2016 Fourth International Symposium on Computing and Networking

Toward a Database Refactoring Support Tool

Kohei Hamaji*, and Yukikazu Nakamoto*
*Graduate School of Applied Informatics, University of Hyogo, Japan

***Abstract*—Database schema changes in system development cannot be avoidable. We need refactoring for the schema. Refactoring criteria for the schema, however, has not been clear and the refactoring depends on the skill of database designers. To solve the problems, we consider to finding out refactoring target columns and recommending them as refactoring targets by using clustering techniques with features of database schema and data in the table and implement it as a support tool. We describe the method and the support tool using the method.**

***Keywords*—Database, Refacroting, Machine Learning**

I. Introduction

Database schema changes in system development cannot be avoidable. During the development, system specification change, programming convenience, and performance tuning and bug fixes force to change schema. Change requests of the schema becomes enormous under the development. In these cases, the validity of the schema changes depends on the skill of database designers and they might make wrong changes in the schema. Initially a schema has integrity as a domain and is normalized. The schema will lose, however, its integrity because a lot of changes to the schema have been made. Such situations are the same as source codes.

We need refactoring in tables of database in a real world, the refactoring criteria has not been clear, however. Ambler described that examples needing database refactoring (for short, DB refactoring) have seven bad smells [1]. He did not describe the clear criteria and said that a decision whether refactoring is performed depends on projects, thus DB refactoring is difficult.

To solve the problems, we consider finding out refactoring target columns and recommending them as refactoring candidate by using clustering techniques using features of database schema and data in the table and implement it as a support tool.

There are benefits of using machine learning in the recommendation of the refactoring targets in database schema and implementing as a tool as follows.

- DB refactoring can be carried out based on criteria that do not depend on a database designer who carries out the refactoring.
- DB refactoring can be performed automatically and easily by using the refactoring tool. Therefore a database designer can perform the refactoring as early as possible by iterating DB refactoring and find out wrong schema changes using the refactoring tool.

In this paper, we propose a tool that recommend refactoring targets from features of schema of a table and data in the table. Previously there are a lot of work to support the refactoring work itself in a database and to develop support tools [2], [3]. There are, however, few research on decision support for the refactoring. Addressing this problem leads to facilitating DB refactoring which relies on the skill of some specialist and whose execution is difficult.

The rest of the paper is organized as follows. In Sec. II,we describe features of a table that needs refactoring in a background of the research. Section III describes a proposed method and a tool using machine learning in the recommendation of the refactoring target in database schema. We describe implementation of the tool in Sec. IV. We mention related works in Sec. V and draw conclusions in Sec. VI.

II. Background

A database designer initially designs a table schema in a normal form according to the third normal form. In an actual development process, however, table schemas are repeatedly changed. If those changes are inappropriate, they hinder the development activity and cause problems of software using such tables. The data consistency leads to a compromised table. We addressed the following features of inappropriate database schemata, which Ambler pointed out as bad smells [1]

a) Redundant data: Redundant data is an important problem in operation of database. If data are duplicated in a table, a program to access the table has to update more than two columns of the data, it causes the update missing and the DB consistency is lost. Suppose that there is more than one address column in a customer table. All of the columns must contain an address correctly. When data in one of the address columns is changed, a developer forgets changing the data in the one column and loses to update the data in the other columns despite that multiple address columns must change. This leads to inconsistency of a table. The same situation more likely happens multiple different tables contain columns with the same meaning.

b) Null data: A null data in a database table plays a marker showing that a cell in the database has no value. A null makes it difficult to operate management of the database, and decrease the performance. There is special property for a null data in the database that a value of a null data is not determined and uncertain. Many existences of null in a table in a database mean incorrect design of table or incomplete of normalization of a table [4]. Figure 1 is an example of a table with null cells. One of the causes of Null data occurs is that the table design is inappropriate. That normalization as an

First name	Last name	HomeAddress	BusinessAddress	Vacation address
Tom	Brown	Tokyo....	Tokyo....	Nagano
Mary	May	Osaka...	Osaka...	Null
John	Steve	Kyoto....	Null	null

Fig. 1. Null table

2379-1896/16 $31.00 © 2016 IEEE
DOI 10.1109/CANDAR.2016.65

name	Class name	Grade	Address
Tom	math	80	Tokyo….
Tom	English	60	Osaka…
Mary	Math	80	Kyoto….

Fig. 2. Test result table

First name	Family name	Birth data	Adress	Company Flag
Tom	Brown	1980/8/8	Tokyo….	Null
Mary	May	1977/2/1	Osaka…	Null
ABC shop co.ltd	Null	1990/4/1	Kyoto….	true

Fig. 3. Customer table

example of inappropriate table design is inadequate, and there is such a multi-purpose table, which will be described later. In Fig1 is a table that contains the information of employees in the company. In this table there is more than one column to store the employees of the address In the case of employees who do not have more than one address, this column will be null. In this case, it is possible to prevent the NULL is stored by performing normalization to create another table to separate the address column.

c) Column names with less relationship in a table: If a column name in a table is less relationship with table name, a design of the table might be wrong. A table with such columns causes a conflict to the stored data. Suppose a table of a test result in a class shown in Fig.2. This table has an address column irrelevant to the test result table. Values of the address columns of Tom are Osaka and Tokyo. This occurs because there are two cell of Tom's address and a database operator updates one column of Tom's address, but forgets updating another cell. This causes contradiction. In this case, it is not normal to include an address of the test result table which should have a student name, a subject and a test result. At first, a table is not consider to have an address column. The address column is added to the table to send a test results to a student. If a column name is not related to a table name, the column is often an unnecessary column in the table. Such columns are often added to a table in the late phase of development. It causes a many null cell problem.

d) Multipurpose table: We show an example of a multi-purpose table. Figure 3 is an example of a customer table. The table contains individual customer data and customer company data. Though attributes in the individual customer and in the customer company have different meanings, the same columns are assigned to the attributes in the table. For an example in Fig. 3, a birth date column for the individual customer means a birthday and the same column for customer company are the establishment date of the company. As a result, a user must insert data with different meanings to one column to avoid the above. This column is said to be multipurpose column.

A multipurpose table causes a table with a lot of null value. If one table has multi purposes, there are columns for one purpose. A column for one purpose is not related to another purpose and has a null value in a column of the other purpose. As a result, that table design becomes distorted, the table contains a lot of null cell.

e) A table with too many columns: If there are too many columns in a table, it indicate row data duplication and a multipurpose table. In such table, the number of columns without primary keys much more than one of primary key column and a ratio between them tends to be small. Kageyama reported that if a ratio of primary key columns and ordinary columns in a table is less than 10%, the table has too many columns and is inappropriate because increasing normalization for a table results in that tables becomes smaller and the ratio increases [5].

III. A DATABASE REFACTORING METHOD

In this section, we propose a DB refactoring method. This method has two types of analyses for database schemas and raw data. With the results of these analysis, we identify columns of refactoring. First, we mention k-means clustering technique used in both of two analyses.

A. *K-means clustering*

We use the k-means clustering for clustering columns (eg. [6]). The k-means method splits a certain group into k clusters, where k is the number of clusters and is given in advance. For the split, the method uses the square of the Euclidean distance between the center of a cluster and each group member in the cluster. The general k-means method is as follows:

1) Select k group members randomly to form an initial cluster.
2) Assign the other group member to the cluster by comparing the other members and a center of the clusters.
3) If the cluster assigned has converged or there is no change in the assignment result to the cluster, the algorithm is terminated. Otherwise, recalculate a center of each cluster and return to 2).

B. *Column name clustering*

To solve a problem of the column name with less relationship in a table, we need to identify columns less related to the other columns and a table to determine refactoring targets in the table. We examine relationships between column names and a table name and between column names. A column name clustering that we propose finds out refactoring target by comparing and classifying the similarity of the column name. We perform the clustering with distance between the column names and choose clusters with smaller members as a refactoring candidate.

Figure 4 is a diagram of the column name clustering. The method extracts the refactoring target column with the similarity of the column name. First, we estimate the distances. At the beginning, we attempted to utilize thesauruses on the Internet for the similarity of column names. We found out that thesauruses on the Internet contains large volume of words conceptually related to a certain word, there is, however, few words with part-of relationship of a certain word. An example of the relationship is a company and its address. Thus, we decided to build a co-occurrence word dictionary for our purpose.

We built a co-occurrence word dictionary from the documents in the Web to show degree of similarity between the

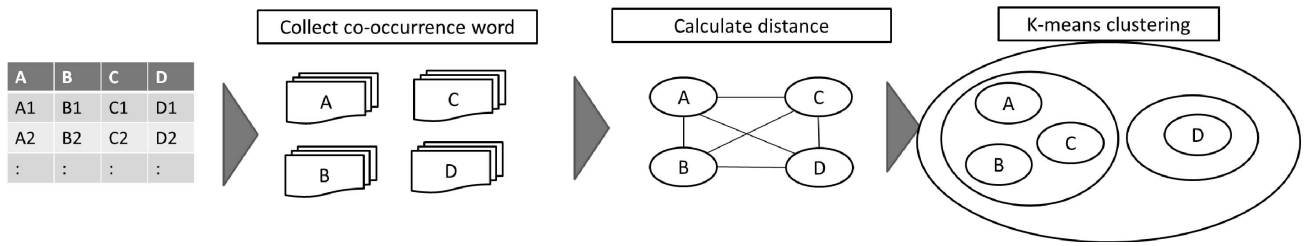

Fig. 4. Column name clustering

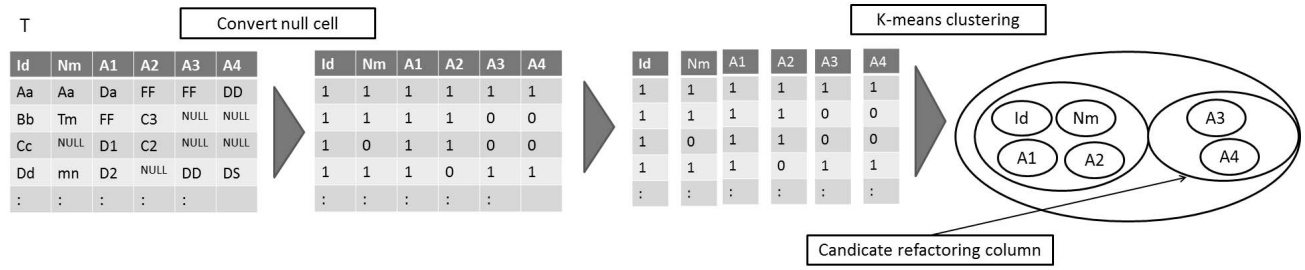

Fig. 5. Null column clustering

column names and carried out by the similarity calculation by the co-occurrence words. Aizawa carried out the words of the similarity calculation from large-scale text corpus and showed the effectiveness[7]. We adopt this technique to compare the similarity of the column name. The method performs search for the column names. From the search results of the TOP l site, the method get the body in the text form, and extract top m words for the column name from the web page as co-occurrence words using KH coder [8], where we set $l = 5$ and $m = 300$ in the first experiment. As a result we determine whether the table should contain the columns for the column refactoring targets. The method counts the number of the same co-occurrence words of each of the column name, and calculates a distance of each column name with the number of the same co-occurrence words. Using the distance of column names, the column name clustering performs k-clustering to form clusters of column names. Column names are candidates for refactoring which belong to separated clusters. Lastly, we seek another way to build a corpus containing similarity between the column names because the search services as mentioned above are limited as of writing this paper.

C. Null column clustering

To reduce null data, normalization of a table and elimination of repeated and duplicated columns are effective. For an example, Ambler described a multipurpose column and a multipurpose table in the database smells, which tend to contain a lot of null data.

Figure 5 is a diagram of a null column clustering that we propose, which extracts null cell patterns and perform the clustering using the null cell pattern for finding out refactoring targets. Table T is a table that contains null data. First, we convert from a table of raw data to a binary table to make it easier to extract a pattern of the null cell location. A binary data table has a two-dimensional data; 1 if the corresponding column is not null, 0 if the corresponding column is null. We perform a k-means clustering for the binary data table. From the result of this clustering, we exclude clusters such as ones with the maximum number of cluster members and ones with primary keys and leave the others as candidates for the refactoring targets. We can see a pattern in A3 and A4 which are different from the others in Fig. 5. Actually, k-means clustering detects A3 and A4 to form a separated cluster.

IV. Implementation

Figure 6 is GUI of a prototype of the DB refactoring tool. This tool has mainly the following three functions.

1) To select a table
   A user selects the refactoring target table from the table list.
2) To display the acquisition and the information in the database to be analyzed. The tool displays the schema information and the data in the table.
3) To extract refactoring target for the selected table and display.
   The tool extracts refactoring candidate columns for the selected table (1), performs the null clustering and the column names clustering mentioned in Sec. III for the selected table and displays the refactoring target column that has been extracted in the results.

V. Related works

DB refactoring is accompanied by various difficulties, such as the migration of data due to the change of the schema. There are several studies on the DB refactoring of support to solve these difficulties. Ambler and Sadalage proposed methods of schema changes, migration of data and application in DB refactoring [4].

Curino, et al. attempted to automate updating of the database schema [2]. They proposed a method and the prototype tool for automatic updating of the query and mapping

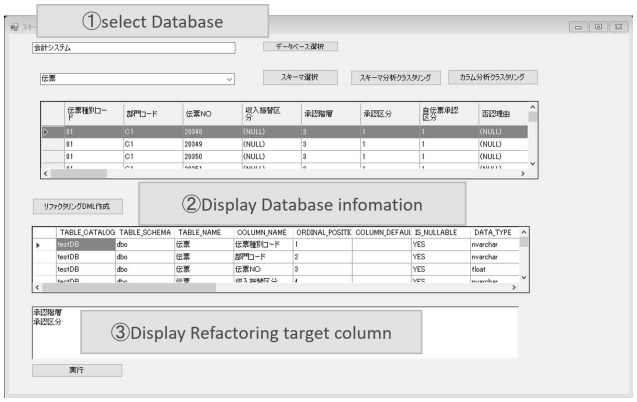

Fig. 6. GUI of the DB refactoring tool

applications of the refactoring process. Moreover Clevea created prototype as a tool to support the re-engineering of the database [3].

Though studies about the tools that support refactoring are done, there are no refactoring tool for targeting columns in a database at our best knowledge.

On the other hand, recommending target for refactoring of source code made well in general. For an example, Goto is going to recommend the refactoring target using a machine learning [9].

VI. Conclusion and Remarks

We proposed two DB refactoring methods, the null column clustering and the column name clustering, to find out refactoring target columns in a table, which uses the clustering techniques using features of database schema and data in the table. Moreover, we implemented the tool to support that.

One of the remaining works is evaluation of our methods. We consider to use the open source and open data library based on complexity of tables such as [10] for the evaluation.

References

[1] S. Ambler, *Agile Database Techniques*. Wiley & Sons., 2003.

[2] C. Curino, H. J. Moon, A. Deutsch, and C. Zaniolo, "Automating the database schema evolution process," *The VLDB Journal*, vol. 22, no. 1, pp. 73–98, 2013.

[3] A. Clevea, M. Goberta, L. Meuricea, J. Maesa, and J. Weberb, "Understanding database schema evolution: A case study," *Science of Computer Programming*, vol. 97, no. 1, pp. 113–121, 2015.

[4] S. J. Ambler and P. J. Sadalage, *Refactoring Databases: Evolutionary Database Design*. Addison-Wesley Professional, 2006.

[5] Y. Kageyama, "Utilization of Database Metrics - Towards Enhancement od System Internal Qualities -," in *Japan Symposium on Software Testing 2013*, 2013.

[6] T. Hastie, R. Tibshiran, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. Springer, 2009.

[7] A. Aizawa, "On Caluculating Word Similarity Using Large Text Corpora," *IPSJ journal*, vol. 49, no. 3, pp. 1426–1436, 2008.

[8] K. Higuchi, "KH Coder," 2001. [Online]. Available: http://khc.sourceforge.net/en/

[9] A. Goto, N. Yoshida, K. Fujiwara, E. Choi, and K. Inoue, "Recommending Extract Method Opportunities Using Machine Learnig," *IPSJ journal*, vol. 56, no. 2, pp. 627–636, 2015.

[10] C. Calero, M. Piattini, and M. Genero, "Mertrics for Controlling Database Complexity," in *Developing Quality Complex Database Systems: Practices, Techniques and Technologies*, S. A. Becker, Ed. Idea Group Pub, 2001, ch. 3, pp. 48–68.