IMPLEMENTING CHECK INTEGRITY CONSTRAINT IN GRAPH DATABASES

[1]KORNELIJE RABUZIN, [2]MARIO KONECKI, [3]MARTINA SESTAK

[1,2,3]Faculty of Organization and Informatics, Varaždin, Croatia
E-mail: [1]krabuzin@foi.hr, [2]mario.konecki@foi.hr, [3]msestak2@foi.hr

Abstract— The issue of maintaining a database in a consistent state is becoming more challenging in situations where users are able to store differentkinds of data in the database without any control. Leaving this process unattended can lead to invalid and duplicate data in the database, which can directly affect the correctness of gathered data. In this paper, an overview of integrity constraints is given with the emphasis on graph databases, as well as the level of support provided by Neo4j, the most popular graph database management system (DBMS).A new integrity constraint, which prevents users from entering data with values outside the permitted predefined set of values, is also presented and elaborated.

Keywords— Integrity constraints, CHECK, Gremlin, Neo4j, Graph databases.

I. INTRODUCTION

Integrity constraints represent a powerful mechanism for maintaining database integrity and consistency. By defining integrity constraints one can prevent situations where low quality data is entered in the database, which are known under the term Garbage In Garbage Out (GIGO).

It is a common situation that, once the integrity constraints are defined, they are not satisfied all the time. Instead, the DBMS ensures that the constraints are satisfied at the end of each database transaction, thus keeping the database in a consistent internal state.

Integrity constraints can be defined as rules, which define the structural and behavioral properties of each object stored in the database[1].

The topic of integrity constraints is elaborated in detail in relational databases, where the integrity constraints can be defined in various levels[2]:

- Column constraints - constraints like PRIMARY KEY, UNIQUE, CHECK, REFERENCES, which are defined for each table column specifically
- Table constraints - constraints like PRIMARY KEY, FOREIGN KEY, which are defined for multiple table columns
- Database constraints - assertions, which are defined for multiple tables or the entire database

According to [3], it is also possible to define domain constraints, which specify the range of values permitted to be entered in a table row for a given table column. To define more complex integrity constraints, one can also use triggers and stored procedures, but their usage is effective only in specific situations, since their evaluation can become very costly.

Integrity constraints can also be classified as static or dynamic [4]. Static integrity constraints define object property values, which must be satisfied in each database state. On the other hand, dynamic integrity constraints represent rules, which define how the database state can change over time, i.e., how object property values can change in different database states.

Recently, NoSQL databases have gained popularity as a good alternative to standard relational databases. They can be classified into four groups of database solutions:

1. Key-value databases
2. Document databases
3. Column-oriented databases
4. Graph databases

In this paper, a more detailed overview of graph databasesis given. The rest of the paper is organized as follows: in section 2, theintegrity constraints in the context of graph databases, and the level of support for defining integrity constraints in the two most commonly used graph query languages - Cypher and Gremlin are discussed. In section 3, the details regarding the implementation of a new CHECK integrity constraint in Neo4j database by using a layered approach are presented and explained. Finally, the paperis concluded by summarizing itscontent.

II. INTEGRITY CONSTRAINTS IN GRAPH DATABASES

As has been already mentioned, graph databases are a category of NoSQL databases based on the graph data model. Benefits of using graph databases become prominent in the context of modeling complex relationships of connected entities in the real world, where the relationship properties cannot be overlooked[5]. In these situations, queries executed in graph databases achieve better performances regarding time, cost and complexity, than the same queries executed in relational databases.

In graph databases, data is stored in nodes (vertices) and relationships (edges). Both nodes and relationships can have properties and labels. Since

graph databases are a relatively new category of NoSQL database solutions, there is still little work done regarding maintaining database consistency.

When it comes to the topic of integrity constraints in graph databases, several papers can be found that describe the topic in general[6][7][8].However, the support for integrity constraints in graph databases is rather low.

According to [9], currently the most commonly used graph DBMS is Neo4j. Besides Neo4j, there are other graph DBMS products in the database market. In [10], anoverview of the level of integrity constraints support in the most popular graph DBMSs has been given. It has been shown that, for instance, Titan and OrientDB support a number of integrity constraints, as opposed to Neo4j, AllegroGraph, or InfiniteGraph DBMSs.

The most popular languages that are used for graph databases are Cypher and Gremlin.

Cypher is a query language for graph databases, and it is known as the official query language for the Neo4j DBMS. It is not too complicated since many clauses in the statements look similar or have the similar meaning to the clauses used in SQL (some of them even share the same name). For instance, CREATE statement is used to create nodes and relationships, LIMIT clause can be used to limit the number of rows that the query returns, ORDER BY clause is used to sort the results, etc.

However, the support for integrity constraints definition is minimal. At this point of time, one can specify the unique property constraint. However, that constraint can only be defined for nodes, but not relationships. The purpose of this constraint is to ensure that nodes with the same label have unique property values, and it can be defined by using the CREATE CONSTRAINT clause[11]. For instance, to specify that the title of a book must be unique, one would execute the following statement:

CREATE CONSTRAINT ON (b:Book) ASSERT b.Title IS UNIQUE

After defining this constraint, if the user tries to insert a new book with the title that already exists in the database, then the user will receive an error message about constraint violation.

Gremlin is another language used for querying graph databases by performing graph traversals[12]. When compared to Cypher, Gremlin is more complex to understand, as the user is required to specify the entire query traversal path.

Regarding the integrity constraint support, at this point of time it is not possible to define any integrity constraints by executing Gremlin queries.

In some of previous research papers it has been elaborated how integrity constraints in graph databases could be implemented (technical implementation details) [2], and it has been explored how to implement the UNIQUE constraint as well [10].

In this paper, it has beendemonstrated how CHECK constraint could be implemented and used in graph databases.

III. IMPLEMENTING CHECK INTEGRITY CONSTRAINT IN NEO4J

In this section, it isdescribed how CHECK integrity constraint has been implemented in Neo4j graph DBMS. Neo4j has been selected because it is one of the most popular systems at this point of time, but with few modifications the results could be used with other graph database management systems as well.

In relational databases, the CHECK integrity constraint is a column constraint, which means it can be specified only for a given table column. It is often used to test whether the table row value for a given column is NULL, whether the given value is in range (BETWEEN), or in a set (IN) of values allowed for that table column.

In this paper, a newimplemented integrity constraint, which checks whether a given node property value is in the allowed range or set of values, is presented and elaborated.

Similar to relational databases where the CHECK constraint is specified for a column, in the context of graph databases the CHECK constraint was implemented to test whether the value of a given node property is in the range or in the set of values allowed for that property.

In order to implement CHECK constraint, a web application has been built by using Spark, a Java web framework, and Apache Tinkerpop, a graph computing framework for graph databases, which contains classes and methods for executing Gremlin queries.

The CHECK integrity constraint is defined as an additional graph DBMS layer, which behaves as a mediator between the graph database and the web application.

The user first needs to define a new constraint, which includes defining the constraint name, selecting the node type, its property, the check type, i.e., whether the property value must be in a range or in a set of values, and setting the range/set of allowed values. For instance, a constraint called “AuthorGenderConstraint” can be defined, whichpurpose is to ensure that the value entered for property “Gender” of node labeled “Author” is either“male” or “female” (Fig. 1).

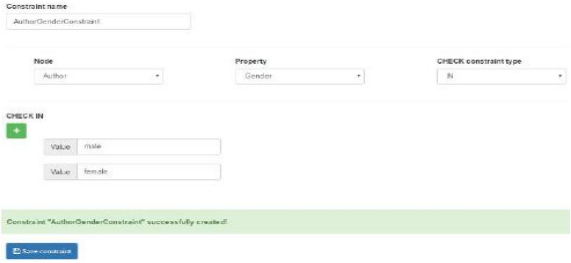


Fig.1. CHECK BETWEEN constraint definition

Once the constraint is defined, it is stored in a list of database constraints. When the user wants to create a new node, the first step of the method responsible for creating nodes is to iterate over the list of all constraints, and check if there is a constraint defined for the node with that label.

In this case, if the user wants to create a new node labeled "Author", the referenced method responsible for creating "Author" node will find the "AuthorGenderConstraint", which has been previously defined. After that, the list of allowed values for the "Gender" property is fetched, and compared to the value entered by the user (Fig. 2).

```
for (Field f:
        Author.class.getDeclaredFields() ) {
    if(f.getName().equalsIgnoreCase(property)){
        f.setAccessible(true);
        try {
            String fieldValue = f.get(nodeParams).toString();

            for (Object o:
                 values) {
                if(fieldValue.equalsIgnoreCase(o.toString())){
                    constraintresultIN = true;
                }
            }
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
    }
}
```

Fig. 2. Checking if the entered property value is allowed

If the entered value is an element of that list, the *constraintresultIN* variable is set to *true*, and the new node labeled "Author" is successfully created in the database by calling the *createNode()* API method, as shown in Fig. 3.

```
try(Transaction tx = db.beginTx()){
    node = db.createNode(Types.Label.Author);
    node.setProperty("Firstname",nodeParams.getFirstname());
    node.setProperty("Lastname", nodeParams.getLastname());
    node.setProperty("Gender", nodeParams.getGender());

    tx.success();
}
```

Fig. 3. Creating new "Author" node in the database

The result of the executed Cypher query is a new "Author" node in the database (Fig. 4).



Conversely, if onewants to define the range of values for a given node property, one needs to use BETWEEN check type. For instance, to create a constraint called "BookYearConstraint", which will ensure that the user can enter books published between 1990 and 2015, one needs to set this range of values as a parameter of the constraint, as shown in Fig. 5.

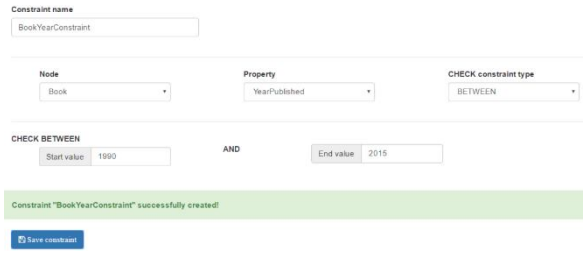


Fig.1. CHECK BETWEEN constraint definition

In this case, the method responsible for creating "Book" node will find the "BookYearConstraint" in the constraints list, and check if the entered value for "yearPublished" property is in the range of values defined by that constraint (Fig. 6).

```
for (Field f:
    Book.class.getDeclaredFields() ) {
    if(f.getName().equalsIgnoreCase(property)){
        f.setAccessible(true);
        try {
            int fieldValue = Integer.valueOf(f.get(nodeParams).toString());

            if(fieldValue >= Integer.valueOf(values.get(0).toString())
                    && fieldValue <= Integer.valueOf(values.get(1).toString())){
                constraintResultBetween = true;
            }
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
    }
}
```

If the entered value is in the range of allowed values, the *constraintresultBETWEEN* variable will be set to *true*, and a new "Book" node will be created in the database. However, if the value for the "YearPublished" property is not within the specified range, the node will not be created, and an error message shown in Fig. 7 will be displayed to the user.

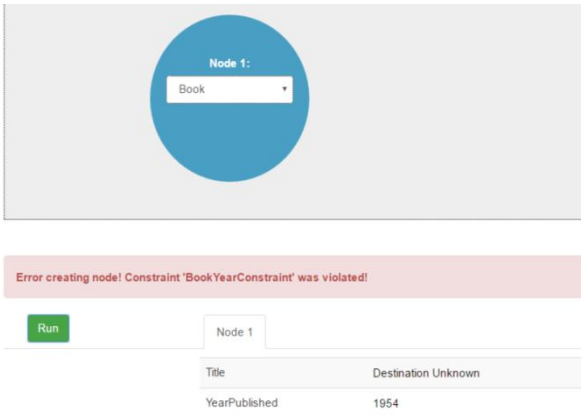


Fig. 7. Error message for constraint violation

CONCLUSIONS

In this paper, integrity constraints in databases with special emphasis on graph databases have been discussed. As elaborated above, the support for integrity constraints in graph databases is still minimal as current graph query languages do not have statements and/or clauses for constraint

implementation. However, integrity constraints can preventstoring of inconsistent data within the database. Hence, the integrity constraints are very important. In this paper, it has been shownhow CHECK integrity constraint can be implemented in graph databases. As a part of future researchit will be explored how other types of integrity constraints can be implemented in graph databases as well.

REFERENCES

[1] L. C. Rivero, J. H. Doorn, and V. E. Ferraggine, "Database Integrity", in Database Integrity, IGI Global, 1AD, pp. 17–65.

[2] K. Rabuzin, M. Šestak, and M. Novak, "Integrity constraints in graph databases – implementation challenges", 2016.

[3] H. Decker and D. Martinenghi, "Database Integrity Checking",Database Technol., pp. 961–966, 2009.

[4] D. Martinenghi, H. Christiansen, and H. Decker, "Integrity Checking and Maintenance in Relational and Deductive Database and Beyond", in Intelligent Databases, IGI Global, 1AD, pp. 238–285.

[5] I. Robinson, J. Webber, and E. Eifrem, Graph Databases: New Opportunities for Connected Data. " O'Reilly Media, Inc.", 2015.

[6] R. Angles, "A comparison of current graph database models", in Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on, 2012, pp. 171–177.

[7] R. Angles and C. Gutierrez, "Survey of Graph Database Models".

[8] J. H. Doorn, L. C. Rivero, and V. E. Ferraggine, "Introduction to Database Integrity", in Database Integrity, IGI Global, 1AD, pp. 1–16.

[9] "DB-Engines Ranking - popularity ranking of graph DBMS". [Online]. Available at: http://db-engines.com/en/ranking/graph+dbms. [Accessed: 16-Sep-2016].

[10] K. Rabuzin, M. Šestak, and M. Konecki, "Implementing UNIQUE integrity constraint in graph databases", 2016.

[11] Neo4j Inc., "Chapter 3. Cypher query language - The Neo4j Developer Manual v3.0.". [Online]. Available at: https://neo4j.com/docs/developer-manual/current/cypher/. [Accessed: 16-Sep-2016].

[12] Titan:db by Aurelius, "Gremlin Query Language", 2014. [Online]. Available at: http://s3.thinkaurelius.com/docs/titan/0.5.4/gremlin.html. [Accessed: 16-Sep-2016].

★★★