Towards a Complete Direct Mapping From Relational Databases To Property Graphs

Abdelkrim Boudaoud, Houari Mahfoud, Azeddine Chikh
Abou-Bekr Bellkaid University & LRIT Laboratory
Tlemcen, Algeria
{abdelkrim.boudaoud, houari.mahfoud, azeddine.chikh}@univ-tlemcen.dz

Abstract. It is increasingly common to find complex data represented through the graph model. Contrary to relational models, graphs offer a high capacity for executing analytical tasks on complex data. Since a huge amount of data is still presented in terms of relational tables, it is necessary to understand how to translate this data into graphs. This paper proposes a complete mapping process that allows transforming any relational database (schema and instance) into a property graph database (schema and instance). Contrary to existing mappings, our solution preserves the three fundamental mapping properties, namely: information preservation, semantic preservation and query preservation. Moreover, we study mapping any SQL query into an equivalent Cypher query, which makes our solution practical. Existing solutions are either incomplete or based on non-practical query language. Thus, this work is the first complete and practical solution for mapping relations to graphs.

Keywords: Direct mapping, Complete mapping, Relational database, Graph database, SQL, Cypher

1 Introduction

Relational databases (RDs) have been widely used and studied by researchers and practitioners for decades due to their simplicity, low data redundancy, high data consistency, and uniform query language (SQL). Hence, the size of web data has grown exponentially during the last two decades. The interconnections between web data entities (e.g. interconnection between YouTube videos or people on Facebook) are measured by billions or even trillions [5] which pushes the relational model to quickly reach its limits as querying high interconnected web data requires complex SQL queries which are time-consuming. To overcome this limit, the graph database model is increasingly used on the Web due to its flexibility to present data in a normal form, its efficiency to query a huge amount of data and its analytic powerful. This suggests studying a mapping from RDs to graph databases (GDs) to benefit from the aforementioned advantages. This kind of mapping has not received more attention from researchers since only a few works [3,4,12,13] have considered it. A real-life example of this mapping has been discussed in [12]: “investigative journalists have recently found, through
graph analytics, surprising social relationships between executives of companies within the Offshore Leaks financial social network data set, linking company officers and their companies registered in the Bahamas. The Offshore Leaks PG was constructed as a mapping from relational database (RDB) sources. In a nutshell, the proposed mappings suffer from at least one of the following limits: (a) they do not study fundamental properties of mapping; (b) they do not consider a practical query language to make the approach more useful; (c) they generate an obfuscated schema.

This paper aims to provide a complete mapping (CM) from RDs to GDs by investigating the fundamental properties of mapping, namely: information preservation (IP), query preservation (QP), and semantic preservation (SP). In addition to data mapping, we study the mapping of SQL queries to Cypher queries which makes our results more practical since SQL and Cypher are the most used query languages for relational and graph data respectively.

Contributions and Road-map. Our main contributions are as follows: (1) we formalize a CM process that maps RDs to GDs in the presence of schema; (2) we propose definitions of schema graph and graph consistency that are necessary for this mapping; (3) we show that our CM preserves the three fundamental mapping properties (IP, SM, and QP); (4) in order to prove QP, we propose an algorithm to map SQL queries into equivalent Cypher queries. To our knowledge, this work is the first complete effort that investigates mapping relations to graphs.

Related Work. We classify previous works as follows:

Mapping RDs to RDF Data. Squeda et al. [11] studied the mapping of RDs to RDF graph data and relational schema to OWL ontology’s. Moreover, SQL queries are translated into SPARQL queries. They were the first to define a set of mapping properties: information preservation, query preservation, semantic preservation and monotonicity preservation. They proved first that their mapping is information preserving and query preserving, while when it comes to the two remaining properties, preserving semantics makes the mapping not monotonicity preserving.

Mapping RDs to Graph Data. De Virgilio et al. [3][4] studied mapping a) RDs to property graph data (PG) by considering schema both in source and target; and b) any SQL query into a set of graph traversal operations that realize the same semantic over the resulted graph data. We remark that the proposed mapping obfuscates the relational schema since the resulted graph schema is difficult to understand. Moreover, the mapping does not consider typed data. From the practical point of view, the graph querying language considered is not really used in practice and the proposed query mapping depends on the syntax and semantics of this language which makes hard the application of their proposal for another query language. In addition, they apply an aggregation process that maps different relational tuples to the same graph vertex in order to optimize graph traversal operations. However, this makes the mapping not information preserving and can skew the result of some analytical tasks that one would like to apply over the resulted data graph.
Table 1: Comparative table of related works.

| Type               | Work                     | Mapping rules | Preserved properties | Schema | Instance |
|--------------------|--------------------------|---------------|----------------------|--------|----------|
| RDs → RDF          | Sequeda et al. [11]       | ✓             | ✓                    | ✓      | ✓        |
| De vergilio et al. [3,4] | ✓                      | ✓             | ✓                    | ✓      | ✓        |
| Stoica et al. [12,13] | ✓                      | ✓             | ✓                    | ✓      | ✓        |
| RDF → PG           | Angeles et al. [2]        | ✓             | ✓                    | ✓      | ✓        |
| RDs → PG           | O. Orel et al. [10]       | ✓             | ✓                    | ✓      | ✓        |
| S. Li et al. [8]   | ✓                        |               |                      | ✓      | ✓        |
| Our Work           | ✓                        | ✓             | ✓                    | ✓      | ✓        |

Stoica et al. [12,13] studied the mapping of RDs to GDs and any relational query (formalized as an extension of relational algebra) into a G-core query. Firstly, the choice of source and destination languages hinders the practicability of the approach. Moreover, it is hard to see if the mapping is semantic preserving since no definition of graph data consistency is given. Attributes, primary and foreign keys are verbosely represented by the data graph, which makes this later hard to understand and to query.

O. Orel et al. [10] discussed mapping relational data only into property graphs without giving attention neither to schema nor mapping properties.

The Neo4j system provides a tool called Neo4j-ETL [1], which allows users to import their relational data into Neo4j to be presented as property graphs. Notice that the relational structure (both instance and schema) is not preserved during this mapping since some tuples of the relational data (resp. relations of the relational schema) are represented as edges for storage concerns (as done in [4]). However, as remarked in [12], this may skew the results of some analytical tasks (e.g. density of the generated graphs). Moreover, Neo4j-ETL does not allow the mapping of queries. S. Li et al. [8] study an extension of Neo4j-ETL by proposing mapping of SQL queries to Cypher queries. However, their mapping inherits the limits of Neo4j-ETL. In addition, no detailed algorithm is given for the query mapper which makes impossible the comparison of their proposal with others. This is also the limit of [9].

Finally, Angeles et al. [2] studied mappings RDF databases to property graphs by considering both data and schema. They proved that their mapping ensures both information and semantic preservation properties.

Table 1 summarizes most important features of related works.

2 Preliminaries

This section defines the several notions that will be used throughout this paper.

Let $\mathcal{R}$ be an infinite set of relation names, $\mathcal{A}$ is an infinite set of attribute names with a special attribute name $\text{tid}$, $\mathcal{T}$ is a finite set of attribute types ($\text{String}$, $\text{Int}$, etc.)
Date, Integer, Float, Boolean, Object), $\mathcal{D}$ is a countably infinite domain of data values with a special value $\text{NULL}$.

## 2.1 Relational Databases

A relational schema is a tuple $S = (R, A, T, \Sigma)$ where:

1. $R \subseteq \mathcal{R}$ is a finite set of relation names;
2. $A$ is a function assigning a finite set of attributes for each relation $r \in R$ such that $A(r) \subseteq \mathcal{A}\{\text{tid}\}$;
3. $T$ is a function assigning a type for each attribute of a relation, i.e. for each $r \in R$ and each $a \in A(r) \setminus \{\text{tid}\}$, $T(a) \subseteq \mathcal{T}$;
4. $\Sigma$ is a finite set of primary (PKs) and foreign keys (FKs) defined over $R$ and $A$. A primary key over a relation $r \in R$ is an expression of the form $r[a_1, \cdots, a_n]$ where $a_1 \leq \cdots \leq a_n \in A(r)$. A foreign key over two relations $r$ and $s$ is an expression of the form $r[a_1, \cdots, a_n] \rightarrow s[b_1, \cdots, b_n]$ where $a_1 \leq \cdots \leq a_n \in A(r)$ and $s[b_1, \cdots, b_n] \in \Sigma$.

An instance $I$ of $S$ is an assignment to each $r \in R$ of a finite set $I(r) = \{t_1, \cdots, t_n\}$ of tuples. Each tuple $t_i : A(r) \cup \{\text{tid}\} \rightarrow \mathcal{D}$ is identified by $\text{tid} \neq \text{NULL}$ and assigns a value to each attribute $a \in A(r)$. We use $t_i(a)$ (resp. $t_i(\text{tid})$) to refer to the value of attribute $a$ (resp. tid) in tuple $t_i$. Moreover, for any tuples $t_i, t_j \in I(r)$, $t_i(\text{tid}) \neq t_j(\text{tid})$ if $i \neq j$.

For any instance $I$ of a relational schema $S = (R, A, T, \Sigma)$, we say that $I$ satisfies a primary key $r[a_1, \cdots, a_n]$ in $\Sigma$ if: 1) for each tuple $t \in I(r)$, $t(a_1 \leq \cdots \leq a_n) \neq \text{NULL}$; and 2) for any $t' \in I(r)$, if $t(a_1 \leq \cdots \leq a_n) = t'(a_1 \leq \cdots \leq a_n)$ then $t = t'$ must hold. Moreover, $I$ satisfies a foreign key $r[a_1, \cdots, a_n] \rightarrow s[b_1, \cdots, b_n]$ in $\Sigma$ if: 1) $I$ satisfies $s[b_1, \cdots, b_n]$; and 2) for each tuple $t \in I(r)$, either $t(a_1 \leq \cdots \leq a_n) = \text{NULL}$ or there exists a tuple $t' \in I(s)$ where $t(a_1 \leq \cdots \leq a_n) = t'(b_1 \leq \cdots \leq b_n)$. The instance $I$ satisfies all integrity constraints in $\Sigma$, denoted by $I \models \Sigma$, if it satisfy all primary keys and foreign keys in $\Sigma$.

Finally, a relational database is defined with $D_R = (S_R, I_R)$ where $S_R$ is a relational schema and $I_R$ is an instance of $S_R$.

## 2.2 Property Graphs

A property graph (PG) is a multi-graph structure composed of labeled and attributed vertices and edges defined with $G = (V, E, L, A)$ where: 1) $V$ is a finite set of vertices; 2) $E \subseteq V \times V$ is a finite set of directed edges where $(v, v') \in E$ is an edge starting at vertex $v$ and ending at vertex $v'$; 3) $L$ is a function that assigns a label to each vertex in $V$ (resp. edge in $E$); and 4) $A$ is a function assigning a nonempty set of key-value pairs to each vertex (resp. edge). For any edge $e \in E$, we denote by $e.s$ (resp. $e.d$) the starting (resp. ending) vertex of $e$. 
2.3 SQL queries and Cypher queries

We study in this paper the mapping of relational data into PG data. In addition, we show that any relational query over the source data can be translated into an equivalent graph query over the generated data graph. To this end, we model relational queries with the SQL language [7] and the graph queries with the Cypher language [6] since each of these languages is the most used in its category. To establish a compromise between the expressive power of our mapping and its processing time, we consider a simple but very practical class of SQL queries and we define its corresponding class of Cypher queries. It is necessary to understand the relations between basic SQL queries and basic Cypher queries before studying all the expressive power of these languages.

The well-known syntax of SQL queries is “Select $I$ from $R$ where $C$” where: a) $I$ is a set of items; b) $R$ is a set of relations names; and c) $C$ is a set of conditions. Intuitively, an SQL query selects first some tuples of relations in $R$ that satisfy conditions in $C$. Then, the values of some records (specified by $I$) of these tuples are returned to the user.

On the other side, the Cypher queries considered in this paper have the syntax: “Match patterns Where conditions Return items”. Notice that a Cypher query aims to find, based on edge-isomorphism, all subgraphs in some data graph that match the pattern specified by the query and also satisfy conditions defined by this latter. Once found, only some parts (i.e. vertices, edges, and/or attributes) of these subgraphs are returned, based on items specified by Return clause. Therefore, the Match clause specifies the structure of subgraphs we are looking for on the data graph; the Where clause specifies conditions (based on vertices, edges and/or attributes) these subgraphs must satisfy; and finally, the Return clause returns some parts of these subgraphs to the user as a table.

Example 1. Fig.1 depicts a data graph where vertices represent entities (i.e. Doctor, Patient, Diagnostic and Admission); inner information (called attributes) represent properties of this vertex (e.g. Speciality); and edges represent relationships between these entities. For instance, an Admission vertex may be connected to some Patient, Doctor and Diagnostic vertices to specify for some patient: a) his doctor; b) information about his admission at the hospital; and c) diagnostics made for this patient.

The following Cypher query returns the name of each patient who is admitted at some date:

\[
\text{MATCH} \ (a : \text{Admissions}) < -[: \text{Admissions} - \text{Patients}] - (p : \text{Patients}) \\
\text{WHERE} \ a.\text{Adm}i._\text{date} \ = \ "30/11/2021" \\
\text{RETURN} \ p.\text{Name}
\]
2.4 Direct Mapping (DM)

Inspired from [11][13], we define in this section the direct mapping from a relational database into a graph database and we discuss its properties. Given a relational database $D_R$ composed of $S_R$ and $I_R$, a direct mapping consists of translating each entity in $D_R$ into a graph database without any user interaction. That is, any $D_R=(S_R, I_R)$ (with possibly empty $S_R$), is translated automatically into a pair of property graphs $(S_G, I_G)$ (with possibly empty $S_G$), that we call a graph database. Let $D_R$ be an infinite set of relational databases, and $D_G$ be an infinite set of graph databases. Based on these notions, we give the next definition of direct mapping and its properties.

**Definition 1.** A direct mapping is a total function $DM : D_R \rightarrow D_G$. $\square$

Intuitively, for each $D_R \in D_R$, $DM(D_R)$ produces a graph database $D_G \in D_G$ that aims to represent the source relational database (i.e. instance and optionally schema) in terms of a graph.

We define next fundamental properties that a direct mapping must preserve [11], namely: information preservation, query preservation and semantic preservation. The two first properties ensure that the direct mapping does not lose neither information nor semantic of the relational database being translated. The last property ensures that the mapping does not hinder the querying capabilities as any relational query can be translated into a graph query.

2.4.1 Information Preservation A direct mapping $DM$ is information preserving if no information is lost during the mapping of any relational database.
**Definition 2 (Information preservation).** A direct mapping $DM$ is information preserving if there is a computable inverse mapping $DM^{-1} : D_G \rightarrow D_R$ satisfying $DM^{-1}(DM(D_R)) = D_R$ for any $D_R \in D_R$. \[\square\]

### 2.4.2 Query Preservation
Recall that both SQL and Cypher queries return a result modeled as a table where columns represent entities requested by the query (using Select clause in case of SQL, or Return clause in case of Cypher), while each row assigns values to these entities. In addition, the result of both SQL and Cypher queries may contain repeated rows.

Let $I_R$ be a relational instance and $Q_s$ be an SQL query expressed over $I_R$. We denote by $[Q_s]_I_R$ the result table of $Q_s$ over $I_R$. Similarly, $[Q_c]_{I_G}$ is the result table of a Cypher query $Q_c$ over an instance graph $I_G$. Moreover, we denote by $[Q_s]_{I_R}^*$ (resp. $[Q_c]_{I_G}^*$) the refined table that has no repeated row.

A direct mapping $DM$ is query preserving if any query over the relational database $D_R$ can be translated into an equivalent query over the graph database $D_G$ that results from the mapping of $D_R$. That is, query preservation ensures that every relational query can be evaluated using the mapped instance graph.

Since SQL and Cypher languages return results in different forms, proving query preservation consists to define a mapping from the SQL result to the Cypher result. This principle was proposed first in [13] between relational and RDF queries. Therefore, we revise the definition of query preservation as follows:

**Definition 3 (Query preservation).**
A direct mapping $DM$ is query preserving if, for any relational database $D_R = (S_R, I_R)$ and any SQL query $Q_s$, there exists a Cypher query $Q_c$ such that: each row in $[Q_s]_{I_R}^*$ can be mapped into a row in $[Q_c]_{I_G}^*$ and vice versa. By mapping a row $r$ into a row $r'$, we assume that $r$ and $r'$ contain the same data with possibly different forms. \[\square\]

### 2.4.3 Semantics Preservation
A direct mapping $DM$ is semantics preserving if any consistent (resp. inconsistent) relational database is translated into a consistent (resp. inconsistent) graph database.

**Definition 4 (Semantic preservation).** A direct mapping $DM$ is semantic preserving if, for any relational database $D_R = (S_R, I_R)$ with a set of integrity constraints $\Sigma$, $I_R \models \Sigma$ iff: $DM(D_R)$ produces a consistent graph database. \[\square\]

Notice that no previous work have considered semantic preservation over data graphs. That is, no definition of graph database consistency have been given. We shall give later our own definition.

### 3 Complete Mapping (CM)
In this section, we propose a complete mapping $CM$ that transforms a complete relational database (schema and instance) into a complete graph database (schema and instance). We call our mapping Complete since some proposed mappings (e.g. [10]) deal only with data and not schema.
Definition 5 (Complete Mapping). A complete mapping is a function \(CM : DR \rightarrow DG\) from the set of all RDs to the set of all GDs such that: for each complete relational database \(DR = (SR, IR)\), \(CM(DR)\) generates a complete graph database \(DG = (SG, IG)\). □

In order to produce a complete graph database, our \(CM\) process is based on two steps, schema mapping and instance mapping, which we detail hereafter.

3.1 Schema Graph and Instance Graph

Contrary to relational data, graph data still have no schema definition standard. Hence, we extend the property graph definition in order to introduce our schema graph definition.

Definition 6 (Schema Graph). A schema graph is an extended property graph defined with \(SG = (Vs, Es, Ls, As, Pk, Fk)\) where: 1) \(Vs\) is a finite set of vertices; 2) \(Es \subseteq Vs \times Vs\) is a finite set of directed edges where \((v, v') \in Es\) is an edge starting at vertex \(v\) and ending at vertex \(v'\); 3) \(Ls\) is a function that assigns a label to each vertex in \(Vs\) (resp. edge in \(Es\)); 4) \(As\) is a function assigning a nonempty set of pairs \((ai : ti)\) to each vertex (resp. edge) where \(ai \in A\) and \(ti \in T\); 5) \(Pk\) is a partial function that assigns a subset of \(As(v)\) to a vertex \(v\); and finally 6) for each edge \(e \in Es\), \(Fk(e, s)\) (resp. \(Fk(e, d)\)) is a subset of \(As(s, e)\) (resp. \(As(d, e)\)). □

The functions \(Pk\) and \(Fk\) will be used later to incorporate integrity constraints over graph databases.

Definition 7 (Instance Graph). Given a schema graph \(SG = (Vs, Es, Ls, As, Pk, Fk)\), an instance \(IG\) of \(SG\), called an instance graph, is given by a property graph \(IG = (Vi, Ei, Li, Ai)\) where:

1. \(Vi\) and \(Ei\) are the set of vertices and the set of edges as defined for schema graph;
2. for each vertex \(vi \in Vi\), there exists a vertex \(vs \in Vs\) such that: a) \(Li(vi) = Ls(vs)\); and b) for each pair \((a : c) \in Ai(vi)\) there exists a pair \((a : t) \in As(vs)\) with \(type(c) = t\). We say that \(vi\) corresponds to \(vs\), denoted by \(vi \sim vs\).
3. for each edge \(ei = (vi, wi)\) in \(Ei\), there exists an edge \(es = (vs, ws)\) in \(Es\) such that: a) \(Li(ei) = Ls(es)\); b) for each pair \((a : c) \in Ai(ei)\) there exists a pair \((a : t) \in As(es)\) with \(type(c) = t\); and c) \(vi \sim vs\) and \(wi \sim ws\). We say that \(ei\) corresponds to \(es\), denoted by \(ei \sim es\). □

As for relational schema, a schema graph determines the structure, meta-information and typing that instance graphs must satisfy. It is clear that an instance \(IG\) of \(SG\) assigns a set of vertices (resp. edges) to each vertex \(vs\) (resp. edge \(es\) in \(SG\) that have the same label as \(vs\) (resp. \(es\)). Moreover, a vertex \(vi\) (resp. edge \(ei\)) in \(IG\) corresponds to a vertex \(vs\) (resp. edge \(es\)) in \(SG\) if the value \(c\), attached to any attribute \(a\) of \(vi\) (resp. edge \(ei\)), respects the type \(t\) given for \(a\) within \(vs\) (resp. \(es\)).
Example 2. Fig. 2 depicts an example of a schema graph where each vertex (resp. edge) is represented naturally with its label (e.g. vertex Admissions, edge Admissions-Doctors) and a list of typed attributes (e.g. AdmiNo:Integer). As a special case, the value of the attribute Pk on some vertex refers to the value of the function Pk on this vertex (e.g. Pk:AdmiNo on vertex Admissions). Moreover, the values of attributes s and d on some edge e refer to the values of the function Fk(e, s) (resp. Fk(e, d)) at this edge (e.g. s: Doc_No and d: DoctorNo on edge Admissions-Doctors). The use of these special attributes (Pk, s and d) will be detailed later. One can see that the data graph of Fig. 1 is an instance of the schema graph of Fig. 2 since each vertex (resp. edge) of the former corresponds to some vertex (resp. edge) of the latter. □

Finally, a graph database is defined with $D_G = (S_G, I_G)$ where $S_G$ is a schema graph and $I_G$ is an instance of $S_G$.

3.2 Schema Mapping (SM)

Given a relational schema $S_R = (R, A, T, \Sigma)$, we propose a schema mapping (SM) process that produces a schema graph $S_G = (V_S, E_S, L_S, A_S, Pk, Fk)$ as follows:

1. For each relation name $r \in R$ in $S_R$, there exists a vertex $v_r \in V_S$ such that $L_S(v_r) = r$;
2. For each $a \in A(r)$ with $T(a) = t$, we have a pair $(a : t) \in A_S(v_r)$;
3. For each primary key $r[a_1, \ldots, a_n] \in \Sigma$, we have $Pk(v_r) = \text{“}a_1, \ldots, a_n\text{”}$;
4. For each foreign key $r[a_1, \ldots, a_n] \rightarrow s[b_1, \ldots, b_n] \in \Sigma$, we have an edge $e = (v_r, v_s) \in E_S$ such that: a) $L_S(e) = (s - r)$; b) $Fk(e, s) = \text{“}a_1, \ldots, a_n\text{”}$; and c) $Fk(e, d) = \text{“}b_1, \ldots, b_n\text{”}$. 
5. A special pair \((\text{vid:Integer})\) is attached to each vertex of \(S_G\) for storage concerns.

Example 3. Fig. 3 depicts (A) a relational schema and (B) its corresponding schema graph. One can see that our schema mapping rules are respected: 1) each relation is mapped to a vertex that contains the label of this relation, its primary key and a list of its typed attributes; 2) each foreign key between two relations (e.g. relations \(\text{Admissions}\) and \(\text{Patients}\) in part A) is represented by an edge between the vertices of these two relations (e.g. edge \(\text{Admissions-Patients}\) in part B).

3.3 Instance Mapping (IM)

Given a relational database \(D_R = (S_R, I_R)\), we propose an instance mapping (IM) process that maps the relational instance \(I_R\) into an instance graph \(I_G = (V_I, E_I, L_I, A_I)\) as follows:

1. For each tuple \(t \in I(r)\), there exists a vertex \(v_t \in V_I\) with \(L_I(v_t) = r\). We denote by \(v_t\) the vertex that corresponds to the tuple \(t\);
2. For each tuple \(t \in I(r)\) and each attribute \(a\) with \(t(a) = c\), we have: \((a : c) \in A_I(v_t)\) if \(a \neq \text{tid}\); and \((\text{vid : c}) \in A_I(v_t)\) otherwise.
3. for each foreign key \(r[a_1, \cdots, a_n] \rightarrow s[b_1, \cdots, b_n]\) defined with \(S_G\) and any tuples \(t \in I(r)\) and \(t' \in I(s)\), if \(t(a_i) = t'(b_i)\) for each \(i \in [1, n]\), then: there is an edge \(e = (v_t, v_{t'}) \in E_I\) with \(L_I(e) = r - s\).

Example 4. An example of our IM process is given in Fig. 4 where part (A) is the relational instance and part (B) is its corresponding instance graph.

It is clear that the special attribute \(\text{vid}\) is used to preserve the tuples identification (i.e. the value of attribute \(\text{tid}\)) during the mapping process.
Complete Direct Mapping From Relational Databases To Property Graphs

Fig. 4: Example of instance mapping.

We notice that our data mapping process is query language independent in the sense that any query language (e.g. Cypher, Gremlin, PGQL) can be applied over the resulting data graph.

4 Properties of CM

We show that our CM satisfies the three fundamental mapping properties \[13\]: information preservation, query preservation and semantics preservation.

4.1 Information Preservation

First, we explain that CM does not lose any part of the information in the relational instance being translated:

Theorem 1. The direct mapping CM is information preserving.

Proof. Theorem\[1\] can be proved easily by showing that there exists a computable mapping \(CM^{-1}: D_G \rightarrow D_R\) that reconstructs the initial relational database from the generated graph database. Since our mapping CM is based on two steps (schema and instance mappings), then \(CM^{-1}\) requires the definition of \(SM^{-1}\) and \(IM^{-1}\) processes. Due to the space limit, the definition of \(CM^{-1}\) is given in Appendix A.

4.2 Query Preservation

Second, we show that the way CM maps relational instances into instance graphs allows one to answer the SQL query over a relational instance by translating it into an equivalent Cypher query over the generated graph instance.
Algorithm 1 S2C Algorithm

Input: A simple SQL query $Q_s$, A relational schema $S_R$
Output: Its equivalent Cypher query $Q_c$.
1: Rename relations (via AS-clause) in $Q_s$ if not applied;
2: Extract the Select-clause (SC), the From-clause (FC) and the Where-clause (WC) from $Q_s$;
3: Generate a Match-clause (mc) from FC:
   a) by translating each relation name $r$ in SC into a vertex with label $r$; and
   b) by representing each join in $Q_s$ with an edge in mc basing on $S_R$;
4: Generate a Where-clause (wc) from WC by translating each condition over a relation (attribute) in WC into a condition over the corresponding vertex (attribute);
5: Generate a Return-clause (rc) from SC by translating each relation (attribute) in SC into a corresponding vertex (attribute);
6: Generate a Cypher query $Q_c$ by combining mc, wc and rc;
7: Return $Q_c$;

Theorem 2. The direct mapping CM is query Preserving.

Proof

4.3 Semantic Preservation

Finally, we show that CM is semantic preserving by checking consistency (resp. inconsistency) of relational database and graph database. Recall that a direct mapping is semantic preserving if any consistent (resp. inconsistent) relational database is translated into a consistent (resp. inconsistent) graph database. While consistency of relational database is well-known, no definition is given for graph database since there exists no standard for (schema) graph definition. To overcome this limit, we added the functions $P_k$ and $F_k$ to our schema graph definition in order to make possible the consistency checking for graph databases.

Definition 8 (Graph consistency). For any graph database $D_G = (S_G, I_G)$, the instance $I_G$ is said to be consistent w.r.t $S_G$ if:

- For each vertex $v_s \in V_s$ with $P_k(v_s) = \langle a_1, ..., a_n \rangle$ and each vertex $v_i \in V_i$ that corresponds to $v_s$; there exists no pair $(a_i : \text{NULL}) \in A_i(v_i)$ with $i \in [1, n]$. Moreover, for each $v'_i \in V_i \setminus \{v_i\}$ that corresponds to $v_s$, the following condition must not hold: for each $i \in [1, n]$, $(a_i : c) \in A_i(v_i) \cap A_i(v'_i)$. 


For each edge \( e_s \in E_s \) with \( Fk(e_s, s) = \langle a_1, ..., a_n \rangle \) and \( Fk(e, d) = \langle b_1, ..., b_n \rangle \), if \( e_s = (v_1, v_2) \in E_f \) is an edge that corresponds to \( e_s \) then we have: \( (a_i : c) \in A_f(v_1) \) and \( (b_i : c) \in A_f(v_2) \) for each \( i \in [1, n] \). □

Intuitively, the consistency of graph databases is inspired from that of relational databases.

**Theorem 3.** The direct mapping CM is semantic Preserving.

**Proof.** The proof of Theorem 3 is straightforward and can be done by contradiction based on the mapping rules of IM (Section 3.3). Given a relational database \( D_R = (S_R, I_R) \) and let \( D_G = (S_G, I_G) \) be its equivalent graph database generated by CM. We suppose that CM is not semantic preserving. This means that either (A) \( I_R \) is consistent and \( I_G \) is inconsistent; or (B) \( I_R \) is inconsistent while \( I_G \) is consistent. We give only proof of case (A) since that of case (B) can be done in a similar way.

We suppose that \( I_R \) is consistent w.r.t \( S_R \) while \( I_G \) is inconsistent w.r.t \( S_G \).

1) There exists a vertex \( v_1 \in V_I \) that corresponds to a vertex \( v_s \in V_s \) where: a) \( Pk(v_s) = \langle a_1, ..., a_n \rangle \); and b) \( (a_i : NULL) \in A_I(v_i) \) for some attribute \( a_i \in [1, n] \).

Based on mapping rules of IM process, \( v_i \) corresponds to some tuple \( t \) in \( I_R \) and attributes \( a_1, ..., a_n \) correspond to a primary key defined over \( I_R \) by \( S_G \). Then (b) implies that the tuple \( t \) assigns a \( NULL \) value to the attribute \( a_i \) which makes \( I_R \) inconsistent. However, we supposed that \( I_R \) is consistent.

2) There are two vertices \( v_1, v_2 \in V_I \) that correspond to a vertex \( v_s \in V_s \) where: a) \( Pk(v_s) = \langle a_1, ..., a_n \rangle \); and b) \( (a_i : c) \in A_I(v_i) \cap A_I(v_2) \) for each \( i \in [1, n] \).

Based on mapping rules of IM process, \( v_1 \) (resp. \( v_2 \)) corresponds to some tuple \( t_1 \) (resp. \( t_2 \)) in \( I_R \) and attributes \( a_1, ..., a_n \) correspond to a primary key defined over \( I_R \) by \( S_G \). Then (b) implies that the tuples \( t_1 \) and \( t_2 \) assign the same value to each attribute \( a_i \) which makes \( I_R \) inconsistent. However, we supposed that \( I_R \) is consistent.

3) There exists an edge \( e = (v_1, v_2) \) in \( E_I \) that corresponds to an edge \( e = (v_s, v_d) \) in \( E_s \) where: a) \( Fk(e, s) = \langle a_1, ..., a_n \rangle \); b) \( Fk(e, d) = \langle b_1, ..., b_n \rangle \); and c) there exists some attribute \( a_i \in [1, n] \) with \( (a_i : c_1) \in A_I(v_1) \), \( (a_i : c_2) \in A_I(v_2) \), and \( c_1 \neq c_2 \). Based on mapping rules of IM process, \( v_1 \) (resp. \( v_2 \)) corresponds to some tuple \( t_1 \) (resp. \( t_2 \)) in \( I_R \), \( v_s \) (resp. \( v_d \)) corresponds to some relation \( s \) (resp. \( d \)) in \( S_R \), the function \( Fk \) over edge \( e \) refers to a foreign-key \( s[\langle a_1, ..., a_n \rangle] \rightarrow d[\langle b_1, ..., b_n \rangle] \) defined over \( I_R \) by \( S_G \). Then (b) implies that the tuple \( t_1 \) assigns a value to some attribute \( a_i \) that is different to that assigned by tuple \( t_2 \) to attribute \( b_i \). This means that there is a violation of foreign-key by tuple \( t_1 \) which makes \( I_R \) inconsistent. However, we supposed that \( I_R \) is consistent.

Therefore, each case of \( I_G \) inconsistency leads to a contradiction, which means that if \( I_R \) is consistent then its corresponding \( I_G \) cannot be inconsistent.

By doing proof of part (B) in a similar way, we conclude that if \( I_R \) is consistent then its corresponding instance graph \( I_G \) must be consistent. This completes the proof of Theorem 3. □
5 Conclusion and Future works

In this paper, we proposed a complete mapping process that translates any relational database into an equivalent graph database by considering both schema and data mapping. Our mapping preserves the information and semantics of the relational database and maps any SQL query, over the relational database, into an equivalent Cypher query to be evaluated over the produced graph database.

We plan to extend our model to preserve another mapping property, called monotonicity [13], which ensures that any update to the relational database will not require generating the corresponding graph database from scratch. Also, we will enrich the definition of the relational schema with more integrity constraints. We are conducting an experimental study based on real-life databases to check our approach’s efficiency and effectiveness.

References

1. Neo4j ETL. https://neo4j.com/developer/neo4j-etr/
2. Renzo Angles, Harsh Thakkar, and Dominik Tomaszuk. Mapping RDF databases to property graph databases. IEEE Access, pages 86091–86110, 2020.
3. Roberto De Virgilio, Antonio Maccioni, and Riccardo Torlone. Converting relational to graph databases. page 1, 2013.
4. Roberto De Virgilio, Antonio Maccioni, and Riccardo Torlone. R2g: A tool for migrating relations to graphs. pages 640–643, 2014.
5. Wenfei Fan, Xin Wang, and Yinghui Wu. Answering pattern queries using views. IEEE Trans. Knowl. Data Eng., pages 326–341, 2016.
6. Nadine Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindauer, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. pages 1433–1445, 2018.
7. Paolo Guagliardo and Leonid Libkin. A formal semantics of SQL queries, its validation, and applications. Proc. VLDB Endow., pages 27–39, 2017.
8. Shunyang Li, Zhengyi Yang, Xianhang Zhang, Wenjie Zhang, and Xuemin Lin. SQL2Cypher: Automated Data and Query Migration from RDBMS to GDBMS. 2021.
9. Shota Matsumoto, Ryota Yamanaka, and Hirokazu Chiba. Mapping rdf graphs to property graphs. pages 106–109, 2018.
10. Ognjen Orel, Slaven Zakošek, and Mirta Baranović. Property oriented relational-to-graph database conversion. Automatika, pages 836–845, 2017.
11. Juan F. Sequeda, Marcelo Arenas, and Daniel P. Miranker. On directly mapping relational databases to RDF and OWL. pages 649–658, 2012.
12. Radu Stoica, George Fletcher, and Juan F. Sequeda. On directly mapping relational databases to property graphs. pages 1–4, 2019.
13. Radu-Alexandru Stoica. R2pg-dm: A direct mapping from relational databases to property graphs. Master’s thesis, Eindhoven University of Technology, 2019.
APPENDIX

A Proof of Theorem 1

We prove Theorem 1 by providing a computable mapping $CM^{-1} : D_G \rightarrow D_R$ that reconstructs the original relational database from the generated graph database. To do so, $CM^{-1}$ must consist of two processes: 1) $SM^{-1}$ that reconstructs the original relational schema from the generated graph schema; and 2) $IM^{-1}$ that reconstructs the original relational instance from the generated instance graph. We provide hereafter complete definitions of $SM^{-1}$ and $IM^{-1}$.

**SM$^{-1}$ transformation rules.** Given a schema graph $S_G = (V_s, E_s, L_s, A_s, Pk, Fk)$, the reversed schema mapping $SM^{-1}$ produces a relational schema $S_R = (R, A, T, \Sigma)$ as follows:

- For each vertex $v_r \in V_s$, we create a relation $r \in R$ with name $L_s(v_r)$. We denote by $v_r$ the vertex that corresponds to the relation $r$.
- For each pair $(a : t) \in A_s(v_r)$, we add the attribute $a$ to $A(r)$ with $T(a) = t$.
- For each $Pk(v_r) = "a_1, ..., a_n"$, we add $r[a_1, \ldots, a_n]$ to the set $\Sigma$.
- For each edge $e = (v_r, v_s)$ with $Fk(e, s)="a_1, ..., a_n"$ and $Fk(e, d) = "b_1, ..., b_n"$, we add $r[a_1, \ldots, a_n] \rightarrow s[b_1, \ldots, b_n]$ to the set $\Sigma$.

Notice that the special pair $(vid : Integer)$ is ignored by the mapping $SM^{-1}$ since it has no equivalent part in the relational schema. However, it allows instance graphs to preserve the tuples identification.

**IM$^{-1}$ transformation rules.** Given an instance graph $I_G = (V_I, E_I, L_I, A_I)$, the reversed instance mapping $IM^{-1}$ produces the original relational instance $I_R$ as follows:

- For each vertex $v \in V_I$ with $L_I(v) = r$, we create a tuple $t \in I(r)$ such that:
  - $t(tid) = c$ where $(vid : c) \in A_I(v)$; and
  - $t(a) = c$ for each $(a : c) \in A_I(v)$.

From the definition of IM mapping (Section 3.3), one can verify that for any two vertices $v_1, v_2 \in V_I$, if $L_I(v_1) = L_I(v_2)$ then $v_1$ and $v_2$ have different values for attributes $vid$. Based on this remark, tuples generated by $IM^{-1}$ mapping satisfy the following condition: for any two tuples $t_1, t_2 \in I(r)$, $t_1(tid) \neq t_2(tid)$.

B Running example of query mapping by S2C 1

Example 5. Fig. 5 represents (A) an SQL query $Q_s$ and (B) its equivalent Cypher query $Q_c$. It is clear to see that each SQL clause is translated into a Cypher clause as explained by algorithm S2C. Precisely, each relation name (Admissions, Doctors, Patients) in From-Clause is translated into a vertex in mc labeled...
Fig. 5: Example of (A) an SQL query $Q_s$ and (B) its equivalent Cypher query $Q_c$.

| Admi_Date |
|------------|
| 30/11/2021 |

(A)  

| a.Admi_Date |
|-------------|
| 30/11/2021  |

(B)  

with name of this relation. Moreover, each join operation between two relations is mapped to an edge between vertices corresponding to these relations. Each condition (e.g. $a.DocNo = b.DoctorNo$) in Where-Clause is translated into an equivalent condition in $wc$. Finally, the selected items to return are extracted from the Select-clause. The final Cypher query $Q_c$ is composed by combining the three clauses Match ($mc$), Where ($wc$) and Return ($rc$).

Consider the relational instance $IR$ of Fig.4 (A) and its equivalent instance graph $IG$ in Fig.4 (B). When we apply $Q_s$ over $IR$ (resp. $Q_c$ over $IG$), we obtain the result table depicted in Fig.6 (A) (resp. (B)). It is easy to see that both $Q_s$ and $Q_c$ return the same information. □