# Reversible Mapping of Relational and Graph Databases

**A. M. Palagashvili[a,*] and S. A. Stupnikov[b,**]**

[a] *Lomonosov Moscow State University, Moscow, 119991 Russian Federation*

[b] *Institute of Informatics Problems of the Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, Moscow, 119333 Russian Federation*

**e-mail: 0583463@gmail.com*

***e-mail: sstupnikov@ipiran.ru*

**Abstract**—In the contemporary world, a large amount of heterogeneous data are accumulated, which have different nature and require specific approaches to their processing and storage. Even within one information system, it is often required to process data represented in different data models from the same knowledge domain. One way to solve this problem is multimodel databases, which simultaneously support several data models. These database management systems generally imply the division into "primary" and "secondary" data models, as well as require explicit mapping of data schemas. The relational data model appeared a long time ago; it is well studied and widely used. On the other hand, graph data models, which are suitable for social networks, recommender services, transport networks, etc., are become increasingly popular. In this paper, we propose algorithms for mapping relational and graph databases the composition of which is an identity mapping. These algorithms form a basis for creating multimodel graph-relational database management systems.

**Keywords:** relational data model, graph data model, database mapping, multimodel database management systems

**DOI:** 10.1134/S1054661823020098

## INTRODUCTION

Modern information systems are used to store and process large amounts of heterogeneous data. To effectively process data of different nature, suitable data models are required. When using several database management systems (DBMSs) in one information system, a number of difficult problems arise, e.g., data synchronization and data mapping from one model to another.

The relational data model [2] appeared quite a long time ago; it is the most popular data model and has the widest support in modern DBMSs. The basic entities in this model are relations, which have attributes and contain tuples of values corresponding to these attributes. For unique identification of tuples, primary keys are used. To create links between relations, foreign keys are created. The most popular relational DBMSs are Oracle, Microsoft SQL Server, PostgreSQL, and MySQL. Each of them uses its own dialect of the structured query language (SQL). The latest ISO standard[1] for the SQL [1] was released in 2016.

The graph model appeared somewhat later; however, it has already managed to find its niche in the software development industry.[2] The basic entities of the graph model[3] are nodes and edges. Edges connect pairs of nodes; both nodes and edges can be assigned attributes with values of built-in types (integer, float, string, etc.). Edges can be either directed or undirected. Both edges and nodes are labeled to reflect the semantics of the objects to be modeled. Currently, one of the most popular graph DBMSs is Neo4j, which uses its own Cypher query language [3]. A number of other graph DBMSs also support this query language. However, there is currently no unified standard for graph query languages. For instance, some DBMSs use SPARQL, an RDF query language recommended by the World Wide Web Consortium (W3C). In 2019, the development of a unified standard for the graph query language (GQL)[4] began; this language is largely based on the experience of Cypher.[5]

One of the ways to integrate heterogeneous data, in particular, those stored in relational and graph models, is the use of multimodel DBMSs. A specific feature of these DBMSs is the simultaneous support of several data models. For instance, AgensGraph DBMS allows one to create graph and relational databases and to provide graph and relational languages to work with them. Typically, these DBMSs have "primary" and "secondary" data models.

[1] https://www.iso.org/standard/63555.html.

[2] https://neo4j.com/customers.

[3] We consider the model of attributed graphs, which is used in most of the modern graph DBMSs. The entities like hypergraphs or hypervertices fall outside the scope of this paper.

[4] https://www.gqlstandards.org.

[5] https://neo4j.com/blog/gql-standard-query-language-property-graphs.

Received December 6, 2022; revised December 6, 2022; accepted December 6, 2022

Graph-relational DBMSs generally perform very well in simultaneous analysis of heterogeneous (relational and graph) data [6]. For instance, in e-commerce applications, data on users and their orders are accumulated in a relational DBMS, while data on their social connections are accumulated in a graph DBMS. In this case, processing an analytical query "goods ordered by friends of a user whose credit limit exceeds 3000" requires a combination of relational and graph data from different DBMSs. For this purpose, it is required to create an integrating superstructure over the DBMSs with different models and reconcile structural differences between the models. In this case, the bottleneck is the query execution on the data with significant structural differences. However, with the multimodel approach, methods to overcome these structural differences are already built into a single DBMS. Thus, the user accesses one interface of one platform, which reduces the cost of data integration.

Even though there are many multimodel DBMSs that support graph and relational models, each of them has its flaws. For instance, some of them use document models as primary ones, while the other models are supported by mapping data to views. Other DBMSs allow one to create independent databases with different data models, access each of them using its own language, and provide their integration through hybrid queries.

Thus, the problems of developing multimodel graph-relational DBMSs remain important. The formal basis of these DBMSs, including reversible mapping of relational and graph data models, requires thorough investigation.

In this paper, we consider algorithms that implement relational-to-graph and graph-to-relational mappings the sequential compositions of which is an identity mapping. Such mapping is called *reversible* in the paper. The paper is organized as follows. Section 1 contains an overview of existing multimodel DBMSs that simultaneously support graph and relational models. Section 2 discusses some related works. Section 3 provides formal definitions of graph and relational databases, describes the mapping algorithms, and proves their properties. Section 4 details the software implementation of the mappings and considers an application example for the Northwind database. The directions for further research are discussed in Conclusions.

## 1. EXISTING MULTIMODEL DATABASE MANAGEMENT SYSTEMS

A review of modern multimodel DBMSs that support graph and relational models was presented in [6, 7], so this section is confined to updating this review. According to the db-engines rating,[6] the following multimodel DBMSs that support graph and relational data models are currently the most popular: AgensGraph, MarkLogic, OrientDB, Virtuoso, Fauna, and TypeDB.

AgensGraph is based on Postgres DBMS. AgensGraph separately stores relational and graph data; for their processing, Cypher and SQL are used. To integrate data models, AgensGraph uses Cypher queries in the FROM section of SQL queries. In addition, the use of SQL subqueries in Cypher queries is permitted if the result of these subqueries is one record. It is not possible to access graph data using SQL and relational data using Cypher.

MarkLogic DBMS uses a document model as the primary one. Data are stored as documents included in collections. MarkLogic allows one to create relational and graph (RDF[7]) data representations by using mapping templates. The resulting representations can be queried using SQL and SPARQL, respectively.

OrientDB also uses a document model as the primary one. To support the graph model, OrientDB provides node and edge classes, which are also documents. To work with nodes and edges, an SQL dialect with graph query extensions is used. The Gremlin graph functional language is also supported. OrientDB does not support links typical of relational DBMSs. Instead, this DBMS provides a mechanism for creating links between documents. These links can be used to model relations between entities.

Virtuoso is a hybrid between a DBMS engine and a mediator for other DBMSs, e.g., MySQL, PostgreSQL, Microsoft SQL Server, ODBC/JDBC-compatible DBMSs, etc. Virtuoso is based on an object-relational data model. To form a graph model, Virtuoso maps relations to an RDF graph by means of automatic ontology generation. The resulting graph can then be processed using SPARQL. However, this platform does not allow one to work with the graph database as with the primary one.

Fauna DBMS supports graph, relational, and document models. The basic entities of this DBMS are collections and documents. As a query language, it uses its own functional language (FQL). Different data models are supported by combining (in FQL) the graph and relational approaches. This DBMS cannot be called either fully graph or fully relational. Moreover, the use of the proprietary query language complicates its integration into existing information systems.

TypeDB is a DBMS that combines elements of relational and graph models. For data organization, TypeDB uses the following abstractions: entity, relation, and attribute. On the basis of these abstractions, user-defined types can be created and related. The query language of TypeQL allows one to query data in a template matching manner, which is similar to the approach used in Cypher. However, it also cannot be said that this DBMS fully implements both relational and graph models.

[6] https://db-engines.com/en/ranking/graph+dbms.

[7] https://www.w3.org/RDF.

## 2. RELATED WORKS

In [4], an approach to extracting schemas from graph databases, which takes into account most of the concepts typical of the graph data model, was proposed.

In [8], the authors shared their experience of migration from relational to graph databases. Some simplest ideas for mapping relational databases to graph databases were proposed, all tuples from relations were mapped to graph nodes.

In [9], an algorithm for mapping a relational database to a graph one was presented. In particular, the authors proposed an idea of extracting relations that model many-to-many relations, which is also used in this study. In addition, the authors aggregated values from different tuples in one node to speed up graph database queries.

In [5], the conversion of a relational database to a graph database on big data was implemented. In that case, the mapping of a relational scheme to a graph scheme was defined by the user.

Finally, in the paper [7] preceding this one, some basic ideas for reversible mapping of graph and relational databases were proposed. In particular, relations with two foreign keys were mapped to edges. In this paper, this approach is refined, formalized, and implemented in software, with the main goal being to ensure the identity of compositions of database mappings.

## 3. ALGORITHMS FOR MAPPING RELATIONAL AND GRAPH DATABASES

### *3.1. Database Definitions*

Before describing the algorithms, we provide formal definitions of graph and relational databases.

Relational database.

• Relational database $RDB = \{R_1, R_2, \ldots, R_n\}$ consists of many relations.

• $R_i = (name, attrs, PK, FK, t)$ is a relation: *name* is the name of the relation, *attrs* is the list of its attributes, *PK* is the primary key, *FK* is the set of foreign keys, and *t* is the set of tuples.

• $attrs = ((name, type), \ldots)$ is the list of attributes, where *name* is the name of an attribute and *type* is the type of an attribute.

• $PK = \{name_1, \ldots\}$ is the set of names of attributes that constitute the primary key of a relation.

• $FK = \{(name, Frel, Fattr), \ldots\}$ is the set of foreign keys, i.e., tuples that include the name of an attribute (*name*), the name of a foreign relation (*Frel*), and the name of an attribute in a foreign relation (*Fattr*).

• $t = \{(value_1, value_2, \ldots), \ldots\}$ is the set of tuples, where $value_i$ is a value corresponding to the *i*th attribute of a relation.

It should be noted that, in this paper, we do not consider recursive relations, i.e., in which relation *Frel* from a foreign key coincides with the relation itself.

**Graph database.** Let us first list the constraints to which we adhere in the following discussion:

• Each node has exactly one *label* (symbolic tag), and the set of nodes with the same label is called a *node class*.

• All nodes with the same label have the same set of attributes.

• Each edge has exactly one label and the set of edges with the same label is called an *edge class*.

• All edges with the same label have the same set of attributes.

• Edges have no direction.

• For edge class *C*, there can be no more than one pair of node classes *A*, *B* such that edges of class *C* connect nodes from classes *A* and *B*.

• For a pair of node classes *A*, *B*, there can be no more than one edge class *C* the edges of which connect nodes from classes *A* and *B*.

Let us now define the graph database.

• $GDB = (NC, EC)$ is a graph database, which includes a set of node classes (*NC*) and a set of edge classes (*EC*).

• $NC = \{Nc_1, \ldots, Nc_n\}$ is the set of node classes.

• $EC = \{Ec_1, \ldots, Ec_m\}$ is the set of edge classes.

• $Nc_i = (label, PK, V)$ is a node class where all nodes have the same label.

• *PK* is the primary key for the node class.

• $V = \{N_1, N_2, N_3, \ldots\}$ is the set of nodes.

• $N_i = (label, attrs, values)$ is a node that has a certain label, set of attributes (*attrs*), and *values* corresponding to them.

• $Ec_i = (label, m2m, E)$ is an edge class where all edges have the same label and *m2m* is a Boolean flag that indicates whether this edge class is an m2m class (see the definition below).

• $E = \{E_1, E_2, E_3, \ldots\}$ is the set of edges.

• $E_i = (label, attrs, values, node_a, node_b)$ is an edge that has a label *L*, set of attributes (*attrs*) with their *values*, and links to $node_a$ and $node_b$ connected by this edge.

• $attrs = \{(name_1, type_1), \ldots\}$ is a set of attributes, where $name_i$ is an attribute name, $type_i$ is an attribute type, and *values* is a set of values that corresponds to the set of attributes *Attrs*.

### *3.2. Requirement for the Composition of the Algorithms and Many-to-Many Relations*

Below in this section, we describe two algorithms: for mapping a relational database to a graph one (φ) and for mapping a graph database to a relational one

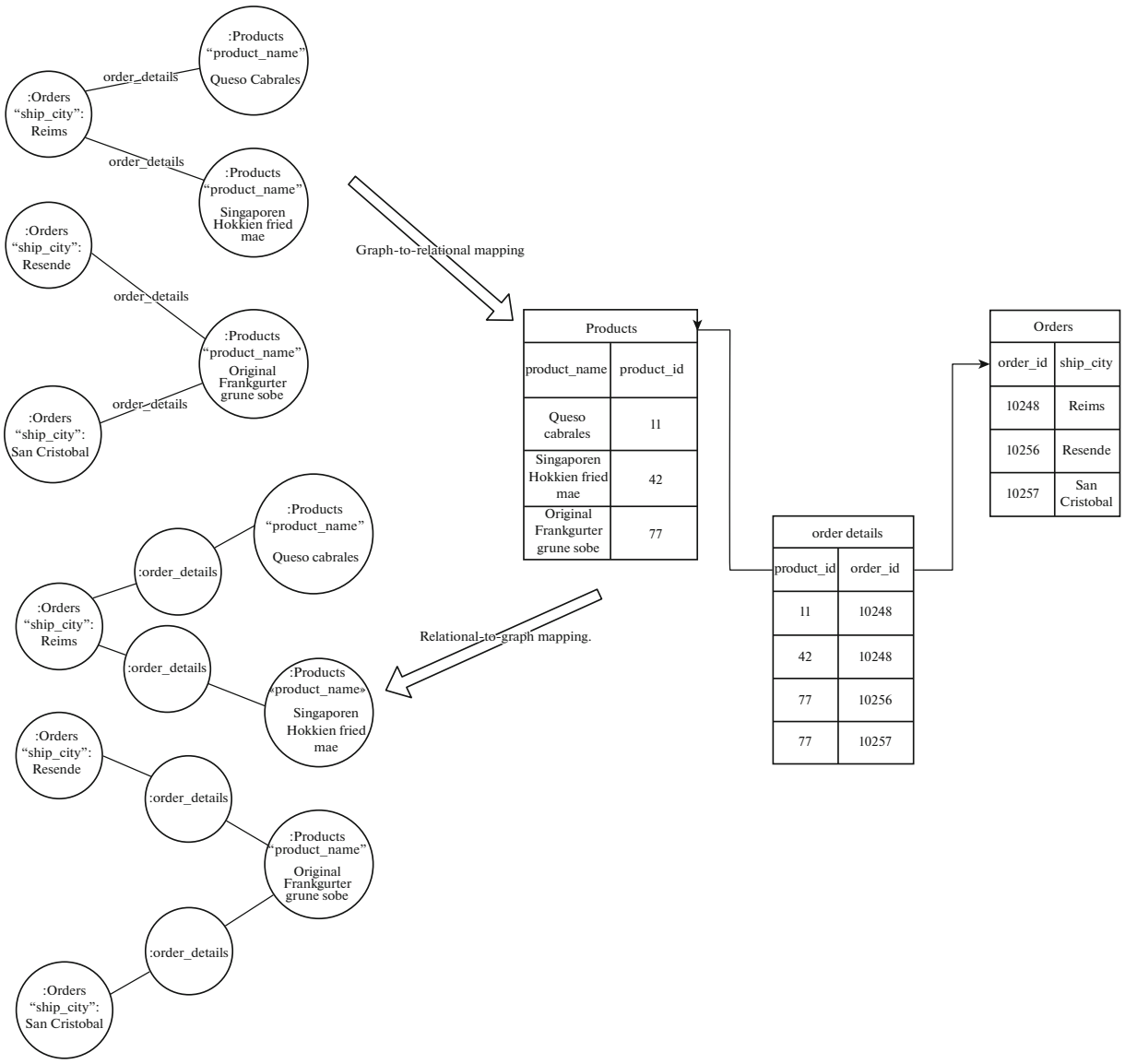


**Fig. 1.** Successive application of the mappings to the graph and relational databases when modeling the many-to-many relation.

(ψ). When developing these algorithms, one of the main requirements was that the compositions of the algorithms be identity mappings, i.e., $\psi(\varphi(rdb)) = rdb$ for any relational database *rdb* and $\varphi(\psi(gdb)) = gdb$ for any graph database *gdb*.

In this case, the mappings provide the equivalent representation of data in relational and graph databases, which is important when developing a multimodel graph-relational DBMS. In particular, this eliminates the need to select a primary database in a multimodel DBMS.

To meet the requirement mentioned above, both the mappings must be considered simultaneously to ensure their consistency. It turns out that one of the main problems in constructing these mappings is the modeling of many-to-many relations. These relations (unlike one-to-one or one-to-many relations) cannot be modeled in a relational database by using foreign keys; an additional relation is required. That is why a straightforward approach to mapping a relational database to a graph database whereby tuples of all relations are represented by nodes does not allow us to construct a pair of mappings the compositions of which are identity mappings. An example is shown in Fig. 1.

Here, the many-to-many relation between orders and products, which is given by the *order_details* edges, is represented in the relational database by the additional relation *Table*. In the case of inverse mapping to the graph database, the tuples of this relation are represented as nodes, and the identity of the mapping composition is violated.

Thus, when constructing the mappings, it is necessary to take into special account many-to-many relations and, when mapping data from the relational model to the graph one, to represent the correspond-

ing relations with edges. Let us introduce the definitions required to represent many-to-many relations in relational and graph databases.

**Definition.** Relation $R$ is called an *m2m relation* if it has exactly two foreign keys referring to different relations and no other relation has foreign keys referring to relation $R$:

- $R.FK = \{fk_1, fk_2\} \wedge fk_1.rel \neq fk_2.rel$.
- $\forall R' \in RDB, F \in R'.FK$ $(R' \neq R \rightarrow F.rel \neq R.name)$

It should be noted that relations $fk_1.rel$ and $fk_2.rel$ are not m2m relations because relation $R$ refers to them (which contradicts the second condition).

**Definition.** Edge class $Ec$ is called an *m2m class* if one of the following two conditions is met:

- this edge class models a many-to-many relation (as in the example shown in Fig. 1);
- $\exists e \in Ec : |e.values| > 0$.

For convenience, all edge classes where edges have attributes are regarded as m2m classes, because these edge classes cannot be represented in the relational model by foreign keys (only by an additional relation).

### *3.3. Mapping a Relational Database to a Graph Database*

Algorithm 1 represents the basic steps of mapping a relational database (*rdb*) to a graph database (*gdb*).

**Algorithm 1.** Mapping a relational database to a graph database.

```
   Function Relational2Graph(rdb)
1  gdb := {}
2  for each r: rdb do
3    if m2m(r) = false then
4      nc := ()
5      nc.label := r.name
6      nc.PK := r.PK
7      for each tuple: r.t do
8        node := ()
9        node.label := r.name
10       node.attrs := r.attrs
11       node.values := tuple
12       insert(nc.V, node)
13     insert(gdb, nc)
14
15 for each r: rdb do
16   if m2m(r) = true then
17     ec := ()
18     ec.label := r.name
19     ec.m2m := true
20     gdb := gdb.insert(ec)
21     for each tuple: r.t do
22       edge := ()
23       edge.label := r.name
24       edge.attrs := r.attrs
25       edge.values := tuple
26       edge.node_a, edge.node_b :=
27         get_incidental_nodes(r, tuple, gdb)
28       insert(ec, adge)
29
30 for each r: rdb do
31   if m2m(r) = false then
32     for each f: r.FK do
33       ec := ()
34       ec.label := f.Fattr
35       ec.m2m := false
36       create_edges(f, r, rdb, gdb)
37       gdb := gdb.insert(ec)
38 return gdb
39
```

An empty instance of the graph database is created (line 1). For each relation *r* in database *rdb* that is not an m2m relation, a node class of the same name is created (lines 2–6 and 13). For each tuple from *r*, a node with the same attribute values is created (lines 8–12).

Then, for each m2m relation, an edge class of the same name is created (lines 15–20). For each tuple from *r*, an edge with the same attribute values is created (lines 21–25).

Given the m2m relation *r* and *tuple* from this relation, function *get_incidental_nodes*(*r*, *tuple*, *gdb*) returns the nodes previously mapped to *gdb* (lines 2–13) from the tuples referenced by *tuple* through foreign keys of relation *r* (lines 27 and 28).

Then, for each relation *r* in *rdb* that is not an m2m relation, edges corresponding to foreign key references are created. For each foreign key *f*, an edge class is created (lines 33–35 and 37) with a label corresponding to the name of the foreign key attribute (*f.Fattr*). Given relation *r* and its foreign key *f*, function create_edges(*f*, *r*, *rdb*, *gdb*) creates edges between the nodes mapped from the tuples of this relation and the nodes mapped from the tuples of relation *f.rel* referenced by *f* (line 36). The edges are added to the corresponding class.

### *3.4. Mapping a Graph Database to a Relational Database*

Algorithm 2 represents the basic steps of mapping a graph database (*gdb*) to a relational database (*rdb*).

**Algorithm 2.** Mapping a graph database to a relational database.

```
   Function Graph2Relational(gdb)
1  rdb := {}
2  for each nc: gdb.NC do
3  |  r := ()
4  |  r.name := nc.label
5  |  r.PK := nc.PK
6  |  r.attrs := extract_schema_from_node_class(nc)
7  |  for each node: nc.V do
8  |  |  tuple := extend_tuple(node.values, r.attrs)
9  |  |  insert(r.t, tuple)
10 |  insert(rdb, r)
11
12 for each ec: gdb.EC do
13 |  if m2m(ec) = true then
14 |  |  r := ()
15 |  |  r.name := ec.label
16 |  |  r.attrs := extract_schema_from_edge_class(ec)
17 |  |  for each edge: ec.E do
18 |  |  |  tuple := extend_tuple(edge.values, r.attrs)
19 |  |  |  create_m2m_fk(edge, tuple)
20 |  |  |  insert(r.t, tuple)
21 |  |  insert(rdb, r)
22
23 for each ec: gdb.EC do
24 |  if m2m(ec) = false then
25 |  |  for each edge: ec.E do
26 |  |  |  create_non_m2m_fk(edge, rdb)
27 return rdb
28
```

An empty instance of the relational database is created (line 1).

For each node class *nc*, a relation *r* of the same name is created (lines 3–5 and 10).

Function *extract_schema_from_node_class*(*nc*) traverses all nodes from class *ec* and forms a set of all attributes that nodes from this class can have. This set becomes the set of attributes of relation *r* (line 6).

For each node of class *nc*, a tuple of relation *r* is created (lines 7–9). The conversion of the set of attribute values (*node.values*) into a tuple corresponding to the complete list of attributes of relation *r.attrs* is implemented by function *extend_tuple* (line 8).

**Table 1.** Type compatibility for the algorithms and DBMSs

| Neo4j types | PostgreSQL types |
|---|---|
| String | nvarchar, nchar, text |
| Integer | int, smallint |
| Float | real |

Similarly, for each m2m edge class *ec*, a relation *r* of the same name is created (lines 12–16); for each edge, a tuple in relation *r* is created (lines 17–20). Function *create_m2m_fk* adds foreign keys to *tuple* in accordance with the nodes connected by *edge*.

Then, for each edge class *ec* that is not an m2m class, function *create_non_m2m_fk* creates, for each *edge* from the *ec* class, foreign keys for the corresponding relations in the *rdb* database (lines 23–25).

### *3.5. Properties of the Mapping Compositions*

Suppose that φ is the mapping of a relational database to a graph database that is implemented by Algorithm 1 and ψ is the mapping of a graph database to a relational database that is implemented by Algorithm 2. In the proof given below, the references to the lines of the algorithms have the following form: (X.Y1-Y2), where X is the algorithm number (1 or 2), while Y1 and Y2 are line numbers.

**Statement 1.** For any instance of relational database, $I = \psi(\varphi(I))$.

**Proof.**

Suppose that $I' = \psi(\varphi(I))$. To prove the statement, it is required to prove that $I \subseteq I'$ and $I' \subseteq I$.

1.1. Let us prove that $I \subseteq I'$. We consider an arbitrary tuple $t \in R$, $R \in I$. It is required to prove that the same tuple belongs to a relation of the same name in exemplar $I'$.

The following two cases are possible.

If $R$ is not an m2m relation, then φ maps $t$ to some node $n = \varphi(t)$ (1.7-12).

If $R$ is an m2m relation, then φ maps $t$ to some edge $e = \varphi(t)$. The foreign keys of $t$ are mapped to the incident nodes of the edge. It should be noted that the nodes to be connected by edge $e$ already exist in the graph database at the time of its creation. They are created in lines (1.7-12) because the relations to which $R$ is linked by foreign keys are not m2m relations. According to line (1.19), the class of these edges has the true value of the m2m flag.

Now let us consider inverse mapping ψ. In the first case, $e$ is mapped to tuple $t' \in R'$, $R' \in I'$. In this case, $R.name = n.label = R'.name$ (1.9, 2.4) and $R.attrs = n.attrs = R'.attrs$ (1.10, 2.6); i.e., $R$ and $R'$ are the same-name relations with the same signatures because all nodes of the class with *n.label* are generated from tuples of $R$. In addition, $t = n.values = t'$ (1.11, 2.8); i.e., attribute values are preserved.

In the second case, edge $e$ is also mapped to tuple $t'' \in R''$, $R'' \in I'$. Similarly, $R.name = e.label = R''.name$ (1.18, 2.15), $R.attrs = e.attrs = R''.attrs$ (1.24, 2.16), and $t = e.values = t''$ (1.25, 2.18).

Therefore, $I \subseteq I'$.

1.2. Let us prove that $I' \subseteq I$. We consider an arbitrary tuple $t' \in R'$, $R' \in I'$. It is required to prove that the same tuple belongs to a relation of the same name in instance $I$.

Indeed, tuple $t'$ results from applying ψ in one of two ways:

1. Preimage of $t'$ is some node $n$ (2.7-9);

2. Preimage of $t'$ is some edge $e$ that belongs to an m2m class (2.17-19).

In the first case, $n = \varphi(t)$, where $t \in R$, and $R \in I$. By reasoning as in 1.1, we have $R.name = n.label = R'.name$, $R.attrs = n.attrs = R'.attrs$, and $t = n.values = t'$.

In the second case, edge $e$ can occur only for φ when mapping some tuple $t$ of m2m relation $R$ (1.21-28) because the flag of the m2m edge class is set to true only in (1.19). By reasoning as in 1.1, we arrive at $R.name = e.label = R'.name$, $R.attrs = e.attrs = R'.attrs$, and $t = e.values = t'$.

Therefore, $I' \subseteq I$.

**Statement 2.** For any exemplar of graph database $G$, $G = \varphi(\psi(G))$.

The proof is similar to that of Statement 1.

## 4. SOFTWARE IMPLEMENTATION OF THE MAPPINGS AND AN EXAMPLE OF APPLICATION

For the software implementation of the mappings, we used PostgreSQL (relational DBMS) and Neo4j (graph DBMS with its own query language Cypher).

It should be noted that, since the current version of Neo4j does not support undirected edges, they are represented by pairs of oppositely directed edges.

In addition, Section 4 makes no distinction between the built-in types of the relational and graph models. The implementation uses type compatibility shown in Table 1.

For our experiments, we used the Northwind[8] relational database, which has 12 relations with the number of tuples from several to 2000 in each (more than 4000 tuples in total).

The software implementation of the mappings is carried out in Python[9] and is organized as follows:

1. Data are read from the DBMS and the corresponding abstractions (relations, tuples, nodes, and edges) are represented in RAM.

[8] https://github.com/microsoft/sql-server-samples/tree/master/samples/databases/northwind-pubs.

[9] https://github.com/fibersel/multi-rel-graph/tree/master/schemas.

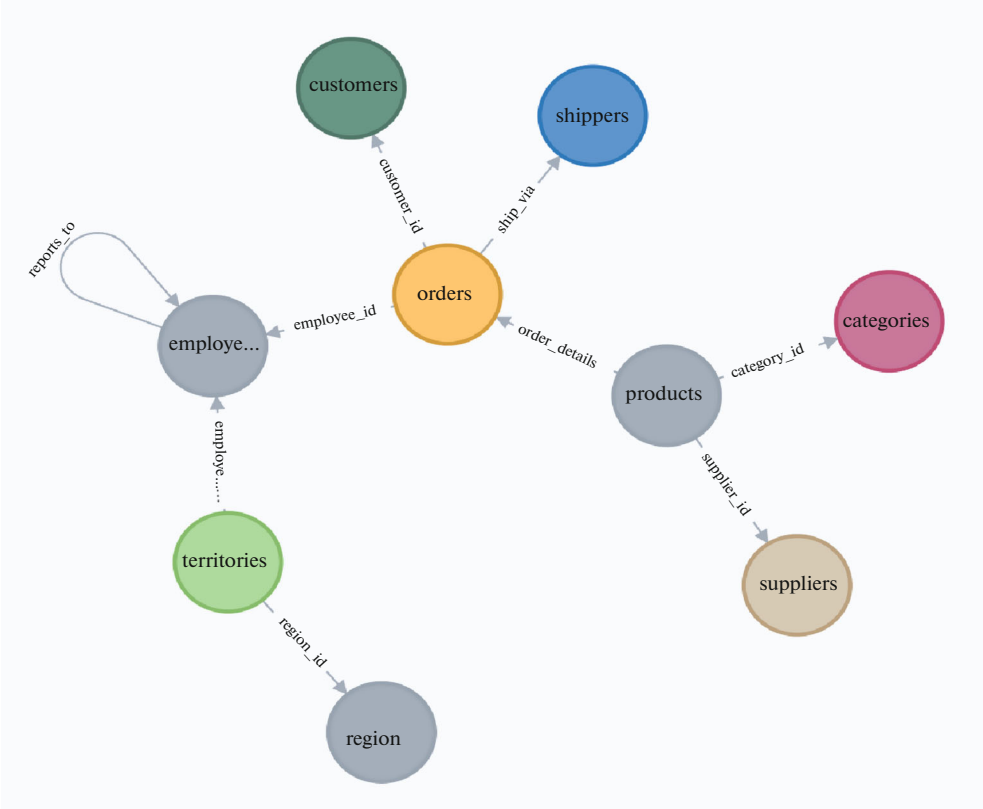


**Fig. 2.** Result of mapping the Northwind database to the Neo4j graph model, visualized using Neo4j Browser.

2. Queries in the corresponding DBMS language are generated to create all necessary structures and fill them with data.

For example, when mapping the tuple (10248, 11, 14, 12, 0) from the *order_details* relation, which has the signature *(order_id, product_id, unit_price, discount)*, the following Cypher query is generated:

```
MATCH (a:products), (b:orders)
WHERE a.product_id = 11 AND b.order_id = 10248
CREATE (a)-[:order_details {unit_price: 14.0, quantity: 12, discount: 0.0}]->(b);
```

This is because *order_details* in Northwind is an m2m relation.

On a computer with Intel Core i7 2.6 GHz and 16 GB RAM, mapping the Northwind relational database to its graph version takes 1 min 40 s, while mapping the graph database to the relational database takes 50 s.

The result of mapping the relational database to the graph one is the data schema shown in Fig. 2.

## CONCLUSIONS

Algorithms for relational-to-graph and graph-to-relational mapping of databases have been developed and the identity of the compositions of these mappings under certain conditions imposed on graph databases has been proved. The algorithms have been implemented in Python for PostgreSQL relational DBMS and Neo4j graph DBMS. The relational-to-graph mapping has been tested on the Northwind database.

The directions for further research include the generalization of the developed algorithms to relational databases with recursive relations and composite foreign keys, as well as graph databases with directed edges and no constraints on node and edge labels, and the reversible mapping of relational and graph query languages consistent with the developed database mappings. In these query languages, we intend to implement support for selection, projection, join, nested queries, recursion, and grouping. Presumably, the query language mappings will be based on a graph-relational algebra.

## ACKNOWLEDGMENTS

We are grateful to Boris Asenovich Novikov, Professor at the Higher School of Economics in St. Petersburg, for the idea of this work.

## FUNDING

This work was supported by the Russian Science Foundation, project no. 22-21-00692 (https://rscf.ru/project/22-21-00692).

## CONFLICT OF INTEREST

The authors declare that they have no conflicts of interest.

## REFERENCES

1. D. Chamberlin, "SQL," in *Encyclopedia of Database Systems*, Ed. by L. Liu and M. T. Özsu (Springer, New York, 2017), pp. 1–9. https://doi.org/10.1007/978-1-4899-7993-3_1091-2
2. D. W. Embley, "Relational model," in *Encyclopedia of Database Systems*, Ed. by L. Liu and M. T. Özsu (Springer, New York, 2018). https://doi.org/10.1007/978-1-4614-8265-9_306
3. N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, "Cypher: An evolving query language for property graphs," in *Proc. 2018 Int. Conf. on Management of Data (SIGMOD '18), Houston, Texas, 2018* (Association for Computing Machinery, New York, 2018), pp. 1433–1445. https://doi.org/10.1145/3183713.3190657
4. A. Augusto Frozza, S. Rodrigues Jacinto, and R. dos Santos Mello, "An approach for schema extraction of NoSQL graph databases, in *IEEE 21st Int. Conf. on Information Reuse and Integration for Data Science (IRI), Las Vegas, 2020* (IEEE, 2020), pp. 271–278. https://doi.org/10.1109/IRI49571.2020.00046
5. S. Lee, B. H. Park, S.-H. Lim, and M. Shankar, "Table2Graph: A scalable graph construction from relational tables using map-reduce," in *IEEE First Int. Conf. on Big Data Computing Service and Applications, Redwood City, Calif., 2015* (IEEE, 2015), pp. 294–301. https://doi.org/10.1109/BigDataService.2015.52
6. J. Lu and I. Holubová, "Multi-model databases: A new journey to handle the variety of data," ACM Comput. Surv. **52**, 55 (2019). https://doi.org/10.1145/3323214
7. A. Osheev, "Two-way mapping of graph and relational data models for multi-model databases," CEUR Workshop Proc. **2790**, 369–386 (2020). http://ceur-ws.org/Vol-2790/paper33.pdf
8. Ye. Unal and H. Oguztuzun, "Migration of data from relational database to graph database," in *Proc. 8th Int. Conf. on Information Systems and Technologies, Istanbul, 2018* (Association for Computing Machinery, New York, 2018), p. 6. https://doi.org/10.1145/3200842.3200852
9. R. De Virgilio, A. Maccioni, and R. Torlone, "Converting relational to graph databases," in *First Int. Workshop on Graph Data Management Experiences and Systems, New York, 2013* (Association for Computing Machinery, New York, 2013), p. 1. https://doi.org/10.1145/2484425.2484426

*Translated by Yu. Kornienko*

**Abuli Mikhailovich Palagashvili.** Born November 20, 1998. Graduated from the Moscow State University, Faculty of Computational Mathematics and Cybernetics, in 2020. Graduated from the Yandex School of Data Analysis in 2021. Scientific interests: data models, machine learning, recommender systems, and machine-learned ranking.

**Sergey Alexandrovich Stupnikov.** Born May 4, 1978. Graduated from the Moscow State University, Faculty of Mechanics and Mathematics, in 2000. Received Candidate's degree in theoretical foundations of informatics at the Institute of Informatics Problems of the Russian Academy of Sciences in 2006. Since 2000, has been working at the Russian Academy of Sciences. Head of Department and Leading Researcher at the Federal Research Center "Computer Science and Control", Russian Academy of Sciences, Moscow, Russian Federation. Scientific interests: integration of heterogeneous data, formal semantics, mapping, data model transformation, formal verification of data integration, research infrastructure support, and problems at the interface of computer science and fields with intensive use of data. Author and coauthor of more than 100 papers in peer-reviewed journals and conference proceedings indexed in Web of Science, Scopus, and RSCI. Scopus: 56 publications, 139 citations in 82 papers, *h*-index 7. RSCI: 97 publications, 480 citations, *h*-index 8. Participated in 40 international and Russian scientific conferences. Since 2018, Deputy Head of the Coordinating Committee and Co-Chair of the Program Committee of the Conference "Data Analytics and Management in Data-Intensive Domains" (DAMDID/RCDL), Member of the Steering Committee of the ADBIS Conference, and Member of the Program Committees of the Conferences ADBIS, DATA, I-ESA, SEIM, and APSSE. Since 2015, Moderator of the scientific seminar of the Moscow ACM SIGMOD Chapter; since 2018, Head of the Chapter. Since 2009, employee at the Moscow State University, delivering lectures on object-oriented databases, as well as virtual and materialized data integration, in the framework of master's degree programs at the Faculty of Computational Mathematics and Cybernetics. Since 2018, Deputy Head of the Program "Big Data: Infrastructures and Methods of Problems Solving." Moderator of a special seminar of the Program. Supervises master's and candidate-of-science dissertations at the Moscow State University and Federal Research Center Computer Science and Control, Russian Academy of Sciences.