21st International Research/Expert Conference
"Trends in the Development of Machinery and Associated Technology"
TMT 2018, Karlovy Vary, Czech Republic, 18th – 22nd September, 2018

MAPPING BETWEEN RELATIONAL DATABASE MANAGEMENT SYSTEMS AND GRAPH DATABASE FOR PUBLIC TRANSPORTATION NETWORK

Faruk Serin
Munzur University, Department of Computer Engineering,
62000, Tunceli, Turkey

Suleyman Mete, Muhammet Gul, Erkan Celik
Munzur University, Department of Industrial Engineering,
62000 Tunceli, Turkey

ABSTRACT
Analysis, model, optimization and simulation (AMOS) of public transportation network (PTN) is critical for urban transportation services (UTS). When PTN is analyzed, it is seen stations and routes are prominent factors since they have important role in almost all stage of UTS. The graph theoretical approaches have been commonly used to model PTN. In the basic graph G(V, E, R) of PTN, the station declared as vertex (V) and directed edges (E) are generated between stations. The routes (R) of the graph include the set of edges. However, PTN data is generally stored and managed in relational database management systems (RDBMS). Thus, the graph theoretical AMOS of PTN in RDBMS requires extra effort such as converting relationships to edges. Thanks to advancement in technology, graph databases management system (GDBMS) such as Neo4j make AMOS of PTN easy and efficient. Thus, the usage of graph compute engines has been increasing in order to manage PTN dynamically. In this circumstance, the requirement of mapping between RDBMS and GDBMS arises. This study addresses the mapping between PostgreSQL RDBMS and Neo4j GDBMS.
Keywords: Public transportation, graph theory, graph database, PostgreSQL, Neo4j

1. INTRODUCTION

Transportation is one of the most significant issue for aspect of city planners. It has a direct impact on the all aspect of the community such as education, economy, health and entertainment activities, and these cannot be maintained without an effective city infrastructure of transportation. For this reason, using of public transportation is gaining more and more importance day by day with growing population [1]. There are a lot of disadvantage during use private car such as noise and air pollution, stress and traffic problems, excessive and unreliable travel times. Therefore, people are generally tending use public transportation due to disadvantage of special car. In this case, passengers need to plan their trip in order to save money and time. One of urban transportation services providers' tools is database management systems (DBMS) to meet these numerous and dynamic needs of passengers. DBMS affects the performance of the other tools. The databases play an important role to unified information from sources like maps, vehicles, stops timetables, etc. By this way, information can be storing into data warehouse or available to end users for analyzing frequently used routes, bottlenecks and discovering. This information is generally used by an ordinary user on a mobile device application [2]. Thus, the choose of DBMS is important.

The prominent factors of public transportation network (PTN) is stations and routes since they have important role in almost all stage of urban transportation services (UTS). The graph theoretical approaches have been commonly used to model PTN. In the basic graph G(V, E, R) of PTN, the station declared as vertex (V) and directed edges (E) are generated between stations. The routes (R) of the graph include the set of edges. However, extra efforts such as converting relationships to edges is required to apply these algorithms if RDBMS is used to manage PTN data. Thanks to advancement in

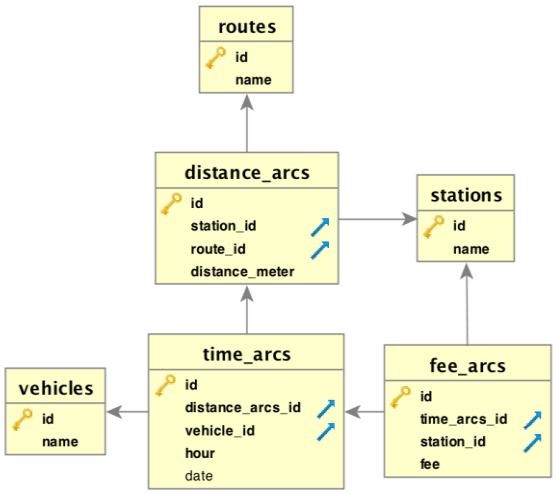
technology, graph databases management system (GDBMS) such as Neo4j facilitates analysis, model, optimization and simulation (AMOS) of PTN.
This study addresses the mapping between PostgreSQL RDBMS and Neo4j GDBMS. The paper presents the graph database structure to store and process data and its implementation for efficient trip planning in PTN. One of the most important elements of modern intelligent transportation systems constitute the advanced databases. The rest of the paper is given as follows: material and method are detailed in Section 2. Conclusion is provided in Section 3.

2. MATERIAL AND METHOD

Nowadays, storing, retrieving, transmitting, and manipulating data is very important in almost every area. For this, database management systems such as RDBMS and GDBMS are used. Each database management system has advantages and disadvantages depending on the purpose of usage. The purpose of this work is to address the databases facilitating the management and processing of data required for public transportation. Since the appropriate databases for this purpose are RDBMS and GDBMS, PostgreSQL [3] and Neo4j [4] are considered for RDBMS and GDBMS respectively. RDBMS is a widely used database management system. Data is stored as tables with rows and columns in RDBMS and processed using structured query language (SQL). The main processes of RDBMS are queries of create, select (read), update and delete (CRUD). Data is stored as graph structures in GDBMS and provide more efficiency for graph algorithms. In this study, Neo4j is selected as GDBMS. Neo4j as a graph platform uses Cypher [5], Neo4j's query language, to process data.

2.1 Mapping Between RDBMS and GDBMS

In RDBMS, first, the tables are created, then each row (object) is inserted. In GDBMS, each object of nodes or edges is created as stated in section 2.2. The mapping between RDBMS and GDBMS can be performed by using equivalent CRUD operations. The export statement (COPY … TO …) in PostgreSQL and import statement (LOAD CSV …) in Neo4j can also be used in order to transfer existing data from PostgreSQL to Neo4j.



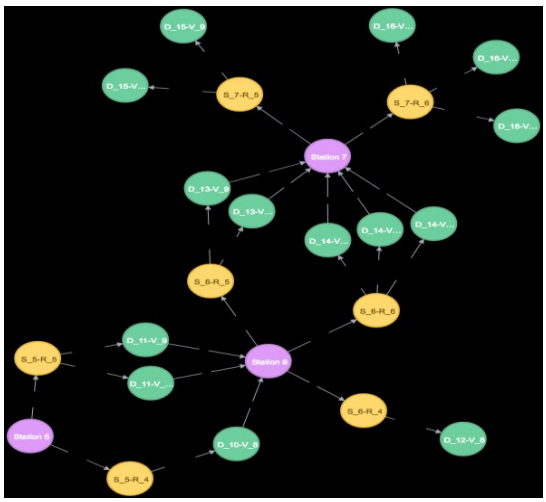


Figure 1. The basic ER diagram of RDBMS for PTN Figure 2. The example Neo4j graph of PTN

The basic ER diagram of PostgreSQL for PTN database is illustrated in Figure 1. An example Neo4j graph of PTN is shown in Figure 2. The CRUD for RDBMS and its equivalents for Neo4j are given as follows. SQL statement CREATE TABLE is used once for each entity. SQL statement INSERT INTO and its equivalents Cypher statement CREATE is run for each object, but, only an example is given here. **SQL statement-X** is equivalent to **Cypher statement-X**.

SQL statement-1:

CREATE TABLE public.stations (id bigint, name character varying(50), CONSTRAINT stations_pkey PRIMARY KEY (id));

INSERT INTO public.stations(id, name) VALUES (1, 'Station 1');
INSERT INTO public.stations(id, name) VALUES (2, 'Station 2');

Cypher statement-1:

CREATE (:Stations {id: 1, name: 'Station 1'});
CREATE (:Stations {id: 2, name: 'Station 2'});

SQL statement-2:

CREATE TABLE public.routes(id bigint,name character varying, CONSTRAINT routes_pkey PRIMARY KEY (id));
INSERT INTO public.routes (id, name) VALUES (1, 'Route 1');

Cypher statement-2:

CREATE (:Routes {id: 1, name: 'Route 1'});

SQL statement-3:

CREATE TABLE public.vehicles(id bigint,name character varying(50),CONSTRAINT vehicles_pkey PRIMARY KEY (id));
INSERT INTO public.vehicles(id, name) VALUES (1, 'Vehicle 1');

Cypher statement-3:

CREATE (:Vehicles {id: 1, name: 'Vehicle 1'});

SQL statement-4:

CREATE TABLE public.distance_arcs (id bigint,station_id bigint,route_id bigint,distance_meter double precision,CONSTRAINT distance_arcs_pkey PRIMARY KEY (id),CONSTRAINT route_id_fk FOREIGN KEY (route_id)REFERENCES public.routes (id) MATCH SIMPLE, CONSTRAINT station_id_fk FOREIGN KEY (station_id) REFERENCES public.stations (id) MATCH SIMPLE);
INSERT INTO public.distance_arcs(id,station_id,route_id,distance_meter) VALUES (1,1,1,200);

Cypher statement-4:

CREATE (:Distance_arcs {id: 1,name:'S_1-R_1', station_id: 1,route_id: 1});
MATCH (s:Stations),(d:Distance_arcs) WHERE s.id=d.station_id and d.id=1 CREATE (s)-[:Distance_edge{distance_meter:200}]->(d);

SQL statement-5:

CREATE TABLE public.time_arcs(id bigint,distance_arcs_id bigint,vehicle_id bigint,hour time without time zone,date date, CONSTRAINT time_arcs_pkey PRIMARY KEY (id), CONSTRAINT distance_arc_id_fk FOREIGN KEY (distance_arcs_id) REFERENCES public.distance_arcs (id) MATCH SIMPLE, CONSTRAINT vehicle_id_fk FOREIGN KEY (vehicle_id) REFERENCES public.vehicles (id) MATCH SIMPLE);
INSERT INTO public.time_arcs(id, distance_arcs_id, vehicle_id, hour, date) VALUES (1,1,1,'10:00','10/03/2018');

Cypher statement-5:

CREATE (:Time_arcs{id:1, name:'D_1-V_1', distance_arcs_id:1, vehicle_id:1});
MATCH (d:Distance_arcs),(t:Time_arcs) WHERE d.id=t.distance_arcs_id and t.id=1 CREATE (d)-[:Time_edge{hour:'10:00', date:'10/03/2018'}]->(t);

SQL statement-6:

CREATE TABLE public.fee_arcs(id bigint,time_arcs_id bigint, station_id bigint, fee double precision,CONSTRAINT fee_arcs_pkey PRIMARY KEY (id),CONSTRAINT station_id_fk FOREIGN KEY (station_id) REFERENCES public.stations (id) MATCH SIMPLE,CONSTRAINT vehicle_arcs_id_fk FOREIGN KEY (time_arcs_id)REFERENCES public.time_arcs (id) MATCH SIMPLE);
INSERT INTO public.fee_arcs(id, time_arcs_id, station_id, fee) VALUES (1,1,2,3);

Cypher statement-6:

CREATE (:Fee_arcs{id:1, name:'T_1-S_2',time_arcs_id:1, station_id:2});
MATCH (t:Time_arcs),(f:Fee_arcs),(s:Stations) WHERE t.id=f.time_arcs_id and f.station_id=s.id and f.id=1 CREATE (t)-[:Fee_edge{fee:3}]->(s);

SQL statement-7: SELECT * FROM public.stations where id='1';
Cypher statement-7: MATCH (s:Stations) where s.id=1 return s
SQL statement-8: UPDATE public.stations SET name='Station 111' WHERE id=1;
Cypher statement-8: MATCH (n:Stations { id: 1 }) SET n.name = 'Station 111' RETURN n.name;

SQL statement-9: DELETE FROM public.stations WHERE id=2;
Cypher statement-9: MATCH (n:Stations{id:2}) DELETE (n);

PostgreSQL and Neo4j store data in different formats as explained in the previous sections. When the Cypher statements 1-9 given in Section 2 are run, a graph as in Figure 3 are generated and the data is stored as graph structure. The path from Station 5 to Station 7 is calculated using **Cypher statement-10**, and the result of query is shown in Figure 4.

Cypher statement-10: MATCH (from_s:Stations { name: 'Station 5' }),(to_s:Stations { name: 'Station 7' }), p = shortestPath((from_s)-[*]->(to_s)) RETURN p

Figure 3. A simple graph of PTN in Neo4j

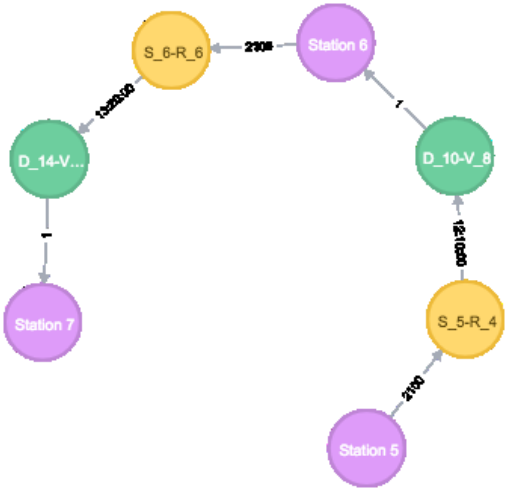


Figure 4. A path from Station 5 to Station 7 in Neo4j.

3. CONCLUSION

Public transport networks data is commonly stored in RDBMS, although the networks have graph structure. In this case, it is necessary to transform the relational structures into graph structures in order to apply graph methods efficiently. However, this brings extra workload, especially in a dynamic system. The data can be stored in GDBMS to get rid of this transformation and the performance is improved by directly running the graph algorithms. In this study, it is explained how basic information for public transportation can be managed in Neo4j GDBMS and how data in PostgreSQL RDBMS can be transferred to Neo4j.

4. REFERENCES

[1] Rakocevic, S., Dragasevic, Z., & Montenegro, P. (2011). Parametric programming for the transportation problem. In The 15th International Research/Expert Conference "Trends in the Development of Machinery and Associated Technology" TMT 2011 (pp. 925-928).

[2] Czerepicki, A. (2016). Application of graph databases for transport purposes. Bulletin of the Polish Academy of Sciences Technical Sciences, 64(3), 457-466.

[3] https://www.postgresql.org/files/documentation/pdf/10/postgresql-10-A4.pdf (May 2018)

[4] https://neo4j.com/docs/ (May 2018)

[5] https://neo4j.com/developer/cypher-query-language/ (May 2018)