The study on data migration from relational database to graph database

Zhihong Nan, Xue Bai *

College of Information, ShanXi University of Finance and Economics, Tai Yuan, Shanxi Province, 030006, China

*Corresponding author’s e-mail: 1324692609@qq.com

Abstract. Under the background of big data, using relational databases to manage massive data may have some problems just like storage capacity and query efficiency. So, there is a new kind of databases called NoSQL to store data. However, the data models of NoSQL databases are different from relation databases. In order to finish migrating historical data from relational databases to NoSQL databases, in terms of graph database in the NoSQL databases, this paper takes ER diagram as the original model, graph model as target, and makes some transformational rules by using the relationships of entities. And this paper proposes an algorithm which can finish data migration by traversing ER diagram and using the transformational rules. This method can reduce the impact of model differences between relational databases and graph databases, ensure the integrity constraint of data, and automatically complete data migration. The experimental results show the validity and correctness of the data migration.

1. Introduction

With the development of computer technology and comprehensive coverage of networks, the number of data increases sharply, data sources tend to diversity, data types are mainly semi-structured and unstructured, the relationships among data are more complicated. Relational database is wide using and the most mature in technology, however, there are some disadvantages about limited storage capacity, poor scalability, low availability and so on in systems. Therefore, relational databases can’t satisfy the needs of storing and managing massive data.

To meet the needs of use, NoSQL databases were designed by researchers. By now, because of advantages of NoSQL databases that contain schemaless data model, good expansibility, high availability and more, they become the first choose in using databases. According to the data model, NoSQL databases are divided into key-value database, document-oriented database, column-oriented database and graph database. In this paper, we mainly take graph database into consideration.

For large and complex data, paper[5] compares relational database with graph database from technology maturity, security of system and other aspects, then draws a conclusion that graph database is more adaptable to store network data by analyzing data. Because relational databases were used widely, many data were stored in them. Therefore, the study of migrating historical data from relational databases to graph databases are more and more[7-11]. The paper[8] takes direct and indirect relationships among data into consideration, transforms every tuple of tables into node and connects nodes by foreign keys. Another paper focuses on improving query efficiency when moving data to graph database, and comes up with a method which gets a graph from paths of all relational data[9]. Paper[11] mainly migrates document-based, parent-child hierarchical data from relational database to graph database. About applications of graph database, some researchers design and
develop new social platforms based on graph databases or hybrid databases. In the field of analyzing data, we can use graph model to study data mining, information recommendation and otherwise.

In terms of migrating data to graph database, this paper comes up with an algorithm via traversing ER diagram.

2. Graph database
Graph database is a sort of NoSQL databases which is based on graph theory[1]. In graph database, the basic elements are nodes, edges, attributes and labels. Nodes are entities. Every attribute of node stores in form of key-value, its value can be any type. Edges represent relationships between two nodes, also have attributes and directions. Labels are types of nodes or edges to distinguish them. At present, the most popular graph model is attributed graph model[2]. Figure 1(a) shows the structure of this model. The query language of graph database is Cypher. We can use this language to perform some operations like create, query, update and delete. Compare to relational databases, the query operation in graph databases is traversal. Figure 1(b) shows the relationship of traversal and graph.

3. Model transformation
3.1. Idea of model transformation
In order to move data from relational database to graph database, it is feasible to manually convert tuples in tables to nodes in graph model. But in case of large data, this work is inefficient and may generate redundant data. Therefore, it is necessary to migrating data automatically.

ER(Entity Relation) diagram is a representation of conceptual model, an abstract of things in real world. There are three elements in ER diagram. They are entity, attribute and relationship. We can find there are some correspondences between elements of ER diagram and elements of graph model. As shown in figure 2, they are between entity and node, attribute and attribute, relationship and edge. According to this figure, we can make some rules to help finish migrating data automatically.

3.2. Design of model transformation
There are multiple relationships in ER diagram such as one-to-one, one-to-many, many-to-many. According to these relationships, we make corresponding transformational rules. The symbols used in these rules are defined as follows:

(1) In ER diagram, E means an entity type, R is relationship between an entity to another, e is one entity in an entity set. The set A={A1, A2, ..., An} is a set of attributes of the i-th entity. The set RA={RA11, RA12, ..., RAjm} is a set of attributes of the j-th relationship.

(2) In graph model, G=(N, Eg) is a graph, where N is a set of nodes in this graph and Eg is a set of edges, n is a node. Aij:valuej is a key-value pair of the j-th attribute of the i-th node.
3.2.1. Transformational rule of basic entity. Every entity type has an entity set that contains one or many entities. We suppose that the entity e belongs to an entity type E. We transform e to a node n by setting the name of E as the label of n, the attributes of e as the attributes of n. At the end, we get a node \( n \) in graph model. Figure 3 shows this transformation.

3.2.2. Transformational rules of relationships. In this part, we introduce four transformational rules.

(1) unrelated: There are some single and unrelated entities in ER diagram. We transform every entity of unrelated entity type via using the rule of basic entity. Finally, we get a graph \( G = (N, E) \) converted from \( E = (A_1, A_2, \ldots, A_n) \), \( N = \{n_1, n_2, \ldots, n_k\} \) is the set of node, as figure 4(a).

![Figure 4(a). transformation of unrelated.](image)

The transformation of same entity type also needs to standardize the entity type, transform entities to nodes, and connect nodes. The label of edge is the name of the foreign key.

(2) one-to-one: The one-to-one relationship exists in same entity type or different entity types. For the transformation of different entity types, we suppose that \( E_1 = (A_{11}, A_{12}, \ldots, A_{1n}) \) and \( E_2 = (A_{21}, A_{22}, \ldots, A_{2n}) \).

\[
\begin{align*}
E_1 &= (A_{11}, A_{12}, \ldots, A_{1n}) \\
E_2 &= (A_{21}, A_{22}, \ldots, A_{2n}) \\
E_1.A_{11} &= E_2.A_{11}
\end{align*}
\]  

The attribute \( A_{11} \) is the primary key of \( E_1 \) and a foreign key in \( E_2 \). The attribute \( A_{21} \) is the primary key in \( E_2 \). A value of \( A_{11} \) corresponds to a value of \( A_{21} \). The rule of transformation is described as follows:

1. We need to standardize \( E_2 \). The main step is to retain attributes without foreign key.
2. According to the rule of unrelated, \( E_1 \) and standardized \( E_2 \) can be converted.
3. Connecting node A which belongs to \( E_2 \) and node B that belongs to \( E_1 \) by using foreign key.

![Figure 4(b). transformation for one-to-one of different entity types.](image)

The transformation of same entity type also needs to standardize the entity type, transform entities to nodes, and connect nodes. The label of edge is the name of the foreign key.

(3) one-to-many: The transformation of this relationship is similar to the method of one-to-one. But each value of attribute \( A_{11} \) corresponds to many values of attribute \( A_{21} \). These two transformations are shown as figure 4(c) and figure 4(d).

![Figure 4(c). transformation of one-to-many in same type.](image)

![Figure 4(d). transformation of one-to-many in different entity type.](image)

(4) many-to-many: The relationship of many-to-many is complicated in ER diagram. There is a relational entity to record the relationship of two entity types. We suppose that

\[
\begin{align*}
E_1 &= (A_{11}, A_{12}, \ldots, A_{1n}) \\
E_2 &= (A_{21}, A_{22}, \ldots, A_{2n}) \\
R &= (A_{11}, A_{21}, R_{11}, \ldots, R_{m}) \\
R.A_{11} &= E_1.A_{11}, R.A_{21} &= E_2.A_{21}
\end{align*}
\]

The entity types \( E_1 \) and \( E_2 \) are two different entity types. R is a relational entity about \( E_1 \) and \( E_2 \). Attribute \( A_{11} \) and attribute \( A_{21} \) are respectively the primary key of \( E_1 \) and \( E_2 \), they are foreign
keys in R, and constitute the primary key of R. In this case, it is unnecessary to convert relational entity into node. This specific transformation is as follows:

1. We convert entities of $E_1$ and $E_2$ into nodes by using the rule of unrelated.
2. According to the primary keys in R, we connect nodes with two edges in different directions. These two edges have same label that is the name of R. And we add other attributes of R into edges.

We get a graph $G = (N,E_g)$, where $N = \{N_1, N_2\}$ is a set of nodes from $E_1$ and $E_2$. And $E_g = \{<n_i, n_j>, <n_j, n_i> | n_i, n_j \in N_1, n_i, n_j \in N_2\}$ is a set of edges. Figure 4(e) shows this transformation.

3.3. Algorithm design

The ER model shows all the attributes of every entity type and relationship. Moreover, there are two roles in model. If an entity type has foreign key, it be called child, the referenced entity type is parent.

On the basic of these properties, we can traverse entities and relationships, store their attributes and parent-child relationships, then use them to decide which rule to use and finish transform and data migration. For an entity type E, E.fk is the foreign key, E.parent_child is a set of parent-child relationships. If they are empty, we use the rule of unrelated. If one of them is empty, we need to discuss this under the constraints. In this algorithm, G is a graph, E is the set of entity types. Visited_E is a set of visited entity types, Relation_check is the set of established relationships.

Input: ER diagram
Output: G

1. Visited_E=\{\}, Relation_check=\{\}
2. for $E_i$ in ER diagram
    getting E.pk, E.fk,
    for R in ER diagram
        getting E.parent_child
    for $E_i$ in E
        if $E_i$ \not\in Visited_E
            3.1 if $E_i$.fk = $\phi$ and $E_i$.parent_child = $\phi$ Convert_unrelated($E_i$); Visited_E $\leftarrow \{E_i\}$
            3.2 if $E_i$.fk = $\phi$ and $E_i$.parent_child $\neq$ $\phi$
                find_child_entity($E_i$)$\rightarrow E_j$;
                if $E_j$ \not\in Visited_E and $E_j$.fk contain all $E_i$.pk
                    Convert_one-to-one/many ($E_i,E_j$); Relation_check $\leftarrow \{R_{E_i,E_j}\}$; Visited_E $\leftarrow \{E_i,E_j\}$
            3.3 if $E_i$.fk $\neq$ $\phi$ and $E_i$.parent_child $\neq$ $\phi$
                if $E_i$.fk contain all $E_i$.pk
                    find_parent_entity($E_i$)$\rightarrow$parent: $E_i,E_n$; Convert_many-to-many ($E_i,E_j,E_n$);
                    Visited_E $\leftarrow \{E_n,E_i,E_j\}$; Relation_check $\leftarrow \{R_{E_i,E_j,E_n}\}$
                else
                    find_parent_entity($E_i$)$\rightarrow E_j$;
                    Convert_one-to-one/many ($E_i,E_j$); Relation_check $\leftarrow \{R_{E_i,E_j}\}$; Visited_E $\leftarrow \{E_i,E_j\}$
3.4. Algorithm analysis
In this part, the time complexity of algorithm is given. We assume that the number of entities and relationships in ER diagram is $n$ and $m$ respectively. This algorithm can be divided into two parts. In the first one, the time complexity of getting primary key, foreign key and parent-child relationships is $O(n \times m)$. In the second, for an entity type, we need find its child or parent, execute rule to convert. We suppose that the size of the set of parent-child relationships is $p (p < n)$, the time complexity of this is $O(n \times p)$. The total time complexity is $O(n \times m + n \times p)$, and we can replace this by $O(n^2)$.

4. Experiment

4.1. Environment and data of experiment
This experiment worked on a computer with a 64-bit OS, 8G of memory and 1T hard disk. And we finished work on eclipse by using Java. In this experiment, one data set is “NorthWind” that is a sample database from SQL Server. Another is about movie crawled from IMDB. We used reverse engineering to obtain the ER diagram of existing databases, as figure 5. We used Neo4j in server mode as graph database.

4.2. Result of experiment
Because the speed of server mode is slow, there was a time delay in practice. We finally got a graph in Neo4j. This graph could be viewed in the web of Neo4j. Figure 6(a) shows the final result. Nodes with different labels have different colors. We can set the visible attribute of every node. The results of transformation of single entity, one-to-one/many and many-to-many are shown as figure 6(b), figure 6(c) and figure 6(d).

4.3. Comparison of query performance
When we query some data by using relational database, some join operations may be required. For example, we want to know the date of one order, where the “CompanyName” of Suppliers is
“TokyoTraders”. It is necessary to join table of Order, Order Details, product and Suppliers, as figure 5. The number of relationships is three. However, when we use Cypher to query this data in Neo4j, we only match nodes of supplier, product and order. This query is shown as Figure 7(a). The number of edge is two. We tested two kinds of query performance. The query efficiency of Neo4j is faster than relational database, as figure 7(b).

5. Conclusions
In terms of storing massive data, this paper makes some rules, proposes an algorithm for transforming model and migrating data automatically by traversing ER diagram. This method reduces the impact of model differences between relational databases and graph databases, finishes work with ensuring constraint and integrity of data. And this method will provide support for later research on graph database.

References
[1] Sullivan, D. (2016) Practice Guideline of NoSQL. China Machine Press, Beijing.
[2] Robinson, I., Webber, J., Eifem, E. (2015) Graph Database. Posts & Telecom Press, Beijing.
[3] Shen, D.R., Yu, G., Wang, X.T., et al. (2013) Survey on NoSQL for management of big data. Ruan Jian Xue Bao/Journal of Software, 24:1786-1803.
[4] Fang, W., Zheng, Y., Xu, J. (2014) Big Data: Conceptions, key technologies and application. Journal of Nanjing University of Information Science and Technology: Natural Science Edition, 6:405-419
[5] Wang, Y.L. (2012) Comparison of graphic database NEO4J and relational database. Modern Electronics Technique, 35:77-79
[6] Wang, H.L., Zhang, Y.J. (2016) An algorithm for Schema Conversion from Relational Databases to Document-Oriented Databases. Modern Computer, 6:18-23
[7] Qiu, S.H., Wang, Y.X. (2016) Study on application of modeling technology of graph database on cloud environment. Application Research of Computers, 33:794-797
[8] Ao, B.Y., Hu, Z.Y. (2018) A model of migration data from MySQL to Neo4j database. Inner Mongolia Science Technology & Economy, 3:90-92
[9] Virgilio, R.D., Maccioni, A., Torlone, R. (2013) Converting relational to graph databases. In: International Workshop on Graph Data Management Experiences & Systems. New York, p.6
[10] Singh, M., Kaur, K. (2015) SQL2Neo: Moving health-care data from relational to graph databases. In: Advance Computing Conference. NJ. pp. 721-725
[11] Yelda, Ü., Halit, O. (2018) Migration of data from relation database to graph database. In: The 8th International Conference on information Systems and Technologies. New York. p.5
[12] Zhang, L., Xiong, S.P. (2018) Design and Implementation of Social Network Platform Based on Neo4j, Information Research, 8:77-82
[13] Lu, Y., Luo, R.Q. (2016) Graphic database Neo4J-based research and practice of co-authorship network. Chin J Med Libr Inf Sci, 24:13-16