POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Building reactive processing rules for knowledge graphs

Laurea Magistrale in Computer Science and Engineering

Author: Alessia Gagliardi

Advisor: Prof. Stefano Ceri

Co-advisor: Dr. Anna Bernasconi

Academic year: 2021-2022

1. Introduction

In this thesis, we present a proposal for incorporating reactive rules into property graphs. Our approach involves a feasibility study on the spread of the COVID-19 virus which allowed the implementation of Reactive Knowledge Rules in Neo4j and its querying language, Cypher. Our study builds on the existing work in the field of active databases and draws on the established use of database triggers. We also consider how knowledge is currently managed and how concepts such as old and new state, which are integral to database triggers, can be applied to reactive knowledge rules. The purpose of this executive summary is to provide an overview of the key findings of our research on these topics.

2. State of Art of Active Databases and Knowledge Graphs

The interest in active databases began in the 1970s and 1980s, and eventually, in the 1990s, active databases in traditional database systems were developed and finalized [3]. Active databases automatically respond to events that occur in the database if certain conditions are satisfied. These mechanisms are called rules or **triggers**. Within the SQL3 standard, triggers typically consist of Event-Action-Condition (ECA) rules. The ECA paradigm states that *"whenever an event e occurs, if a condition c is true, then an action a is executed"*

However, with the rise of the internet as a tool for the general public, data began to increase both in volume and interconnectedness, and SQL-based relational databases were no longer considered the best choice.

The NoSQL ("Not only SQL") space brings together many interesting solutions offering different data models and database systems. In this study, we will focus on graph technologies, a topic that has gained recognition from the IT community but has yet to garner large-scale academic study. Graph database models took off in the 1980s and early 1990s.

The rise in popularity of NoSQL's graph databases can be attributed to three main properties of these types of data models: performance, flexibility, and agility. The large participation and attention on graphs databases led also to the start of the ongoing ISO standardization effort, aiming at defining a new standard Graph Query Language (GQL). A few article aimed at incentives the development of GQL have been drafted. Specifically, the articles on `PG-Keys`[1] and `PG-Schema`[2] provide a formal-

ization of two important concepts for property graphs.

3. Knowledge Graph

Knowledge Graphs are flexible, reusable data layers used for answering complex queries across different data domains by creating connections between contextualized data, represented and organized in the form of graphs. They are built with the aim to represent the fluctuating nature of knowledge.
A Knowledge Graph model comprises two elements: nodes and edges. Nodes can be resources with unique identifiers, or they can be values with literal strings, integers, or whatever. The edges (also called predicates or properties) are the directed links between nodes. The "from node" of an edge is called the subject. The "to node" is called the object. Thus, a Knowledge Graph is a directed graph of triple statements.

4. Reactive Knowledge Rules

The main contribution of this thesis is to extend the theory and concepts of active databases in order to build reactive knowledge bases, which respond to the need of reacting to knowledge changes.
With the same approach used for Active Databases, we define a Reactive Knowledge Rule as *triple* (Event, Condition, Action). The event captures the data modifications e.g., the creation and deletion of nodes or relationships. For the Condition and Action parts we introduce two new components: Guards and Alerts.

- The `Guard` is an existential predicate that reveals situations that can be considered interesting and deserve further investigation
- The `Alert` is a program that further analyzes the situation in order to understand if it is critical. If this happens, the Alert produces a new node, labeled Alert, which includes all the information that is necessary to manage the knowledge change. If, instead, the situation is not considered critical, then the Alert has no side effects.

As reactive rules are designed to respond to changes in the graph data structure, each of them typically reference a particular knowledge hub.**Knowledge Hubs** are areas of interest united by the knowledge they share forming an interaction knowledge graph.
We also classify different types of reactive rule through some orthogonal properties. Firstly we distinguish `intra-hub` and `inter-hub` rules which differ in the scope. The former's scope is confined within a single knowledge hub. While the scope of the latter spans over multiple hubs. However it is also crucial for reactive rules to have a temporal reference about the knowledge. Reactive processes are actually equipped with variables denoted by keywords OLD and NEW however those variable are not available in the context of graph database. To distinguish these types of reactive rules we introduced other two properties: a rule whose Alert part can be expressed on the current state takes the name of `single-state`, while a rule whose Alert part requires comparing several states of the knowledge base can be defined as `multi-state`.

5. The COVID-19 Example

Since the outbreak of the COVID-19 virus, the evolution of the virus has been constantly monitored. Given the many milestones reached in the research of property graphs, it is evident that modeling the COVID-19 knowledge as a graph could potentially have an important impact to get a deeper insight into new outbreaks of the virus or simply to monitor the spreading of the mutations. The example that we present in our study, allowed us to better understand what are the current challenges in building active processing rules in these types of scenarios.
The example proposed in the thesis monitors of the spreading of the COVID-19 virus over a geographical region. As shown in Figure 1 we consider nine types of nodes: the collected **Sequences** of the virus, the **Variants** classified by the scientific community, the **Laboratories** in which the virus is being analyzed, the **Mutations** that are found in the sequences, the eventual **Critical Effect** a new mutation can bring, the **Patients** affected by the virus and admitted to **Hospitals** including the eventuality in which they are cured in **Intensive Care Unit**. Finally, we have some **Region** nodes which confine a geographical region with its Labs and Hospitals.
To better understand the distribution of those nodes we can map a partition of the domain of applications as we envision four knowledge hubs. The **Experimental Hub** that studies

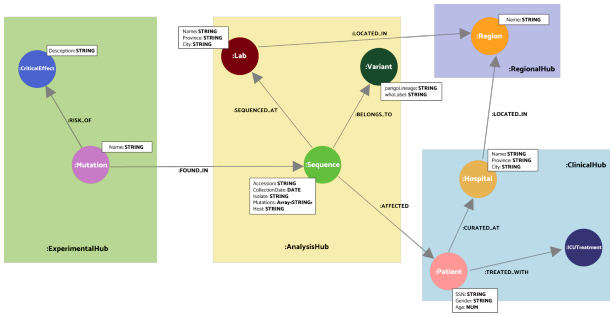

Figure 1: Representation of the graph nodes, relationships and knowledge hubs.

mutational effects. **Analysis hub** that associates sequences produced at a given location with known variants. **Clinical hub** at a given hospital. And **Regional hub** at a given region, possibly hosting many hospitals.

5.1. Reactive Knowledge Rules example

As reactive rules are designed to respond to changes in the graph data structure, each of them typically reference a particular knowledge hub. Each reactive rule refers to the knowledge hub that contains nodes sharing the same label as the "new-node" referenced in the rule.
In our example we propose four reactive rule distributed as follows: one for the Experimental hub, two for the Analysis Hub and one for the Clinical Hub. While proposing those reactive rules we tried to create rules of different types as Table 1 shows.

Reactive rule on **Experimental Hub**	Single-state	Intra-hub
Reactive rules on **Analysis Hub**	Single-state	Inter-hub
Reactive rule on **Clinical Hub**	Multi-states	Inter-hub

Table 1: Classification of Reactive Knowledge Rules proposed in the COVID-19 example.

6. Periods of observation

Before seeing the actual implementation of the example rules in Neo4j we are going to discuss about periods of observation. Focusing on the *multi-state* reactive rules, it is clear that we need to be able to distinguish different state of the knowledge. But since graph databases do not exist a concept of OLD state of the graph we need to think of an out-of-the-box solution. In our study we propose two models: the **Total Replication model** and the **Essential Summary model**.
The Total Replication approach periodically replicates the graph. To differentiate the various graphs we need a reference to the corresponding period to which they are linked. Thus, when a copy of the graph is created, each node of that replication will be connected to a special node that has a reference to the date and time of the creation of the replica. What we are going to have is a graph that we are going to call *"Current"* and other *"Version"* graphs referred to each replica. By following this procedure we can ensure a comprehensive representation of the graph, encompassing the nodes and their relationships. By querying every aspect of the graph, users can obtain a comprehensive view of the data and identify patterns and trends.

In the Essential Summary model we instead keep track of the events and not of the entire graph. The Essential Summary corresponds to a specific data structure created at each execution of the periodic query. Since this approach does not store the entire version of the graph at a specific instant, nodes of the Essential Summary type may need to store additional properties. For instance it may be useful to store a specif value needed for a future comparison with that day.

While a Total Replication allows to obtain a comprehensive view of the data, it presents some downsides. There are drawbacks in the storage due to the large space required to maintain all the replicas. But another issue that this approach can have is towards performance since maintaining consistency across multiple replicas can be challenging and may lead to an increased latency. Therefore it may be beneficial to follow the approach of the Essential Summary model.

7. Neo4j and Cypher

In order to deploy our use case into a concrete example we decided to use the Neo4j, a graph

database management system. Implemented in Java, Neo4j can be accessed from software written in other languages using the Cypher query language via a transactional HTTP endpoint or the binary "Bolt" protocol.

Neo4j belongs to the so called *Labeled Property Graph Models*. A labeled property graph is composed by nodes that are the entities in the graph, relationships that connect two node entities, properties that are key-value pairs containing relevant information about a node and labels which distinguish their different roles and groups entities together.

Its querying language, Cypher, a new expressive (yet compact) graph database query language to create, manipulate, and query data. Cypher is designed to be easily read and understood by developers, database professionals, and business stakeholders; it enables users to ask the database to find data that matches a specific pattern. However, as we will see later, Cypher has limited functionalities. In order to fulfil this need Neo4j and Cypher have opted for the possibility to extend their functions through User Defined Procedures and Functions. Some libraries provide some functionalities to further extend Cypher. In our study we will make use of the APOC library which has become a standard in Neo4j thanks to how well-supported and easy it is to run it through separate functions or to include it in Cypher queries. Two relevant APOC's procedures used in this thesis are `apoc.do` and `apoc.trigger` which we are going to present in the following section.

8. Proof-of-Concept

In this section we are going to discuss procedures and functionalities developed in our study to demonstrate the feasibility of our proposal.

In order to import the data into the Neo4j database server we created some CSV file. This files has a specific structure where each column will be interpreted in our Java application to populate the graph database. For instance, the first column specifies the type of operation to perform (e.g., creation or deletion of nodes). The second column will instead specify the label, and therefore the type, of node on which the operation will be performed. The other columns will instead contain the values of the properties of the node itself or other reference to the node it has a relationship with.

The CSV file will be read by a Java application that we called `NodeGenerator.java` which connects to the Neo4j server through a Bolt protocol. This application takes care of the queries that are going to create or delete the node from the database. For each row read from the CSV file, it interprets the values of each column and fills correctly the query to performed.

8.1. Reactive Knowledge Rules in Neo4j

In this thesis we replicated the Reactive Knowledge Rules in Neo4j through the triggering procedures offered by the APOC library. The `apoc.trigger` procedure collection allows the registration of Cypher statements that are going to be run in the database on the happening of a relevant event. APOC triggers do not distinguish the type of event occurred. In order to capture the correct type of event, these procedures provide a set of parameters to select. The transaction data from Neo4j is turned into appropriate data structures to be consumed as parameters to a statement.

Focusing on the Guard and Alert components we could consider them as a simple `if-then-else structure`. If the Guard responds with TRUE, we move into the Action part, which communicates a significant situation if the target nodes meet certain criteria. Otherwise, a response of the Guard with FALSE indicates that the event did not cause a critical situation and therefore we can move on. This conditional part of the reactive rule is replicated with a conditional query using the `apoc.do.when` procedure.

The actual implementation of Guard and Alert of our use case was done by creating custom functions, which can be crafted for expressing complex Guards and Alerts. These user-defined functions are then called into the apoc.do.when procedure inside the trigger. The result is the definition of a number of functions

dedicated to Guards and Alerts, all of which can be directly called in Neo4j as an extension of Cypher.

9. Conclusions

The aim of this thesis was to model reactive rules providing a way to trigger actions based on changes to the graph database. We define a reactive rule as a triple (Event, Condition, Action) represents an effective solution to monitor changes in graph databases, allowing us to model expressive conditions of arbitrary complexity to best fit the use cases at hand considering all the limitation posed by the querying language used. In conclusion, the modeled triggering system has enabled us to create a monitoring mechanism that is both expressive and powerful. This mechanism serves as a starting foundation for future research, as there are still many aspects that needs further investigation.

References

[1] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Keith W Hare, Jan Hidders, Victor E Lee, Bei Li, Leonid Libkin, Wim Martens, et al. Pg-keys: Keys for property graphs. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2423–2436, 2021.

[2] Angela Bonifati, Stefania Dumbrava, George Fletcher, Jan Hidders, Bei Li, Leonid Libkin, Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savković, et al. Pg-schema: Schemas for property graphs. *arXiv preprint arXiv:2211.10962*, 2022.

[3] Jennifer Widom and Stefano Ceri. *Active database systems: Triggers and rules for advanced database processing*. Morgan Kaufmann, 1995.