Efficient checking of functional dependencies for relations

Hyontai Sug
Division of Computer Eng., Dongseo University, Busan 47011, Korea
sht@dongseo.ac.kr

Abstract. Data consistency is very important in relational databases, because relational databases are adopted by most operational database and inconsistent data can generate inconsistent reports or search results. In order to avoid data inconsistency problem, the underlying relation variables should be at least in the third form. But, the relational variables may not be in the third normal form due to incorrect database design. In this paper an efficient method is suggested to prevent possible data inconsistency problem due to incorrect database design based on functional dependency checking on non-key attributes for stored data. The idea of the method is based on the principle of functional dependency on relational variables and sorting on non-key attribute values. An experiment was performed to show how we may detect possible functional dependencies in real database. A database called HR from Oracle is used for the experiment, where HR account has example relations designed by experts for practice. The experiment showed good results and found several possible functional dependencies between non-key attributes so that a database designer may use the found functional dependencies to improve the structure of the database.

1. Introduction

Functional dependency is an important factor for the data integrity of relational databases. When we design relational tables, we check the structure of the tables and want to see whether the tables are in at least the third normal forms [1]. The theory about normal forms are well developed and applied to the field of relational database design for decades. When a database designer design the structure of tables, he/she allocates appropriate attributes for each relation variable, and checks functional dependencies in the relational variables for possible future data inconsistency and anomalies. The checking process is done manually so that there is always some possibility that there are unnoticed functional dependencies between non-key attributes or functional dependency of non-key attributes on a part of the primary key when the key is composite key. Inappropriate functional dependencies make the relational variables not in the third or second normal forms, and this fact could make the related relation variables have inconsistent data and may generate anomalies as the data are updated, inserted and deleted with time. But, it may not be easy to detect such design mistakes by human, because we are not accustomed to recognize our own mistakes, and sometimes the database designer may not be very familiar with the domain of the database. Therefore, in this paper we want to find how we can check possible functional dependencies between non-key attributes efficiently when data have been stored for a while. If such possibilities are found, we may modify the structure of related relation variables.

Discovering functional dependency (FD) efficiently for data sets in table form has attracted a lot of attention from researchers as an optimization problem of time complexity, because we can have $2^n$
combinations of attributes for a table having \( m \) attributes, and possible FDs have to be checked based on possibly very large data in the table. The efficient discovering process can be divided into two or three categories. Top-down, bottom-up [2], and hybrid approach [3]. Top-down approach like TANE [4] or TANE-based incremental algorithm [5] generate the lattice of attributes first to generate candidate FDs, and the candidate FDs are tested for validity using data in the related table. Bottom-up approach like FastFD [6] generates difference sets and agree sets of attributes based on some tuple pairs in the table. The sets are used to drive all functional dependencies. Hybrid approach mixes the good points of the two approaches and generates better performance [3]. FDEP generates negative cover and positive cover based on FD-tree by pair-wise comparison of all tuple pairs in a table [7]. In [8] sampling-based algorithm is suggested to find approximate functional dependencies. In [9] the principles and performances of seven different representative algorithms to find FDs are compared by experiment. All in all, the time complexity of the algorithms are polynomial times because they check the validity of candidate FDs of all length repeatedly based on data. Moreover, most of the experiments are based on data sets not in real relations like the data sets in UCI machine learning database [10]. In other words, the data sets in the database are not from real relational databases, even though the data sets are in table shape. Note that we expect that relational tables should be in the third form, but most of the data sets in UCI machine learning database are prepared for data mining so that they are not good example relations to check functional dependency between attributes especially for the purpose of better database structure. Note that the purpose of data mining is to find hidden knowledge, for example, in rule form like if \( A=a \) then \( B=b \) with confidence of 0.7, which means that there are possibility of \( B=b \) with confidence of 0.7 and there are also other possibility of \( B\neq b \) with confidence of 0.3 when \( A=a \). Here, large characters mean attribute names and small characters mean attribute values. But, functional dependency represents many to one correspondence between attribute values so the confidence is always 1.0, so that the meaning of the two are different. Therefore, we want to find a better method to find FDs for real relations, and use the FDs to improve the related relation scheme or relation variable.

In section 2 we will discuss our method to surmount the problem, and result of experiments will be discussed, and in section 3 conclusions will be presented.

2. The method and experiment
When we design relation variables, we may draw entity-relationship diagram based on requirement analysis, and after that we determine the relation schemes, and each relation scheme is called relation variable [11]. Therefore, the design process is very rely on the mindset of database designer. During the design process determining functional dependencies in relation variables is important task to observe the regulations of normal forms. In section 2.1 we’ll see needed inference rules of our task, and in section 2.2 the algorithm of our task will be suggested, and in section 2.3 the result of experiment will be presented.

2.1. Functional dependencies
Functional dependency can be checked by Armstrong’s axioms. Armstrong’s axioms have three basic inference rules.

Armstrong’s axioms:
Let \( R \) be a relation variable over the set of attributes \( U \), and \( X, Y, Z, W \) be any subset of \( U \).

1) Reflexivity: if \( Y \subseteq X \), then \( X \rightarrow Y \).

2) Augmentation: if \( X \rightarrow Y \), then \( XZ \rightarrow YZ \).

3) Transitivity: if \( X \rightarrow Y \) and \( Y \rightarrow Z \), then \( X \rightarrow Z \).
The above three rules can be used to infer another set of functional dependencies called the closure of given functional dependencies [1]. But, there are additional rules for convenience.

4) Self-determination: X → X.

5) Decomposition: X → YZ, then X → Y and X → Z.

6) Union: if X → Y and X → Z, then X → YZ.

7) Composition: if X → W and Y → Z, then XY → WZ.

(Proposition) when we check functional dependencies between non-key attributes, functional dependencies that have the same attributes for left had side (LHS) or right hand side (RHS) can be combined together.

(Proof) 1. Assume that we have X → Z and Y → Z, then XY → ZZ by rule 7). Because ZZ → Z and rule 3) then XY → Z.

2. Assume that we have X → Y and X → Z, then X → YZ by rule 6). Q.E.D.

The above proof 1 says that if we have found two different functional dependencies with the same RHS, then the two can be combined, and the above proof 2 says that if we have found two different functional dependencies with the same LHS, then the two can be combined. Because X or Y can be attribute sets having one attribute, if we found functional dependencies, we can combine the two functional dependencies together if they have the same RHS or the same LHS.

If non-key composite attributes have key-like property like a composite key, they can be a LHS of a FD. But, we are interested in finding functional dependencies of LHS and RHS consisting of one attribute, because it is very rare that composite attributes of non-key attributes seldom have key-like property in real relations. Therefore, our basic task is to find functional dependencies between each single non-key attribute. Checking it needs checking functional dependencies between \(m \times C_2\) combinations of attributes where m is the number of non-key attributes. The direction for functional dependency check is bi-directional. That is, we should check functional dependency of X → Y and Y → X for \(m \times C_2\) combinations of attributes.

2.2. Algorithm

We can check possible functional dependencies based on data. As explained in section 2.1 we have to test on all the couples of two attributes from the set of non-key attributes that we are interested. When we design relational variables, we may have several keys for a relational variable. The keys are called candidate keys, and among the candidate keys one key attribute can be selected as the primary key. Keys in a relation variable have functional dependencies on non-key attributes. So these candidate keys do not need to be checked for possible functional dependencies.

Another thing to consider is key-like attributes. Key-like attributes are attributes that have almost unique value for each tuple. For example, if we have employee table, we may have name attribute, and because almost everyone has different name, name attribute looks like a candidate key. But, two different employee may have the same name accidently, the name attribute cannot become a candidate key.
For functional dependency to exist between two attributes, we need to check many to one correspondence between values of two attributes either ways. Note that one to one correspondence is a special case of many to one correspondence of functional dependency. The following is the algorithm to check possible functional dependency for the chosen set of attributes interested.

INPUT: a relation $r$, chosen set of attributes $\{A_1, A_2, ..., A_m\}$
OUTPUT: many to one correspondences between attributes

Begin

$n = |r|$ //total number of tuples
For $i = i$ to $m-1$
    For $j = i-1$ to $m$
        // checking for possible FD in ‘→’ direction
        $r' <= \text{Project } r \text{ with respect to } A_i \text{ and } A_j$;
        Sort $r'$;
        not_found := false;
        For $k = 1$ to $n$
            If one to many correspondence is found in $r'$ Then
                not_found := true;
                Exit;
            End If;
        End For;
        If not_found ≠ true Then
            Output the possible FD;
        End If;

        // checking for possible FD in ‘←’ direction
        $r'' <= \text{Project } r \text{ with respect to } A_j \text{ and } A_i$;
        Sort $r''$;
        not_found := false;
        For $k = 1$ to $n$
            If one to many correspondence is found in $r''$ Then
                not_found := true;
                Exit;
            End If;
        End For;
        If not_found ≠ true Then
            Output the possible FD;
        End If;

    End For;
End For;
End.

The for-loop using control variable $k$ stops whenever it confronts one to many correspondence between the two attribute values so that we do not have to do further comparisons. The time complexity of the algorithm is $O(n\log n \cdot m^2)$ where $n$ is the total number of tuples and $m$ is the number of attributes to compare in the table. The main computation in the algorithm is the sorting, and the time complexity of the fastest sorting algorithm is $O(n\log n)$ [12], and $m$ is usually small numbers in relation variables, therefore, the time complexity is almost $O(n\log n)$.

2.3. Experiment
An experiment was performed using an example database provided by Oracle, a leading database company [13]. When we install Oracle 11g [14], we have HR account as an example account for practice. HR account has ready made seven tables and one view as examples. Users may use the tables for practice. The followings are the schema of the tables. Primary keys are indicated by underlines.

**EMPLOYEES**
- EMPLOYEE_ID
- FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER
- HIRE_DATE, JOB_ID, SALARY, COMMISSION_PCT, MANAGER_ID, DEPARTMENT_ID

**DEPARTMENTS**
- DEPARTMENT_ID
- DEPARTMENT_NAME, MANAGER_ID, LOCATION_ID

**LOCATIONS**
- LOCATION_ID
- STREET_ADDRESS, POSTAL_CODE, CITY, STATE_PROVINCE, COUNTRY_ID

**COUNTRIES**
- COUNTRY_ID
- COUNTRY_NAME, REGION_ID

**REGIONS**
- REGION_ID
- REGION_NAME

**JOBS**
- JOB_ID
- JOB_TITLE, MIN_SALARY, MAX_SALARY

**JOB_HISTORY**
- EMPLOYEE_ID
- START_DATE, END_DATE, JOB_ID, DEPARTMENT_ID

We use example data that have been stored already in the tables. Because EMPLOYEES table has many attributes, we want to check possible functional dependencies between non-key attributes. EMPLOYEES table has 107 tuples, and some attributes have null values. Null values are ignored when we check the possible functional dependencies between attributes. We may skip the attributes, FIRST_NAME, LAST_NAME, EMAIL, and PHONE_NUMBER, because the attributes have mostly unique values like key-like attributes. Key-like attributes have high possibility of functional dependency for non-key attributes. Note also that a key attribute has functional dependency for all non-key attributes in a relation variable. So, we check functional dependencies between attributes, \{HIRE_DATE, JOB_ID, SALARY, COMMISSION_PCT, MANAGER_ID, DEPARTMENT_ID\}.

We have found possible functional dependencies between attributes, \{HIRE_DATE, COMMISSION_PCT\}, \{JOB_ID, DEPARTMENT_ID\}, and \{COMMISSION_PCT, DEPARTMENT_ID\}. Table 1 shows the found values in the tables.

| HIRE_DATE | COMMISSION_PCT |
|-----------|----------------|
| 2004-01-30| 0.35           |
| 2004-03-04| 0.35           |
| 2004-05-11| 0.3            |
| 2004-08-01| 0.35           |
| 2004-10-01| 0.4            |
| 2005-01-30| 0.3            |
| 2005-03-10| 0.3            |
| 2005-03-11| 0.25           |
| 2005-03-19| 0.25           |
| 2005-03-24| 0.25           |
| 2005-08-20| 0.25           |
As we see in the table 1, the correspondence of values between HIRE_DATE and COMISSION_PCT is many to one, so that there exists the possibility of functional dependency, 
HIRE_DATE → COMISSION_PCT.

The other possibility of functional dependency is between JOB_ID and DEPARTEMNT_ID. Table 2 shows the found many to one correspondence in their values.

**Table 2** Found values for possible functional dependency JOB_ID → DEPARTMENT_ID

| JOB_ID     | DEPARTMENT_ID |
|------------|---------------|
| AC_ACCOUNT | 110           |
| AC_MGR     | 110           |
| AD_ASST    | 10            |
| AD_PRES    | 90            |
| AD_VP      | 90            |
| FI_ACCOUNT | 100           |
| FL_MGR     | 100           |
| HR_REP     | 40            |
| IT_PROG    | 60            |
As we see in the table 2, the correspondence of values between JOB_ID and DEPARTMENT_ID is many to one, so that there exists the possibility of functional dependency JOB_ID → DEPARTMENT_ID.

The third possibility of functional dependency is between COMMISSION_PCT and DEPARTMENT_ID. Table 3 shows the found many to one correspondence in their values.

**Table 3** Found values for possible functional dependency COMMISSION_PCT → DEPARTMENT_ID

| COMMISSION_PCT | DEPARTMENT_ID |
|----------------|---------------|
| 0.1            | 80            |
| 0.15           | 80            |
| 0.2            | 80            |
| 0.25           | 80            |
| 0.3            | 80            |
| 0.35           | 80            |
| 0.4            | 80            |

As we see in the table 3, the correspondence of values between COMMISSION_PCT and DEPARTMENT_ID is many to one, so that there exists the possibility of functional dependency COMMISSION_PCT → DEPARTMENT_ID.

Therefore, we found three possible functional dependencies, HIRE_DATE → COMMISSION_PCT, JOB_ID → DEPARTMENT_ID, COMMISSION_PCT → DEPARTMENT_ID. Because RHS of the last two functional dependencies are the same, we may have one more functional dependency, {JOB_ID, COMMISSION_PCT} → DEPARTMENT_ID. Moreover, we can drive HIRE_DATE → DEPARTMENT_ID by the two FDs HIRE_DATE → COMMISSION_PCT and COMMISSION_PCT → DEPARTMENT_ID based on the transitivity in Armstrong’s axioms. From the fact that we could find the above possible functional dependencies based on stored data, so that we may want to redesign the structure of the relation variable to get rid of possible data inconsistency problems and anomalies in the future.
3. Conclusions
Functional dependency is main theory for normal forms of relational databases. The main purpose of
normal forms is to avoid future data inconsistency as updates happen, and suggests to store one fact in
one place. In order to avoid data inconsistency, the underling relation variables should be at least in
the third form. But, the relational variables may not be in the third normal forms due to some
misunderstanding of the domain of the target database or some human mistakes by database designer
when the database designer designed the database. Even though there has been many research results
to find functional dependencies efficiently, the time complexity is rather high, and do unnecessary
tests otherwise avoided. Moreover, most experiments of previous researches have not been done for
real relations. In this paper a more efficient method is suggested to check possible data inconsistency
for stored data in real relations based on functional dependency on non-key attributes. The idea of the
method is based on the principle of functional dependency on relational variables and sorting on non-
key attribute values, and how we may detect possible functional dependencies was shown by
experiment. A database called HR from Oracle is adapted for the experiment. HR account has example
relations designed by experts for practice. The experiment showed good results and found several
possible functional dependencies between non-key attributes even though the database was designed
by experts, so that a database designer may use the found possible functional dependencies to improve
the structure of database. By being able to adapt better structure on databases, we could expect better
and more correct outputs from data. The structural improvement in the database is recommended as
early as possible, because correcting the structure of very large databases requires a lot of costs. Future
research could be more efficient methods for the sorting algorithm of our special purpose when
database is large.

References
[1] Elmasri R and Navathe SB 2017 Fundamentals of Database Systems 7th ed. (Pearson)
[2] Liu J, Li J, Liu C, and Chen Y 2012 IEEE Transactions on Knowledge and Data Engineering
vol 24 no 2 pp 251-264
[3] Pepenbrock T and Naumann F 2016 Proc. of the 2016 International Conference on
Management Data pp 821-833
[4] Huhtala Y, Kärkkäinen J, Porkka P, and Toivonen H 1999 The Computer Journal 42 pp 100-
111
[5] Caruccio L, Cirillo S, Deufemia V and Polese G 2019 Proc. of the 27th Italian Symposium on
Advanced Database Systems 2400 (Grosseto) paper-21
[6] Wyss C, Giannella C, and Robertson E 2001 Proc. of the International Conference on Data
Warehousing and Knowledge Discovery pp 101-110
[7] Flach PA and Savnik I 1999 AI Communications 12 pp 139-160
[8] Kruse S and Naumann F 2018 Proc. Of the VLDB Endowment vol 11 no 7 pp 759-772
[9] Pepenbrock T, Ehrlich J, Marten J, Neubert T, Rudolph JP, Schönberg, Zwiener J, and Naumann
F 2015 Proc. of the VLDB Endowment vol 8 pp 1082-1093
[10] Dua D and Graff C 2019 UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]
Irvine, CA, University of California, School of Information and Computer Science
[11] Date CJ 2003 Introduction to Database Systems 8th ed. (Pearson)
[12] Cormen TH, Leiserson CE, Rivest RL, and Stein C 2009 Introduction to Algorithms
(Cambridge, MA: The MIT Press) p 167
[13] Kolonko K 2018 Performance comparison of the most popular relational and non-relational
database management systems (Blekinge Institute of Technology, Sweden)
[14] Oracle Technology Network / Database / Database Express Edition / Overview
https://www.oracle.com/technetwork/es/database/express-edition/overview/index.html
(Oracle)