Graph Database for Collaborative Communities

Rania Soussi, Marie-Aude Aufaure, Hajer Baazaoui

Graph Database For collaborative Communities

Rania Soussi[1, 2], Marie-Aude Aufaure[1], Hajer Baazaoui[2]

[1]Ecole Centrale Paris, Applied Mathematics & Systems Laboratory (MAS), SAP Business Objects Academic Chair in Business Intelligence

[2]Riadi-GDL Laboratory, ENSI – Manouba University, Tunis

Abstract Data manipulated in an enterprise context are structured data as well as unstructured data such as emails, documents, social networks, etc. Graphs are a natural way of representing and modeling such data in a unified manner (Structured, semi-structured and unstructured ones). The main advantage of such a structure relies in the dynamic aspect and the capability to represent relations, even multiple ones, between objects. Recent database research work shows a growing interest in the definition of graph models and languages to allow a natural way of handling data appearing. In this chapter, we give a survey of the main graph database models and the associated graph query languages. We then present an application using a graph database to extract social networks.

1.1 Introduction

We have now entered the knowledge era, where people work in a collaborative way and manipulate structured as well as unstructured data. More and more information about communications among people are available. This mass of information should be used in companies to optimize the business process, for example using information about people to constitute the best team for a particular project.
These tremendous amounts of data need storage and analysis. This data can resides in multiple locations and may change over time. Moreover, the data sources do not have a unified schema or their schemas cannot be controlled. Current representation and storage systems are not very flexible in dealing with dynamic changes and are not very efficient to manipulate complex data. Besides, data manipulation systems cannot easily work with structural or relational data.

Graphs are a powerful representation formalism for both structured and unstructured data, and can be seen as a unified data representation. Data in multiple domains can be naturally modeled as graphs like Semantic Web (Shadbolt *et al*. 2006), images, social networks (Xu *et al.*, 2008), bioinformatics, etc. Thus, recent database research shows a growth of interest in the definition of graph models and languages to allow a natural way of handling data appearing in these applications. Indeed, Graph database leads to a more natural modeling (graph structures) and offers a flexible support for dynamic data (Social network, web, etc...). It also facilitates data query using graph operations. Explicit graphs and graph operations allow a user to express a query at a very high level of abstraction. Queries about paths and shortest path between two nodes are performed efficiency with graph database techniques.
In this chapter, we present the main graph database models and the associated graph query languages; we will also discuss two related models that do not fit properly as graph database models, but use graphs, for example, for navigation, for defining views, or as language representation. We discuss in each section the capacity of these models and query languages to present or to query communities data especially information found on social networks. Then we show an application using a graph database for modeling social networks.

1.2 Graph database: Models and query languages

1.2.1Brief overview of Graph database models

A graph database is defined (Angles *et al.* 2008) as a "database where the data structures for the schema and/or instances are modeled as a (labeled) (directed) graph, or generalizations of the graph data structure, where data manipulation is expressed by graph-oriented operations and type constructors, and has integrity constraints appropriate for the graph structure." More formally, a graph database schema is in the form of a graph $G_{db}=(N,E,\psi,\lambda)$ where: N is a set of nodes and E is a set of edges; ψ is an incidence function from E into $N \times N$;V is a set of labels and λ is a labeling function from $N \cup E$ into V . There is a variety of models for Graph database (for more details see (Angles et al. 2008)). All these models have their formal foundation as variations of the basic mathematical definition of a graph. The structure used for modeling entities and relations influences the way to query and visualize data. In this section, we made a comparison between existing models to find the more suitable for storing and representing a Social Network. We will focus on the representation of entities and relations in these models. We present in what follows some models classified according to the data structure used to model entities and relations.

1.2.1.1 Models based on simple node

Data are represented in these models by a (directed or undirected) graph with simple nodes and edges. Most of these models (GOOD (Gyssens et al. 1990), GMOD (Andries et al. 1992), etc.) represent both schema and instance database as a labeled directed graph. Moreover, LDM (Kuper and Vardi 1993) represents the graph schema as a directed graph where leaves represent data and whose internal nodes represent connections among the data. LDM instances consist of two-column tables, one for each node of the schema. Entities, in these model, are represented by nodes labeled with type name and also with type value or object identifier (in the case of instance graph). Some models have nodes for explicit representation of tuples and sets (PaMaL (Gemis and Paredaens 1993), GDM (Hidders 2003)), and n-ary relations (GDM). Relations (attributes, relations between entities) are generally represented in these models by the mean of labeled edges. LDM and PaMaL use tuple nodes to describe a set of attributes which are used to define an entity. GOOD defines edges to distinguish between mono-valued (functional edge) and multi-valued attributes (nonfunctional edge). Nevertheless, these models do not allow the presentation of nested relations and are not very suited for modeling complex objects.

1.2.1.2 Models based on complex node

In these models, the basic structure of a graph (node and edge) and the presentation of entities and relations are based on hypernodes (and hypergraphs). Indeed, a hypernode is a directed graph in which nodes can themselves be graphs (or hypernodes). Hypernodes (Levene and Poulovassilis, 1990) can be used to represent simple (flat) and complex objects (hierarchical, composite, and cyclic) as well as mappings and records. A hypergaphs is a generalized notion of graph where the notion of edge is extended to hyper edge, which relates to an arbitrary set of nodes. The Hypernode Model (Levene and Loizou, 1995) and GGL (Graves et al, 1995) emphasize the use of hypernodes for representing nested complex objects. GROOVY (Levene and Poulovassilis, 1991) is centered on the use of hypergraphs. The hypernode model is characterized by using nested graphs at the schema and instance levels. GGL introduces, in addition to its support for hypernodes (called Master-nodes), the notion of Master-edge for encapsulation of paths. It uses hypernodes as an abstraction mechanism consisting in packaging other graphs as an encapsulated vertex, whereas the Hypernode model additionally uses hypernodes to represent other abstractions like complex objects and relations. Most models have explicit labels on edges. In the hypernode model and GROOVY, labeling can be attained by encapsulating edges, that represent the same relation, within one hypernode (or hyperedge) labeled with the relation name.

1.2.1.3 Discussion

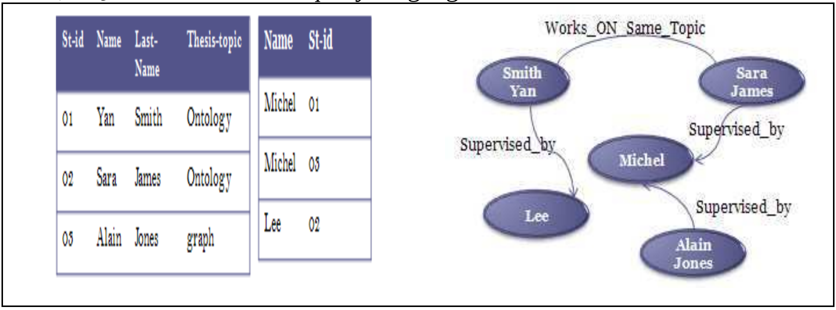
The purpose of this graph database models reviewing of is to find the most suited one to model many complex data objects and their relationships, such as social networks. Social Network is an explicit representation of relationships between people, groups, organizations, computers or other entities (Barnes, 1954). As other networks, it can be represented as a complex graph (Xu et al. 2008), $G = (V, E)$, where V is the set of nodes representing people and E is the set of edges ($V \times V$) meaning the different kind of relationships among people.
Indeed, the social network structure can contain one or more types of relations, one or more types or levels of entities and many attributes over the entities. This structure is dynamic: growth of the volume, change on attributes and relations.
Then, we have compared the previous graph database models using some characteristics related to social network: the ability to present dynamic and complex objects, nested and neighborhood relations and the ability to give a good visualization of social network. We resume the comparison on Table 1.1 where "+" indicates the graph model support, "-" indicates that the graph model doesn't support and "+/-" partial support. From this comparison, we have concluded that models based on hypernodes can be very appropriate to represent complex and dynamic object. Specially, the hypernode model with its nested graphs can provide an efficient support to represent every real-world object as a separated database entity. Moreover, models based on simple graph cannot be suitable for complex networks where entities have many attributes and multiple relations.

Table 1.1 Graph database model comparison

	Entity			Relation	Visualization
	Complex	Dynamic	Nested	Neighborhood	
Hypernode	+	+	+	+	+
Groovy	+	+	+	+	-
GGL	+	+	+	+	-
GOOD	-	+	-	-	+
GMOD	-	+	-	-	+
PaMaL	+	+	-	+	+/-
GDM	+	+	-	-	+
LDM	+	+	-	-	-

1.2.2 Graph database languages

A query language is a collection of operators or inference rules which can be applied to any valid instance of the model data structure types, following the objective of manipulating and querying data in those structures in any desired combination (Codd 1980). In this section, we review some proposals for graph database query languages found in the literature. We concentrate this study on visual, semantic, SQL-like and Formal query languages.



St-id	Name	Last-Name	Thesis-topic
01	Yan	Smith	Ontology
02	Sara	James	Ontology
03	Alain	Jones	graph

Name	St-id
Michel	01
Michel	03
Lee	02

Fig.1.1 PHD student and their supervisors (Tables and corresponding graph)

For each category, we will run some queries using the following example about a PhD student database as shown in Fig.1.1. We will show how these graph database languages support graph features (path, neighborhood, etc.).

1.2.2.1 Visual query languages

Visual query languages aim at providing the functionality of textual query languages to users who are not technical database experts, and also to improve the productivity of expert database users. In general, these languages allow users to draw a query as a graph pattern with the help of a graphical interface. The result is the collection of all subgraphs of the database matching the desired pattern (Blau et al. 2002), (Cruz et al. 1987), (Cruz et al. 1989).

a. G, G+ and GraphLog

G (Cruz et al. 1987) is a visual query language based on regular expressions that allow simple formulation of recursive queries. G enables users to pose queries, including transitive closure, which is not expressible in relational query languages. A graphical query Q (example Fig1.2) is a set of labeled directed multi-graphs, in which the node labels of Q may be either variables or constants, and the edge labels are regular expressions defined over n-tuples of variables and constants. A path is expressed on a G query initially by the means of two types of edges: dashed edges correspond to paths of arbitrary length in the graph and solid edges correspond to paths of fixed length. In G, simple paths are traversed using certain non-Horn clause constructs available in Prolog. Although, it does not support cycles, finding the shortest path or calculating node distance. In addition, G does not support aggregation functions.

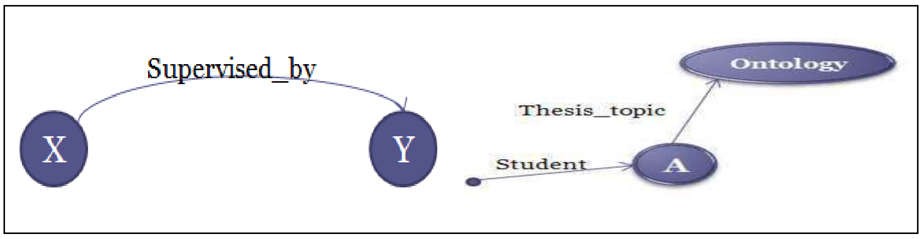


Fig. 1.2 G query to find student and supervisors and query GraphLog query to find all students working on Ontology

G evolved into a more powerful language called G+ (Cruz et al. 1989), in which a query graph remains as the basic building block. A simple query in G+ has two elements, a query graph that specifies the class of patterns to search, and a summary graph, which represents how to restructure the answer obtained by the query graph. G+ provides primitive operators like depth-first search, shortest path, transitive closure and connected components. It can easily find regular simple path. The language contains also aggregate operators that allow finding path length and node degree. The graph-based query language G+ provided a starting point for GraphLog (Consens and Mendelzon 1989). GraphLog differs from G+ with a more general data model, the use of negation, and the computational traceability. GraphLog queries are graph patterns which ask for patterns that must be present or absent in the database graph. Edges in queries represent edges or paths in the database. Each pattern defines a set of new edges (i.e., a new relation) that are added to the graph whenever the pattern is found. An edge used in a query graph either represents a base relation or is itself defined in another query graph. GraphLog supports computing aggregate functions and summarizing along paths. Fig.1.2 shows an example of a GraphLog query.

b. Hyperlog

Hyperlog (Levene and Poulovassilis 1991) is a declarative query and update language for the Hypernode Model (Fig.1.3). It visualizes schema information, data, and query output as sets of nested graphs, which can be stored, browsed and queried in a uniform way.

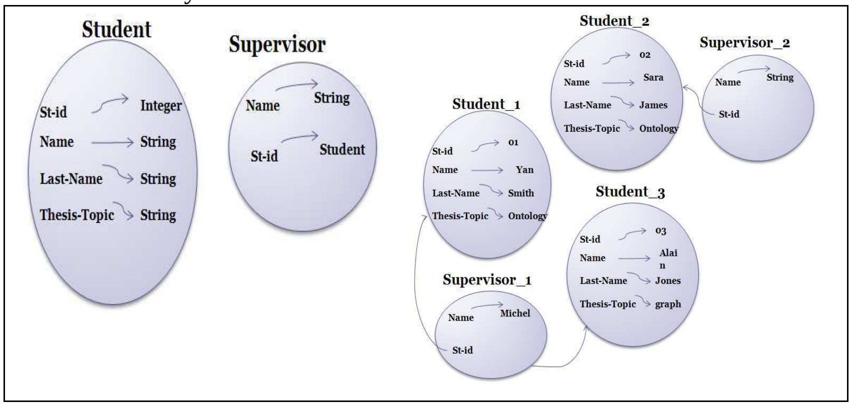


Fig.1.3 Hypernode database schema and instance

A hyperlog query consists of a number of graphs (templates) which are matched against the hypernodes and which generate graphical output.

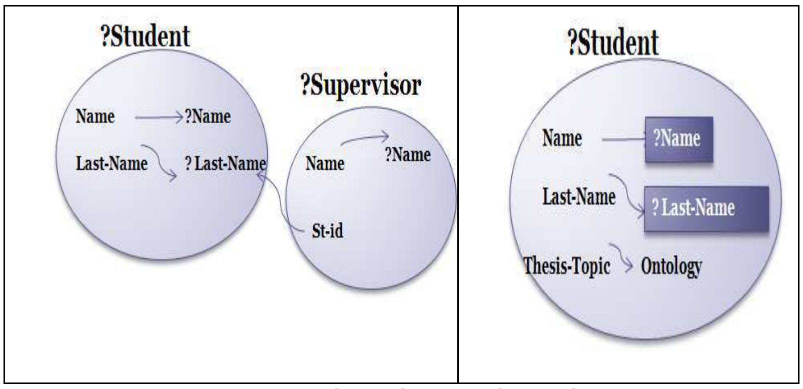


Fig.1.4 Template and query with Hyperlog

The user chooses which variables in the query should have their instantiations output in the query result. Hyperlog programs contain sets of rules. The body of a rule is composed of a number of queries, which may contain variables. The head of a rule is also a query and indicates the updates (if any) to be undertaken for each match of the graphs in the body. In order to illustrate the template and the query in the Hyperlog query language, we give an example in Fig.1.4: the template can find the students and their supervisors; the query can find the students working on Ontology. Hyperlog does not offer a special notation or expression to express paths. The existent rules can just find simple ones. The absence of aggregation functions explains the absence of answers of query about node degree or path lengths.

c. QGRAPH

QGRAPH (Blau et al., 2002) query is a labeled connected graph in which the vertices correspond to objects and the edges to links with a unique label. The query specifies the desired structure of vertices and edges. It may also place Boolean conditions on the attribute values of matching objects and links, as well as global constraints. A query consists of match vertices and edges and optional update vertices and edges. The former determine which subgraphs in the graph database constitute a match for the query. The latter determine modifications made in the matching subgraphs. A query with both match and update vertices and edges can be used for attribute calculation and for structural modification of the database. The query processor first finds the matching subgraphs using the query's match elements, and then makes changes to those subgraphs as indicated by the query's update elements.

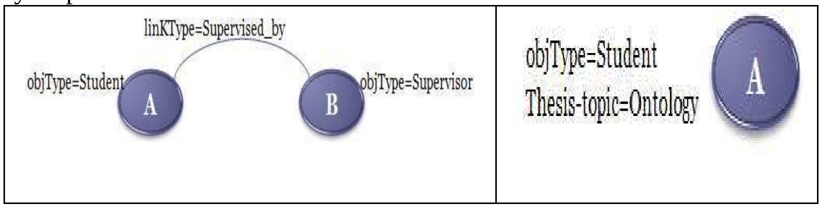


Fig. 1.5 Quries with QGRAPH

QGRAPH offers a good support to express paths by the means of sub-queries, conditions and annotations on edges and nodes. However, it does not offer operator for aggregation. Fig.1.5 contains two queries: the right query finds all subgraphs with a supervised link between a Student and a Supervisor; the left one finds just the students that have the ontology as Thesis-topic.

d. GOOD and languages based on GOOD

The Good (Gyssens *et al.* 1990) data transformation language is a database language with graphical syntax and semantics. This query language is used for the GOOD graph-based data model (Fig.1.6). GOOD query language is based on graph-pattern matching and allows the user to specify node insertions and deletions in a graphical way.

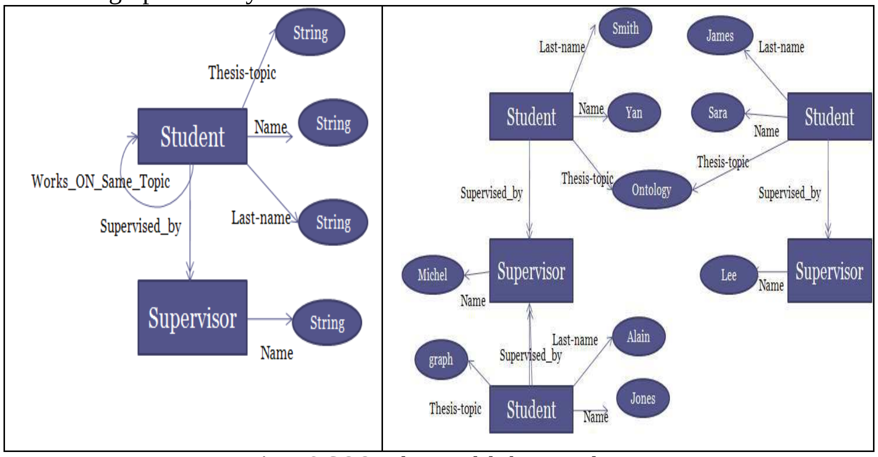


Fig. 1.6 GOOD data model shema and instance

Good contains five operators. Four of them correspond to elementary manipulation of graphs: addition of nodes and edges, deletion of nodes and edges. The fifth operation called abstraction is used to group nodes on the basis of common functional or non-functional properties. The specification of all these operations relies on the notion of pattern to describe subgraphs in an object base instance. GOOD presents other features like macros (for more succinct expression of frequent operations), computational-completeness of the query language, and simulation of object-oriented characteristics like encapsulation and inheritance.

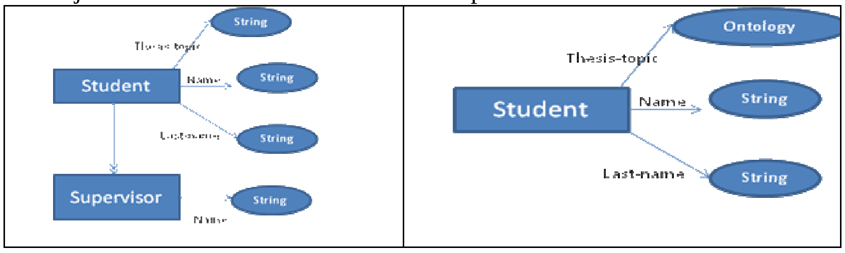


Fig. 1.7 GOOD queries

Simple path can be exprimed by using pattern. Moreaver, GOOD are not adapted to find path with no fixed length. Fig.1.7 illustrates two examples of GOOD query: First query to find student and their supervisor the secand one to find student working on ontology topic. This language was followed by the proposals GMOD (Andries et al. 1992), PaMaL (Gemis and Paredaens 1993) and GOAL (Hidders and Paredaens 1993). These languages use GOOD principal's features and add some new functionality.

1.2.2.2 SQL-like languages

SQL-like languages are declarative rule query languages that extend traditional SQL and propose new SQL-like operators for querying graphs and objects.

a. Lorel

Lorel (Abiteboul et al. 1997) is implemented as the query language of the Lore prototype database management system at Stanford (http://www-db.stanford.edu/lore).
It is used for the OEM (Object Exchange Model) data model (Fig.1.8). A database conforming to OEM can be thought as a graph where Object-IDs represent node-labels and OEM-labels represent edge-labels. Atomic objects are leaf nodes where the OEM-value is the node value. Lorel allows expressing flexible path expressions, which allow querying without precise knowledge of the structure. Path expressions are built from labels and wildcards (place-holders) using regular expressions, allowing the user to specify rich patterns that are matched to actual paths in the graph database. Lorel also includes a declarative update language.

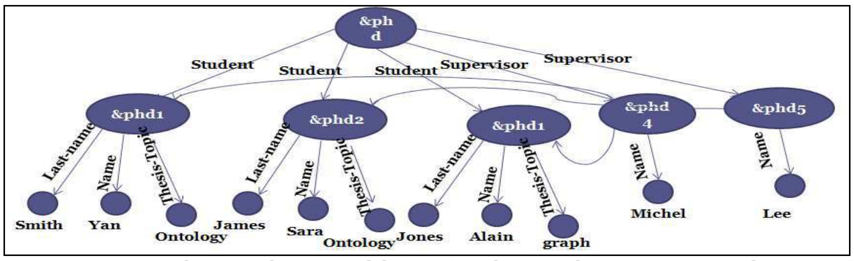


Fig. 1.8 Object Exchange Model (OEM). Schema and instance are mixed.

b. GraphDB

Guting (Güting 1994) proposes an explicit model named GraphDB, which allows simple modeling of graphs in an object oriented environment. A database in GraphDB is a collection of object classes where objects are composed of identity and tuple structure; attributes may be data or object-valued. There are three different kinds of object classes called simple classes, link classes, and path classes. Simple objects are just objects, but also play the role of nodes in the database graph. Link objects are objects with additional distinguished references to source and target simple objects. Path objects are objects with an additional list of references to simple and link objects that form a path over the database graph. GraphDB uses graph algorithms in order to implement graph operations. Shortest path and cycle both were implemented using the A* algorithm. Moreover, nodes, paths and subgraphs are indexed using path classes and index structures like B-Tree and LSD-Tree. GraphDB allows aggregation by using aggregate functions.

c. GOQL

GOQL (Sheng et al. 1999) is an extension of OQL enriched with constructs to create, manipulate and query objects of type graph, path and edge. GOQL is applied to graph database that use an object oriented data model. In this data model, they define similar to GraphDB a special type: node type, edge type, path type and graph type. GOQL is capable to query sequences and paths. In addition to the OQL sequence operators, GOQL uses the temporal operators next, until and connected for queries involving the relative ordering of sequence elements. For processing, GOQL queries are translated into an operator-based language, O-Algebra, extended with new operators. O-Algebra is an object algebra designed for processing object-oriented database (OODB) queries. To deal with GOQL‘s extension for path and sequence expressions, O-Algebra is extended with three temporal operators, corresponding to the temporal operators Next, Connected, and Until.

d. SOQL

SoQL (SOcial networks Query Language), (Ronen and Shmueli 2009) is an SQL-like language for querying and creating data in social networks. SoQL enables the user to retrieve paths to other participants in the network, and use a retrieved path in order to attempt to create a connection with the participant at the end of the path. The main element of a SoQL query is either a path or a group, with subpaths, subgoups and paths within a group defined in the query. Creation of new data is also based on the path and group structures. Indeed, SoQL presents four new operators:

-SELECT FROM PATH query which retrieves paths between network participants, starting at a specific node and satisfying conditions in the path predicates.

- SELECT FROM GROUP query which retrieves groups of participant that satisfy conditions as a set of nodes.

-The CONNECT USING PATH and CONNECT GROUP commands are presented. These commands automate the process of creating connections between participants.

The language uses Operators which specify conditions on a path or a group. It proposes also aggregation functionalities, as well as existential and universal quantifiers on nodes and edges in a path or a group, and on paths within a defined group.

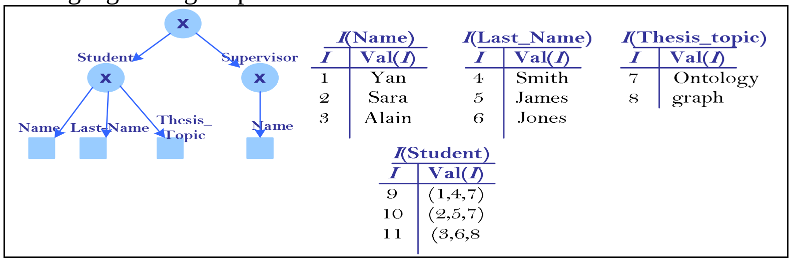
e. GraphQL

GraphQL (He and Sindh 2008) is a graph query language for graphs with arbitrary attributes and sizes. In GraphQL, graphs are the basic unit of information. Then, each operator takes one or more collections of graphs as input and generates a collection of graphs as output. It is based on graph algebra and the FLWR (For, Let, Where, and Return) expressions used in Xquery (see next section). In the graph algebra, the selection operator is generalized to graph pattern matching and a composition operator is introduced for rewriting matched graphs using the idea of neighborhood subgraphs and profiles, refinement of the overall search space, and optimization of the search order.

1.2.2.3 Formal languages

a. LDM

The Logical Database Model (Kuper and Vardi 1993) presents a logic very much in the spirit of relational tuple calculus, which uses fixed sort variables and atomic formulas to represent queries over a schema using the power of full first order languages. Fig1.9 presents the LDM schema and instances.



I(Name)

I	Val(*I*)
1	Yan
2	Sara
3	Alain

I(Last_Name)

I	Val(*I*)
4	Smith
5	James
6	Jones

I(Thesis_topic)

I	Val(*I*)
7	Ontology
8	graph

I(Student)

I	Val(*I*)
9	(1,4,7)
10	(2,5,7)
11	(3,6,8

Fig.1. 9 Logical Data Model, The schema (on the left) and part of instances (on the right)

The result of a query is another LDM schema called query schema which consists of those objects over a valid instance that satisfy the query formula. In addition the model presents an alternative algebraic query language proven to be equivalent to the logical one.

b. Gram

Gram (Amann and Scholl 1992) is an algebraic language based on regular expression and supporting a restricted form of recursion.

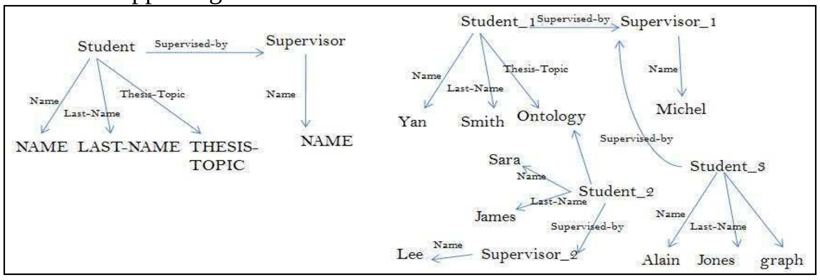


Fig.1. 10 Gram Data Model, the schema (on the left) and the instances (on the right)

Fig.1.10 shows the data model used by Gram. Regular expressions over data types are used to select walks (paths) in a graph. It uses a data model where walks are the basic objects. A walk expression is a regular expression without union, whose language contains only alternating sequences of node and edge types, starting and ending with a node type. The query language is based on hyperwalk algebra with operations closed under the set of hyperwalks.
This hyperwalk facilitates the query of paths and to find adjacent node and edge. A Gram query example is presented on Fig.1.11.

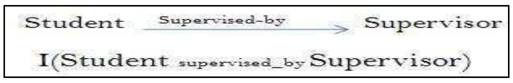


Fig.1. 11 Gram query to find student and their supervisors

c. G-Log

G-Log (Paredaens *et al.* 1995) is a declarative, nondeterministic complete language for complex objects with identity.

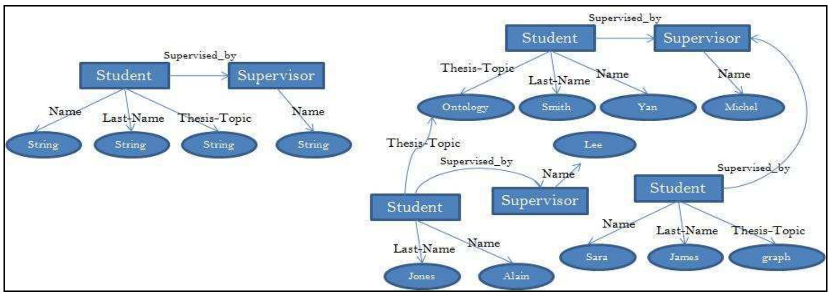


Fig.1. 12 G-Log Data Model: The schema (on the left) and the instances (on the right)

The data model of G-Log is (up to some minor details) the same as that of GOOD (Fig.1.12). The main difference between G-Log and GOOD is that the former is a declarative language, and that the latter is imperative. In G-Log, the basic entity of a program is a rule. Rules in G-Log are graph-based and are built up from colored patterns. A G-Log program is defined as a sequence of sets of G-Log rules.

d. HNQL

HyperNode Query Language (HNQL) is a query and update language for the hypernode model (Levene and Loizou 1995). HNQL consists of a basic set of operators for declarative querying and updating of hypernodes. In addition to the standard deterministic operators, HNQL provides several non-deterministic operators, which arbitrarily choose a member from a set. HNQL is further extended in a procedural style by adding to the said set of operators an assignment construct, a sequential composition construct, a conditional construct for making inferences and, finally, loop and while loop constructs for providing iteration (or equivalently recursion) facilities.

1.2.2.4 Semantic languages

A semantic query language is a query language which is defined for querying a semantic data model.

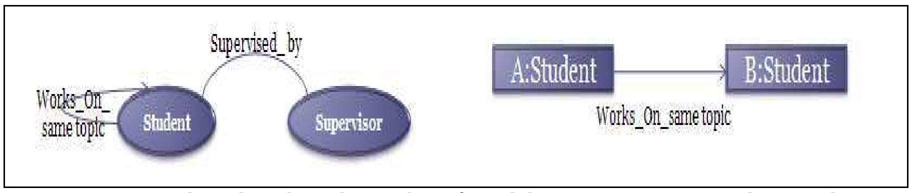


Fig. 1.13 Ontology describing the graph (Left) and the Pattern to extract students working on same topic (Right)

The semantic query language presented in (Kalpan, 2006) provides a foundation for extracting information from the semantic graph where the possible structure of the graph is described by ontology (Fig1.13) that defines the vertex types, the edge types and how edges may interconnect vertices to form a directed graph. It uses a query with a specific format containing function which specifies patterns and conditions for matching graphs in the database. Fig1.13 shows an example of pattern used by kalpan query language.

1.2.2.5 Discussion

Querying social networks turns out to be a non-trivial task due to the intrinsic complexity of the networked data. Also these kinds of querying focus on special type of information. Moreover, information needs from a community or a social network are diverse and can be categorized in two types: (1) values or measures like the centrality, diameter, etc; (2) information about attributes relations and data management on social networks. In this section, we present a comparison of the previous graph database languages and we discuss if they are well adapted to query a social network. The existing graph query languages cannot extract all the characteristics of a social network even those designed for social networks (e.g.SoQL). We resume the main characteristics of the previous languages on the following tables (Table 1.2 and Table 1.3). We put (+) where the language proposes an explicit definition for the characteristics, (-) if not and (+/-) where it try to define it indirectly.

Table 1.2 Graph query languages-1-

	G	G+ Graph-Log	Hyperlog	QGraph	GOOD	Kalpan	HNQL
Basic Unit	nodes/ edges	nodes/ edges	Hypernode	nodes/ edges	nodes/ edges	nodes/ edges	Hypernode
Data Model	graph	graph	Hypernode	graph	GOOD	Semantic graph	Hypernode
Language style	Graphical	Graphical	Graphical	Graphical	Graphical	semantic	formal
Pattern	+	+	Rules	+	+	+	-
Update query	+	+	+	+	+	-	+
Implementation	+	+	+	-	+	-	+
Path	+/-	+	+/-	+/-	+/-	+	+/-
Neighborhood	-	+	+/-	-	+/-	+	+/-
Diameter	-	+	-	-	-	-	-
Distance between nodes	-	+	-	-	-	-	-

These two tables show that:

- Many query languages use pattern to facilitate the information search process especially graphical languages like Good, Qgraph, G, etc.
- Almost languages provide operator or techniques to find path. Nevertheless, graphical languages do not determine path by direct operation.
- The neighborhood characteristic is not well processed by existing languages.
- Graph characteristics based on calculation like diameter or distances between nodes are only treated by G+ and GraphLog.

In practice, users prefer graphical languages because they are easy to use. Moreover, graphical query languages for graph model lack of operation to obtain information about communities. Languages designed for social network like SOQL are based on SQL and can be applied only on simple graphs.

Table 1.3 Graph query languages-2-

	Lorel	GOQL	SOQL	GraphDB	GraphQL	LDM	Gram	G-Log
Basic Unit	Object	nodes/ edges	Group/ path	nodes/ edges	graph	tuple	nodes/ edges	nodes/ edges
Data Model	OEM	graph	graph	graph	graph	LDM	Gram	G-Log
Query style	SQL-like	SQL-like	SQL-like	SQL-like	SQL-like	formal	formal	formal
Pattern	-	-	+	+	+	-	-	+
Update query	+	-	+	+	-	-	-	-
Implementation	+	+	+	+	-	-	+	-

Path	+	+	+	+	+	-	+	+/-
Neighborhood	+/-	+/-	+/-	+/-	+	-	+/-	+/-
Diameter	-	-	+/-	-	-	-	-	-
Distance between nodes	+/-	-	+	-	-	-	-	-

1.3Related Data Model

1.3.1 RDF query languages

RDF (Miller et al., 2004) is a knowledge representation language dedicated to the annotation of documents and more generally of resources within the framework of the Semantic Web. By definition, an RDF graph (Klyne et al., 2004) is a set of RDF triples. An RDF triple is a triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ where *I*, *B*, and *L* are sets that represent (IRIs), Blank nodes, and Literals, respectively). In this triple, *s* is the subject, *p* the predicate, and *o* the object.
RDF models information with graph-like structure, where basic notions of graph theory like node, edge, path, neighborhood, connectivity, distance, degree, and so on play a central role. RDF has been used for presenting communities and social network (e.g FOAF[1], RELATIONSHIP[2], etc). RDF can be a good support to model social network although its query languages do not offer an efficient support to query this kind of data. Indeed, several languages for querying RDF documents have been proposed, some in the tradition of database query languages (i.e. SQL, OQL): RQL (Karvounarakis et al, 2002), SeRQL (Broekstra et al, 2003), RDQL (Seaborne, 2004) SPARQL (Perez et al, 2006). Others more closely inspired by rule languages: Triple (Sintek et al, 2002), Versa[3], N3 and RxPath[4]. The currently available query languages for RDF support a wide variety of operations. However, several important features are not well supported, or even not supported at all. RDF query languages support only querying for patterns of paths which are limited in length and form. Nevertheless, RDF allows representing irregular and incomplete information (e.g the use of blank node). From the original approach just Versa and SeRQL provide built in means for dealing with incomplete information. For example, the SeRQL language provides so-called optional path expressions

1 http://www.foaf-project.org/

2 http://vocab.org/relationship/.html

3 http://wiki.xml3k.org/Versa

4 http://rx4rdf.liminalzone.org/ RxPathSpec

(denoted by square brackets) to match paths whose presence is irregular. Usually, such optional path expressions can be simulated, if a language provides set union and negation. Others works on RDF query languages try to extend the original languages to improve path expressiveness. For example in (Alkhateeb et al, 2009), they allow to query an RDF knowledge base using graph patterns whose predicates are regular expressions. In RDF Path[5], N3 and Graph Path[6], they try to use specifications similar to those in XPATH to query paths in RDF. Moreover, RDF query languages are not well adapted to query path with unknown length or including multiples propriety on RDF graph. Neighborhoods retrieving cannot be well done for languages that do not have a union operator. Many of the existing proposals support very little functionality for grouping and aggregation. Moreover, Aggregated functions like COUNT, MIN, MAX applied to paths could be used to answer queries in order to analyze data (like the degree of a node, the distance between nodes, and the diameter of a graph). We can find exceptions in Versa, RQL and N3which support count functionality Aggregation in path and nodes are not explicitly treated by any languages which need to be considered as a requirement

1.3.2 XML query languages

The Extensible Markup Language (XML) is a subset of SGML. XML data are labeled ordered trees (with labels on nodes), where internal nodes define the structure and leaves the data (scheme and data are mixed.). XML additionally provides a referencing mechanism among elements that allows simulating arbitrary graphs. In this sense XML can simulate semi-structured data. Also, many new extension of XML are designed to represent graphs like GML, GraphML, XGML and etc.

Current query languages (Bonifati et al, 2000) for XML do not support the majority features for graph-structured XML document. The principal feature supported is path. For example, XPath[7] uses path expressions to select nodes or nodesets in an XML document. Also, the set of axes defined in XPath is clearly designed to allow the set of graph traversal operations that are seen to be atomic in XML document trees. An XPath axis is fundamentally a mapping from nodes to nodesets and defines a way of traversing the underlying graph. Each axis encapsulates two things: a type of edge to follow (eg. child vs. attribute) and whether to follow it transitively (e.g. child vs. descendant). Also, XQuery[8] uses XPath to express complex path and supports flexible query semantics. In XML-QL (Deutsch et al,1999), path expressions are admitted within the tag specification and they permit the alternation, concatenation and Kleene-star operators, similar to those used in regular expressions. In XML-GL (Ceri et al, 1999), the only path expres-

5 http://infomesh.net/2003/rdfpath

6 http://www.langdale.com.au/GraphPath/

7 http://www.w3.org/TR/xpath

8 http://www.w3.org/TR/xquery/

sions supported are arbitrary containment, by means of a wildcard* as edge label; this allows traversing the XML-GL graph reaching an element at any level of depth. However, Current query languages for XML are designed for tree-structured XML data and do not support the matching of schema in form of general graph. Even though XPath can express a node with multiple parents by multiple constraints with axis “parent", it cannot express a graph with cycles. While XML won't allow multiple parents, there's nothing in XQuery (or in particular, XPath) which precludes a traversal from parent to child to a different parent. This insufficiency does not allow the presentation and the query of all kind of graphs specially those on social network.

1.4 Social network Extraction from relational database using a graph database

Social Network is an explicit representation of relationships between people, groups, organizations, computers or other entities and it is modeled by a graph (see section 1.2.1.3). There are many ways to obtain a Social Network. The approaches presented in the literature for Social Network extraction use a specific type of data source to extract people and relations among them (Kirchhoff et al. 2008). Most of these data sources come from the Web. However, some problems related to the extraction of Social Network from various information sources available on the World Wide Web still remains. First, a general problem is the identification of people because of different naming standards or same names assigned to different persons. The social context and the type of social interactions among people within these information sources need to be carefully analyzed in order to obtain a meaningful understanding of the underlying Social Network structure. Moreover, data from the web are often not well reliable because anyone can add information; also in some case we cannot easily collect information from the web due to privacy issues.

Nevertheless, in the context of business, important expertise information about people is not stored on the Web. Such information is stored in files, databases and especially relational databases. Relational database is a rich source of data, but it is not well adapted to store and manipulate social network data. Indeed, the relational model was directed to simple record-type data with a structure known in advance. The schema is fixed and extensibility is a difficult task. Thus, they might require very sophisticated and expensive operations, such as renormalization, re-indexing etc., which may not be performed automatically. Schema renormalization in such cases is neither desirable nor easy to do. The standard query and transformation language for the relational database is SQL which does not support paths, neighborhoods and queries that address connectivity (an exception is transitivity). These graph features will facilitate the application of social network analysis algorithm. Also, it will allow to response queries such as who owns the information, who has the leadership, who is an expert in a particular domain and etc.

Such information is very important for business applications. Then, enterprises need to extract their Social Network from the existing relational database to store, update and retrieve information in a simple way as graphs. On the other hand, extracting social network from relational database is not just a translation of relational database into a simple graph structure. The resulting Social Network should contain detailed information about people and their relations. As we have shown in the previous section Graph database can be a good representation for social networks and facilitate its querying. There are many approaches that transform relational databases to other structure having graph-like features like RDF, XML or even ontology, but not into a graph databases. Then, in this section, we will present our approach to transform a relational database to a social network using a graph database.

1.5.1 Converting relational database into hypernode database

Having a graph database instead of relational database will provide a more clear view of existents entities in the initial database. Indeed, all these entities will be presented on the form of nodes and the relations between them will be outlined which facilitate in further steps the selection of the desired entities. Also, nodes in graph database can encapsulate all the attribute of entities in the same node and give us a simple graph of entities. From this graph of entities, a social network can be extracted. Using the comparison between existing graph database models (Table1.1), we have chosen to work with the hypernode model (Levene and Loizou 1995)because the hypernode database with its nested graphs can provide an efficient support to represent every real-world object as a separated database entity.

The relational database transformation into a graph database includes schema translation and data conversion (Maatuk et al. 2008). The schema translation can turn the source schema into the target schema by applying a set of mapping rules. In our work, we propose a translation process which directly transforms the relational schema into a hypernode schema. Data Conversion process of converts data from the source to the target database based on the translated schema. Data stored as tuples (Rows) in relational database are converted into nodes and edges in graph database. This involves unloading and restructuring relational data, and then reloading them into a target database in order to populate the schema generated during the translation process. In what follows, we will detail these two steps.

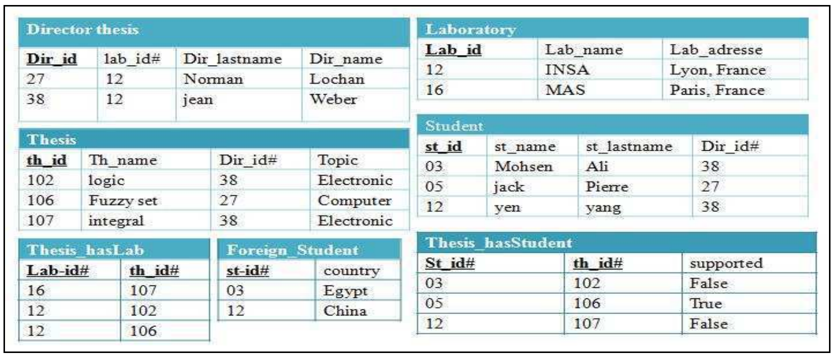
1.4.1.1 Schema Translation

The first step consists in extracting the relational database schema using the schema metadata of the relational database management system (information about tables and columns) which is extracted using SQL queries. The idea is to identify the primary key, composite key(s) and foreign key(s) of each relation.

This information is then used to design the new schema (hypernodes and relations within and between them). This process is performed by the following steps.

Step1: Relational schema extraction. In this step, information from the relational database is extracted using SQL queries. In our approach a relational schema is represented as a set of relations (tables) = $\left\{TR \backslash \mathrm{TR}{:=}\left\langle \mathrm{r_n}, A\ , K_{p,F} \right\rangle\right\}$, where:

- $\mathrm{r_n}$ denotes the name of *TR*.
- *A* denotes a set of attributes of *TR* and gives information about each attribute integrity constraints, A :={ a\a :=<a_n, *t, ce, cp, n, d*>}, where a_n is an attribute name, *t* is its type, *ce* mentions if *a* is a foreign key or not, *cp* mentions if *a* is a primary key or not, *n* mentions if *a* can be null or not and *d* is a default value if one is available.
- $K_{p,F}$ denotes a set of key of *TR* and gives information about each Key integrity constraints, $K_{p,F}$:={ β | β := <*kr, ce, cp, re* ,f_a}>}, where: β represents a key (an attribute which can be a key or a part of a composed key), *kr* is the name of a key attribute, *ce* indicates if β is a foreign key or not, *cp* indicates if β is a primary key or not, *re* is the relation that contains the exported primary key, f_a is the attribute name of the foreign key.


Director thesis

Dir_id	lab_id#	Dir_lastname	Dir_name
27	12	Norman	Lochan
38	12	jean	Weber

Laboratory

Lab_id	Lab_name	Lab_adresse
12	INSA	Lyon, France
16	MAS	Paris, France

Thesis

th_id	Th_name	Dir_id#	Topic
102	logic	38	Electronic
106	Fuzzy set	27	Computer
107	integral	38	Electronic

Student

st_id	st_name	st_lastname	Dir_id#
03	Mohsen	Ali	38
05	jack	Pierre	27
12	yen	yang	38

Thesis_hasLab

Lab-id#	**th_id#**
16	107
12	102
12	106

Foreign_Student

st-id#	country
03	Egypt
12	China

Thesis_hasStudent

St_id#	**th_id#**	supported
03	102	False
05	106	True
12	107	False

Fig. 1.14 Relational database schema (primary key is underlined and foreign key is marked by "#"

This schema provides an image of metadata obtained from an existing relational database and provides more information than a traditional schema. Indeed, it gives information about primary and foreign keys to facilitate relations extraction in further steps. For example for the Table "*Thesis*" (database in Fig.1.14), the relation *Thesis (th_id, Th_name, Topic)* was extracted. Once the schema is extracted, we can generate the corresponding hypernode schema (step 2).

Step 2: Mapping the relational schema to the hypernode schema. We use on this step a hypernode schema which is an extension of the original one.

A hypernode is defined (Levene and Loizou 1995)by *H* = (*N*, *E*), where *N* is a finite set of nodes containing primitive nodes and further hypernodes, and *E* is a set of edges between members of N, Such that $N \subseteq A \cup L$ (where A is the set of atomic values and L the set of labels) and $E \subseteq (N \times N)$.

A Hypernode database (*HD*) is a finite set of hypernodes which satisfies these following conditions:

(1) The hypernode label is unique in *HD*.

(2) $\forall$ H a label in the label set of *HD*, $\exists h \subset HD$ whose defining label is H.

The Hypernode model does not use labeled edges, the task of representing relations (and their names) can be attained by encapsulating edges, that represent the same relation (same label edges), within one hypernode labeled with the relation-name. However, the traditional presentation of social network is labeled node attached with explicit labeled edge. Then, we extend the *HD* to *LHD* (Labeled hypernode database) by adding explicit labels to edges. $LHD = HS \cup ES$, where:

(1) *HS* is a finite set of hypernode.

(2) *ES* is a set of edge where $ES \subseteq (HS \times HS)$ and $\forall\ e \in ES$, *e* has a label.

In this step, we use a hypernode database schema composed by the union of two sets: $\{H \setminus \mathrm{H} := \langle \mathrm{h_n}, Nh \rangle\} \cup \{R_h \setminus \mathrm{R_h} := \langle \mathrm{r}, \mathrm{h_s}, \mathrm{h_d} \rangle\}$

The first one is the set of hypernodes where:

- $\mathrm{h_n}$ denotes the name of *H*
- *Nh* denotes a set of nodes *Nh* :={ n\n :=<$\mathrm{n_n}$, *t, ce, cp*>} where $\mathrm{n_n}$ is the node name, *t* is the type, *ce* mentions if the nodes contains a foreign key (in the relational schema n is a foreign key) and *cp* mentions if the nodes contains a primary key.

The second one is the set of relations where:

- *r* denotes the name of *R*
- $\mathrm{h_s}$ denotes the hypernode source name
- $\mathrm{h_d}$ denotes the hypernode destination name

To extract this schema, we start by identifying the hypernodes then their relations.

Hypernode identification. Using the relational schema, we create from each table *t* $\in$ *TR a* new hyprnode *h.* h owns the same characteristics of *t:* same name and attributes. Indeed, each attribute a from the table TR is transformed into a node n in h where n contains all the characteristics of a (name, type, etc). If the attribute is a foreign key, its type is changed to be the name of the exported relation.

Relation identification. In order to identify the relations between the identified hypernodes, the nodes set *Nh* of each hypernode *h* is analyzed. For each node, we verify if it contains a foreign key in order to search existent dependency with other hypernodes. We have identified four relation types:

- **"IS-A"** relation: if *h* has only one node n_{pf} and no more, that contains a key which is primary (a simple one) and foreign key, then *h* shares the relation "IS-A" with the hypernode mentioned in the n_{pf} type; e.g. the hypernode "Foreign_Student" contains the node "ST_id" which is a primary key and a foreign key, then “Foreign_Student” shares the relation “IS-A” with “Student.”

- ***“Part_of*** ” relation : if *h* has more than one node n_{pf} that contains a key which is primary and foreign key, then h shares a *Part-of* relation with each hypernode mentioned in the n_{pf} type. e.g. “*Student*” and “*Thesis*” are “Part-of” the hypernode “*Thesis_hasStudent*” because “*Thesis_hasStudent*” contains the nodes “*St-id*” with type “*Student*” and “*Th-id*” with type “*Thesis*”.
- ***R*** relation: this kind of relation is a particular case of the ***“Part_of”*** *relation*. When the hypernode is composed only with nodes which contain a key which is primary and foreign key, then we delete this hypernode and we use its name to build relations between the hypernodes mentioned in the n_{pf} type. e.g. the hypernode “*Thesis_hasLab*” is deleted and is transformed into a relation between “*Thesis*” and “*Laboratory*”.
- **“”** relation: if *h* contains a node which contains a foreign key, *h* has a relation with the hypernode mentioned in the type of the node. In this case, we are not able to give a name to this relation.

Considering the initial database, Fig 1.15 shows the resulting Hypernode schema.

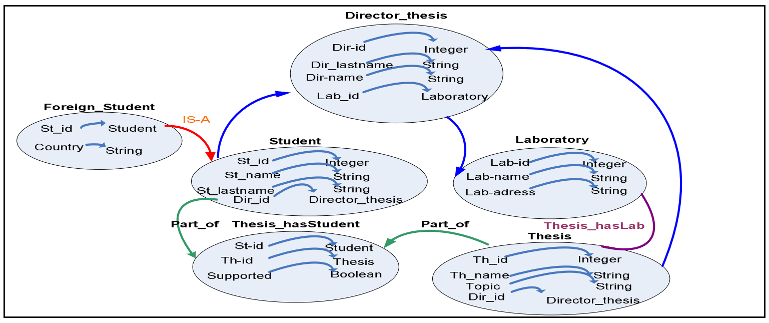


Fig.1.15 Hypernode database schema

1.4.1.2 Data conversion

In order to instantiated the hypernode database schema already identified, Data conversion is performed in three steps. First, the relational database Table’ tuples are extracted. Second, these data are converted to match the target format. Then, for each hypernode in the hypernode database, a set of instances hypernode *HI* is extracted from the relational tuples.

The set of instances hypernode *HI* is defined by $HI = \{H_i \setminus \mathrm{H}_i := \langle \mathrm{h}, \mathrm{h}_\mathrm{i}, N_{hi} \rangle\}$where:

- H_i denotes the instance hypernode
- h denotes the hypernode source name.
- h_i denotes the name of H_i.

- N_{hi} denotes a set of nodes N_{hi} :={ n\n$_i$:=< n$_n$, *t*, *val*>} where n$_n$ is the node name, *t* is the type, and *val* mentions the node value.

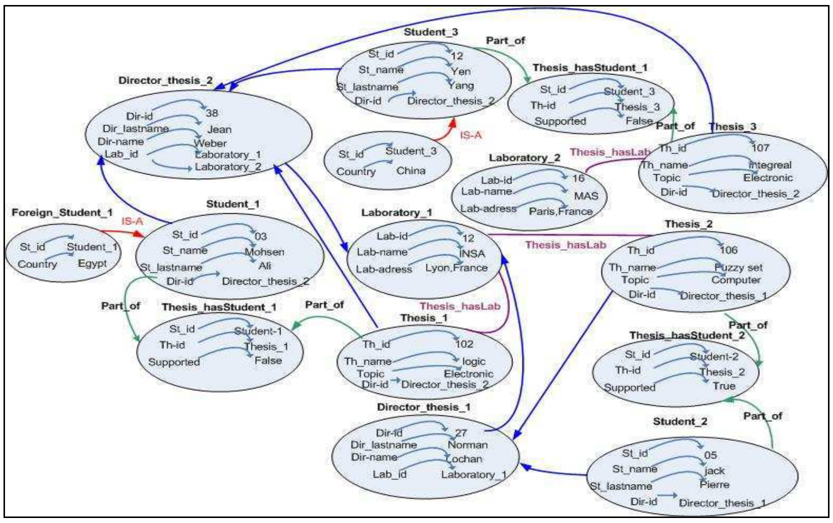


Fig. 1. 16 Part of the Hypernode database instance

For each relation in the *LHD*, a set of instance relations *RI* is extracted using the value of keys on the relational tables. *RI* is defined by *RI*:={ r$_i$\r$_i$:=< r, h$_{is}$, h$_{id}$ >} where:

- r denotes the relation which is intanciated by r$_i$.
- h$_{is}$ denotes the hypernode instance source.
- h$_{id}$ denotes the hypernode instance destination.

Finally, transformed data are loaded into the *LHD* schema. An excerpt of the HD is shown in Fig.1.16.

1.5.2 Social network extraction

Using the result Hypernode database from the previous step, the social network is extracted. This phase passes through two steps: (1) Entities (people) identification and (2) Detection of relations among people.

The social network is defined by: *SN* = (*ESN*, *RSN*) where:

• *ESN* is a finite set of entity such *ESN* :={ e\ e ∈ *HI*, e :=< h, e$_n$, N$_e$>} where h is the hypernode which represents *e*, e_n is *e*'s name and N_e is the set of *e*'s node.

• *RSN* is a finite set of the relations between entities such *RSN*= {r$_{sn}$\ r$_{sn}$:=<n,e$_1$, e$_2$>}where *n* is the relation name, e$_1$ and e$_2$ ∈ *ESN*.

In what fallows, we will describe the two step of the social network extraction.

1.4.2.1 Entities identification

Entities identification is the process to identify hypernodes that contain entities which compose the social network. In this step, we describe the process to identify people. The hypernode database schema is used to extract candidate hypernodes (hypernodes which may be contain persons). Then, the hypernodes instances are used to deeply analyze the candidate hypernodes and detect those containing people.

Candidate hypernodes detection. A person has a number of characteristics like name, surname, birthday, address, email, etc. Some of these characteristics are used when designing databases containing persons. We collect these characteristics from various ontologies such FOAF ontology and person ontology (schema-Web[9]) and we manually build a person ontology (PO) containing all these characteristics and their synonyms (collected from WordNet). Using the person ontology, the set of nodes related to each hypernode in the *LHD* is analyzed.

- If the node's name is one of the PO concepts, the number of characteristics for this hypernode is incremented.
- If the number of characteristics for the hypernode >=1 and one of them contains a name, the hypernode *h* is a candidate to contain persons.

Candidate hypernodes Analysis. Each candidate hypernode has a set of instance hypernodes h_i. In order to analyze the name found in each instances hypernode (we take just the 10 first entities), the name is send to the web search engine (Bing API). The top 10 returned documents is downloaded and parsed using DOM[10]. Each document is analyzed using the NER (Named entity Recognition) proposed by Stanford[11] and which put three kinds of tags (Person, location or organization). We give for each document a rank *rd*. If the name is tagged in the document by Person, the document is ranked by *rd*=1 else *rd*=0. The average assigned to the name found in the hypernode instance h_i (avghi) counts how many times is considered as a person name in the documents (where the tag of this name is Person)

$$avghi = \frac{\sum rd}{number\_documents} \qquad (1.1)$$

The average assigned to the hypernode (avgH) calculates the average where the names found in its hypernode instances are considered as a person name:

$$avgH = \frac{\sum avghi}{number\_hi} \qquad (1.2)$$

In order to identify persons, we use the NER proposed by standford: in which precision is in the average of 90%to find Person entities; so, a hypernode is considered as representative of a person if more than 60% of its instances contains a person name (we take only 60% as a threshold due to problems such as wrong written name use of abbreviations, etc. which decrease the precision of NER).

9 http://ebiquity.umbc.edu/ontology/person.owl

10 http://www.w3.org/DOM/

11 http://nlp.stanford.edu/ner/index.shtml

1.4.2.2 Building relations

After the identification of the entities set *ESN*, we use the existent relations in the hypernode database (the relations which share *ESN* elements with other hypernodes or among them) to find the set of relations *RSN*.

In order to facilitate this step, we have designed a set of patterns to apply this kind of transformation to all the relation on the hypernode database. The pattern will enumerate all the existent relations between persons only by using the hypernode database schema. After the relation pattern identification, we will search the correspondent relations on the instances database.

A pattern relation *Pr* is defined by *Pr*=<*nPr*, hp_1, hp_2, h_{in}> such as *nPr* is the name of the relation, hp_1 and hp_2 the hypernodes which share the relation hypernodes which represent people), h_{in} a mediator for this relation (the hypernode used to identify the relation).

For each relation $R_h \in$ set of *LHD* relations, we check these conditions:

1. If **R_h:=<"IS-A",h_s,h_d> where h_s or $h_d \in$ *ESN*** then h_s or h_d is added to *ESN*. The relation "IS-A" allows to find hidden entities which are not identified in the previous step. In the relation construction process, we start by analyzing this kind of relation to find in the next steps the relations related to the new discovered entity.

2. If **R_h:=<r,h_s, h_d> where h_s and $h_d \in$ *ESN*** then two patterns are identified:
   - 2.1 Pr_1:=<r, h_{is}, h_{id},null>, if two entities (h_s and h_d) are already connected in the *LHD*, we will search if their instances (h_{is} and h_{id})are connected, too.
   - 2.2 Pr_2:=<Same_hd.name, hp_i,hp_j, h_d > where hp_i= h_{is} , hp_j= h_{is} and i!=j. Pr_2 represents the relations between the instances of h_s which can be connected with the same instance of h_d.

3. If **R_h:=<r h_s,h_d> where $h_s \in$ *ESN* and $h_d \notin$ *ESN* then:**
   - 3.1 If r != "Part-of" then pattern Pr_3 is extracted :Pr_3:=<Same_h_d.name, hp_i,hp_j, h_d > where hp_i= h_{is} , hp_j= h_{is} and i!=j. we search the h_s instances which are connected with the same instance of h_d.
   - 3.2 If r = "Part-of" then the hypernodes which are "Part-of" h_d are researched: Firstly, for each $h_j \in$ {h\h has the relation R_h:=<"Part-of", h_j, h_d>}, a new node is added to h_s containing the name of h_j then the pattern Pr_4 is extracted. Pr_4:=<Same_h_j.name, hp_1,hp_2, h_j > where hp_1= h_s , hp_2= h_s. Pr_4 represents the relations between the instances of h_s that share the same value of $h_{j.}$

4. If **R_h:=<r h_s,h_d> where $h_s \notin$ *ESN* and $h_d \in$ *ESN* then** :
   - 4.1 a new node on h_s containing the name of h_d is added.
   - 4.2 if h_s has relations with other entities then for each detected relations, a pattern Pr_5 is extracted as Pr_5:=<Same_hs.name,hp_1,hp_j, h_d > where hp_1=h_d and $hp_j \in$ {e\e has relation with h_s}.

By applying these patterns to our example, we can detect the relations between the detected entities *Student* and *Director_thesis*:

-From the relation, R_h=<"IS-A", *Foreign-Student*, *Student*>, we detect a new entity "*Foreign-Student*" which is added to the set of entities *ESN*.
-From the relation R_{h1}:=<"",Student, Director_thesis >, we identify two patterns :
- Pr_1:<"",Student, Director_thesis, null>: *Student* and *Director_thesis* share the relation R:=<"",Student, Director_thesis > then each *Student* and *Director_thesis* instance can have these relations if they have the same *id_Dir* value (Fig.1.17).

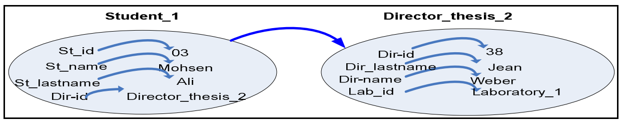


Fig 1.17 Instance of the relation between Student and Director_thesis

- Pr_2:=<Same_ Director_thesis, $Student_i$, $Student_j$, Director_thesis >: two students may have the same *Director_thesis* (same value of *Dir-id*) (Fig1.16).

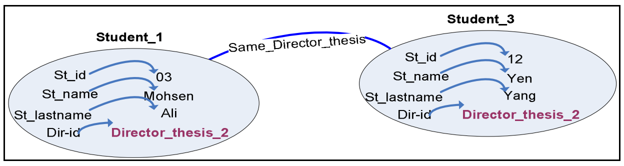


Fig1.16 Relation among Student instances

-From the relation R_{h2}:=<"", Director_thesis, Laboratory>, we identify the pattern:
-Pr_3:=<Same_Laboratory,$Director\_thesis_i$, $Director\_thesis_j$, Laboratory >: using the value of the foreign key *Lab_id* in each hyeprnode instance of the entity Director_thesis, we will link those having the same value of *Lab_id* by the relation *Same_Laboratory*. (Fig1.17)

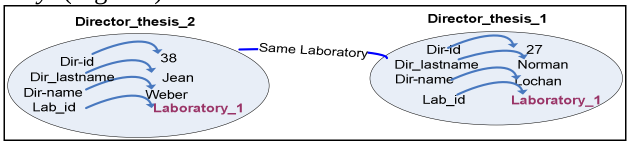


Fig1. 17 Relation between *Director_thesis* instances

-From the relation R_{h3}:=<"Part-of",Student, thesis_hasStudent>
- *Thesis_hasStudent* shares two relations "Part-of" with *Student* and *Thesis*. We add a new node on the hypernode *Student n:<Thesis*, *Thesis_i*>, corresponding to his Thesis. Then, we can apply the pattern Pr_4 (Fig1.18).

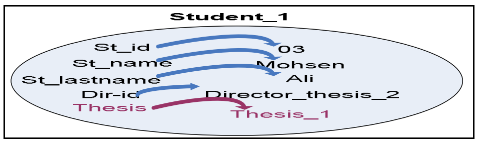


Fig1.18 Adding the node thesis on the *Student* hypernode

-Pr$_4$:=<Same_Thesis, Student$_i$, Student$_j$, *Thesis_hasStudent* >, by this pattern we search all the student which shares the same thesis (not found in our data).
-From the relation R$_{h4}$:=<"", Thesis, Director_thesis >, there are no identified pattern because *Thesis* is not related to other entities
After identifying relations and entities, the final social network is obtained by applying all the previous detection (people and their relations) and giving as tag for each hypernode his type and the name of the corresponding person (see Fig.1.19)

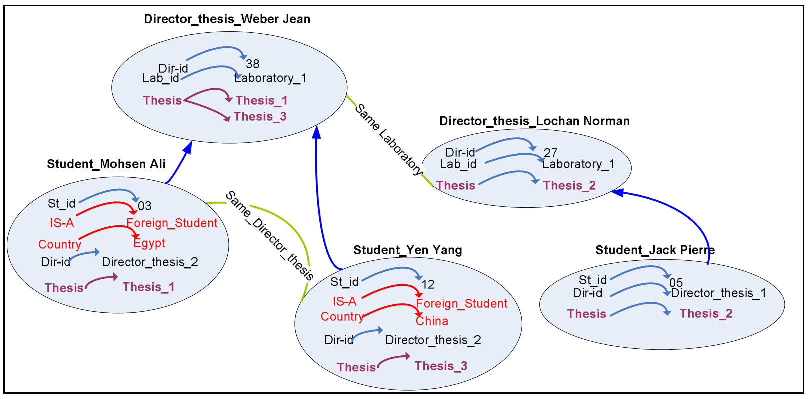


Fig.1.19 Corresponding Social Network

1.5.3Implementation and evaluation

In order to demonstrate the effectiveness and the validity of the proposed approach, a prototype has been developed. The prototype was implemented using Java and PostgreSQL. We have visualized the output database and the social network using SNA[12] (Fig1.20).
The Hypernode database will be stored in an adapted database management system which also allows the storage of voluminous complex graphs.
We experimented (for more details see (Soussi et al, 2010)) the process using the data of a database containing information about PhD students (Administrative and technical information) and also information about actors surrounding them. This database contains 1788 students (+ 80 students per year). To evaluate scalability and performance of our method for converting relational database into hypernode database, a set of SQL queries has been designed to observe any differences between the source relational database and the hypernode database. After comparing

[12] http://www.sapweb20.com/blog/2009/03/sap-enterprise-social-networking-prototype/

the results between the two databases, the hypernode schema is generated without loss or redundancy of data. This proves the correctness of this conversion.

We have used the same method to evaluate the transformation into a social network. For each entity *e* :=<*h*,e$_n$, N$_e$> , we verify that: (1) the same attributes appear in the relational database and in the social network, and (2) we find the same relations. For example, in order to verify entity's attributes: *Select * from h.name where n$_0$.name=n$_0$.val*; (n$_0$ is the first node on the *e*'s node set, and is usually corresponding to the primary key or a part of the primary key on the relational database). For example for the entity *"Student_Mohsen_Ali"*, the correspondent query used: *Select * from Student where St_id=03*;
The results obtained from these queries shows the correctness of the transformation approach. Our approach can transform a relational database into a graph from a social network perspective without lost of information.

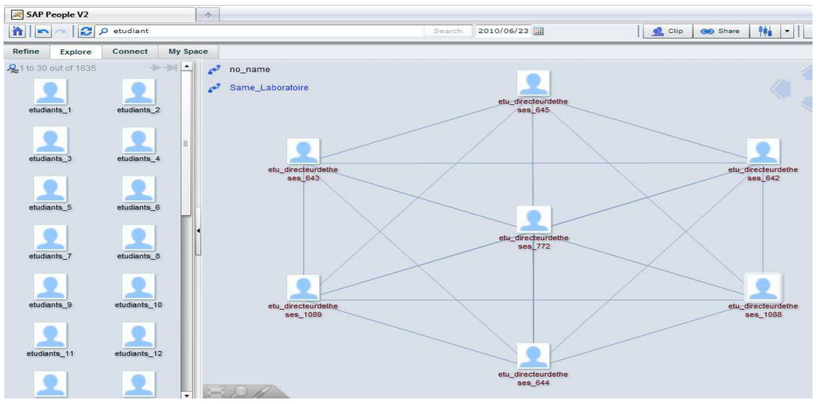


Fig. 1. 20 SNA visualization

Conclusion

In this chapter, we have presented the main graph database models and the associated graph query languages. Graph database models can model communities (social networks) and their activities even they are complex and dynamic (using model based on complex node). Even, Graph query languages are the most suited query languages to query communities (e.g. better than RDF query languages) but they are not well adapted to extract information about communities because they do not use social network analyze methods or most of them do not offer techniques to extract information about path and neighborhood. We have also presented a social network extraction method from relational database which is based on (1) a transformation of the relational database into a hypernode database and (2) a social network extraction from the hypernode database. In our future work, we will focus on how to improve the extraction method by the use of ontologies describing the relations between entities in the relational database. Then, we will try to define a storage system based on the hypernode model and a graph query language more adapted to the social network structure. We will also work on generic transformation rules according to different users' point of view and graphs merging.

References

Abiteboul S, Quass D, McHugh J, Widom J, Wiener J L (1997) The Lorel query language for semistructured data. International Journal on Digital Libraries, 1(1):68–88.

Alkhateeb F, Baget J, and Euzenat J (2009) Extending SPARQL with regular expression patterns (for querying RDF). Web Semantic 7, 2 , 57-73.

Amann B, Scholl M (1992) Gram: A Graph Data Model and Query Language. In: *Proceedings of the* European Conference on Hypertext Technology (ECHT), ACM, pp 201–211.

Andries M , Gemis M, Paredaens J , Thyssens I, Bussche J D (1992) Concepts for Graph-Oriented Object Manipulation. In Proceedings of the 3rd international Conference on Extending Database Technology: Advances in Database Technology, vol. 580. Springer-Verlag, London,pp 21--38.

Angles R and Gutierrez C (2008) Survey of graph database models. ACM Comput. Surv. 40, 1 (Feb. 2008), 1-39. DOI= http://doi.acm.org/10.1145/1322432.1322433

Barnes J A (1954) Class and committees in a Norwegian island parish. Hum. Relat, pp 39--58.

Blau H, Immerman N, Jensen D(2002) A Visual Language for Querying and Updating Graphs. University of Massachusetts Amherst, Computer Science Department Technical Report 2002-037.

Bonifati A and Ceri S (2000) Comparative analysis of five XML query languages. SIGMOD Rec. 29, 1 (Mar. 2000), 68-79.

Broekstra J(2003) SeRQL: Sesame RDF query language, in: SWAP Deliverable 3.2 Method Design.

Ceri S, Comai S, Damiani E, Fraternali P, Paraboschi S, and Tanca L(1999) XML-GL: a graphical language for querying and restructuring XML documents. Comput. Netw. 31, 1171-1187.

Codd E F (1980) Data models in database management. In: Proceedings of the 1980 Workshop on Data Abstraction, Databases and Conceptual Modeling (Pingree Park, Colorado, United States, June 23 - 26, 1980). ACM, New York, NY,pp 112-114.

Consens M P, Mendelzon A O (1989) Expressing Structural Hypertext Queries in Graphlog. In: Proceedings of the 2th International Conference on Hypertext, ACM Press, pp 269–292.

Cruz I F, Mendelzon A O, Wood P T (1987) A graphical query language supporting recursion. SIGMOD Rec. 16(3) :323-330. doi= http://doi.acm.org/10.1145/38714.38749

Cruz I F, Mendelzon A O, Wood P T (1989) G+: recursive queries without recursion. In: Proceedingsof the 2th International Conference on Expert Database Systems (EDS). Addison-Wesley, pp 645–666.

Deutsch A, Fernandez M, Florescu D, Alon Levy, Suciu D (1998) XML-QL: A Query Language for XML. In:Proc. of the Query Languages workshop (QL98), Cambridge, Mass.,

Flesca S, Greco S (2000) Querying Graph Databases. In: Proceedings of the 7th international Conference on Extending Database Technology: Advances in Database Technology (March 27 - 31, 2000). C. Zaniolo, P. C. Lockemann, M. H. Scholl, and T. Grust, Eds. Extending Database Technology, vol. 1777. Springer-Verlag, London, pp 510–524.

Gemis M, Paredaens J (1993) An object-oriented pattern matching language. In: JSSST, Springer-Verlag, vol. 742, pp 339-355.

Graves M, Bergeman E R, Lawrence C B (1995) Graph database systems for genomics. IEEE Eng.Medicine Biol. 14(6):737--745.

Güting R H (1994) GraphDB: Modeling and Querying Graphs in Databases. In: Proceedings of the 20th international Conference on Very Large Data Bases, September 12 - 15. Eds. Very Large Data Bases. Morgan Kaufmann Publishers, San Francisco, CA, pp 297-308.

Gyssens M , Paredaens J, Gucht D V (1990) A graph-oriented object model for database end-user interfaces. SIGMOD Rec. 19(2) :24-33. doi= http://doi.acm.org/10.1145/93605.93616

He H ,Singh A K (2008) Graphs-at-a-time: query language and access methods for graph databases. In: Proceedings of the 2008 ACM SIGMOD international Conference on Management of Data SIGMOD '08. ACM, New York, NY, pp 405–418.

Hidders J, Paredaens J (1993) GOAL, A Graph-Based Object and Association Language. Advances in Database Systems: Implementations and Applications, CISM: 247–265.

Hidders J (2002) Typing Graph-Manipulation Operations. In Proceedings of the 9th International Conference on Database Theory (ICDT). Springer- Verlag. pp 394–409.

Kaplan I (2006)A Semantic Graph Query Language. technical repport, Lawrence Livermore National Laboratory, October 17, 2006 UCRL-TR-255447

Karvounarakis G, Alexaki S, Christophides V, Plexousakis D, and Scholl M (2002) RQL: a declarative query language for RDF. In Proceedings of WWW '02. ACM, NY, 592-603.

Kirchhoff L, Stanoevska-Slabeva K, Nicolai T, Fleck M (2008) Using social network analysis to enhance information retrieval systems. In: Applications of Social Network Analysis (ASNA),

Klyne G, Carrol J J, Andmcbride B (2004) Resource description framework (RDF): Concepts and abstract syntax. W3C recommendation. http://www.w3.org/TR/rdf-concepts/.

Kuper G M , Vardi M Y(1993) The Logical Data Model. ACM Trans. Database Syst,379--413.

Levene M, Poulovassilis A (1990) The hypernode model and its associated query language. In: Proceedings of the Fifth Jerusalem Conference on information Technology (Jerusalem, Israel). IEEE Computer Society Press, Los Alamitos, CA:520--530.

Levene M,Poulovassilis A(1991) An object-oriented data model formalised through hypergraphs. Data Knowl. Eng. 6(3): 205--224.

Levene M, Loizou G(1995) A Graph-Based Data Model and its Ramifications. IEEE Trans. on Knowl. and Data Eng. 7(5): 809-823. doi= http://dx.doi.org/10.1109/69.469818

Maatuk A, Akhtar M, Rossiter B N (2008) Relational Database Migration: A Perspective. In: DEXA'08, pp. 676--683

Miller E, Swick R, and Brickley D (2004) Resource description framework (RDF). Recommendation,W3C.

Paredaens J, Peelman P, Tanca L (1995) G-Log: A Graph-Based Query Language. IEEE Transactions on Knowledge and Data Engineering (TKDE), 7(3):436–453.

Pérez J, Arenas M, and Gutierrez C (2009) Semantics and complexity of SPARQL. ACM Trans. Database Syst. 34, 3 (Aug. 2009), 1-45.

Ronen R, Shmueli O (2009) SoQL: A Language for Querying and Creating Data in Social Networks. In: Proceedings of the 2009 IEEE international Conference on Data Engineering,March 29 - April 02, ICDE. IEEE Computer Society, Washington, DC, pp 1595-1602.

Seaborne A (2004) RDQL—A Query Language for RDF, Member Submission, W3C.

Shadbolt N, Berners-Lee T, Hall W (2006) The semantic web revisited. IEEE Intelligent Systems, 21(3):96–101.

Sheng L, Ozsoyoglu Z M, Ozsoyoglu G(1999) A Graph Query Language and Its Query Processing. In: Proceedings of the 15th Int. Conf. on Data Engineering (ICDE), IEEE Computer Society, pp 572–581.

Sintek M, and Decker S (2002) TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In Proceedings of the First international Semantic Web Conference on the Semantic Web. Lecture Notes In Computer Science, vol. 2342. Springer-Verlag, 364-378

Soussi R, Aufaure M A, Baazaoui H (2010)Towards Social Network Extraction Using a Graph Database, In: Proceedings of Second International Conference on Advances in Databases, Knowledge, and Data Applications, pp 28-34

Xu X, Zhan J, Zhu H (2008) Using Social Networks to Organize Researcher Community. In: Proceedings of the IEEE ISI 2008 Paisi, Paccf, and SOCO international Workshops on intelligence and Security informatics. Springer-Verlag, Heidelberg, pp 421--427