Towards Temporal Graph Databases

Alexander Campos, Jorge Mozzino, and Alejandro Vaisman

Instituto Tecnologico de Buenos Aires
jcamposi,jmozzino,avaisman@itba.edu.ar

Abstract. In spite of the extensive literature on graph databases (GDBs), temporal GDBs have not received too much attention so far. Temporal GBDs can capture, for example, the evolution of social networks across time, a relevant topic in data analysis nowadays. We propose a data model and query language (denoted TEG-QL) for temporal GDBs, based on the notion of attribute graphs. This allows a straightforward translation to Neo4J, a well-known GBD.

Keywords: Neo4J, Graph Database, Temporal Graphs

1 Introduction

Graphs, and, particularly, attributed graphs [8], are becoming increasingly popular to model different kinds of networks (e.g., social networks, sensor networks, and the kind) for analysis in a classical way, and also for Online Analytical Processing (OLAP) on graphs [6, 8]. Also, these kinds of graphs underlie the data model of Neo4J [1], probably the most popular graph database for social network analysis [2]. In spite of the extensive bibliography on graph database models [3], and of the fact that social networks are frequently-changing structures, not much attention has yet been paid to temporal graph databases. In this paper we introduce a temporal data model consisting in a data structure (an attribute graph), and a set of constraints. Over this model, we define a temporal query language, called TEG-QL, an SQL/SPARQL-like style language, aimed at facilitating the translation to Cypher, the query language for Neo4J.

Among the work on temporal graphs, Catutto et al. [5] organize temporal data in so-called frames, associated with a time interval. When changes are frequent, redundancy is a problem in this model, since each frame is connected to all the existing data. Also, changes in attributes are not allowed. Khurana and Deshpande [7] studied methods to efficiently query historical graphs, focusing on *querying the state of a network as of a certain point* (snapshot) in time. They use a model based on versioning, storing the current graph, plus a series of deltas, which contain the graph variation over time. Our model, on the contrary, is based on timestamps, where the complete history is stored in the same graph.

Our *running example* is a network containing two kinds of nodes, representing *persons* and *buildings*. Edges are of two kinds: One, representing *friendship* relationships between people across time; the other one, telling the buildings

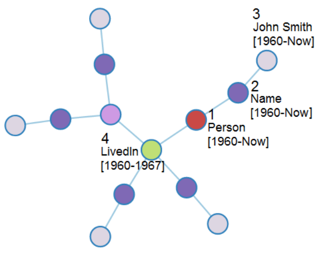

Fig. 1. A temporal graph and its different kinds of nodes

where people had *lived in* through time. Besides information about the `name`, type of building, number of bedrooms in the apartment, etc., nodes have a *temporal attribute*, which is a temporal element indicating the periods of validity of the node. Edges are also labeled with temporal attributes.

2 Data Model

We now define our temporal data model, based on the notion of attribute graphs.

Definition 1 (Temporal graph). *A temporal graph is a structure $G(No, Ne, Na, Nv, E)$ where G is the name of the graph, E is a set of edges, and No, Ne, Na, and Nv are sets of nodes, denoted object nodes, edge nodes, attribute nodes,and value nodes, respectively. Every node is associated with a tuple (`name`, `interval`), where `name` represents the content of the node, and `interval` is a temporal element representing the period(s) in which the node is (was) valid.* □

Object nodes represent entities, edge nodes represent relationships between object nodes, attribute nodes describe entities, and value nodes represent the value of an attribute. Figure 1 depicts a portion of a graph, using our running example. The properties labeling the nodes are, from top to bottom, `id`, `name` and `interval`. This way, the node with `id`=4 (in light green) is an *Edge node*, the node with `id`=1 (in red) is an *Object node*, the node with `id`=2 (purple) is an *Attribute node*, and the one with `id`=3 (grey) is a *Value node*. □

Definition 2 (Constraints). *Consider the following notation. We denote edge nodes as $ne\{na, nb\}$, where ne is an edge node connected to object nodes na and nb; $e\{na, nb\}$ represent edges, where na and nb are nodes connected by edge e; $na\{n\}$ represent attribute nodes, where n is the object or edge node connected to na; $nv\{na\}$ denote value nodes, where na is the attribute node connected to nv.*

For the graph in Definition 1, the following constraints hold:

1. $\forall n, n' \in No,\ n = n' \vee n.id \neq n'.id$
2. $\forall n, n' \in Ne,\ n = n' \vee n.id \neq n'.id$
3. $\forall n, n' \in Na,\ n = n' \vee n.id \neq n'.id$

4. $\forall n, n' \in Nv,\ n = n' \vee n.id \neq n'.id$
5. $\forall nv\{na\}, nv'\{na\} \in Nv,\ nv = nv' \vee nv.value \neq nv'.value$
6. $\forall n \in No,\ e\{n, n'\} \in E \Rightarrow n' \in Ne \bigcup Na$
7. $\forall n \in Ne,\ e\{n, n'\} \in E \Rightarrow n' \in No \bigcup Na$
8. $\forall n \in Na,\ e\{n, n'\} \in E \Rightarrow n' \in No \bigcup Ne \bigcup Nv$
9. $\forall n \in Nv,\ e\{n, n'\} \in E \Rightarrow n' \in Nv$
10. $\forall ne \in Ne,\ if\ \exists\ e\{no, ne\} \wedge \exists e'\{ne, no'\} \Rightarrow \nexists e'' \in E, \nexists no'' \in No \wedge no'' \neq no \wedge no'' \neq no' \wedge e''\{no'', ne\} \wedge e''\{ne, no''\}$
11. $\forall n \in Na(\exists no \in No \exists e \in E(e(no, n) \vee \exists ne \in Ne \wedge e\{ne, n\} \wedge (\not\exists n' \in (Na \bigcup Ne \bigcup Nv \bigcup No) \wedge e' \in E \wedge e'\{n', n\})$
12. $\forall n \in Nv \wedge e\{n', n\} \wedge n \in Na \Rightarrow \nexists ! n'' \in (Na \bigcup Ne \bigcup Nv \bigcup No) \wedge (e''\{n'', n\} \in E \vee e''\{n, n''\} \in E$
13. $\exists e\{n, n'\}, e'\{n, n'\} \in E \Rightarrow e = e'$
14. $\forall ne\{n, n'\} \in Ne, ne.interval \subset n.interval \cap n'.interval$
15. $\forall na\{n\} \in Na, na.interval \subset n.interval$
16. $\forall nv\{na\} \in Nv, nv.interval \subset nv.interval$
17. $\forall nv\{na\}, nv'\{na\}, nv \neq nv', nv.interval \cap nv'.interval = \emptyset$

Constraints 1 through 4 state that no two nodes can have the same id. Constraint 5 requires coalescing all nodes with the same value; Constraints 6 through 9 state that Object nodes can only be connected to edge nodes or attribute nodes; Edge nodes can only be connected to object nodes or attribute nodes; Attribute nodes can be connected to non-attribute nodes; and Value nodes can only be connected to attribute nodes. The cardinalities of these connections is stated by Constraints 10 through 13 (e.g., Edge nodes must be connected to exactly two different object nodes through exactly one edge). Constraints 14 to 16 restrict the values of the `interval` *property. Finally, constraint 17 forces value nodes connected to the same attribute node to have non-overlapping intervals.* □

3 TEG-QL: A Query Language for Graphs

The syntax of TEG-QL resembles the one of SQL, but queries, as usual in graphs, are based on pattern matching. Thus, the `FROM` clause contains one or more paths (of fixed or variable length), over which a selection is performed. The `SELECT` clause may either mention just attributes or paths. The temporal semantics in embedded in the language, i.e., the answer to the query is a temporal graph, although the query may not mention temporal attributes. This can be changed by the `SNAPSHOT` modifier, which allows to retrieve the state of the graph at a certain point in time, or the `IN` modifier, which allows retrieving the status of the graph in a certain interval. Further details can be found in [4].

Consider the query *People and buildings such that a person named John Smith has lived in such buildings.* The TEG-QL query is shown on the left hand side of Figure 2. We can see that we take the paths matching the `FROM` clause, and filter them using the condition in the WHERE clause. Figure 3 (left) shows the result. The center node (in orange) is the `Person` node that represents John

SELECT Person−LivedIn→Building	SELECT Person−friend→P2
FROM Person−LivedIn→Building	FROM Person−Friend→Person as P2
WHERE Person.Name = 'John Smith'	WHERE Person.Name = 'John Smith'

Fig. 2. TEG-QL queries

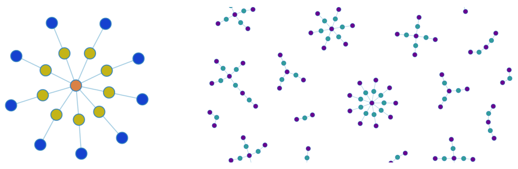

Fig. 3. Query selecting a path (left); Friends of John Smith (right)

Smith, middle nodes (yellow) nodes are the edge nodes representing the Lived In relationships; and outer nodes (blue) are the Building nodes.

The TEG-QL expression for the query *Friends of someone called John Smith* is shown in Figure 2 (right); Figure 3 (right) depicts the clusters of people in the result (the ones who know someone with the name "John Smith").

Queries showing the use of the `SNAPSHOT` and `IN` modifiers are depicted in Figure 4. The query on the left returns *all the people named John Smith, and the buildings where they lived during 1990.* Note that we assume a temporal granularity at the *year* level here (We do not get into the details of how to manipulate granularities here). The IN predicate allows selecting nodes and edges valid in a given interval. The query on the right hand side of Figure 4 is similar to the one above, *just selecting those paths existing between 1986 and 1989.*

SELECT Person−LivedIn→Building	SELECT *
FROM Person−LivedIn→Building	FROM Person−LivedIn→Building
WHERE Person.Name = 'John Smith'	WHERE Person.Name = 'John Smith'
SNAPSHOT 1990	IN [1986-1989]

Fig. 4. TEG-QL queries with SNAPSHOT (left) and IN (right) modifiers

To translate TEG-QL queries into Cypher, we first translate each path in the `FROM` clause. The term `element.alias:OBJECT{title:element.name}` results from the translation of an object node; an Edge node is translated analogously. We then expand the `SELECT` clause with the corresponding attributes. Finally, the `WHERE` clause is addressed splitting conjunctions and disjunctions. Details of the translation process can be found in [4]

Future work will focus on expanding the temporal capabilities of TEG-QL, and, most of on addressing the problem of query optimization.

References

1. Neo4J website, http://www.neo4j.com
2. Angles, R.: A Comparison of Current Graph Database Models. In: Proceedings of ICDE Workshops. pp. 171–177. Arlington, VA, USA (2012)
3. Angles, R., Gutierrez, C.: Survey of graph database models. ACM Computing Surveys (CSUR) 40(1), 1–39 (2008)
4. Campos, A., Mozzino, J., Vaisman, A.: Towards temporal graph databases. CoRR abs/1604.08568 (2016), http://arxiv.org/abs/1604.08568
5. Cattuto, C., Quaggiotto, M., Panisson, A., Averbuch, A.: Time-varying social networks in a graph database. In: Proceedings of GRADES 2013. p. 11. NY, USA (2013)
6. Ghrab, A., Romero, O., Skhiri, S., Vaisman, A., Zimányi, E.: GRAD: Modeling and Querying Data Warehouses. In: Proceedings of ADBIS. pp. 92–105. Poitiers, France (2015)
7. Khurana, U., Deshpande, A.: HiNGE: Enabling Temporal Analytics at Scale. In: Proceedings of SIGMOD. NY, USA (2013)
8. Wang, Z., Fan, Q., Wang, H., Tan, K., Aggrawal, D., Abbadi, A.E.: PArallel GRaph OLap Over Large Scale Attributed Graphs. In: Proceedings of ICDE. Chicago, USA (2014)