D(h(v),p)$ (type-preserving),

(7) for all $v \in V_G, p \in \mathcal{K}_D$, if $\nu_D(h(v),p) = dom(p)$, then there is some $val \in dom(p)$ such that $\nu_G(v,p) = val$ (key properties must exist), and

(8) for all $v \in V_G, w' \in V_D$, if $(h(v),w') \in E_D$, then there is some $w \in V_G$ such that $(v,w) \in E_G$ and $h(w) = w'$ (referential integrity).

Rule (7) does not enforce a domain requirement whenever $\nu_D(h(v),p) = dom_\perp(p)$, as $\perp$ indicates that a value for this property is optional. The order of a node $v \in V_G$ is the order of its image $h(v) \in V_D$. E/R graphs inherit directed acyclicity and referential integrity from their models (property 8). For entity integrity, we specify *E/R keys* for the property graph model $\mathcal{G}_D$ of an E/R diagram $\mathcal{D}$ as expressions $O(C,K)$ with $C = \{O_1,\ldots,O_n\} \subseteq comp(O)$ and $K = \{K_1,\ldots,K_m\} \subseteq attr(O)$, for any object type $O \in V_D$. An *E/R graph G satisfies* $O(C,K)$ iff for all $o,o' \in V_G$ such that $(o,o_i),(o',o_i) \in E_G$ for $i = 1,\ldots,n$, and $\downarrow = \nu_G(o,K_j) = \nu_G(o',K_j) = \downarrow$ for $j = 1,\ldots,m$, and $\lambda_G(o) = O = \lambda_G(o')$ and $\lambda_G(o_i) = O_i$ for $i = 1,\ldots,n$, then $o = o'$. For G to be an E/R graph for $\mathcal{G}_D$ we require that the function h also satisfies condition (9) for all $O \in V_D$ such that $id(O) = C \cup K$ with $C = \{O_1,\ldots,O_n\}$ and $K = \{K_1,\ldots,K_m\}$, G satisfies the E/R key $O(C,K)$ (entity integrity). Fig. 7 illustrates the semantics of E/R key :LINEITEM(:ORDERS, *linenumber*). The two vertices o and o' have values matching on property *linenumber*, and each has an E/R link to an ORDERS node. If o and o' were different vertices, the E/R graph would violate the E/R key. Hence, $o = o'$.

As SQL `UNIQUE`, E/R keys $O(C,K)$ do not stipulate node uniqueness if any property in K is undefined. Entity integrity requires some E/R key for each object type, ensuring objects can be identified and accessed efficiently, as primary keys in relational databases. All E/R keys that express integrity rules ought to be declared.

4.3 Graphical Core Unifying Data Modeling

As mentioned before, the E/R methodology constitutes principled data modeling and best practice. Fig. 8 illustrates that these characteristics transcend to the logical model, in particular relational database schemata in IDNF and their instances. Our concepts of E/R graphs and their models further bridge logical and conceptual modeling with graph modeling using property graphs. As we have shown, E/R diagrams are property graphs and PG-schemata. Furthermore, E/R instances can be mapped into E/R graphs, and vice versa. The mapping can be defined recursively using the order of objects and vertices. Entities e with value $e(A) = val$ on attribute A are mapped to leaf vertices v_e with $\nu(v_e,A) = val$, and relationships r of order $k+1$ with relationship $r' = r(C)$ of order less than $k+1$ on component C and value $r'(A) = val$ on attribute A are mapped to vertices v_r of order $k+1$ with directed edges $(v_r,v_{r'})$ and $\nu(v_r,A) = val$. Within any set of objects, key attributes and components ensure uniqueness, and this transcends directly to E/R graphs. The translation from E/R graphs to E/R instances is similar.

Since logical schemata and their instances translate into E/R diagrams and their instances, respectively, conceptual, logical, and graph modeling have a common core of well-designed databases.

4.4 Undecidability of PG-Keys

We establish undecidability for the implication problem of PG-Key [3] over property graphs that originate from relational translations. Hence, the expressiveness of PG-Keys leaves reasoning about them infeasible, already on property graphs without edges.

First, a *relational translation* maps a set $\Sigma_\mathcal{S}$ of keys and foreign keys over relational database schema $\mathcal{S}$ to a set $\Sigma_{\mathcal{G}_\mathcal{S}} = \{\sigma_{\mathcal{G}_\mathcal{S}} \mid \sigma \in \Sigma_\mathcal{S}\}$ of PG-keys over property graph $\mathcal{G}_\mathcal{S}$ as follows:

- every relation schema $R \in \mathcal{S}$ to a vertex v_R with label $:R$,
- every attribute $A \in R$ to a property A on the node v_R,
- every key $\sigma = \{A_1,\ldots,A_n\}$ over R to PG-Key $\sigma_{\mathcal{G}_\mathcal{S}}$ over v_R, defined by `FOR` $x : R$ `ID` $x.A_1,\ldots,x.A_n$
- every foreign key $\sigma = R[A_1,\ldots,A_n] \subseteq S[B_1,\ldots,B_n]$ to a PG-Key $\sigma_{\mathcal{G}_\mathcal{S}}$ over v_R, defined by `FOR` $x : R$ `MANDATORY` y `WITHIN` $(x),(y : S)$ `WHERE` $x.A_1 = y.B_1,\ldots,x.A_n = y.B_n$.

Instances $\mathcal{I}(\mathcal{S})$ over $\mathcal{S}$ translate into instances $\mathcal{I}(\mathcal{G}_\mathcal{S})$ over $\mathcal{G}_\mathcal{S}$ by mapping every tuple t of every relation $\mathcal{I}(R) \in \mathcal{I}(\mathcal{S})$ to a vertex $v_t \in V_{\mathcal{I}(\mathcal{G}_\mathcal{S})}$ with label $:R$ and properties $\nu(v_t,A) := t(A)$. That is, records simply translate into vertices with properties.

The *implication problem of relational PG-Keys* is to decide whether for every given $(\mathcal{S},\Sigma_\mathcal{S} \cup \{\varphi\})$, every instance $\mathcal{I}(\mathcal{G}_\mathcal{S})$ over $\mathcal{G}_\mathcal{S}$ that satisfies all elements of $\Sigma_{\mathcal{G}_\mathcal{S}}$ will also satisfy $\varphi_{\mathcal{G}_\mathcal{S}}$.

Undecidability follows from the PG-Keys' ability to express keys and foreign keys over relational instances, and the fact that implication for keys by keys and foreign keys is undecidable [16].

COROLLARY 4.1. *Implication of relational PG-Keys is undecidable.*

4.5 E/R Keys as Efficient Core of PG-Key

E/R keys form a special class of PG-Key. Indeed, an E/R key $O(\{O_1,\ldots,O_n\},\{K_1,\ldots,K_m\})$ for $\mathcal{G}_D$ is satisfied by an E/R graph G for $\mathcal{G}_D$ if and only if G satisfies the following PG-Key

$$\texttt{FOR}\ (x : O)\ \texttt{ID}\ x.K_1,\ldots,x.K_m,y_1,\ldots,y_n\ \texttt{WITHIN}$$
$$(x) \to (y_1 : O_1),\ldots,(x) \to (y_n : O_n).$$

The *implication problem for E/R keys* is to decide whether for every E/R graph model $\mathcal{G}_D$ and every set $\Sigma \cup \{\varphi\}$ of E/R keys for $\mathcal{G}_D$, every E/R graph that satisfies all E/R keys in Σ also satisfies φ. The *extension rule* $\frac{O(C',K')}{O(C,K)} C' \subseteq C, K' \subseteq K$ derives any E/R key $O(C,K)$ from an E/R key $O(C',K')$ whenever $C' \subseteq C$ and $K' \subseteq K$, just like super-keys X can be derived from a super-key X' with $X' \subseteq X$. Indeed, the set of E/R keys derived from any set Σ of E/R keys by applications of the extension rule coincides with the set of E/R keys implied by Σ. This entails that Σ implies the E/R key $O(C,K)$ if and only if there is some E/R key $O(C',K')$ in Σ such that $C' \subseteq C$ and $K' \subseteq K$. This means we obtain the following result.

THEOREM 4.2. *The implication problem of E/R keys is finitely axiomatizable and decidable in time linear in the input.*

Current graph database systems only permit `UNIQUE` constraints with multiple properties, excluding components [25]. This makes it necessary to duplicate properties across nodes. Permitting E/R links in keys would advance integrity management to new levels.

4.6 E/R Links for Referential Integrity

A critical novelty of E/R graphs is their ability to handle referential integrity using E/R links. In relational databases, foreign key attributes of child tables reference key attributes of parent tables, and updates to any of their values need to be propagated consistently to maintain referential integrity. As illustration on TPC-H, we will discuss three implementations of referential integrity for the E/R link SUPPLIER $\in$ *comp*(PARTSUPP).

When mapping TPC-H relations to E/R graphs, we may use (1) property *suppkey* on both vertices :SUPPLIER and :PARTSUPP, and an E/R link from :PARTSUPP to :SUPPLIER. We may then use the following PG-Key K to enforce referential integrity:
(1) FOR (x : PARTSUPP) MANDATORY y WITHIN (x) $\longrightarrow$ (y : SUPPLIER) WHERE $x.suppkey = y.suppkey$.

However, it suffices to *either* (2) duplicate property *suppkey* from vertex :SUPPLIER on vertex :PARTSUPP, or (3) use the E/R link. In case (2) PG-Key K reduces to the following *foreign key*:
(2) FOR (x : PARTSUPP) MANDATORY y WITHIN (x), (y : SUPPLIER) WHERE $x.suppkey = y.suppkey$

while case (3) reduces the PG-Key K to the following *E/R link*:
(3) FOR (x:PARTSUPP) MANDATORY y WITHIN (x) $\longrightarrow$ (y:SUPPLIER)

For E/R links, no data values need duplication at all, which is impossible in relational databases or E/R instances because they duplicate attributes just like foreign PG-keys. This has profound implications for data integrity management in E/R graphs.

5 DATA INTEGRITY FOR E/R GRAPHS

Based on the choice of foreign keys or E/R links, we will propose three principled approaches to managing data integrity for E/R graphs in Sec. 5.1. For each approach, we dedicate one section to defining mappings of relational databases in IDNF to well-designed property graphs. This also prepares our experiments in Sec. 6.

5.1 Three Principled Approaches

As seen before, each component $C \in comp(O)$ of an E/R schema provides us with a choice of implementing referential integrity as a foreign key FOR (x:O) MANDATORY y WITHIN (x), (y:C) WHERE $x.P_1 = y.P_1$ AND ... AND $x.P_k = y.P_k$ with $id(C) = \{P_1, \ldots, P_k\}$, or as E/R link FOR ($x$:$O$) MANDATORY y WITHIN (x) $\longrightarrow$ (y:C) in the corresponding E/R graph. The situation is illustrated on the E/R graph model of TPC-H in Fig. 9a. For each non-leaf node, we can keep either the foreign key properties or the E/R link. Keeping foreign keys introduces property redundancy, where updates to any values on key properties need to be propagated consistently.

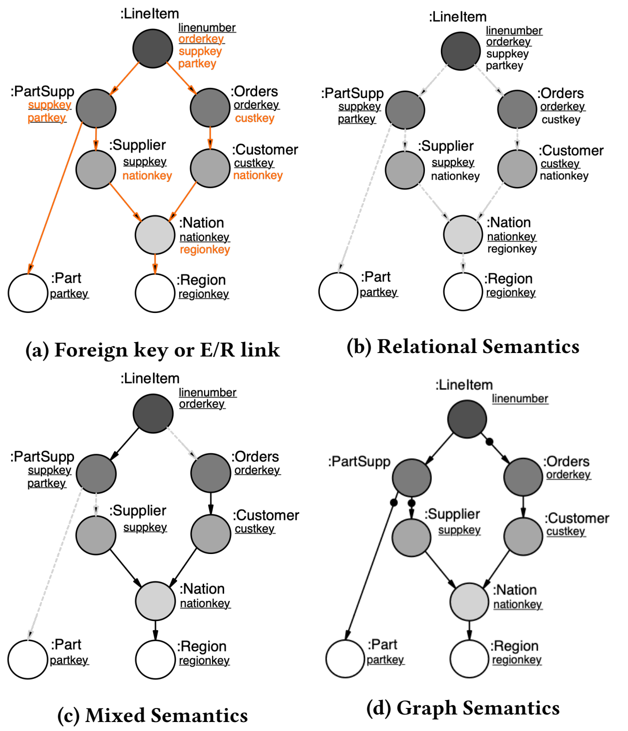


Figure 9: E/R graph models for TPC-H under relational (Sec. 5.2), mixed (Sec. 5.3), and graph semantics (Sec. 5.4)

The remainder of Fig. 9 illustrates three principled approaches, representing varying levels of property redundancy. Fig. 9b illustrates *relational semantics* where we uniformly choose foreign keys. All orange properties from Fig. 9a are now black and all E/R links are grey and dotted, indicating that E/R links have become redundant. Fig. 9c shows *mixed semantics* where we choose foreign keys when their properties contribute to a key, and E/R links otherwise. In this case, only three E/R links (those in grey and dotted) become redundant. Relational and mixed semantics enable us to handle entity integrity by E/R key $O(C,K)$ where $C = \emptyset$. As these keys only use properties, we call them *property keys.* While E/R keys are not supported in graph database systems, property keys are. Fig. 9d introduces *graph semantics* where we do not introduce any property redundancy since we uniformly choose E/R links.

Tab. 1 summarizes how the different approaches. On one extreme, relational semantics relies on foreign keys and property keys for referential and entity integrity, respectively. On the other extreme, graph semantics uses exclusively E/R links to manage referential integrity, but requires E/R keys for entity integrity. Mixed semantics

Table 1: Integrity Management Under Different Semantics

Semantics	Referential integrity	Entity integrity
Relational	Foreign keys	Property keys
Mixed	Mixed	Property keys
Graph	E/R links	E/R keys

minimizes the use of foreign keys for referential integrity, while only relying on property keys for entity integrity.

5.2 Relational Semantics

Given is a relational database schema $\mathcal{S}$ with a set $\Sigma_{\mathcal{S}}$ of keys and foreign keys in IDNF, as well as a relational database $\mathcal{I}(\mathcal{S})$ over $\mathcal{S}$ that satisfies $\Sigma_{\mathcal{S}}$.

Targeting relational semantics, we map the schema, constraints and instances to an E/R graph model, its constraints and E/R graphs, respectively. Firstly, the E/R graph model is obtained by mapping every relation schema $R \in \mathcal{S}$ to a vertex v_R with label R, and

- for every attribute $A \in R$ with $dom(A)$ we have a property-value pair $P_A = dom(A)$ on v_R,
- for every key $\{A_1, \ldots, A_n\}$ of R we have a property key $R(\emptyset, \{P_{A_1}, \ldots, P_{A_n}\})$ implemented as PG-Key `FOR` $(x : R)$ `ID` $x.P_{A_1}, \ldots, x.P_{A_n}$
- for every foreign key $R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$ of R we have the foreign key `FOR` $(x : R)$ `MANDATORY` y `WITHIN` (x),$(y : S)$ `WHERE` $x.P_{A_1} = y.P_{B_1}, \ldots, x.P_{A_m} = y.P_{B_m}$.

Secondly, the E/R graph obtained from an instance $\mathcal{I}(\mathcal{S})$ results from mapping every tuple $t \in \mathcal{I}(R)$ for every $R \in \mathcal{S}$ to a node v_t with label R such that, for all $A \in R$, $\nu(v_t, P_A) := t(A)$.

As an example, the relation schema LINEITEM with attributes *suppkey*, *partkey*, *orderkey*, *linenumber*, …, *comment*, key {*linenumber,orderkey*} and foreign keys

- [*suppkey*, *partkey*] $\subseteq$ PARTSUPP[*suppkey*, *partkey*] and
- [*orderkey*] $\subseteq$ ORDERS[*orderkey*]

would have a corresponding vertex v_{LINEITEM} with label LINEITEM and properties $P_{orderkey}$, …, $P_{comment}$, an E/R key LINEITEM($\emptyset$, {*linenumber*, *orderkey*}) implemented as `FOR` $(x :$ LINEITEM$)$ `ID` $x.P_{linenumber}, x.P_{orderkey}$ and foreign keys

- `FOR` $(x$:LINEITEM$)$ `MANDATORY` y WITHIN (x), $(y :$ PARTSUPP$)$ `WHERE` $x.P_{suppkey} = y.P_{suppkey}, x.P_{partkey} = y.P_{partkey}$, and
- `FOR` $(x :$ LINEITEM$)$ `MANDATORY` y `WITHIN` (x), $(y :$ ORDERS$)$ `WHERE` $x.P_{orderkey} = y.P_{orderkey}$.

5.3 Mixed Semantics

For each relation schema $R \in \mathcal{S}$, let K_R denote the set of attributes in R that are part of some minimal key, and let F_R denote the attributes that participate in some foreign key over R. The set $R_{\text{mix}} := R - (F_R - K_R)$ removes all attributes from R that belong to some foreign key but do not belong to any key of R. Note that every foreign key uniquely identifies objects from the table it references, and that all attributes of this foreign key are required for that. Hence, we cannot have a foreign key where some of its attributes belong to a key and other attributes do not. In summary, for every foreign key $R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$ of R we have either $\{A_1, \ldots, A_m\} \subseteq R_{\text{mix}}$ or $\{A_1, \ldots, A_m\} \cap R_{\text{mix}} = \emptyset$.

Firstly, the E/R graph model is obtained by mapping every relation schema $R \in \mathcal{S}$ to a vertex v_R with label R, such that

- for every attribute $A \in R_{\text{mix}}$ with domain $dom(A)$ we have a property-value pair $P_A = dom(A)$ of v_R,
- for every key $\{A_1, \ldots, A_n\}$ of R, we have property key $R(\emptyset, \{P_{A_1}, \ldots, P_{A_n}\})$ implemented as PG-Key `FOR` $(x : R)$ `ID` $x.P_{A_1}, \ldots, x.P_{A_n}$
- for every foreign key $R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$ of R where $\{A_1, \ldots, A_m\} \subseteq R_{\text{mix}}$, we have the foreign key `FOR` $(x : R)$ `MANDATORY` y `WITHIN` (x),$(y : S)$ `WHERE` $x.P_{A_1} = y.P_{B_1}, \ldots, x.P_{A_m} = y.P_{B_m}$, and
- for every foreign key $R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$ of R where $\{A_1, \ldots, A_m\} \cap R_{\text{mix}} = \emptyset$, we have the foreign key `FOR` $(x : R)$ `MANDATORY` y `WITHIN` $(x) \longrightarrow (y : S)$.

Secondly, the E/R graph obtained from an instance $\mathcal{I}(\mathcal{S})$ results from mapping every (i) tuple $t \in \mathcal{I}(R)$ for every $R \in \mathcal{S}$ to a node v_t with label R such that, for all $A \in R_{\text{mix}}$, $\nu(v_t, P_A) := t(A)$; and (ii) foreign key/key relationship where for $t \in \mathcal{I}(R)$ we have a unique $s \in \mathcal{I}(S)$, such that $t[A_1, \ldots, A_m] = s[B_1, \ldots, B_m]$, based on the foreign key $R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$ on R where $\{A_1, \ldots, A_m\} \cap R_{\text{mix}} = \emptyset$, to a directed edge (v_t, v_s).

Continuing our example, the relation schema LINEITEM from before has a corresponding vertex v_{LINEITEM} with label LINEITEM and properties $P_{orderkey}$, $P_{linenumber}$, …, $P_{comment}$ (properties $P_{suppkey}$ and $P_{partkey}$ are not required), a property key LINEITEM($\emptyset$, {*linenumber*, *orderkey*}) implemented as `FOR` $(x :$ LINEITEM$)$ `ID` $x.P_{linenumber}, x.P_{orderkey}$ and further foreign keys `FOR` $(x :$ LINEITEM$)$ `MANDATORY` y `WITHIN` $(x) \longrightarrow (y :$ PARTSUPP$)$, and `FOR` $(x :$ LINEITEM$)$ `MANDATORY` y `WITHIN` $(x), (y :$ ORDERS$)$ `WHERE` $x.P_{orderkey} = y.P_{orderkey}$. Note that relation schema PARTSUPP has the key {*partkey,suppkey*}.

5.4 Graph Semantics

For each relation schema $R \in \mathcal{S}$, let $R_g := R - F_R$ consist of all attributes from R that are not part of any foreign key. Furthermore, let C_R denote the set of all relation schemata $S \in \mathcal{S}$ such that there is some foreign key $R[X] \subseteq S[Y] \in \Sigma_{\mathcal{S}}$. For each relation schema, there is a distinguished key Z (the primary key), dictating which components are labeled as key components in the resulting E/R graph model.

Firstly, the E/R graph model is obtained by mapping every relation schema $R \in \mathcal{S}$ to a vertex v_R with label R, and

- for every attribute $A \in R_g$ with domain $dom(A)$, we have a property-value pair $P_A = dom(A)$ of v_R,
- for the distinguished key Z of R in Σ, we have an E/R key $R(C_Z, P_Z)$ where $C_Z := \{S \in C_R \mid Z \cap S \neq \emptyset\} = \{S_1, \ldots, S_m\}$ and $P_Z := \{P_A \mid A \in Z\} - \{P_A \mid A \in S, S \in C_Z\} = \{P_{A_1}, \ldots, P_{A_n}\}$, implemented as directed edges $(v_R, v_{S_1}), \ldots, (v_R, v_{S_m})$ with label $\bullet$ each, and `FOR` $(x$:$R)$ `ID` $x.P_{A_1}, \ldots, x.P_{A_n}, y_1, \ldots, y_m$ `WITHIN` $(x) \longrightarrow (y_1 : S_1), \ldots, (x) \longrightarrow (y_m : S_m)$,
- for every foreign key $R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$ of R where $S \notin C_Z$, we have the directed edge (v_R, v_S), and foreign key `FOR` $(x : R)$ `MANDATORY` y `WITHIN` $(x) \longrightarrow (y : S)$.

Table 2: E/R keys used for entity integrity checking

Semantics	E/R key	PG-Key	Cypher Query	Index
Relational and Mixed	$\mathrm{PS}_k(r) = \mathrm{PS}_k(m)$: PS($\emptyset$,{*partkey*, *suppkey*})	`FOR` (ps:PS) `ID` ps.*partkey*, ps.*suppkey*	`MATCH` (ps1:PS), (ps2:PS) `WHERE` id(ps1) < id(ps2) `AND` ps1.*partkey* = <*partkey*> `AND` ps2.*partkey* = <*partkey*> `AND` ps1.*suppkey* = <*suppkey*> `AND` ps2.*suppkey* = <*suppkey*> `RETURN` ps1, ps2	PS(*partkey, suppkey*)
Graph	$\mathrm{PS}_k(g)$: PS({P, S},$\emptyset$)	`FOR` (ps:PS) `ID` p, s `WITHIN` (s:S) ⟵ (ps) ⟶ (p:P)	`MATCH` q1 = (s:S)<-[ ]-(ps1:PS)-[ ]->(p:P), q2=(s:S)<-[ ]-(ps2:PS)-[ ]->(p:P) `WHERE` id(ps1) < id(ps2) `AND` p.*partkey* = <*partkey*> `AND` s.*suppkey* = <*suppkey*> `RETURN` q1, q2	no index
Relational and Mixed	$\mathrm{L}_k(r) = \mathrm{L}_k(m)$: L($\emptyset$,{*orderkey*, *linenumber*})	`FOR` (l:L) `ID` l.*orderkey*, l.*linenumber*	`MATCH` (l1:L), (l2:L) `WHERE` id(l1) < id(l2) `AND` l1.*orderkey* = <*orderkey*> `AND` l2.*orderkey* = <*orderkey*> `AND` l1.*linenumber* = <*linenumber*> `AND` l2.*linenumber* = <*linenumber*> `RETURN` l1, l2	L(*orderkey, linenumber*)
Graph	$\mathrm{L}_k(g)$: L({O},*linenumber*)	`FOR` (l:L) `ID` l.*linenumber*, o `WITHIN` (l) ⟶ (o:O)	`MATCH` q = (l1:L)-[ ]->(o:O)<-[ ]-(l2:L) `WHERE` id(l1) < id(l2) `AND` o.*orderkey* = <*orderkey*> `AND` l1.*linenumber* = <*linenumber*> `AND` l2.*linenumber* = <*linenumber*> `RETURN` q	no index

- for other keys X of R, we have E/R key $R(C_X, P_X)$ where $C_X := \{S \in C_R \mid X \cap S \neq \emptyset\} = \{S_1, \ldots, S_m\}$ and $P_X := \{P_A \mid A \in X\} - \{P_A \mid A \in S, S \in C_X\} = \{P_{A_1}, \ldots, P_{A_n}\}$, implemented as `FOR` $(x{:}R)$ `ID` $x.P_{A_1}, \ldots, x.P_{A_n}, y_1, \ldots, y_m$ `WITHIN` $(x) \longrightarrow (y_1{:}S_1), \ldots, (x) \longrightarrow (y_m{:}S_m)$.

Secondly, the E/R graph is obtained from an instance $\mathcal{I}(\mathcal{S})$ by mapping every (i) tuple $t \in \mathcal{I}(R)$ for every $R \in \mathcal{S}$ to a node v_t with label R such that, for all $A \in R_g$, $\nu(v_t, P_A) := t(A)$; and (ii) foreign key/key relationship where for $t \in \mathcal{I}(R)$ we have a unique $s \in \mathcal{I}(S)$, such that $t[A_1, \ldots, A_m] = s[B_1, \ldots, B_m]$, based on the foreign key $R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$ of R, to a directed edge (v_t, v_s).

In our example, LINEITEM results in vertex v_{LINEITEM} with label LINEITEM and properties $P_{linenumber}, \ldots, P_{comment}$ ($P_{suppkey}$, $P_{partkey}$, $P_{orderkey}$ not needed), E/R key LINEITEM({ORDERS}, {*linenumber*}) implemented as `FOR` $(x{:}\text{LINEITEM})$ `ID` $x.P_{linenumber}, y$ `WITHIN` $(x) \longrightarrow (y{:}\text{ORDERS})$ and foreign key `FOR` $(x{:}\text{LINEITEM})$ `MANDATORY` y `WITHIN` $(x) \longrightarrow (y{:}\text{PARTSUPP})$. ORDERS has key {*orderkey*}, and PARTSUPP has key {*partkey,suppkey*}.

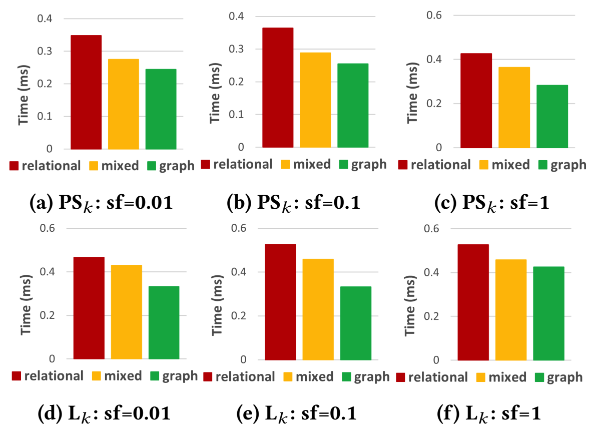
6 EXPERIMENTS

We will analyze the performance of data integrity maintenance and query evaluation on translations of TPC-H into Neo4j, using scaling factors (sf) small (0.01), medium (0.1) and large (1) under relational, mixed, and graph semantics. TPC-H is in IDNF, and capable of quantifying advantages of the different semantics, due to i) key/foreign key chains with different lengths, ii) E/R and property keys, iii) different data volumes and iv) a wide range of operations.

Sec. 6.1 explains the set up, entity and referential integrity are analyzed in Sec. 6.2 and 6.3, with queries discussed in Sec. 6.4.

6.1 Set up

For details of the experiments with results and artifacts, we refer the reader to our Github repository[1]. We used Neo4j's query language Cypher together with Python 3.9.13. As RDBMS we used MySQL 8.0. Experiments were conducted on a 64-bit operating system with an Intel Core i7 Processor with 16GB RAM. The following table shows how many nodes and edges are present in each translation of TPC-H.



(a) PS_k: sf=0.01 (b) PS_k: sf=0.1 (c) PS_k: sf=1

(d) L_k: sf=0.01 (e) L_k: sf=0.1 (f) L_k: sf=1

Figure 10: Entity Integrity Checking in TPC-H

Semantics	*sf=0.01*		*sf=0.1*		*sf=1*	
	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$
Relational	86,805	0	866,602	0	8,661,245	0
Mixed	86,805	76,800	866,602	766,597	8,661,245	7,661,240
Graph	86,805	152,975	866,602	1,527,169	8,661,245	15,262,455

6.2 Checking Entity Integrity

Our experiments measure the runtime of Cypher queries that check the validity of E/R keys when inserting new nodes. We selected the E/R key PS_k = PS({PART, SUPP}, $\emptyset$) on PARTSUPP, and the E/R key L_k = L({ORDERS}, {*linenumber*}) on LINEITEM. Under relational (r) and mixed (m) semantics, we can specify these keys as `unique` constraints within Neo4j, and benefit from the resulting index. Under graph semantics (g), the corresponding E/R keys are not supported by Neo4j[2] (or any other graph database system), so we cannot benefit from any index structure. Under all semantics, the corresponding Cypher query returns all nodes that duplicate the given key values. Hence, the entity integrity check is successful if and only if the query answer is empty.

For each translation, Tab. 2 lists the E/R key, its PG-Keys, Cypher queries and index structure (if available). We used shortcuts PART (P), SUPPLIER (S), PARTSUPP (PS), ORDERS (O) and LINEITEM (L); and

[1]https://github.com/graphdbexperiments/er_graph_experiments

[2]https://neo4j.com/docs/cypher-manual/current/constraints

Table 3: Time (in ms) for Entity Integrity Checks on TPC-H

E/R		*sf=0.01*		*sf=0.1*		*sf=1*	
key	Sem	index	no index	index	no index	index	no index
PS_k	r	0.348	5.617	0.364	14.949	0.426	203.713
	m	0.275	4.042	0.288	13.694	0.364	142.031
	g	–	0.244	–	0.255	–	0.282
L_k	r	0.466	34.877	0.526	206.668	0.527	26721.735
	m	0.43	33.783	0.458	215.784	0.468	15260.388
	g	–	0.332	–	0.368	–	0.426

$K(s)$ for the E/R key K under semantics s, such as $\text{PS}_k(r)$ for the E/R key PS_k under relational (r) semantics.

Fig. 10 shows the performance of entity integrity checking under each semantics and scaling factor. Each runtime reported has been averaged over ten runs. While checking entity integrity is efficient across all semantics and scaling factors, graph semantics performs best. This is very interesting since relational and mixed semantics enjoy index support based on property keys while graph semantics does not. We attribute these results to the strength of graph database systems on exploiting edges.

Tab. 3 shows the runtime (in ms) explicitly, including those for relational (r) and mixed (m) semantics (sem) without a `unique` index. This constitutes a great motivation for implementing E/R keys in graph database systems and make entity integrity checking even faster.

6.3 Referential Integrity

A fundamental impact of graph databases is their ability to eliminate property redundancy, which transcends to tremendous time savings when maintaining referential integrity. As we will quantify, this is best achieved by our concepts of E/R graphs, keys, and links under our graph semantics.

In maintaining referential integrity, our experiments propagate updates on properties according to the translations of E/R links on the TPC-H dataset under different semantics. We selected the following chains resulting from E/R links between the tables:

P_c: Part(P) ⊂ PartSupp(PS) ⊂ Lineitem(L)
S_c: Supplier(S) ⊂ PartSupp(PS) ⊂ Lineitem(L)
C_c: Customer(C) ⊂ Orders(O)
O_c: Orders(O) ⊂ Lineitem(L) .

These chains are diverse in how they handle referential integrity across the semantics. As example, for the chain S_c we have updated the property *suppkey* for different percentages of nodes labelled Supplier (for all semantics) which requires corresponding updates for property *suppkey* on PartSupp nodes (for relational and mixed semantics) and again for nodes labelled Lineitem (for relational semantics). In particular, as *suppkey* is a key attribute on PartSupp, it is duplicated on PartSupp nodes under relational and mixed semantics because they only use property keys, but not under graph semantics as it uses E/R keys. Tab. 4 shows the update operations in Cypher syntax for the chain S_c under different semantics.

We conducted the same experiments for the other three chains. The chain P_c has very similar characteristics as the chain S_c. For C_c the property *custkey* is already absent on Orders nodes under mixed semantics, hence updates under mixed and graph semantics are the same. For O_c the property *orderkey* is only present on Lineitem nodes under relational and mixed semantics but not under graph semantics. Hence, updates under relational and mixed semantics remain the same.

Table 4: Update propagation for $S \subset PS \subset L$ by semantics

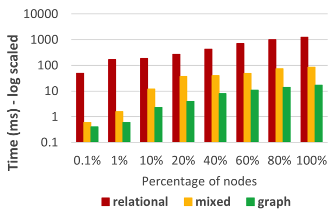
Sem	Update propagation
r	`MATCH` (s:S), (ps:PS), (l:L) `WHERE` ps.*suppkey* = s.*suppkey* `AND` l.*suppkey* = s.*suppkey* `SET` s.*suppkey* = `RIGHT`((’00000000’ + `toString`(s.*suppkey*)), 8), ps.*suppkey* = `RIGHT`((’00000000’ + `toString`(ps.*suppkey*)), 8), l.*suppkey* = `RIGHT`((’00000000’+`toString`(l.*suppkey*)), 8)
m	`MATCH` (s:S), (ps:PS) `WHERE` ps.*suppkey* = s.*suppkey* `SET` s.*suppkey* = `RIGHT`((’00000000’ + `toString`(s.*suppkey*)), 8), ps.*suppkey* = `RIGHT`((’00000000’ + `toString`(ps.*suppkey*)), 8)
g	`MATCH` (s:S) `SET` s.*suppkey* = `RIGHT`((’00000000’ + `toString`(s.*suppkey*)), 8)

Our main question concerns the impact of our semantics on update times. Tab. 5 lists the database hits (db hits) and times (in ms) required for different update chains on one percent of the respective nodes of the TPC-H dataset under different semantics (sem) and for different scaling factors (sf). In Neo4j, database operators send a request to its storage engine to do work, such as retrieving or updating data. A database hit is an abstract unit of this storage engine work. The db hits estimate huge performance gains independent of the database size. Indeed, the query engine recognizes that property redundancy is reduced in mixed and eliminated in graph semantics. Moreover, the estimates manifest themselves in actual runtime, too.

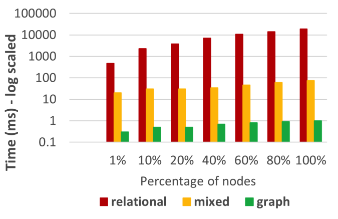
Fig. 11 shows the update times across different semantics and for each fixed combination of a chain and scaling factor. In each scenario, we show the average runtime (in ms) over ten runs for all the different percentages of nodes we update. Due to the big difference in times between the semantics, we show each graph on a log scale. Noticeably, the performance differs at orders of magnitude, showing the superior handling of referential integrity by graph over mixed semantics, and that of mixed over relational semantics. The different magnitudes result from how the data are distributed and the characteristics of our translations. Fig. 11 (b) is missing results for 0.1%, as here 1% corresponds to a single node labelled Supplier already.

For the chain $P_c : P \subset PS \subset L$, performance under mixed semantics is closer to that under graph semantics when compared to the performance difference for the chain $S_c : S \subset PS \subset L$. The simple reason is that there are fewer suppliers than parts but the combinations of suppliers and parts in the PartSupp table are the same. For our chain $C_c : C \subset O$, graph and mixed semantics coincide, so any difference in update performance is negligible. For the chain $O_c : O \subset L$ we duplicate the property *orderkey* on Lineitem nodes under both relational and mixed semantics. Hence, performance for those semantics are closely aligned.

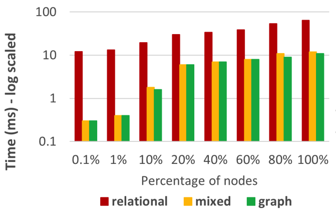
Our experiments quantify the penalty of maintaining referential integrity in terms of property redundancy. Indeed, relational semantics incurs the highest penalty, followed by mixed semantics, with high efficiency gains in many cases. Graph semantics eliminates property redundancy and therefore shows best performance. Like



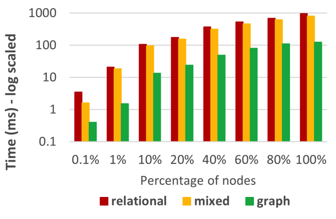
(a) *sf=0.01*: $P \subset PS \subset L$



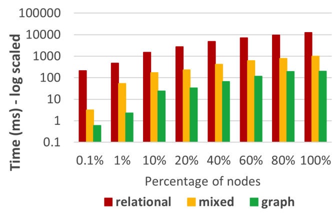
(b) *sf=0.01*: $S \subset PS \subset L$



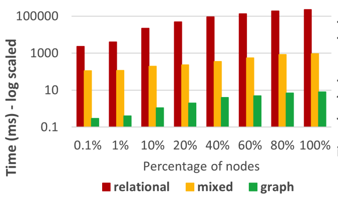
(c) *sf=0.01*: $C \subset O$



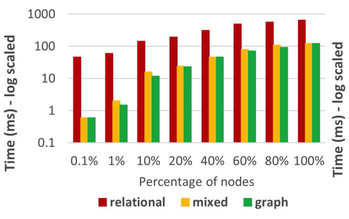
(d) *sf=0.01*: $O \subset L$



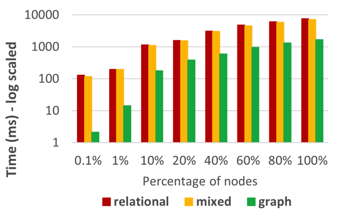
(e) *sf=0.1*: $P \subset PS \subset L$



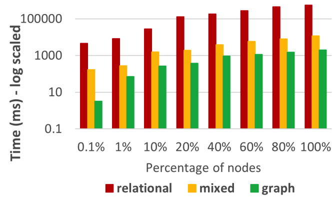
(f) *sf=0.1*: $S \subset PS \subset L$



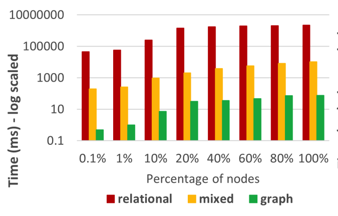
(g) *sf=0.1*: $C \subset O$



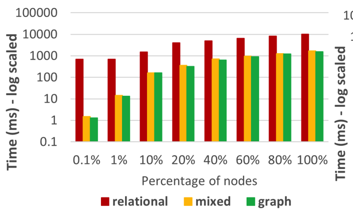
(h) *sf=0.1*: $O \subset L$



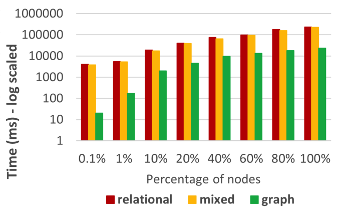
(i) *sf=1*: $P \subset PS \subset L$



(j) *sf=1*: $S \subset PS \subset L$



(k) *sf=1*: $C \subset O$



(l) *sf=1*: $O \subset L$

Figure 11: Comparing referential integrity maintenance across different propagation chains and different semantics

Table 5: Results for update propagation in TPC-H

		sf=0.01		*sf=0.1*		*sf=1*	
Chain	Sem	db hits	time (ms)	db hits	time (ms)	db hits	time (ms)
P_c	r	311,642	168	1,816,084	469	8,243,302	8,418
	m	761	1.6	83,402	54	834,002	276
	g	101	0.6	601	2.3	6,001	73
S_c	r	884,973	470	5,958,206	4,101	47,587,284	57,795
	m	24,641	20	243,201	117	864,102	261
	g	6	0.3	31	0.4	301	1.0
C_c	r	75,131	13	456,155	60	4,621,519	709
	m	76	0.4	451	2.0	4,501	15
	g	76	0.4	451	1.5	4,501	13
O_c	r	65,161	21	626,036	198	6,256,283	5,447
	m	65,111	19	626,011	195	6,256,090	5,237
	g	751	1.5	4,501	14	45,001	173

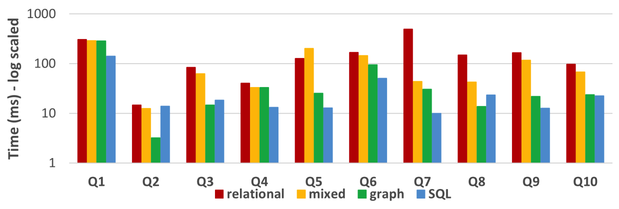
before our experiments made full use of `unique` indices for property keys, again highlighting strong prospects for further performance gains once index support becomes available for E/R keys.

6.4 Benchmark queries and refresh operations

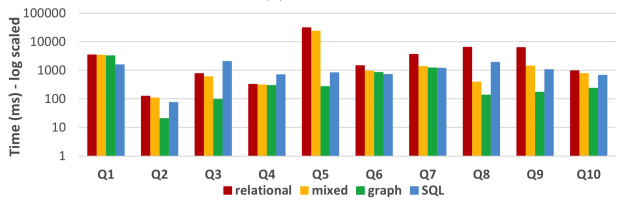
We analyze the impact of our semantics on query and refresh operations of the TPC-H benchmark, and we also compared their performance to that in MySQL.

Fig. 12 shows the runtime for ten queries in different semantics and SQL, averaged over ten runs. With one exception, we can see speed ups from relational to mixed semantics, and from mixed to graph semantics. For Q5, mixed takes longer than relational semantics for the small scaling factor. This may be due to the fact that only on larger data the benefits of matching complicated graph patterns outperform the processing of complex join operations. Q5 also takes significantly more time under relational and mixed semantics than other queries, especially for larger scaling factors. Again, this might be due to the processing of expensive join conditions which still exist under mixed semantics for this query. Q1 and Q6 do not involve any join and therefore have the same translation into Cypher under all semantics. The slight improvements for mixed and graph semantics result from having to scan fewer properties on LINEITEM nodes. MySQL shows slightly better performance on queries with no joins (Q1 and Q6), compared to graph semantics. However, graph semantics in Neo4j outperforms MySQL for most queries that require joins (Q2-Q5, Q7-Q10). Finally, Q2-Q4 and Q7-Q10 show a similar trend where relational takes longer than mixed semantics, and graph semantics shows further improvements. These vary depending on scaling factor and number of joins. For example, Q4 shows the least difference between the different semantics.

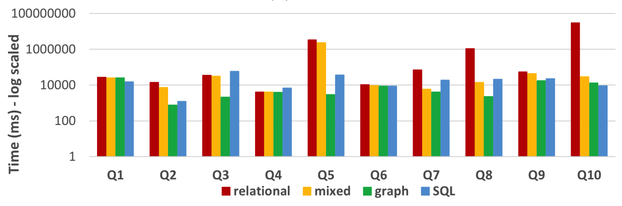
The two refresh operations of the TPC-H benchmark include RF1, which inserts new records for customer orders and corresponding lineitem information, while RF2 deletes existing orders and their associated lineitem details. Fig. 13 shows the average time required for these operations over ten runs. As expected, insertion is most expensive for graph semantics followed by mixed and then relational semantics. For the deletion of records we observe the opposite trend. These extra costs for node insertion are due to additional



(a) sf=0.01

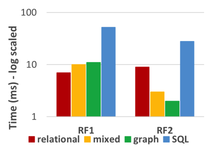


(b) sf=0.1

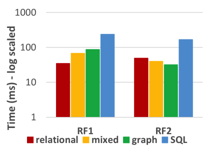


(c) sf=1

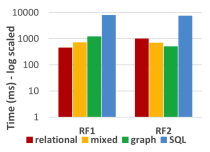
Figure 12: Runtime of TPC-H queries



(a) sf=0.01



(b) sf=0.1



(c) sf=1

Figure 13: Runtime of TPC-H refresh operations

costs for matching nodes and creating relationships on top of the new nodes, while for the deletion of records it is more expensive to match and delete ORDERS and LINEITEM nodes than just matching and deleting ORDERS nodes along with all linked LINEITEM nodes. Neo4j outperforms MySQL on RF1 and RF2.

Tab. 6 compares the total runtimes over all benchmark queries Q1-Q10, and also over both refresh operations under different semantics in Neo4j and MySQL, and for different scaling factors. We also show the performance ratio, which we always compare to graph semantics. For Q1-Q10 the ratios clearly show how graph semantics outperforms the other semantics in both speed and scale. Indeed, the performance ratios grow with larger scaling factors with one exception: MySQL performs better than Neo4j under graph semantics for the small scaling factor, but this reverses for medium and large scaling factors. Regarding refresh operations we see that for growing scaling factors, relational and mixed semantics perform better than graph semantics. However, graph semantics outperforms MySQL for all scaling factors.

Table 6: Results for query and refresh operations in TPC-H

		sf=0.01		*sf=0.1*		*sf=1*	
	Sem	time (ms)	ratio (x/g)	time (ms)	ratio (x/g)	time (ms)	ratio (x/g)
Q1	r	1639.6	3.01	54,090.4	8.31	34,178,162.3	425.91
-	m	1,012.8	1.86	32,604.4	5.01	2,537,165.5	31.62
Q10	g	545.5	1	6512.7	1	80,248.3	1
	SQL	318.7	0.58	10,743.5	1.65	198,704.1	2.48
RF1	r	7	1.23	84	0.71	1,434	0.85
and	m	13	1	108	0.91	1,384	0.82
RF2	g	13	1	119	1	1686	1
	SQL	80	6.15	406	3.41	15,153	8.99

7 CONCLUSION AND FUTURE WORK

We have proposed E/R graphs as instances of classical E/R diagrams. These constitute a core contribution to conceptual data modeling as E/R graphs are the first graph semantics for the E/R model. Their core advantage is the elimination of attribute/property redundancy through the use of E/R links and E/R keys. We have shown that E/R diagrams are (i) graph models for well-designed property graphs, which are directed acyclic graphs with no value and no property redundancy, (ii) a sub-class of PG-Schema, and their instances define homomorphisms on them. This constitutes a fundamental contribution to schema languages for property graphs. Indeed, E/R diagrams and graphs combine best practice in conceptual, logical, and graph modeling. They constitute the core of PG-Schema that represents well-designed property graphs, equivalent to relational database schemata in Inclusion Dependency Normal Form whose instances are free from data redundancy but carry attribute redundancy. Our third contribution advances entity and referential integrity maintenance fundamentally. We propose three alternative approaches: graph semantics eliminates property redundancy by using E/R keys, relational semantics fully utilizes property redundancy and only requires property keys, and mixed semantics minimizes property redundancy while still requiring property keys only. Our experiments firmly quantify how efficient the different semantics handle TPC-H workloads within native graph databases. Our results establish E/R keys as an efficient core fragment for best data modeling practice within the class of PG-Keys for which we showed that implication is undecidable. Altogether, the elimination of property redundancy and its resulting benefits appear to be a particular advantage of graph over other types of databases. This transcends to the operational level where benchmark workloads, including queries, can be handled efficiently. In particular, TPC-H join queries process faster and scale better in Neo4j under graph semantics than their corresponding queries do in MySQL for relational databases. We have therefore shown that the classical E/R framework provides a principled and natural methodology for well-designed property graphs, firmly driving the further uptake of graph databases.

Our work motivates the implementation of E/R modeling within graph database systems, including the use of E/R keys and the development of index structures that support their maintenance. Identifying other useful fragments of PG-Keys and PG-Schema appears to be another important direction of future research.

REFERENCES

[1] Serge Abiteboul, Peter Buneman, and Dan Suciu. 1999. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann.

[2] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Alastair Green, Jan Hidders, Bei Li, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savkovic, Michael Schmidt, Juan Sequeda, Slawek Staworko, Dominik Tomaszuk, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Dusan Zivkovic. 2023. PG-Schema: Schemas for Property Graphs. *Proc. ACM Manag. Data* 1, 2 (2023), 198:1–198:25.

[3] 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 *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. 2423–2436.

[4] Renzo Angles and Claudio Gutierrez. 2008. Survey of graph database models. *ACM Comput. Surv.* 40, 1 (2008), 1:1–1:39.

[5] Nelly Barret, Tudor Enache, Ioana Manolescu, and Madhulika Mohanty. 2024. Finding the PG schema of any (semi)structured dataset: a tale of graphs and abstraction. In *40th International Conference on Data Engineering, ICDE 2024 - Workshops, Utrecht, Netherlands, May 13-16, 2024*. 365–369.

[6] Nelly Barret, Ioana Manolescu, and Prajna Upadhyay. 2024. Computing Generic Abstractions from Application Datasets. In *Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28*. 94–107.

[7] Nimo Beeren and George Fletcher. 2023. A Formal Design Framework for Practical Property Graph Schema Languages. In *Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023*. 478–484.

[8] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. 1983. On the Desirability of Acyclic Database Schemes. *J. ACM* 30, 3 (1983), 479–513.

[9] Zohra Bellahsene, Angela Bonifati, and Erhard Rahm (Eds.). 2011. *Schema Matching and Mapping*. Springer.

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

[11] Peter P. Chen. 1976. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Trans. Database Syst.* 1, 1 (1976), 9–36.

[12] Peter P. Chen. 1997. English, Chinese and ER Diagrams. *Data Knowl. Eng.* 23, 1 (1997), 5–16.

[13] Stavros S. Cosmadakis and Paris C. Kanellakis. 1984. Functional and Inclusion Dependencies: A Graph Theoretic Approach. In *Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, April 2-4, 1984, Waterloo, Ontario, Canada*. 29–37.

[14] Gwendal Daniel, Gerson Sunyé, and Jordi Cabot. 2016. UMLtoGraphDB: Mapping Conceptual Schemas to Graph Databases. In *Conceptual Modeling - 35th International Conference, ER 2016, Gifu, Japan, November 14-17, 2016, Proceedings*. 430–444.

[15] Victor Martins de Sousa and Luís Mariano del Val Cura. 2018. Logical Design of Graph Databases from an Entity-Relationship Conceptual Model. In *Proceedings of the 20th International Conference on Information Integration and Web-based Applications & Services, iiWAS 2018, Yogyakarta, Indonesia, November 19-21, 2018*. 183–189.

[16] Wenfei Fan and Leonid Libkin. 2002. On XML integrity constraints in the presence of DTDs. *J. ACM* 49, 3 (2002), 368–406.

[17] Wenfei Fan and Ping Lu. 2019. Dependencies for Graphs. *ACM Trans. Database Syst.* 44, 2 (2019), 5:1–5:40.

[18] Ewout Gelling, George Fletcher, and Michael Schmidt. 2023. Bridging graph data models: RDF, RDF-star, and property graphs as directed acyclic graphs. *CoRR* abs/2304.13097 (2023). https://doi.org/10.48550/arXiv.2304.13097

[19] Henning Köhler and Sebastian Link. 2017. Inclusion dependencies and their interaction with functional dependencies in SQL. *J. Comput. Syst. Sci.* 85 (2017), 104–131.

[20] Phokion G. Kolaitis. 2005. Schema mappings, data exchange, and metadata management. In *Proceedings of the Twenty-fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 13-15, 2005, Baltimore, Maryland, USA*. 61–75.

[21] Mark Levene and George Loizou. 2003. Why is the snowflake schema a good data warehouse design? *Inf. Syst.* 28, 3 (2003), 225–240.

[22] Mark Levene and Millist W. Vincent. 2000. Justification for Inclusion Dependency Normal Form. *IEEE Trans. Knowl. Data Eng.* 12, 2 (2000), 281–291.

[23] Heikki Mannila and Kari-Jouko Räihä. 1992. *Design of Relational Databases*. Addison-Wesley.

[24] John C. Mitchell. 1983. The Implication Problem for Functional and Inclusion Dependencies. *Inf. Control.* 56, 3 (1983), 154–173.

[25] Inc. Neo4j. 2024. *Neo4j constraints*. https://neo4j.com/docs/cypher-manual/current/constraints/

[26] Jaroslav Pokorný. 2016. Conceptual and Database Modelling of Graph Databases. In *Proceedings of the 20th International Database Engineering & Applications Symposium, IDEAS 2016, Montreal, QC, Canada, July 11-13, 2016*. 370–377.

[27] Sherif Sakr, Angela Bonifati, Hannes Voigt, Alexandru Iosup, Khaled Ammar, Renzo Angles, Walid G. Aref, Marcelo Arenas, Maciej Besta, Peter A. Boncz, Khuzaima Daudjee, Emanuele Della Valle, Stefania Dumbrava, Olaf Hartig, Bernhard Haslhofer, Tim Hegeman, Jan Hidders, Katja Hose, Adriana Iamnitchi, Vasiliki Kalavri, Hugo Kapp, Wim Martens, M. Tamer Özsu, Eric Peukert, Stefan Plantikow, Mohamed Ragab, Matei Ripeanu, Semih Salihoglu, Christian Schulz, Petra Selmer, Juan F. Sequeda, Joshua Shinavier, Gábor Szárnyas, Riccardo Tommasini, Antonino Tumeo, Alexandru Uta, Ana Lucia Varbanescu, Hsiang-Yun Wu, Nikolay Yakovets, Da Yan, and Eiko Yoneki. 2021. The future is big graphs: a community view on graph processing systems. *Commun. ACM* 64, 9 (2021), 62–71.

[28] Philipp Skavantzos and Sebastian Link. 2023. Normalizing Property Graphs. *Proc. VLDB Endow.* 16, 11 (2023), 3031–3043.

[29] Bernhard Thalheim. 2000. *Entity-relationship modeling - foundations of database technology*. Springer.

[30] Roberto De Virgilio, Antonio Maccioni, and Riccardo Torlone. 2014. Model-Driven Design of Graph Databases. In *Conceptual Modeling - 33rd International Conference, ER 2014, Atlanta, GA, USA, October 27-29, 2014. Proceedings*. 172–185.