Analytics-Aware Graph Database Modeling

Amine Ghrab[1,2], Oscar Romero[3], Sabri Skhiri[1], and Esteban Zimányi[2]

[1] Eura Nova R&D, Mont-Saint-Guibert, Belgium
firstname.lastname@euranova.eu
[2] Université Libre de Bruxelles, Belgium
ezimanyi@ulb.ac.be
[3] Universitat Politècnica de Catalunya , Spain
oromero@upc.edu

Abstract. Graphs are a fundamental structure for modeling many real world domains and applications. They have emerged in various fields such as social, informational and transportation networks. The heterogeneity and dynamicity of these networks pose challenges to traditional techniques for data modeling, storage and analysis of data.
Managing graph-structured data using native graph structures and algorithms is the key for its efficient analysis. Therefore, the graph should be modeled using nodes and edges, and explored using graph algorithms, such as pattern matching and k-neighborhood.
In this paper, we introduce a novel model for management of graph data. The aim of our model is to provide analysts with a set of simple, well-defined, and adaptable components to perform complex graph modeling and analysis tasks.

1 Introduction

With the rise of BigData there has been a growing demand to design models and tools capable of handling the volume, variety and velocity of data. Particularly, the variety of data structures urges the need for equipping analysts with modeling and querying tools that are aware of the specific nature of each data model. The well-established relational model and widespread XML technologies have been developed and matured for decades. However, these technologies were pushed to their limits as the one-size-fits-all data management solutions. We experience an urgent need for techniques and industry-grade tools for efficient management of graph data. Managing and processing graph data with models where graphs are not a first-class citizen leads to poor performance. The wide adoption of the emerging NoSQL and NewSQL solutions by industry, while not yet as mature as relational model, proved the need to push databases into fields beyond the traditional business applications.

Particularly, graph databases are a major family of NoSQL databases that received a lot of interest from both industry and academia. The great expressive power of graphs, along with their solid mathematical background, encourages their use for modeling domains having complex structural relationships. Graphs

have been used for decades to model diverse domains. In biology, graphs are used for modeling metabolic pathways, genetic regulations, known as the transduction signal network, and protein interactions in a single significant graph [1]. In social networks, they are used for modeling relationships between users and mining uncovered structural information such as churn prediction [2] or marketing strategies [3].
Analysis of graph properties was deeply studied in the data mining community. A large set of algorithms computing specific graph structures and investigating specific problems was developed [4]. In the database community, multiple graph databases were designed to enable graph management. Many of them are built on top of non-native graph models, such as object-oriented or the relational models [5]. However, in recent years, the trend shifted to the development of efficient,native and relationship-oriented graph databases.

A graph database model is essential for the efficient management of graph data and is the cornerstone for building graph databases. A database model, as defined by Codd in [6], consists of a set of (1) data structures, (2) integrity constraints, and (3) operators. In a native graph database model, both the schema and the instances are modeled as graphs. Nodes and edges are first-class citizens. The model equips users by data as well as topology-aware manipulation operators. The operators handle entire graphs and do not simply return sets of disconnected nodes. Finally, the consistency of the graph data should be guaranteed by integrity constraints defined over the graph structures and maintained through the different algebraic operations [5]. The constraints should be applied on whole sub-graphs and not just single nodes or edges.

Most of the graph database models proposed on the literature ignore at least one of the three components of a complete data model [5]. The currently known graph databases, categorized under the NoSQL umbrella, are mostly built on top of property graphs.[4] A property graph is a directed, edge-labeled, attributed multi-graph. Property graphs are ubiquitous and enable embedding rich information on nodes and edges. Networks shown on the top of Figure 1 are modeled as property graphs. They are sufficient to answer queries about the average rating of a movie or detection of communities of users. Although property graphs are generic and widely accepted, the data structures they introduce are simple and oriented for storage and operational workload. For advanced analysis, richer graph structures should be introduced. Therefore, current commercial graph databases still need more improvements to meet Codd's definition. Especially the integrity constraints, guaranteeing database consistency, and a declarative language for online-querying of graph data [7].
The benefits of a data model specifically designed for graphs are manifold. The modeling phase is more natural since the structures are similar to the way end-users perceive the domain. Querying is more user-friendly and less error-prone since the operators target directly the network structure and examine specific graph measures. Moreover, this approach avoids transformation and mapping problems at both design and querying phases. Compared to manipulating the

[4] https://github.com/tinkerpop/blueprints/wiki/Property-Graph-Model

graph structures using other models, native graph databases reduce impedance mismatch problem, widely encountered by object-oriented programmers [8]. This problem is mainly perceived at two levels: the modeling level and the programming level. Here we refer to the modeling aspects, i.e. data representation and manipulation. For example, operations such as traversals are easily expressible, and are executed at a much lower cost than relational joins. Definition and implementation of concepts such as assertions are more straightforward using graph patterns, as discussed on Section 4. To the best of our knowledge, such a notion is not implemented on most current relational databases. Furthermore, implementation aspects such as indexing, storage, and matching algorithms specifically devised and tuned for graph workloads lead to better performances [9–11].

Despite the need for native graph database modeling frameworks was recognized by previous works, a complete graph database model compliant to Codd's definition is still missing. Such a framework would support the analyst on his task of modeling, integrating and getting insights on heterogeneous unstructured graph data. The aim of this paper is to provide a general framework for modeling and on-line querying of graph data. The model is generic and lays the foundations for building graph database management systems. It could nevertheless be extended to support specific applications requirements. Along the paper, we illustrate our model with concrete examples of modeling and querying scenarios. The input to our framework is a directed, labeled, attributed multi-graph.

The contributions of our work are summarized as follows:

- At the *data structures* level, we introduce a meta-model that assists the designer on his task of representing and organizing the real world entities using graphs as first-class citizens. The model is analysis-oriented, with the analysis process centered around special analytical structures, namely *analytics hypernodes* and *classes*. We embed rich semantics on the graph edges such as hierarchical and composition relationships.
- At the *integrity constraints* level, we define the constraints specific for these graph data structures. These constraints guarantee the integrity and consistency of the graph database with regard to both the meta-model and the domain.
- At the *operators* level, we propose a set of algebraic operators specific for online graph querying and analysis. Graphs are the operands and the return type of all the operators. All operations preserve the integrity constraints, this allows users to safely browse and query the network in a very flexible manner.

The rest of the paper is organized as follows. In Section 3, we introduce the meta-model and its special data structures. In Section 4, we define the different integrity constraints ensuring the graph database consistency. In Section 5, we present the algebra for querying and manipulating the graph data. Related work and extensions of our model are discussed in Section 7. Finally, Section 8 concludes the paper.

2 Running Example

We consider as a running example for this paper a movie rating network *HetRec 2011*. Initially, the network data is scattered across three different heterogeneous web sources. However, they all provide information about the same movies but from different perspectives. Each data source provides part of the context of the complex real world data entity. If the scattered information is put together, we get a more accurate and complete context about the entity. The integration enlarges the spectrum of possible queries that could not be easily expressed on separate datasets. Analysts could therefore gain new insights into the data and perform better decision making.
Our data model is equipped with flexible structures and operations to assist in the process of data integration. Networks shown on Figure 1a, 1b and 1c depict the original scattered data, while Figure 1d shows the full network, generated by the integration of the three original networks. We will describe in Section 3 the notation used for the integrated network. The dataset is an extension of MovieLens10M dataset, published by GroupLens research group.[5] It integrates the movies of MovieLens with their corresponding web pages at Internet Movie Database (IMDb)[6] and Rotten Tomatoes[7] movie review systems. The network contains movies, classified by genres, and filmed across multiple locations. Movies are linked to directors and actors ranked by importance. User provide rating and tagging for their watched movies.

3 Graph Data Model Structures

Real world graphs are heterogeneous and dynamic. Data flowing in graph data management systems either include new real-world entities, or additional information about existing entities. For example, in the case of a social network, users have their profiles continuously enriched with new personal information or new connections. The overall graph structure continues to grow with the introduction of new users, groups of users or communities etc. To support seamless integration of new information, the graph schema we adopt is semi-structured. We start by defining special graph data structures that are flexible and support integration of incoming data.

3.1 Data Strucutres

Based on our previous work [12], we propose the following types of graph elements. Note that some notation changed and elements description is enriched to fit better the analysis workload.

[5] http://www.grouplens.org
[6] http://www.imdb.com
[7] http://www.rottentomatoes.com

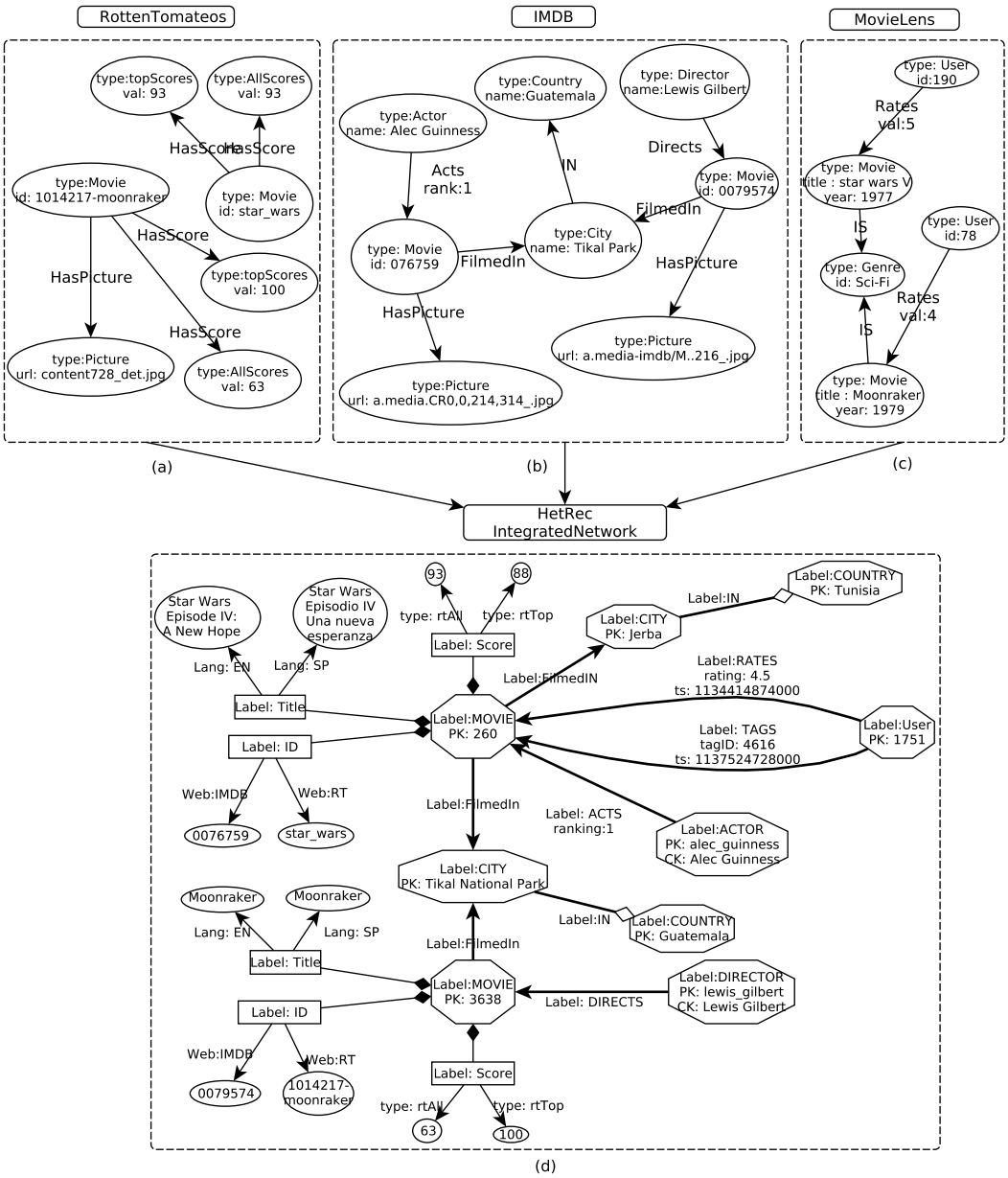


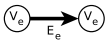
Fig. 1. Integration of Networks of Movies

Definition 1 An **entity node** is defined by a triple $\langle type, PI, SI \rangle$ where (1) *type* denotes the class of the entity node , such as user or movie, (2) PI is the primary identifier that univocally identify the entity node among the nodes with the same label, and SI is the secondary identifier. The attributes of this tuple are immutable. The set of entity nodes is denoted as V_e. □

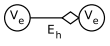
Definition 2 An **attribute node** denotes an attribute of the entity node that is not immutable. An attribute node contains only a label describing its name and reflecting the original attribute it represents. Changes are treated as punctual events and denote the values the attribute had. As in the working example, an attribute node might be the score, the title of a movie provided in multiple languages or its ID on each website. The set of attribute nodes is denoted as V_a, (in graph theory, indegree of attribute nodes is 1). □

Definition 3 Values of the attributes of real world entities are subject to change and might have different values according to circumstances change or context enrichment. A **literal node** records the value of its corresponding attribute node in a specific given context. For example, a movie might have different scores according to the rating website. The set of literal nodes is denoted as V_l, (indegree of literal nodes is 1, and outdegree is 0). □

We adopt the UML notation for relationships to represent the edges of the graph.[8] With regard to the nodes they link, we classify edges as follows.
Edges linking entity nodes are of two types:

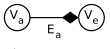
Definition 4 An **entity edge** (denoted by 

) describes the *association* between two entity nodes. As an example, relationship between movies and actors is represented by an entity edge, with the label ACTS and the attribute *ranking*, for the actor's ranking in the movie. The set of entity edges is denoted as E_e ($E_e \subseteq V_e \times V_e$). □

Definition 5 An **hierarchical edge** (denoted by 

) depicts an *aggregation* (i.e., part-of) relationship between two entity nodes. This type of nodes could be seen as an extension of entity nodes, with the support for hierarchies. Relationship between cities and countries, label IN, is hierarchical since each city is part of only one country. The set of hierarchical edges is denoted as E_h ($E_h \subseteq V_e \times V_e$). □

Both of the above edge types have a single label, a multiplicity, and a set of attributes. We denote an entity and hierarchical edge as a triple $\langle label, multiplicity, Atts \rangle$, where *label* is the name of the relationship, multiplicity is the cardinality of incoming and outgoing edges between a specific couple of nodes, and *Atts* is the set of its attributes.

Definition 6 An **attribute edge** (denoted by 

) represents a *composition* relationship, i.e. a life-cycle dependency between nodes. It keeps track of the changing attributes extracted as new nodes and do not embed attributes or labels. The set of attribute edges is denoted as E_a ($E_a \subseteq V_e \times V_a$). □

[8] http://www.omg.org/spec/UML/2.3/Superstructure/PDF/

Definition 7 A **literal edge** (denoted by ) denotes a *directed association* between an attribute node and a literal node. Literal edges are attributed, where each attribute describes part of the context for the literal node. For example, it indicates the specific language for each movie title, while the title itself is stored in the related literal node. The set of literal edges is denoted as E_l ($E_l \subseteq V_a \times V_l$). □

We introduce next two new concepts useful for analytics queries.

Definition 8 Each real world entity is represented by a specific type of hypernodes [13] which we denote as analytics hypernode. An **analytics hypernode** is an induced subgraph[9,10] grouping an entity node, all its attribute and literal nodes, and all the edges between them. This abstraction provides a coarse view of the graph. It enables high-level design and analysis of the network structure. For example, an analyst starts designing or analyzing the graph focusing on coactorship relations will not dive into the specific attributes of movie nodes. An analytics hypernode whose entity node is v is denoted as $\Gamma_v = (V, E)$, where $V \subseteq (V_e \cup V_a \cup V_l)$ and $E \subseteq (E_a \cup E_l)$. □

Definition 9 A **class** is a label-based grouping of analytics hypernodes whose underlying entity nodes share the same label. For example, all movie nodes are grouped in the Movie class. This abstraction enables straight forward grouping of network elements. □

3.2 Logical Graph Metamodel

With the graph elements clearly defined, we introduce the graph model as follows.

Definition 10 We consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \alpha, \beta, \Lambda, \lambda)$, where:

- $\mathcal{V} = \{V_e, V_a, V_l\}$ is the set of nodes.
- $\mathcal{E} = \{E_e, E_h, E_a, E_l\}$ is the set of edges.
- $\alpha : (V_e \cup V_a) \longrightarrow L_V$ is the function that returns the label for each entity or attribute node, where L_V is the set of labels of entity and attribute nodes.
- $\beta : (E_e \cup E_h) \longrightarrow L_E$ is the function that returns the label for each entity or hierarchical edge. L_E is the set of labels of entity and hierarchical edges.
- $\Lambda_{key} : (V_e \cup V_l) \longrightarrow Dom(value)$ is the function that returns the value of a node's attribute given its *key*. Λ is applied only to entity and literal nodes. $Dom(value)$ denotes the domain of *value*.
- $\lambda_{key} : (E_e \cup E_h) \longrightarrow Dom(value)$ is the function that returns the value of an edge's attribute given its *key*. λ is applied only to entity and hierarchical edges. $Dom(value)$ denotes the domain of *value*.

[9] $G_2 = (V_2, E_2)$ is a subgraph of $G_1 = (V_1, E_1)$ if $V_2 \subseteq V_1$ and $E_2 \subseteq E_1$

[10] G_2 is an induced subgraph of G_1 if all edges between V_2 present in E_1 are in E_2

The metamodel of the graph is shown on Figure 2. Figure 3 depicts the graph model of the running example shown on Figure 1d. In this graph model, *Movie* and *User* are entity nodes, while *Score* is an attribute node attached to the movies. The score keeps track of the different values of the movie's rating by website.

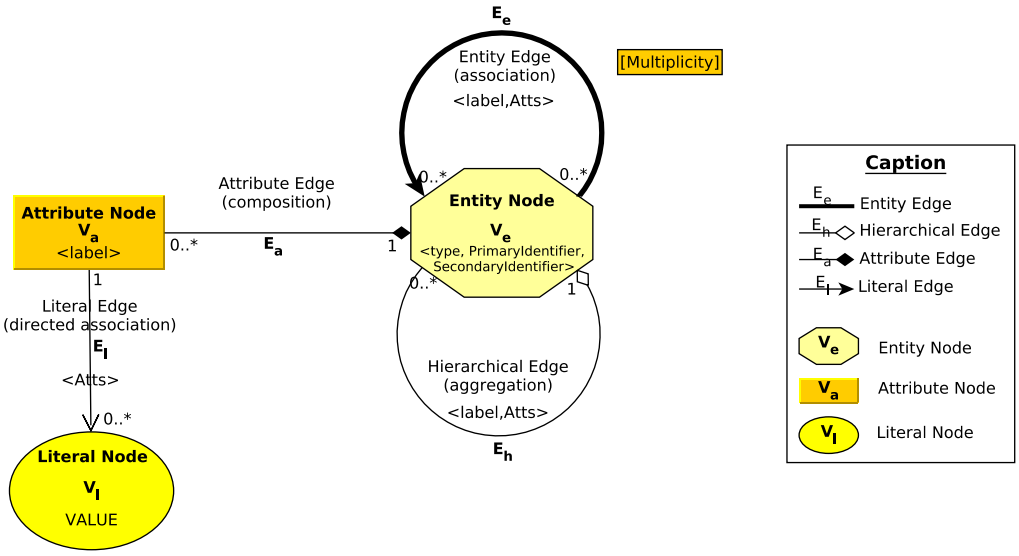


Fig. 2. Graph Representation of the Meta Model

In graphs, the distinction between schema and data instances is blurred. Current graph databases do not handle an explicit schema for the graph data, and assume that the schema is application-implicit. We adopt a semi-structured schema for graph data management. The main advantage of a semi-structured schema is the flexibility it offers for users to introduce new entities and relations without tight conditions on their internal structures and attributes. The schema is not meant to restrict the data, but rather provides a high-level description of its organization.

UML is a well-defined and accepted standard that gained a wide adoption in the modeling and database community. For a more standard representation, we introduce the UML class diagram of the data model on Figure 4.

4 Integrity Constraints

From a data modeling perspective, Codd defined integrity constraints as the set of general rules ensuring database consistency at its different states [6]. The semi-structured graph schema we propose provides mechanisms for ensuring consistency rules defined at both the meta-model and model levels. Rules at the

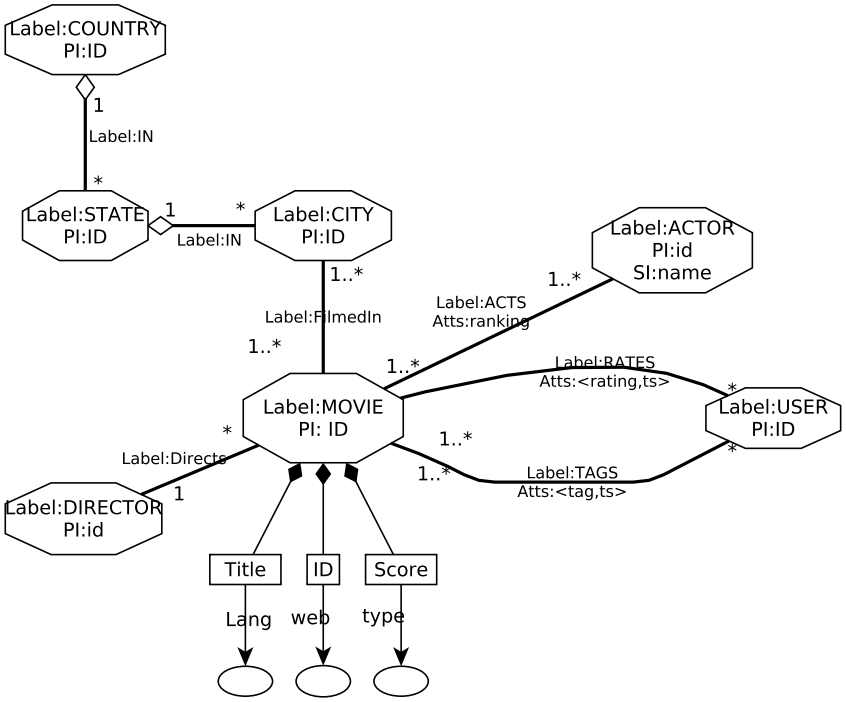


Fig. 3. Sample Network Model for Movie Rating

meta-model level serve for checking well-formedness of the graph data, while rules at the model level rather ensure the validity of the graph data.

4.1 Meta-model Constraints

The meta-model constraints describe how real world elements should be modeled using the previously defined data structures. At this level, the goal is to guarantee consistency of the graph data with regards to the meta-model defined above. Three constraints should be respected: (1) consistent typing, (2) compliant topology, and (3) existence of mandatory elements. These constraints will be used to verify the *well-formedness* of the graph.

Consistent Typing: Any element of the graph $\mathcal{G}$, must be typed using one of the following limited and predefined types:

- Nodes: $\forall v \in \mathcal{V}$, $v \in \{V_e, V_a, V_l\}$
- Edges: $\forall e \in \mathcal{E}$, $e \in \{E_e, E_h, E_a, E_l\}$

Note that the difference between type and label is that types are defined at meta-model level while labels are given by the designer at the model level.

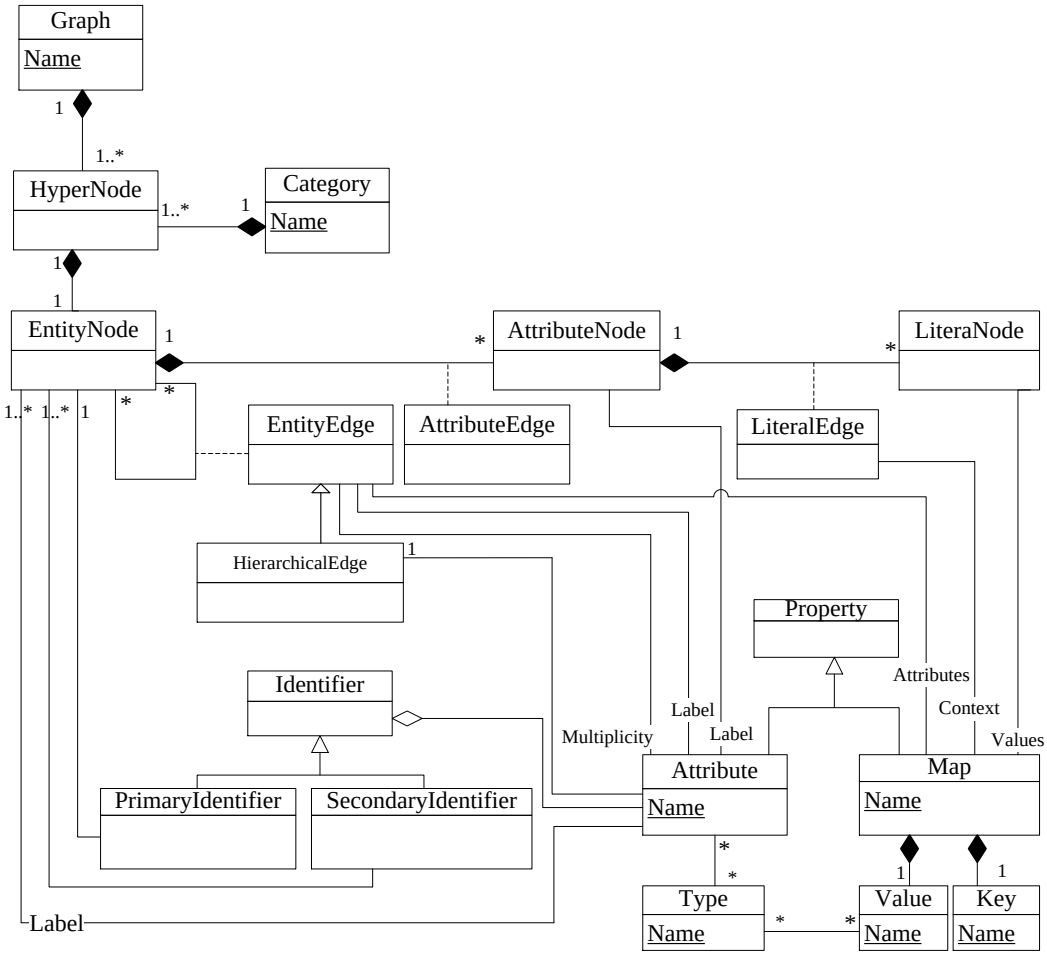


Fig. 4. UML Representation of the Data Model

Graph Topology: The graph topology is governed by the following rules:

- The definition of edges (see Section 3) strictly states the types of nodes they relate.
- Each node (resp., attribute and literal edge) is part of only one hypernode: $\forall u \in \mathcal{V}$ (resp., $e \in E_a \cup E_l$), $\exists! \Gamma_v \mid u \in \Gamma_v$ (resp., $e \in \Gamma_v$).
- Attribute nodes are only linked to their entity nodes, and InDegree(V_a)=1: $\forall u \in V_e, v \in V_a | u \in \Gamma_u$, if $\exists e \in E_a | e = (u, v)$ then $\{e, u, v\} \in \Gamma_u$ and $\exists! e | e = (u, v)$.
- Literal nodes are only linked to their attribute nodes, and InDegree(V_a)=1, and OutDegree(V_v)=0 : $\forall u \in V_a, v \in V_l, w \in V_e | u \in \Gamma_w$, if $\exists e \in E_l | e = (u, v)$ then $\{e, u, v\} \in \Gamma_w$ and $\exists! e | e = (u, v)$.
- Two edges with the same label cannot link the same pair of EnityNodes.
- An entity node has at most one hierarchical edge: OutDegree$(V_e)_{E_h} \leq 1$.

Mandatory Elements: The following properties should exist within the graph elements:

- Each EntityNode should have a PI: $\forall u \in V_e \ \Lambda_{PI}(u) \neq \emptyset$
- Each EntityNode and AttributeNode must have exactly one label: $\forall u \in (V_e \cup V_a), \alpha(u) \neq \emptyset$
- Each EntityEdge and HierarchicalEdge should have exactly one label: $\forall e \in (E_e \cup E_h), \beta(e) \neq \emptyset$

4.2 Model Constraints

The goal of the model level constraints is to guarantee the compliance of the instances with regards to the designed schema (i.e. the domain model). It ensures correct labeling, multiplicities, and assertions. This information is provided by the designer based on his knowledge of the domain. These constraints are used to verify *the validity* of the graph.

- Each class should have exactly one label different from all the labels of the other classes.
- Multiplicities define the min/max number of relations an EntityNode can have with other EntityNodes. For example we can state that an actor should participate to at least one movie, and a movie should have at least one actor. The multiplicity on the EntityEdge relating movies to actors would be: $[1..*, 1..*]$
- Assertions are graph patterns that reflect domain rules. The pattern depicts constraints to be respected by the incoming subgraphs, before being successfully added to the existing graph. For example, we can state that for a movie to be successfully added, it should have an attribute node called "Title", and be related to at least one actor and location, and exactly one director. Assertions are based on patterns, and multiplicities could be seen as a specific case of assertions.

4.3 Identification Constraints

Entity identification guarantees that a real world entity is represented only once in the model, and provides identification mechanism. This prevents redundancy of data, and help fulfilling consistent update and deletion of data entities.

- Each EntityNode PI is unique within the class of that node.
- Each EntityNode is uniquely identified by the tuple: $< label, PI >$
- EntityEdge and HierarchicalEdge are identified by the tuple: $< label, ID(V_{e1}), ID(V_{e2})) >$.
- V_a is identified by its label and its related EntityNode: $< label + ID(V_e) >$.
- E_a, E_l are identified by the nodes they are linking.

5 Graph Algebra

With the data model structures and integrity constraints defined, we introduce in this section the operators. Graph analysis enables extracting information from both the data and the network topology. It inspects both global properties, such as graph diameter and density, and local properties, such as centrality and the degree of transitivity. The analysis is mainly performed through graph pattern matching, traversals, neighborhood and reachability exploration.

Our task is to define a query language, based on an algebra that states how to express the analysis tasks described above with a limited set of operators. In this paper, we adopt GraphQL [14] algebra and adapt it to the data structures we introduced in Section 3. GraphQL operates on collections of simple graphs, similar to property graphs. The algebra was built on the lines of the relational algebra, but operates directly on graphs. The authors introduced five graph operations: Selection, Composition, Cartesian Product, Union, and Difference. Graphs are the operands and the return type of all these operators. Other shorthand operators could be defined atop, such as join and intersection.
Another interesting characteristic of GraphQL is that the authors provide a mapping between the algebraic expressions and FLWR expressions from XQuery. This gives a graph-oriented language, declarative, user-friendly, powerful in expressiveness, and specifically optimized for graph structures.

5.1 Selection

The first major operation is the selection. It is a subgraph extraction operation based on pattern matching: $\sigma_{\mathrm{P}}(\mathcal{C}) = \{\Phi_P(G) | G \in C\}$, P being the required pattern, and Φ the pattern matching function.
Subgraph selection consists on finding all subgraphs $G' = (V', E') \subseteq \mathcal{G}$ that are isomorphic to P. Subgraph isomorphism is NP-complete, however indexing techniques are used to accelerate the algorithms. Selection is a critical operation in scenarios like extracting the k-neighborhood of a node. For example, we can query for the list of co-actors and their joint movies. This could be used further

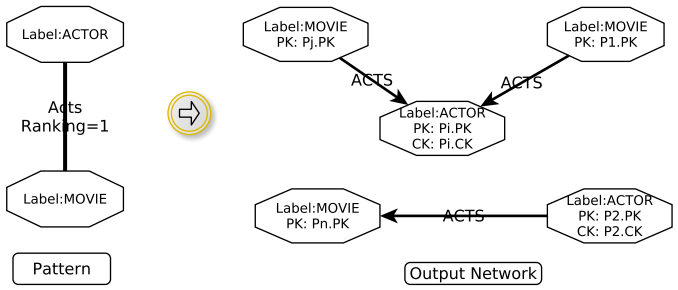


Fig. 5. Top actors network

for recommendation of actors for upcoming movies. A similar query would be the list of actors playing in movie with rank one, such as depicted on Figure 5.

For our model, we enrich selection with three specific operators to perform selection of subgraphs. They were previously defined on [12], and provide a shorthand to assist users in common graph analysis tasks.
All the proposed operators perform subgraph extraction operations. Given an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \alpha_1, \beta_1, \Lambda_1, \lambda_1)$, we denote the produced subgraph as $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \alpha_2, \beta_2, \Lambda_2, \lambda_2)$ where:

- $\mathcal{V}' \subseteq \mathcal{V}$, and $\mathcal{E}' \subseteq \mathcal{E}$
- $\alpha_2(u) = \alpha_1(u), \forall u \in \mathcal{V}'$
- $\beta_2(e) = \beta_1(e), \forall e \in \mathcal{E}'$
- $\Lambda_2(u) = \Lambda_1(u)$, $\forall u \in V'_e \cup V'_l$
- $\lambda_2(u) = \lambda_1(u)$, $\forall e \in E'_e \cup E'_h$
- $\Gamma_v \subseteq \mathcal{G}'$, iff $v \in V'_e$.

These conditions are valid for the three following operators. For the remainder of the paper, asterisk $(*)$ denotes an optional parameter that could be supplied many times and $|S|$ denotes the cardinality of a set S. We start by examining the restriction.

Definition 11 A **restriction** $\varsigma_{([\text{NLabel,AttVals}]*;\ [\text{ELabel,AttVals}]*)}(\mathcal{G})$ is a *partial*[11] subgraph extraction operation. It is applied on analytics hypernodes and the edges linking their entity nodes. It takes as input a list of the labels ($NLabel$) of the entity nodes V_e, underlying the targeted analytics hypernodes, (resp., a list of labels ($ELabel$) of their edges E_e and E_h) and the corresponding values of their targeted attributes $AttVals$. A restriction returns a partial subgraph $\mathcal{G}'$ of $\mathcal{G}$ where :

- $\alpha_2(u) \in NLabel, \forall u \in V'_e$
- $\beta_2(e) \in ELabel, \forall e \in (E'_e \cup E'_h)$
- $u \in V'_e$ iff $\alpha_1(u) \in NLabel$ and $\exists (k_i, v_i) \in AttVals | \Lambda_{1_{key}}(u) = V_i$, $\forall key = k_i$

[11] G_2 is a partial subgraph of G_1 if a subset of the edges between V_2 from E_1 is in E_2

- $e \in E'_e \cup E'_h$ iff $\beta_1(e) \in ELabel$ and $\exists(k_i, v_i) \in AttVals \mid \lambda_{1_{key}}(e) = v_i$, $\forall key = k_i$
- $u \in \mathcal{V}'$, iff $\exists \Gamma_v \subseteq \mathcal{G}' \mid u \in \Gamma_v$.

Definition 12 A **projection** $\pi_{(\text{EvLabel, } \{\text{ValSet}\})}\{\mathcal{G}, NLabel\}$ is an *induced* subgraph extraction operation. It is applied on a single class of analytics hypernodes, selected through $NLabel$. Other analytics hypernodes remain untouched by this operation. Attribute nodes whose label is not in $EvLabel$ are removed from the targeted analytics hypernodes. It further narrows the range of values in the resulting subgraph by specifying for each literal edge a key/value map of the requested values, $\{ValSet\}$. A projection returns an induced subgraph $\mathcal{G}'$ where:

- $\mathcal{E}' = \mathcal{E} \cap (\mathcal{V}' \times \mathcal{V}')$
- $u \in V'_e$ iff $\alpha_1(v) \in NLabel$
- $u \in V'_a$ iff :$\alpha_1(u) \in EvLabel$ and $\exists \Gamma_v \subseteq \mathcal{G}' \mid u \in \Gamma_v$
- $u \in V'_l$ iff: $\exists \Gamma_v \subseteq \mathcal{G}' \mid u \in \Gamma_v$ and $\exists\ e = (u, u_e), e \in E'_l \mid u_e \in \Gamma_v$ and $\exists(k_i, v_i) \in ValSet | \lambda_{1_{key}}(e) = v_i$, $\forall key = k_i$.

Definition 13 A **traversal** $\tau_{(\text{Start,Pattern})}$ is a subgraph extraction operation. A traversal starts from an entity node and follows a guided navigation on the graph according to given rules. Traversal navigates through entity nodes according to navigation rules that describe the navigation path. A navigation path is a finite sequence of conditions dictating the navigation steps. At each node ($u \in V_e$) and edge ($e \in E_e \cup E_h$), the next step is defined by an expression applied on the labels or the attributes of the current element.

5.2 Cartesian Product and Join

The cartesian product is between two collections of graphs. The output is a collection of graphs putting together couples of subgraphs of the original collections, without concatenation or unification.

A join is a selection over a cartesian product: $\mathcal{C} \bowtie_P \mathcal{D} = \sigma_{\text{P}}(\mathcal{C} \times \mathcal{D})$. Here we focus on the structural join, where the output is a new graph generated from unification of nodes and edges, according to the join condition.

5.3 Composition

The composition enables generating new graphs from existing ones. It extracts data from the original graph through pattern matching. Then it populates a graph template τ_P with the data from matched graphs:
$\omega_{\tau_P}(\text{C}) = \{\tau_P(G) | G \in C\}$.
The idea is analogous to having a function with predefined instructions, and the output depends on the actual parameters values. The goal is to reuse the information extracted from input graphs, for creating new ones. Composition is a candidate for implementing view on top of graphs. In graph databases, a view

is useful for multiple scenarios such as security and access control, data transformation and integration, and query optimization. Composition is a powerful operation that will be useful as an ETL building block for graph data warehouse. For example, in the Hetrec network, the composition enables generating the graph of co-actors from the original graph, as depicted on Figure 6.

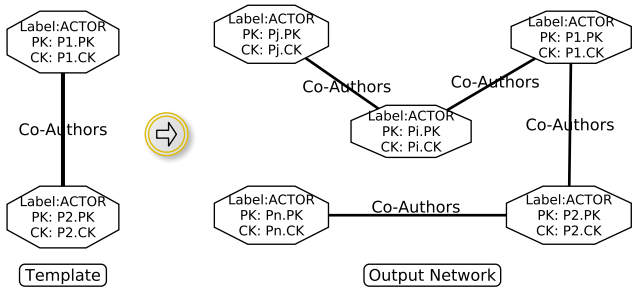


Fig. 6. Co-actorship network generation

5.4 Set operators : Union and Difference

Set operations manipulate collections instead of sets. Union simply puts together two collections of graphs. Since a collection can include the same element twice, then duplicate graphs are allowed.

5.5 Algebra: Closure, Completeness, minimalism

The original GraphQL algebra, as defined by He et al. satisfies the following two properties:

- Completeness: i.e. any query or new operator could be expressed using the previous five operators. Additional operations, encapsulating a combination of the core ones, could be defined to increase user-friendliness. However, they do not increase the expressiveness. GraphQL have equal expressive power as the relational algebra, and its nonrecursive version is equivalent to the relational algebra [14].
- Minimalism: None of the five operators could be expressed using a combination of the others.

Algebraic laws regarding commutativity and associativity remain valid since GraphQL is built on the lines of relational algebra.

The third property that should be verified for the algebra is the *closure*. It guarantees that each operator's output is a valid input for the others. In other words, the operations could be pipe-lined or nested within each other and the

result is a graph compliant to the integrity constraints defined at the meta-model level. Since we adopt more advanced data structures, we propose to adapt the operators while guaranteeing the closure property. To achieve this, we state for each operator the pre-condition, post-condition, and the rules not to violate while applying the transformation. The pre-condition and post-condition for all operators is the same: the input and output graphs should be compliant to the meta-model constraints. Compliance to the model is to be ensured by the user when applying the operators.

Union and Cartesian products are closed operations since they do not alter the internal structure of the hypernodes neither add new edges. We investigate next the remaining operators.

Difference : The model we propose dictates that if an entity node is removed, the total hypernode is removed, and if an attribute node is removed then all its literal nodes are removed. The general rule that if a node is removed its edges are removed remains valid for all nodes.

The two remaining operations are based on graph patterns, specified as follows $P = (Motif, Predicate)$:

- *Selection* extracts a subgraph according to a given pattern. The closure is satisfied if the pattern is compliant to the meta-model. The condition for selection is that the graph pattern is compliant to the meta-model. If the pattern is not compliant, the selection could extract valid data, but the result cannot be safely piped to other operations as it violates their pre-conditions.
- *Composition* instantiates a graph template, using graph data extracted from the input data using a specified pattern. The composition is closed if the template is compliant to the meta-model.

We tolerate disconnection of graphs between hypernodes. However, since hierarchies are transitive, we suggest that broken hierarchical edges could be reconstructed between remaining nodes, in order to preserve summarization.

5.6 Shorthand operations

Join is a major shorthand operation for combining subgraphs based on common criteria, and was already presented above. The author in [15] stated some common graph querying operations such as:

- Graph Summarization: It consists on consolidating a set of graph elements into a single one. This operation requires techniques capable of aggregating both data and network structures. Graph summarization is a challenging operation as it incurs topological changes to the graph. Many works have provided preliminary techniques for performing such an operation such as the k-SNAP and GraphCube operations [16, 17]. In the case of the running example, new insights on the graph data could be explored such as the shape of graph where movies are grouped by genre, or locations grouped by countries.

– Adjacency: Examines the node and edge adjacency, and explores K-neighborhood
– Reachability: Tests whether a node is reachable from a start-point within a fixed-length path, a regular simple path, or compute the shortest path.

6 Graphs for decision making

Graphs for data integration Graphs provide a generic and powerful abstraction tool for knowledge representation. The networked data model is more generic than relational tables or XML documents. Many current models are represented using graph structures. The entity relationship model [18] is a set of entity nodes related with relationships. Ontologies are also represented as graphs of interrelated primitive entities from the same domain. XML files are structured as trees, which are also specific graphs, focused on the hierarchical model rather than the networked model. The graph model we propose provides a more advanced abstraction than primitive graphs, while remaining generic. We believe it is a good candidate for modeling domains initially modeled using the other formats.

Data integration is a critical task of multiple data management systems. Our model equips analysts with powerful operators enabling the integration process while ensuring consistency of the database. Authors in [19], investigated techniques for graph integration at both instance and schema levels. For instance, Master Data Management (MDM) tools are a potential candidate for our model. MDM are used in industry as a middleware to integrate multiple data management systems and provide a unified view of the master data. Master data are critical entities shared across the organization's infrastructure. They evolve with the introduction of new data sources by, for instance, mergence or acquisitions of organizations. Designing an MDM using the model we propose enables a smooth integration of heterogeneous data structures, support for complex and rich relationships semantics such as compositions and hierarchies, and graph-oriented analysis scenarios, such pattern discovery and paths and neighborhood exploration.

Multidimensional modeling and analysis of Graphs Data warehouses provide a particularly interesting ground for graph data analysis. They provide a subject-oriented and integrated view of data. This makes them a suitable backbone for common analysis techniques such as OLAP analysis, reporting and data mining. The model we propose in this paper efficiently supports data consolidation and could serve for the data warehouse schema definition. The goal of the ETL process is the integration of the operational data into the data warehouse. ETL covers tasks such as identification, transformation, matching and insertion of the incoming nodes and edges in the graph data warehouse. As explained through the paper, the algebra, and specifically the composition and join operations are especially suited for the implementation of the transformation phase. Moreover, data quality could be ensured using the integrity constraints checking mechanisms. As in the running example, let's consider the websites as operational

graph databases. The first step is to use apply the meta-model at the movie rating network, describing the data organization inside the data warehouse. Then, we can transform and integrate the data using the operators we stated above. The integrity constraints guarantee the quality of the inserted graph entities.

7 Related Work

Graph analytics are gaining a lot of momentum in the data management community in recent years. In [5], Angles et al. surveyed a set of graph database models based on Codd's definition.

In industry, a plethora of graph database tools are developed with multiple management features. Neo4j[12] and DEX [20] are both centralized graph databases. Titan[13] is a distributed graph database. All the three are oriented for online-querying of graph data, and support similar features to relational databases such as ACID properties, indexing and query languages. They provide Blueprints[14]-compliant interfaces, and are built on top of the property graphs abstraction. HypergraphDB [21] is different in that it is built on hypergraphs, where a single edge might link an arbitrary number of nodes. Trinity [22] is distributed, in-memory key/value store supporting both online and offline graph analytics. imGraph [23] is a blueprints-compliant, distributed, in-memory graph database. These databases do not formally define integrity constraints or complete graph algebra, and therefore do not provide a formal graph database model to compare with. A major key factor in the success of relational databases is the relational algebra and its mapping to the SQL query language. However, querying of current graph databases is mainly done through their API. This approach is more programming-oriented, is application-dependent and requires more programming skills, in contrast with the common declarative approach. Graph databases such as Neo4j defined their own query language, namely Cypher. However, Cypher do not support querying simultaneously on a collection of graphs. managing and thus also querying sets of graphs. The return-type is tables instead of graphs and therefore do not support piping of graph operations.
Zhao et al. studied the issue of querying large heterogeneous information graphs. Their goal was to define an SQL-like declarative language specific for information networks. They designed two operators P-Rank, and SPath for approximate and exact subgraph matching respectively [24]. They also defined GraphCube for computing OLAP cubes aggregations on information networks [17].
However, to the best of our knowledge, apart from GraphQL, none of the previously cited works proposed a complete algebra, and formally defined a declarative query language. Hence assessing their complexity, expressive power and completeness remains difficult [7]. Moreover, as reported by lee et al., GraphQL is the only algorithm to complete all subgraph isomorphism queries, although at a slower performance compared to some of the other tested algorithms [25].

[12] http://neo4j.org/
[13] http://thinkaurelius.github.com/titan/
[14] https://github.com/tinkerpop/blueprints/wiki

Graphs on alternative database models Two approaches are widely used for managing graph data, (1) native graph data models and database engines, or (2) transformation of the graph data to fit other models, notably the relational model. The first being to specifically design native and intuitive modeling and analysis tools for graph data. The advantage of the second approach is that once data is loaded in a relational system, it gains all the benefits of the well-established relational model, with a smooth integration with the wide range of relational platforms. Relational database models being currently the most mature, Figure 7 shows the graph data management stack that we built on the lines of its relational counterpart. However, traditional relational databases fell short of meeting the requirements for complex graph data management. Due to inherent difference between the two models, transformation of graph data to relational model is a manual, complicated process. It incurs a high risk of information loss during the transformation process. Moreover, it causes a performance loss for graph oriented analysis. In relational databases, the join operation introduces a heavy workload especially for highly connected tables [7]. In graphs, the cost of running traversals is much lower than the equivalent joins in relational tables [8]. This makes graph databases more suitable for managing highly connected data compared to relational tables. Moreover, native graph models flexibility allows rich topological semantic description, such as representation of hierarchical relationships and assertions.

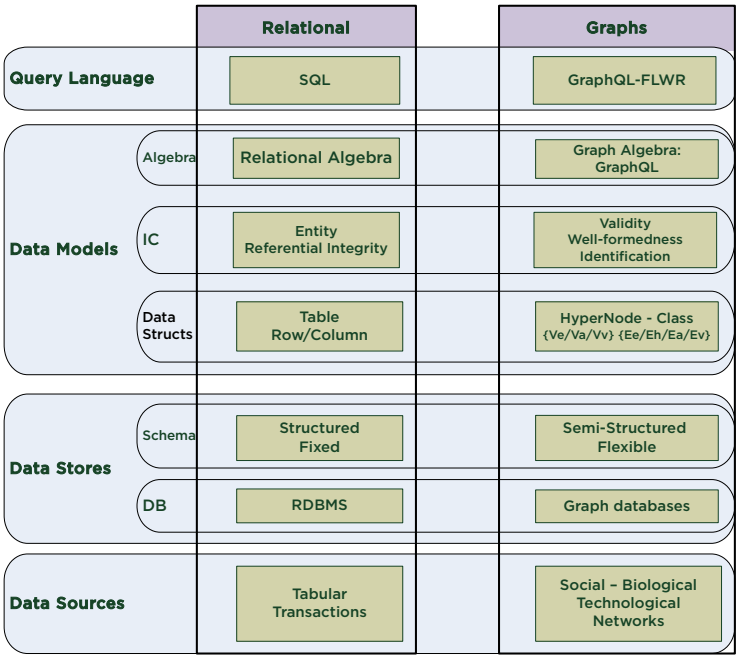


Fig. 7. Relational vs Graph Database Model

Object-oriented databases were also considered as a candidate for graph data management, with data represented as a graph of objects. But the model is focused on the objects and not their inter-relationships, and is not adapted for querying structural properties of collections of graphs. Moreover, it forces a predefined schema on the graph data [5]. At the physical layer, our model could be implemented using current graph databases built on top of property graphs. RDF is a widespread data model in the semantic web community, and could be an implementation candidate as well.

In this paper we do not argue that graph data models are a one size fits all solution. We believe that the proper way to handle BigData is to adopt a polyglot persistence approach [8]. Polyglot persistence is a paradigm, where multiple database technologies coexist in the same organization. Each technology is used to handle a subset of the data. The choice is made according to the native type of the data entities and the application managing the data. In such environments, a relational driver would manage tabular data where most queries are scans, while a graph driver would manage highly connected data where most queries are traversals.

Open Challenges The model we propose in this paper is a general model for graph database management, and provides a foundation for building atop multiple graph data management systems. Extensions could be applied at the structures, operators, or constraints to fit a certain application requirements. For example, our model could be extended to support spatio-temporal graph databases. A further step could be the support of evolving graph databases, that takes into account both data and schema changes. Graph data warehousing is also a potential application for our model. As stated earlier, master data management and ETL process are potential candidates for our model. Further extensions to the model to support OLAP-like aggregations should also be considered. This opens the research to new types of dimensions and aggregation axis on which multi-levels, multi-perspectives analysis could be performed.

8 Conclusions and Future Work

In this paper, we propose a graph database model compliant to Codd's definition. In this model, real world entities are represented as graphs and relationships are first-class citizen. We propose analysis-oriented graph structures. The algebra is designed such that graphs are the operands and the output of all operators. We defined the integrity constraints at multiple levels to guarantee data consistency. Moreover, the model equips users with mechanisms for supporting data integration and enabling ETL.

Efficient query processing on large dynamic graphs is an emerging research field that brings many challenges for future research. The model we defined here provides a foundation for future graph data management platforms. In future works, we plan to extend the model, and build atop a graph data warehousing framework. The graph data warehouse will cover both multidimensional design

and OLAP querying of graph data. Finally, the physical layer poses its open challenges. An optimized graph processing paradigm tuned to our data model as well as specific storage and indexing mechanisms are still to be developed.

Acknowledgment

This work has been partially funded by the Wallonia Region in Belgium (Grant FIRST-ENTERPRISES N° 6850).

References

1. Zimányi, E., Skhiri dit Gabouje, S.: Semantic visualization of biochemical databases. In: Semantics for GRID Databases, Proceedings of the International Conference on Semantics for a Networked World, ICSNW2004. Number 3226 in Lecture Notes in Computer Science. Springer-Verlag, Paris, France (June 2004) 199–214
2. Dasgupta, K., Singh, R., Viswanathan, B., Chakraborty, D., Mukherjea, S., Nanavati, A.A., Joshi, A.: Social ties and their relevance to churn in mobile telecom networks. In: Proceedings of the 11th international conference on Extending database technology: Advances in database technology. EDBT '08, New York, NY, USA, ACM (2008) 668–677
3. Gomez Rodriguez, M., Leskovec, J., Krause, A.: Inferring networks of diffusion and influence. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. KDD '10, New York, NY, USA, ACM (2010) 1019–1028
4. Han, J., Kamber, M., Pei, J.: Data mining: concepts and techniques, Third Edition. Volume 2. Morgan kaufmann (2011)
5. Angles, R., Gutierrez, C.: Survey of graph database models. ACM Comput. Surv. **40**(1) (2008) 1:1–1:39
6. Codd, E.F.: Data models in database management. In: Workshop on Data Abstraction, Databases and Conceptual Modelling. (1980) 112–114
7. Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., Wilkins, D.: A comparison of a graph database and a relational database: a data provenance perspective. In: Proceedings of the 48th Annual Southeast Regional Conference, ACM (2010) 42:1–42:6
8. Sadalage, P.J., Fowler, M.: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley Professional (2012)
9. Zhao, P., Yu, J.X., Yu, P.S.: Graph Indexing: Tree + Delta <= Graph. In: Proceedings of the 33rd International Conference on Very Large Data Bases. VLDB '07, VLDB Endowment (2007) 938–949
10. Tian, Y., Patel, J.M., Nair, V., Martini, S., Kretzler, M.: Periscope/GQ: a graph querying toolkit. PVLDB **1**(2) (2008) 1404–1407
11. He, H., Singh, A.K.: Closure-Tree: An Index Structure for Graph Queries. In: Proceedings of the 22Nd International Conference on Data Engineering. ICDE '06, Washington, DC, USA, IEEE Computer Society (2006) 38–
12. Ghrab, A., Skhiri, S., Jouili, S., Zimányi, E.: An analytics-aware conceptual model for evolving graphs. In Bellatreche, L., Mohania, M., eds.: Data Warehousing and Knowledge Discovery. Volume 8057 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 1–12

13. Levene, M., Poulovassilis, A.: The hypernode model and its associated query language. In: Proceedings of the fifth Jerusalem conference on Information technology. JCIT, Los Alamitos, CA, USA, IEEE Computer Society Press (1990) 520–530
14. He, H., Singh, A.: Query language and access methods for graph databases. In Aggarwal, C.C., Wang, H., eds.: Managing and Mining Graph Data. Volume 40 of Advances in Database Systems. Springer US (2010) 125–160
15. Angles, R.: A comparison of current graph database models. In Kementsietsidis, A., Salles, M.A.V., eds.: ICDE Workshops, IEEE Computer Society (2012) 171–177
16. Tian, Y., Hankins, R.A., Patel, J.M.: Efficient Aggregation for Graph Summarization. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, ACM (2008) 567–580
17. Zhao, P., Li, X., Xin, D., Han, J.: Graph cube: on warehousing and OLAP multidimensional networks. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, ACM (2011) 853–864
18. Chen, P.P.S.: The Entity-Relationship Model-Toward a Unified View of Data. ACM Transactions on Database Systems (TODS) **1**(1) (1976) 9–36
19. Lim, E.P., Sun, A., Datta, A., Chang, K.: Information integration for graph databases. In Yu, P.S., Han, J., Faloutsos, C., eds.: Link Mining: Models, Algorithms, and Applications. Springer New York (2010) 265–281
20. Martínez-Bazan, N., Muntés-Mulero, V., Gómez-Villamor, S., Nin, J., Sánchez-Martínez, M.A., Larriba-Pey, J.L.: Dex: high-performance exploration on large graphs for information retrieval. In: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management. CIKM '07, New York, NY, USA, ACM (2007) 573–582
21. Iordanov, B.: Hypergraphdb: A generalized graph database. In Shen, H., Pei, J., Özsu, M., Zou, L., Lu, J., Ling, T.W., Yu, G., Zhuang, Y., Shao, J., eds.: Web-Age Information Management. Volume 6185 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2010) 25–36
22. Shao, B., Wang, H., Li, Y.: Trinity: A distributed graph engine on a memory cloud. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. SIGMOD '13, New York, NY, USA, ACM (2013) 505–516
23. Jouili, S., Reynaga, A.: imgraph: A distributed in-memory graph database. In: Social Computing (SocialCom), 2013 International Conference on. (Sept 2013) 732–737
24. Zhao, P., Han, J., Sun, Y.: P-rank: A comprehensive structural similarity measure over information networks. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management. CIKM '09, New York, NY, USA, ACM (2009) 553–562
25. Lee, J., Han, W.S., Kasperovics, R., Lee, J.H.: An in-depth comparison of subgraph isomorphism algorithms in graph databases. PVLDB **6**(2) (2012) 133–144