Developing Multiple Nested Schema to Reduce Data Redundancy in CONoSQLDB Schema

by Dr. Sutedi, S.kom., M.t.i

Submission date: 27-Sep-2021 06:23PM (UTC+1000)
Submission ID: 1658588142
File name: Nested_Schema_to_Reduce_Data_Redundancy_in_CONoSQLDB_Schema.pdf (6.91M)
Word count: 4491
Character count: 24797

Developing Multiple Nested Schema to Reduce Data Redundancy in CONoSQLDB Schema

Sutedi
Department of Electrical Engineering and Information Technology
Universitas Gadjah Mada
Yogyakarta, Indonesia
Department of Information System
Institut Informatika dan Bisnis
Darmajaya
Lampung, Indonesia
sutedi.s3te15@mail.ugm.ac.id
sutedi@darmajaya.ac.id

Teguh Bharata Adji
Department of Electrical Engineering and Information Technology
Universitas Gadjah Mada
Yogyakarta, Indonesia
adji@ugm.ac.id

Noor Akhmad Setiawan
Department of Electrical Engineering and Information Technology
Universitas Gadjah Mada
Yogyakarta, Indonesia
noorwewe@ugm.ac.id

***Abstract*—The Corporate's interest of cloud computing based system continues accelerating due to its efficiency, promising performance and better scalability. Most of these systems use the Not Only SQL (NoSQL) database. The database has different characteristics from the relational database. Data migration from a relational database to a column-oriented NoSQL database (CONoSQLDB) can use the concept of Multiple Nested Schema. The concept has proved to be effectual to boost query performance but at the cost of high data redundancy. This study proposes a new conversion method called Enhanced Multiple Nested Schema, developed of Multiple Nested Schema by adding rules to identify and eliminate inter-table transitive dependency—one of the major causes of data redundancy. We aim to reduce data redundancy in CONoSQLDB Schema. The objective is to generate CONoSQLDB schema much better. Previous research did not pay attention to transitive dependency problems when doing schema conversion. The proposed conversion mechanism produces an efficient CONoSQLDB database schema. However, the schema guarantees complete data and allows queries quickly. This means that the additional rules proposed in this study can improve the CONoSQLDB schema generated using the previous method.**

Keywords—conversion, migration, SQL to NoSQL, transitive dependency, column-oriented NoSQL

I. Introduction

Cloud computing is becoming more and more popular among corporations as a means to reduce business operating cost as the system offers efficiency in terms of resources and cost [1] as well as effective performance and high scalability [2]. The efficiency and effectiveness are accomplished through data management using Not Only SQL (NoSQL) database. This type of database was developed to deal with the inability of relational database to meet the demands for scalability, high readiness, and well-structured contents [3]. Compared to relational database, NoSQL database also has distinct schema characteristics [4]. Therefore, to perform successful data migration from relational database to NoSQL database, an appropriate conversion method is highly required.

There have been various studies on the development of data conversion from relational database to NoSQL database. One notable result from the studies is the concept of Multiple Nested Schema, which was developed by G. Zhao, L. Li, Z. Li, and Q. Lin to run data conversion from relational database to HBase, a type of column-oriented NoSQL databases (CONoSQLDB) [5]. Based on the data model, there are four types of NoSQL databases: (1) column-oriented, (2) key-value, (3) document-oriented, and (4) graph, with column-oriented NoSQL database (CONoSQLDB) bearing the most similarity to relational database [6]. Column-oriented NoSQL database (CONoSQLDB) organizes data in a structured way and store it in a column family [7]. Despite similar mechanism in data organization, CoNoSQLDB carries highly different schema design from that of relational database. For this reason, the concept of Multiple Nested Schema plays a vital role as a framework for data conversion into CoNoSQLDB.

While the concept of Multiple Nested Schema has dramatically improved query performance, the problem with the schema is that the conversion result has high redundancy rate. Thus, we aim to further develop the concept of Multiple Nested Schema so that data redundancy is greatly reduced without losing reliable query performance.

II. Relevant Studies

Many studies have tried to develop data conversion methods from relational database to CONoSQLDB. Reference [8] offered a data translation mechanism from relational database to HBase, which is built on the concept of entity relation model (ERM) and column-based database schema (CBDS). The study described five types of translation: (1) entity translation for a table that has no relationship with another table, (2) 1-1 translation for tables with a one-to-one relationship, (3) 1-m translation for tables with a one-to-many relationship, (4) m-n translation for tables with a many-to-many relationship, and (5) R-R translation for tables with multiple-join keys. However, the produced schema was not able to maximize query performance and the number of resulting column families was superfluous.

Reference [9] suggested a mechanism to manage SQL database schema denormalization by following the design principles of DDI (denormalization, duplication, and intelligent keys). Denormalization and duplication involve aggregating referenced SQL tables into a big table, followed by selecting a set of intelligent keys to identify rows uniquely (row keys). The mechanism operates on the tall-narrow design pattern, so that the selected row keys must keep their highest cardinality. As a result, NoSQL table row keys have to be an aggregate of primary keys on the SQL table. Data migration into NoSQL database is run autonomously by creating an automatic NoSQL schema after SQL schema analysis has been completed. The result of this conversion can in fact optimize query average time, yet the row key composition is inefficient.

To reduce overhead incurred in the determination of the row key composition during data migration from SQL to NoSQL [9], in [6] recommended an automatic mechanism to transform an SQL schema to NoSQL. The mechanism also follows NoSQL's DDI design principles, but the row key selection is executed automatically by combining all of the SQL table's primary keys located in the longest sequence. Though row key selection has been able to be run automatically, the composition remains inefficient.

Similarly, in [10] offered a method to transform relational structure into non-relational structure of column-oriented database. The method involves four main steps: (1) creating the waiting list, (2) creating the adjacency matrix, (3) selecting the appropriate starting point, and (4) creating column-oriented tables and selecting the appropriate completion path. However, the method has not produced a schema that will eventually maximize query performance. Besides, the row keys remain inefficient because they are made up of unnecessary composite attributes.

Reference [11] made use of Sqoop to migrate data from Oracle 11g to HBase. The converted data is stored in XML/CSV or other compatible formats and HBase maps the tables according to the data that will be accessed by any related applications. This data mapping mechanism depends fully on the system. Even the schema mapping pattern cannot handle either nested or multiple nested conditions hence the query needs to refer to several tables in relational database via join keys [12].

To migrate relational database to NoSQL, in [13] offered an approach which consists of three main stages: (1) data extraction, (2) data processing, and (3) data conversion. However, the study did not provide any real example of its application, making its effectiveness questionable.

Reference [12], an enhanced HBase schema was applied to migrate data from relational database. This enhancement supports multiple nested conditions that indicate join reference relationships between three tables or more. At first, such conditions were difficult to manage because HBase schema supports column family nest only. With the enhancement, data is easily acquired only with a query on a single HBase table, which in turn leads to increased query efficiency, particularly in a join query. Also, the schema provides index migration that supports faster query. In spite of such advantages, the conversion results using Multiple Nested Schema carry high rate of data redundancy, which is caused by overlooked inter-table relationships that potentially generate data redundancy. One of these inter-table relationships is the inter-table transitive dependency [14].

III. Intertable Transitive Dependency

In relational database, inter-table transitive dependency occurs when the primary keys of Table T2 and Table T3 become foreign keys in Table T1. Similarly, the primary key of Table T3 becomes a foreign key in Table T2 or vice versa, in which the foreign key is identical to one of the foreign keys in Table T1. The situation is described in the case example in Figure 1. In the table, the scenarios of T1 is Table Student, T2 is Table Specialty, and T3 is Table Department.

The relational database in Figure 1 contains six tables as follows:

a) Table Type_of_Bill consists of the attributes Type_of_Bill and Type_of_Bill_ID (the primary key).

b) Table Billing consists of the attributes Bill_Dates, Type_of_Bill_ID, Student_ID, Amount, Chasier_ID, and Billing_ID (the primary key).

c) Table Cashier consists of the attributes Cashier_Name and Cashier_ID (the primary key).

d) Table Student consists of the attributes

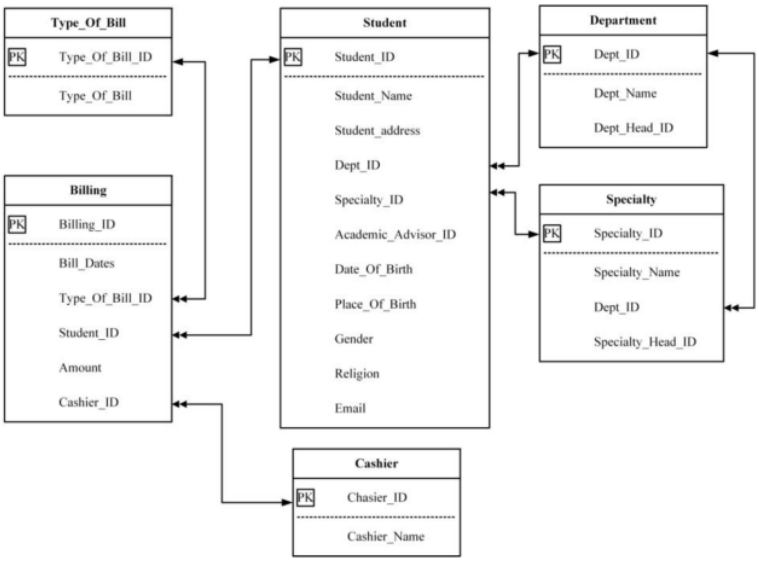

Figure 1. Case example of relational database that contains inter-table transitive dependency

Student_Name, Student_address, Dept_ID, Specialty_ID, Academic_Advisor_ID, Date_of_Birth, Place_of_Birth, Gender, Religion, Email, and Student_ID (the primary key).

e) Table Specialty consists of the attributes Specialty_Name, Dept_ID, Specialty_Head_ID and Specialty_ID (the primary key).

f) Table Department consists of the attributes Dept_Name, Dept_Head_ID, and Dept_ID (the primary key).

In the case example above, the primary key of Table Specialty (Specialty_ID) and that of Table Department (Dept_ID) become foreign keys in Table Student, which clearly indicates that Table Student refers to both Table Specialty and Table Department. Furthermore, the primary key of Table Department becomes foreign keys in Table Specialty, thus the Table Specialty also refers to Table Department. As a result, Table Student is transitively dependent on Table Department because Table Student can obtain part or all of the data from Table Department via Table Specialty.

IV. The Proposed Mechanism

This study proposes a relational database conversion method to the CONoSQLDB schema by adding rules that are used to identify and eliminate inter-table transitive dependency in the concept of Multiple Nested Schema. It is reasonable to eliminate inter-table transitive dependency in a relational database due to its redundant nature. Elimination of such relationships is required to prevent unnecessary storage usage. Therefore, our proposed mechanism of data conversion begins with the process of identifying and eliminating inter-table transitive dependency in relational database, followed by the conversion of relational tables into column families in CoNoSQLDB. The simplified illustration of the mechanism is displayed in Figure 2. In this figure, the conversion process begins with examining the relationships between tables in the relational database. If there is a transitive dependency then the foreign key that causes the dependency is removed. If the relational database does not have transitive dependencies, then all the relational tables are converted into column families in CoNoSQLDB starting from the outermost reference table. Furthermore, the row key for the resulting NoSQL schema is taken from the primary key in the innermost transaction table.

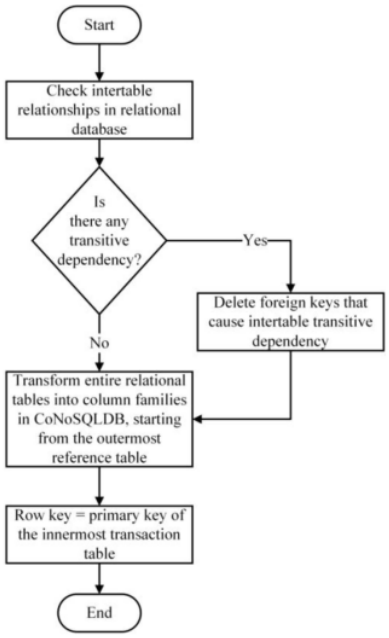

Figure 2. The proposed conversion mechanism

Table 1. Adjacency Matrix For The Case Example In Figure 1

Refer \ Reference	Table Type_of_Bill	Table Student	Table Department	Table Billing	Table Specialty	Table Cashier
Table Type_of_bill	0	0	0	0	0	0
Table Student	0	0	1	0	1	0
Table Department	0	0	0	0	0	0
Table Billing	1	1	0	0	0	1
Table Specialty	0	0	1	0	0	0

To help identify the outermost reference table and the innermost transaction table in a relational database, an adjacency matrix as exemplified in Table 1 may come in handy. In the adjacency matrix (see Table 1 illustration), the intersection between a row and a column in a table will form an index for each matrix element aij, which will equal 1 if either of the following conditions are met [10]:

a) Table i, and Table j are in a relationship where Table i is on "many" side.

b) Table i, and Table j are in a one-to-one relationship.

Therefore, if the case as illustrated in Figure 1 put into a matrix, the result in Table 1. The innermost transaction table, or the root table, always refer to one or more other tables, and no referred by any other table within the database. In the matrix illustrated in Table 1, the innermost transaction table is a table that has the most entries of the value of 1 in its row and has no entry of the value of 1 in its column. In the above case example, the innermost transaction table is Table Billing, in which there are three values of 1 in the billing row and all values of zero in the billing column. On the contrary, one or more tables refer the outermost reference table. The outermost reference table does not refer to any other table and lies in the farthest path of the root table. In a matrix, such table has no entry of the value of 1 in its row but has the most entries of the value of 1 in its column. In the case example above, Table Department is the outermost reference table.

V. Result and Discussion

The complete conversion of the case in Figure 1 is explained below:

a) By eliminating transitive inter-table dependencies, the relational database structure as shown in Figure 1 will be altered into a relational database structure as shown in Figure 3.

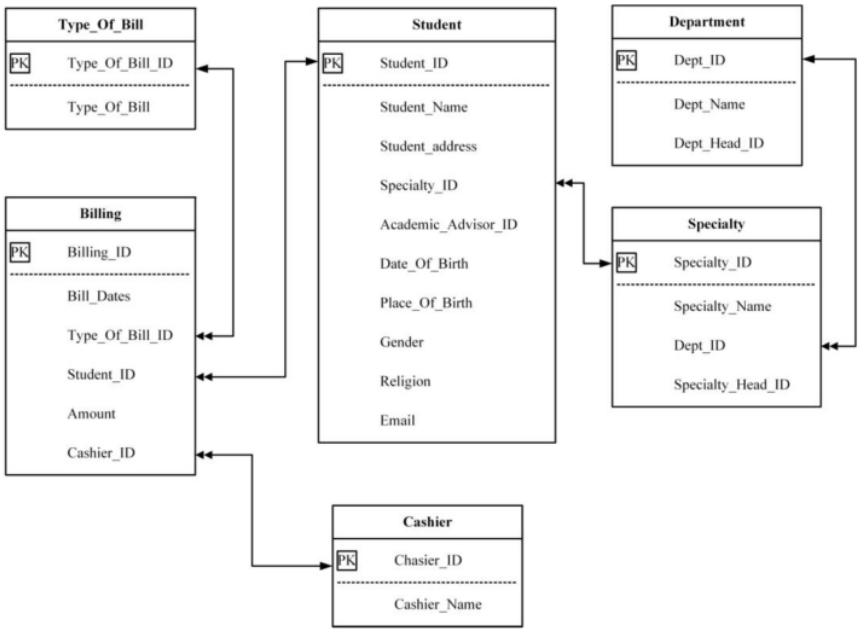

Figure 3. Database structure that contains no inter-table transitive dependency

b) The tables are then converted into column families in the CONoSQLDB schema. The process starts from the outermost reference table and the table that refers to it then continues with the next table until it reaches the root table. If there are still relational tables that have not been converted into column families, repeat these steps until all relational tables are included in the CONoSQLDB schema. Thus, the sequence of conversion begins with Table Department and Table Specialty, followed by Table Student then Table Billing. Next, Table Cashier and Table Type_of_Bill are converting into column families in the CONoSQLDB schema. For a detailed explanation, see Figure 4.

c) Every time a merger is made between column families, the foreign key used to connect the two column families must be eliminated.

d) The row keys for the resulting CONoSQLDB schema are derived from the primary key of the root table. In the case example in Figure 1, the root table is Table Billing and thus the row key is Billing_ID. Figure 5 shows a CONoSQLDB schema that forms the final product of data conversion from the case example shown in Figure 1.

Compared to the schema displayed in Figure 6, which is a conversion obtained using the concept of Multiple Nested Schema [12], the conversion schema in our study has the same quantity of column families but has lower data redundancy. Similarly, compared to the conversion results utilizing DDI [6][9][10] as shown in Figure 7, the resulting schema has less column families and lower data redundancy. Moreover, the schema in our study is more efficient in term of row key selection, and relatively successful at retaining relationship between entities in accordance with the original relational database.

The amount of data redundancy that occurs in the conversion result is measured using Equation 1 [14] below:

$$Rd=Rc(Tr)*\sum(Sz(Tci)-Pk(Tci)) \quad (1)$$

Rd is the number of redundancies that occur, $Rc(Tr)$ is the number of records (Rc) in the root table (Tr), $Sz(Tci)$ is the record size (Sz) of each table-i (Tci), and $Pk(Tci)$ is the primary key (Pk) from each table-i (Tci). Before applying the Equation 1 to calculate the amount of data redundancy, it is necessary to know in advance the sequence of conversion processes for each method used. The sequence of the conversion process using Enhanced Multiple Nested Schema is given in Equation 2.

$$A=(((((E\cup F)\cup(D-(D\cap F))\cup C)\cup B)\cup A) \quad (2)$$

Set A represents Table Billing, Set B represents Table Type_Of_Bill, Set C represents Table Cashier, Set D represents Table Student, Set E represents Table Specialty, and Set F represents Table Department. Thus, the amount of data redundancy can be calculated as follows.

$$Rd=Rc(A)*\sum((Sz(E)-Pk(E))+(Sz(F)-Pk(F))+(Sz(D)-Pk(D))+(Sz(C)-Pk(C))+(Sz(B)-Pk(B))$$

$$=219.720*((56-4)+(62-2)+((192-2)-10)+(33-10)+(52-2)$$

$$=80.197.800 \text{ Byte}$$

$$=76,48 \text{ MB}$$

Next, the sequence of the conversion process using Multiple Nested Schema is given in Equation 3.

$$A=(((((E\cup F)\cup D)\cup F)\cup C)\cup B)\cup A) \quad (3)$$

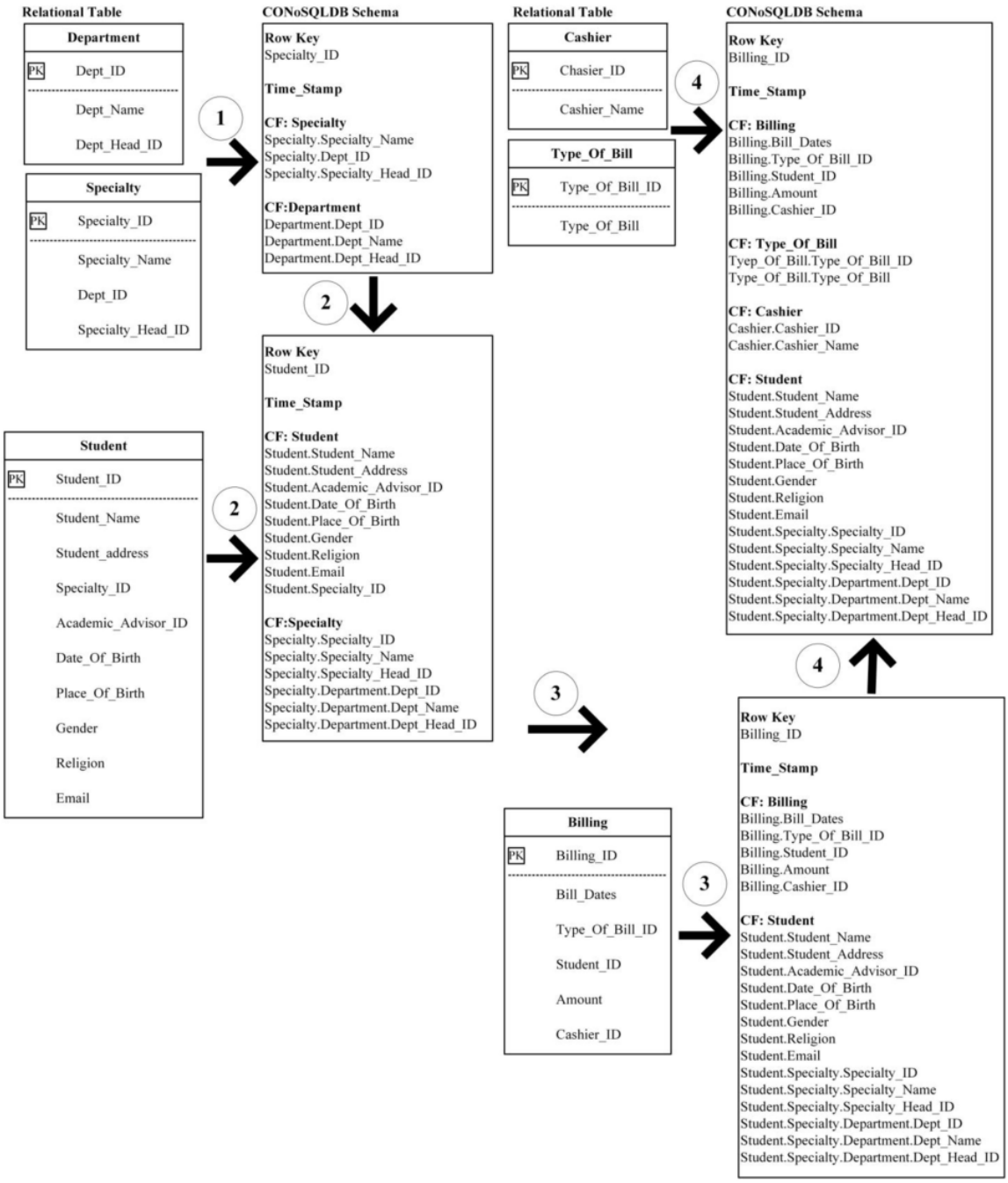

Figure 4. Detailed stages of the data conversion process from the case examples shown in Figure 1

Thus, the amount of data redundancy can be calculated as follows.

$$Rd = Rc(A) * \sum ((Sz(E)-Pk(E))+(Sz(F)-Pk(F))+(Sz(D)-Pk(D))+(Sz(F)-Pk(F))+(Sz(C)-Pk(C))+(Sz(B)-Pk(B))$$

$$= 219.720*((56-4)+(62-2)+(192-10)+(62-2)+(33-10)+(52-2)$$

$$= 93.820.440 \text{ Byte}$$

$$= 89,47 \text{ MB}$$

Row Key
Billing_ID

Time_Stamp

CF: Billing
Billing.Bill_Dates
Billing.Type_Of_Bill_ID
Billing.Student_ID
Billing.Amount
Billing.Cashier_ID

CF: Type_Of_Bill
Type_Of_Bill.Type_Of_Bill_ID
Type_Of_Bill.Type_Of_Bill

CF: Cashier
Cashier.Cashier_ID
Cashier.Cashier_Name

CF: Student
Student.Student_ID
Student.Student_Name
Student.Student_Address
Student.Academic_Advisor_ID
Student.Date_Of_Birth
Student.Place_Of_Birth
Student.Gender
Student.Religion
Student.Email
Student.Specialty.Specialty_ID
Student.Specialty.Specialty_Name
Student.Specialty.Specialty_Head_ID
Student.Specialty.Department.Dept_ID
Student.Specialty.Department.Dept_Name
Student.Specialty.Department.Dept_Head_ID

Figure 5. Schema obtained using Enhanced Multiple Nested Schema

Row Key
Billing_ID

Time_Stamp

CF: Billing
Billing.Bill_Dates
Billing.Type_Of_Bill_ID
Billing.Student_ID
Billing.Amount
Billing.Cashier_ID

CF: Type_Of_Bill
Type_Of_Bill.Type_Of_Bill_ID
Type_Of_Bill.Type_Of_Bill

CF: Cashier
Cashier.Cashier_ID
Cashier.Cashier_Name

CF: Student
Student.Student_ID
Student.Student_Name
Student.Student_Address
Student.Dept_ID
Student.Specialty_ID
Student.Academic_Advisor_ID
Student.Date_Of_Birth
Student.Place_Of_Birth
Student.Gender
Student.Religion
Student.Email
Student.Specialty.Specialty_ID
Student.Specialty.Specialty_Name
Student.Specialty.Dept_ID
Student.Specialty.Specialty_Head_ID
Student.Specialty.Department.Dept_ID
Student.Specialty.Department.Dept_Name
Student.Specialty.Department.Dept_Head_ID
Student.Department.Dept_ID
Student.Department.Dept_Name
Student.Department.Dept_Head_ID

Figure 6. Schema obtained using Multiple Nested Schema

Row Key
Type_Of_Bill_ID
Cashier_ID
Student_ID

Time_Stamp

CF: Billing
Billing_ID
Billing.Bill_Dates
Billing.Type_Of_Bill_ID
Billing.Student_ID
Billing.Amount
Billing.Cashier_ID

CF: Type_Of_Bill
Type_Of_Bill.Type_Of_Bill

CF: Cashier
Cashier.Cashier_Name

CF: Student
Student.Student_Name
Student.Student_Address
Student.Dept_ID
Student.Specialty_ID
Student.Academic_Advisor_ID
Student.Date_Of_Birth
Student.Place_Of_Birth
Student.Gender
Student.Religion
Student.Email

CF: Specialty
Specialty_ID
Specialty.Specialty_Name
Specialty.Dept_ID
Specialty.Specialty_Head_ID

CF: Department
Dept_ID
Department.Dept_Name
Department.Dept_Head_ID

Figure 7. Schema obtained using DDI Concept

Furthermore, the sequence of the conversion process using the DDI Concept is given in Equation 4.

$$A=(((((E \cup F) \cup D \cup C) \cup B) \cup A) \quad (4)$$

Therefore, the amount of data redundancy can be calculated as follows.

$$Rd=Rc(A)*\sum((Sz(E)-Pk(E))+(Sz(F)-Pk(F))+(Sz(D)-Pk(D))+(Sz(C)-Pk(C))+(Sz(B)-Pk(B))$$

$$=219.720*((56-4)+(62-2)+(192-10)+(33-10)+(52-2)$$

$$=80.637.240 \text{ Byte}$$

$$=76{,}90 \text{ MB}$$

Figure 8 shows a comparison of the amounts of data redundancy—calculated using Equation 1—of each CONoSQLDB schemas, which generated from the conversion process using Enhanced Multiple Nested Schema, Multiple Nested Schema [12], and DDI Concept [6] [9] [10]. Based on the experiment, Enhanced Multiple Nested Schema generated a CONoSQL schema that is lower than else. In Figure 8, the CONoSQLDB schema generating from Enhanced Multiple Nested Schema has a data redundancy of 76.48 MB. Meanwhile, the Multiple Nested Schema Concept generates a CoNoSQLDB schema with data

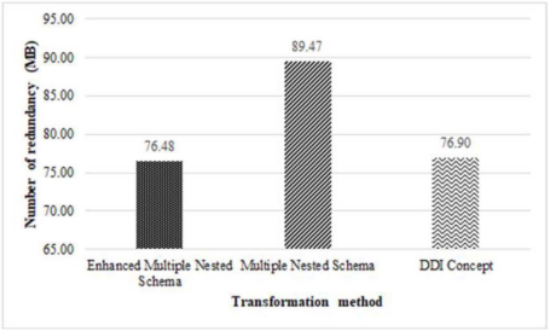

Figure 8. Comparison of the amounts of data redundancy

redundancy of 89.47 MB. In the same case, the DDI Concept generates a CONoSQLDB scheme with data redundancy of 76.90 MB. Thus, the CONoSQLDB schema generated from Enhanced Multiple Nested Schema has lower data redundancy than the resulting schema using Multiple Nested Schema with a difference of 14.52%, and lower than the schema produced using the DDI Concept with a difference of 0.54%.

VI. Conclusion

In the concept of Multiple Nested Schema, the addition of rules to solve inter-table transitive dependency issue can really minimize data redundancy within a CONoSQLDB schema environment. It is proven that the proposed CONoSQLDB schema has a lower amount of data redundancy compared to the schema generated using the Multiple Nested Schema which does not use additional proposed rules. In addition, the proposed schema also has a lower amount of data redundancy and simpler row keys than the schema produced using the DDI Concept. However, future research is needed to address other problems such as multiple relationships between tables, unary relationships, and subtypes/super-type relationship. These problems must be handled properly to ensure that an efficient and complete CONoSQLDB schema can be generated.

References

[1] G. Karnitis and G. Arnicans, "Migration of Relational Database to Document-Oriented Database : Structure Denormalization and Data Transformation," in 7th International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN), 2015, pp. 113–118.

[2] C. Li, "Transforming Relational Database into HBase : A Case Study," in International Conference on Software Engineering and Service Sciences (ICSESS), 2010, pp. 683–687.

[3] L. Rocha, F. Vale, E. Cirilo, D. Barbosa, and F. Mouroa, "A Framework for Migrating Relational Datasets to NoSQL," in ICCS 2015 International Conference On Computational Science, 2015, vol. 51, pp. 2593–2602.

[4] G. Zhao, Q. Lin, L. Li, and Z. Li, "Schema Conversion Model of SQL Database to NoSQL," in Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2014, pp. 355–362.

[5] F. Yang, J. Cao, and D. Milosevic, "An Evolutionary Algorithm For Column Family Schema Optimization In HBase," in IEEE International Conference on Big Data Computing Service And Applications, 2015, vol. 6, pp. 439–445.

[6] C. H. Lee and Y. L. Zheng, "Automatic SQL-to-NoSQL Schema Transformation over the MySQL and HBase Databases," in International Conference on Consumer Electronics-Taiwan (ICCE-TW), 2015, pp. 426–427.

[7] Y. Hu and S. Dessloch, "Extracting deltas from column oriented NoSQL databases for different incremental applications and diverse data targets," Data Knowl. Eng., vol. 93, pp. 42–59, 2014.

[8] M. Dagar, S. Mittal, and M. Singh, "Conversion from Relational-Based Database to Column-Based Database," Int. J. Sci. Res. Comput. Sci., vol. 1, no. 1, pp. 29–35, 2013.

[9] C. H. Lee and Y. L. Zheng, "SQL-to-NoSQL Schema Denormalization and Migration : A Study on Content Management Systems," in IEEE International Conference on Systems, Man, and Cybernetics, 2015, pp. 2022–2026.

[10] D. Mohammadpur and A. Zeid Abadi, "Column-Oriented Storage Optimization in Multi-Table Queries," Rev. Tec. la Fac. Ing. Univ. del Zulia, vol. 38, no. 2, pp. 62–68, 2015.

[11] V. Seshagiri, M. L. Vadaga, J. J. Shah, and P. Karunakaran, "Data Migration Methodology from SQL to Column Oriented Databases (HBase)," Int. J. Adv. Res. Comput. Eng. Technol., vol. 5, no. 11, pp. 2631–2635, 2016.

[12] G. Zhao, L. Li, Z. Li, and Q. Lin, "Multiple Nested Schema of HBase for Migration from SQL," in Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2014, pp. 338–343.

[13] S. A. T. Mpinda, L. G. Maschietto, and P. A. Bungama, "From Relational Database to Column-Oriented NoSQL Database : Migration Process," Int. J. Eng. Res. Technol., vol. 4, no. 05, pp. 399–403, 2015.

[14] Sutedi, T. B. Adji, and N. A. Setiawan, "Enhanced Graph Transforming Algorithm to Solve Transitive Dependency between Vertices," in International Conference on Signals and Systems (ICSigSys), 2017, pp. 250–255.

Developing Multiple Nested Schema to Reduce Data Redundancy in CONoSQLDB Schema

ORIGINALITY REPORT

7%	2%	6%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

MATCH ALL SOURCES (ONLY SELECTED SOURCE PRINTED)

3%

★ Sutedi, Teguh Bharata Adji, Noor Akhmad Setiawan. "Enhanced Graph Transforming Algorithm to solve transitive dependency between vertices", 2017 International Conference on Signals and Systems (ICSigSys), 2017

Publication

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		

Developing Multiple Nested Schema to Reduce Data Redundancy in CONoSQLDB Schema

GRADEMARK REPORT

FINAL GRADE

/0

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7