STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIX**, Number 2, 2014

FINDING, MANAGING AND ENFORCING CFDS AND ARS VIA A SEMI-AUTOMATIC LEARNING STRATEGY

KATALIN TÜNDE JÁNOSI-RANCZ

Abstract. This paper describes our strategy, which finds Conditional Functional Dependencies (CFDs) and Association Rules (ARs), and instead of using them to clean dirty data we use them to prevent their appearance in the database. We achieve this by differentiating strict CFDs/ARs from apparent CFDs/ARs. If we know about a CFD/AR that it will be valid in the future, we can rely on them by creating constraints which guarantee that the CFD-rule will not be breached by insertions or modifications. Along with complete management of CFDs/ARs our implemented application called DependencyManager also uses Formal Concept Analysis (FCA) methods to analyze the strict CFDs/ARs and draw useful conclusions, helping the users of the application to prevent inconsistencies, fix bugs and optimize their queries and applications by providing a lattice of CFDs/ARs, using usefulness as the relation.

1. Introduction

It is not enough to store data, as the correctness of stored data is also very important. Data cleanups are lowering the frequency of inconsistencies but they are also introducing new inconsistencies, because it's impossible to guarantee that these algorithms are choosing the correct pattern from more patterns. Data quality rules can be defined in the form of FDs, CFDs [6] and ARs among others.

An **FD** $X \to Y$ asserts that any two tuples that agree on the values of all the attributes in X must agree on the values of all the attributes in Y. However, many interesting constraints hold conditionally, that is, on only a subset of the relation. For example if in a dataset we have customers data from different countries, than for customers in the UK, Zip code determines

Received by the editors: August 15, 2014.

2010 *Mathematics Subject Classification.* 68P15.

1998 *CR Categories and Descriptors.* H.2.8 [**Database Management**]: Database Applications – *Data mining*.

Key words and phrases. Dependency Mining, Conditional Functional Dependency, Formal Concept Analysis, Association Rules.

Street, which is not true for other countries. This will be a CFD: $[Country =' UK', ZipCode] \rightarrow [Street]$.

A **CFD** φ on R is a pair $(R : A \rightarrow B, T_p)$, where (1) A, B are sets of attributes in $Attr(R)$, (2) $A \rightarrow B$ is a standard FD, referred to as the FD embedded in φ; and (3) T_p is a tableau with attributes in A and B, referred to as the pattern tableau of φ, where for each X in $A \cup B$ and each tuple $t \in T_p$, $t[X]$ is either a constant a in $Dom(X)$, or an unnamed variable "_" that draws values from $Dom(X)$. CFDs can be described as sets of ARs, which are dependencies holding on particular values of attributes.

An **AR** is a dependency $c \Rightarrow A = a$, where $A = (A_1, \dots, A_n)$ and $a = (a_1, \dots, a_n)$, knowing that c is a condition, A is a set of columns and a is a set of constants. In our implementation only those conditions are supported, which can be represented in the form of $C = ct$, where C is a column and ct is a constant. ARs can be defined as special CFDs with an additional property, namely the determinant column set is empty set.

If a formula for a dependency is almost met, because the number of records which are inconsistent with the rule is very small comparing to the total number of affected records, then one should not consider it to be a CFD/AR just yet, because the inconsistence of the rule's exceptions with the rule might be caused by incorrect or correct data. This leads to the conclusion that the correctness of the rule's exceptions must be checked. If there is at least a correct exception, then the rule is not correct. Otherwise the exceptions should be cleaned up and the rule is a CFD/AR. As a consequence we believe that a dependency learner should only consider some rules to be dependencies if there is no valid exception from the rule. After dependencies were successfully learned they should be validated. By validating dependencies we can determine whether they are Strict Dependencies (SDs) or Apparent Dependencies (ADs). ADs are all dependencies which are applicable to the current dataset but there is no guaranty that after any correct inserts or updates the dependency will be still applicable. Most of the previous works omitted the step of validation, they considered all CFDs/ARs to be valid, but that is a false presumption.

Example 1. *Let us consider the example of a company where in a given moment the position determines the salary; all managers have a salary of X, all technicians have a salary of Y and all cleaners have a salary of Z. If this is not a company policy, just a coincidence, then the FD is invalid, as there is no guarantee that the position will determine the salary in the future too. The functional dependency of work-position determining the salary is an AD.There is no way to automatically determine the validity of such dependencies, because programs are not aware of the nature of the entities stored as records and cannot make logical conclusions. This is the job of human specialists, who* *understand the business logic of the system and the nature of the records stored in the table.*

1.1. **Our contributions.** As part of our research we have implemented an application called DependencyManager, which learns FDs, CFDs and ARs from an arbitrary database. We can validate the learned dependencies by accepting or rejecting them, thus modifying its data from candidate to Strict(SD) or Apparent Dependencies(AD). We have created support for the simplification of the validation by using the relation of MU as the most useful candidates should be taken into account from all the other dependencies, therefore we drastically reduce the number of dependencies to be taken into account and we only store the most useful SDs because all the other dependencies are implied by them. The result of our research is a system which is able to prevent all the possible inconsistencies resulting from inserts or updates which do not comply to the schema of the accepted SDs using constraints. We use FCA methods to analyze the strict CFDs/ARs and draw useful conclusions.

Our approach is useful, because: 1. It prevents inconsistencies to appear in the database. 2. If an error occurs because of mishandling data where a strict CFD/AR is applicable, then in the error logs the problem can be identified and potentially bugs can be found and fixed. 3. Knowing that a CFD/AR is strict, using all the known value sets for determinant columns and dependent columns we can determine the values of dependent columns using a value set of determinant columns. This way data-processing and backup-creating algorithms can be optimized.

2. Related work

CFDs have recently been proposed for pattern-finding purposes; CFD-patterns are frequently used by data-cleaners. In [6, 3, 11, 7], the authors employed exact and approximate CFDs to characterize the semantics of data, for example, discovering and verifying the confidence, support, and parsimony of CFDs on a given relation instance. Recent research on CFDs focused mainly on implication analysis, consistency and axiomatizability, see [7]. In [5], the authors studied how to efficiently estimate the confidence of a CFD with a small number of passes over the input using small space. Fan at al in [9] developed three algorithms for discovering minimal CFDs and a novel optimization technique via closed-item-set mining. A hierarchy of CFDs, FDs and ARs has been proposed in [16] with some theoretical results on pattern tableaux equivalence. Many integrity constraints have been studied for data cleaning in [4, 17, 2, 13]. Existing data repair techniques do not guarantee to find correct fixes in data monitoring, because they may deteriorate the data introducing new errors when trying to repair it. [8] proposed editing rules that, compared to constraints used in data cleaning, are capable to find precise fixes by updating input tuples with master data. [17] proposed a framework called GDR (Guided Data Repair), to handle the problem of data cleaning. This framework involves the user in controlling the cleaning process side by side with existing automatic cleaning techniques through an interactive process. In these previous works we have seen algorithms which find the CFDs and for each CFD, the presence of CFD was periodically checked. Previous works focused on finding dirty data and cleaning them. We believe it is easier to prevent the appearance of dirty data instead of cleaning an already dirty database. So, our algorithm, instead of finding CFDs and applying them for cleaning data, finds CFDs and if a CFD is validated to be an SD, then constraints are generated to prevent any insertion/modification that results in breaking the rule.

It is well known that, FDs admit interesting characterizations in terms of FCA [10, 12]. Different authors in [1, 14] considered that the internal logic of data can also be displayed, by means of the so-called implications, which proved to be the proper framework to describe functional dependencies with FCA tools. In [1], the authors studied the lattice characterization and its properties for Armstrong and symmetric dependencies. In [15], the authors present the relation between CFDs and FCA. They showed that the lattice of CFDs is a synthetic representation of the concept lattice. In contrast to these works, which focused mainly on application of FCA in mining different types of dependencies, we analyze the properties of CFDs and ARs discovered with our application by an implemented FCA. In other words, we exploit FCA here to derive ontology containing concepts.

3. Learning, Presentation, Validation and Forgetting

In this section we share our strategy, which is based on automatic learning, semi-automatic validation and automatic forgetting.

3.1. **Learning.** To gather CFDs in a database, we must iterate all the conditions of all columns of all tables. In our case a condition is of the form of $R.C = ct$, where R is a table, C is a column and ct is a constant.

ARs can be found like CFDs; we just have to iterate every condition of every column of every table. For each condition we must find the ARs.

If A is the set of determinant columns $R.A = \{R.A_1, \dots, R.A_m\}$; B is the set of dependent columns $R.B = \{R.B_1, \dots, R.B_n\}$; $a = \{a_1, \dots, a_m\}$ is an m-dimensional constant and the condition can be checked by a Boolean function c for any record r, then the CFD is valid if and only if it is inconsistent to insert/update record r_1 in such a way that the following criteria holds: $\exists\ r_2 \in R$ such as

$$\begin{aligned} & ((r_1.A_1 = r_2.A_1) \wedge \cdots \wedge (r_1.A_m = r_2.A_m) \\ \wedge \quad & ((r_1.B_1 \neq r_2.B_1) \vee \cdots \vee (r_1.B_n \neq r_2.B_n))) \wedge c(r_2) \wedge c(r_1) \end{aligned}$$

and the AR is valid if and only if it is inconsistent to insert/update record r_1 in such a way that the following criteria holds:

$$((r_1.A_1 \neq a_1) \vee \cdots \vee (r_1.A_m \neq a_m)) \wedge c(r_1).$$

Periodical checking for new ARs and CFDs is essential if one intends to handle them. Newly found dependencies are stored in the database. As new rules are discovered and stored, this periodical action is called *learning*. If a dependency is already known by the system, then the rule will not be learned again. Essentially all the rows of any table in any given moment can be described as a record which fulfills a dependency. This would generate many dependencies, slowing down the learning algorithm and storing a lot of dependencies, while most of them are ADs. As a consequence, constraints are needed to increase the speed of the algorithm and store only relevant ARs and CFDs. For now, we have created two constraints: *MinOccurrenceFrequency* and *MinOccurrenceRate*. With these constraints we can prevent over-learning. DependencyManager allows its users to set the values of these constraints. The first step is to discover conditions. Condition Occurrence is the number of occurrence of a condition in a table. Condition Occurrence depends on a Relation, a Column and a Value and determines the occurrence number of the condition defined by R, C and V. The formula of $R.C = V$ is considered to be a condition if the constraints of $MinOccurrenceFrequency$ and $MinOccurrenceRate$ are met. Let us introduce the notations

$$CO(R, C, V) = \text{card}(r \in R : r.C = V) and$$

$$\begin{cases} c : R \times C \times V \to \{0; 1\}, \\ c(R, C, V) = (R.C = V). \end{cases}$$

$$\begin{aligned} \text{Observe that } CO(R, C, V) &\geq MinOccurrenceFrequency, \\ \frac{CO(R, C, V)}{\text{card}(r \in R)} &\geq MinOccurrenceRate. \end{aligned}$$

The learner in a cycle iterates through all the columns of all tables and learns the potential conditions. If the set of columns of R is $Cols$ and $c(R, C, V)$ is a condition, where C is a column from $Cols$, then the possible AR and CFD scenarios can be described as:

$$\begin{aligned} ARScenarios(R, C) &= SubCols \subset (Cols \setminus \{C\}) \text{ and } SubCols \neq \emptyset. \\ CFDScenarios(R, C) &= \text{Determinant} \subset (Cols \setminus \{C\}) \text{ and Dependent} \subset (Cols \setminus \{C\}), \end{aligned}$$

such as Determinant $\cap$ Dependent $= \emptyset$, Determinant $\neq \emptyset$, Dependent $\neq \emptyset$. We have defined an algorithm (Algorithm 3.1) which is used by the application to learn new CFDs and ARs. Intuitively speaking the algorithm iterates all the interesting conditions of all columns of all tables in the database except the system tables. In each step, all the possible AR and CFD scenarios are generated and if no equivalent is found as a CFD, SD or AD, then the

Algorithm 3.1 Sd-candidate learner

```
1: for all $R\,in\,Tables$ do
2:   $Conditions \leftarrow \emptyset$
3:   for all $C\ in\ R.Cols$ do
4:     $Conditions.Clear()$
5:     $Conditions.AddRange(GetConditions(R, C))$
6:     for all $c\,in\,Conditions$ do
7:       $ARScenarios \leftarrow GetARScenarios(R, C)$
8:       for all $ARScenario\,in\,ARScenarios$ do
9:         if $((IsAR(c, R, C, ARScenario))\ And$
           $(Not(IsAlreadyLearnedAR(c, R, C, ARScenario))))$ then
10:          $StoreAR(c, R, C, ARScenario)$
11:        end if
12:      end for
13:      $CFDScenarios \leftarrow GetCFDScenarios(R, C)$
14:      for all $CFDScenario\,in\,CFDScenarios$ do
15:        if $((IsCFD(c, R, C, CFDScenario))And$
           $(Not(IsAlreadyLearnedCFD(c, R, C, CFDScenario))))$ then
16:          $StoreCFD(c, R, C, CFDScenario)$
17:        end if
18:      end for
19:    end for
20:  end for
21: end for
```

given scenario will be considered to be a new CFD. In Algorithm 3.1 Conditions is the set of conditions applicable on $R.C$, fulfilling the constraints of $MinOccurrenceFrequency$ and $MinOccurrenceRate$. Complexity of Algorithm 3.1. can be calculated knowing the number of tables, the average number of table columns and the average number of column conditions, while its space complexity depends on the number of dependencies, the average number of determinant columns and the average number of dependent columns.

$$\begin{aligned}
& \text{Complexity of Algorithm 3.1} \\
= \; & \Theta\left(T \cdot AFN \cdot ACN \cdot (\mathrm{card}(ARScenarios(T, AFN))\right. \\
& \left. + \,\mathrm{card}(CFDScenarios(R, AFN)))\right) \\
= \; & \Theta(T \cdot AFN \cdot ACN \cdot (2^{AFN-1} - 1 + 3^{AFN-1} - 2 \cdot 2^{AFN-1})) \\
= \; & \Theta(T \cdot AFN \cdot ACN \cdot (3^{AFN-1} - 2^{AFN-1} - 1)),
\end{aligned}$$

where T is the number of Tables, AFN is the average number of table columns, ACN is the average number of column conditions.

Space complexity of Algorithm 3.1 =
$\Theta((\mathrm{card}(Dependency)) \cdot (1 + \mathrm{avg}(\mathrm{card}(Determinant)) + \mathrm{avg}(\mathrm{card}(Dependent))))$

One of our experiment were conducted on a database containing exceptions and we will present this example in entire paper to illustrate the steps of our method. For our experiments, we used the attribute Message, AppID, CrossAppException and OperatingSystemException, where *Message* represented an Exception message, *AppID* was the ID of the Application where the given Exception occurred, *CrossAppException* and *OperatingSystem-Exception* were Boolean values. Possible values for *AppID* were 1 (Forum), 2 (Wiki), 3 (Desktop), 4 (Website) and 5 (Engine). Table 3.1. shows some SDs, a part of accepted dependencies.

Id	Condition	Determinant	Dependent
3	e.Message='Authentication needed'	e.CrossAppException e.OpSystException	CrossAppException='1' and OpSystException='0'
6	e.Message='Connection to DB failed'	e.CrossAppException e.OpSystException e.AppID	AppID='3' CrossAppException='0' OpSystException='0'
5	e.Message='File not found'	e.CrossAppException e.OpSystException e.AppID	AppID='3' CrossAppException='0' OpSystException='1'
2	e.Message='Thread was already closed'	e.CrossAppException e.AppID e.OpSystException	AppID='1' CrossAppException='0' OpSystException='0'
4	e.OpSystException='1'	e.CrossAppException e.AppID e.Message	Message='File not found' AppID='3' CrossAppException='0'

TABLE 1. SDs, accepted CFDs

3.2. **Usefulness.** In this section we define the More Useful relation. Let D_1 and D_2 be two CFDs, and $D_1.c$, $D_1.A$, $D_1.B$ and $D_2.c$, $D_2.A$, $D_2.B$ be the condition, determinant column set and dependent column set of D_1 and D_2 respectively. We define the More Useful (MU) relation as:

$$D_1\,\mathrm{MU}\,D_2 \Leftrightarrow (D_1.c = D_2.c) \wedge (D_1.A \subseteq D_2.A) \wedge (D_1.B \supseteq D_2.B).$$

It is impossible to compare the usefulness of D_1 and D_2 if their conditions are not the same, because they are not applicable on the same record set. Also, if D_1 is more useful than D_2, then D_1 is more descriptive, because $D_1.A$ cannot contain any column outside of $D_2.A$ and $D_2.B$ cannot contain any column outside of $D_1.B$. MU is not a strict relation; $D_1\,\mathrm{MU}\,D_1$ is true to guarantee reflexivity and anti-symmetry. If D_1 is a CFD and there is no $D_2 \neq D_1$ which is more useful than D_1, then D_1 is a most useful CFD. Note that a non-AR cannot be more useful than an AR, because the determinant column set of an AR is empty set, the smallest set in the "$\subseteq$" relation. MU is reflexive, transitive and anti-symmetric. On a set of CFDs defined for a database MU is an order.

3.3. **Validation.** Validation is not always easy for users, because there can be many dependencies learned, but we used the More Useful relation, which reduces significantly the difficulty of validation if used properly. As we mentioned earlier, the user sees a list of most useful CFDs/ARs. He can decide whether they are SDs or ADs. If the user accepts a dependency D_1, then all D_2 will instantly become invalid which meets the criteria of $D_1 \,\mathrm{MU}\, D_2$. As a result any D_2 less useful than D_1 is redundant with D_1. If the user rejects a CFD/AR D_1, then it will become invalid and all CFDs/ARs D_2 which meet the criteria of

$$(D_1 \,\mathrm{MU}\, D_2) \wedge (\nexists D_3 \text{ such that } D_1 \,\mathrm{MU}\, D_3 \,\mathrm{MU}\, D_2)$$

will be shown to the user, because they will become most useful ARs/CFDs. DependencyManager lets the user decide whether a dependency is an SD or AD. The following example illustrates the relation of usefulness between some CFDs.

Example 2. *D_1 is a CFD, $D_1.c = (Exception.CrossAppException = '0')$,*
$D_1.A = (Exception.Message)$, $D_1.B = (Exception.AppID)$.
D_2 is a CFD, $D_2.c = D_1.c$, $D_2.A = (Exception.Message)$,
$D_2.B = (Exception.OperatingSystemException, Exception.AppID)$.
D_3 is a CFD, $D_3.c = D_1.c$, $D_3.B = (Exception.AppID)$,
$D_3.A = (Exception.OperatingSystemException, Exception.Message)$.
D_4 is a CFD, $D_4.c = D_1.c$, $D_4.A = (Exception.Message)$,
$D_4.B = (Exception.OperatingSystemException)$.
D_5 is a CFD, $D_5.c = D_1.c$, $D_5.A = (Exception.AppID, Exception.Message)$,
$D_5.B = (Exception.OperatingSystemException)$.
In our experiment D_1 was SD and D_2, D_3, D_4, D_5 were ADs.
$(D_1.c = D_2.c) \wedge (D_1.A = D_2.A) \wedge (D_2.B \supset D_1.B) \Rightarrow D_2 \,\mathrm{MU}\, D_1$.
$(D_1.c = D_3.c) \wedge (D_1.A \subset D_3.A) \wedge (D_1.B = D_3.B) \Rightarrow D_1 \,\mathrm{MU}\, D_3$.
$D_2 \,\mathrm{MU}\, D_1 \,\mathrm{MU}\, D_3 \Rightarrow D_2 \,\mathrm{MU}\, D_3$.
$(D_2.c = D_4.c) \wedge (D_2.A = D_4.A) \wedge (D_2.B \supset D_4.B) \Rightarrow D_2 \,\mathrm{MU}\, D_4$.
$(D_2.c = D_5.c) \wedge (D_2.A \subset D_5.A) \wedge (D_2.B \supset D_5.B) \Rightarrow D_2 \,\mathrm{MU}\, D_5$.
$(D_4.c = D_5.c) \wedge (D_4.A \subset D_5.A) \wedge (D_4.B = D_5.B) \Rightarrow D_4 \,\mathrm{MU}\, D_5$.
D_2 is a most useful element. The user did not see D_1, D_3, D_4 and D_5 because D_2 was more useful. When we invalidated D_2, then D_1 and D_4 appeared on the view, because after we excluded D_2 from the set, D_1 and D_4 became most useful. D_5 was still hidden, because D_4 is more useful than D_5 and D_3 was hidden, because D_1 is more useful than D_3.

The size of needed space depends on the number of dependencies. If there are n dependencies, where k columns appear in a dependency on average,

Case	Type	Description	Automatically detectable
1	SD-candidate	It is no longer a CFD/AR	Yes
2	AD	It is no longer a CFD/AR	Yes
3	AD	It was mistakenly rejected	No
4	SD	It was mistakenly accepted	No
5	SD	A more useful SD appeared	Yes
6	SD	Schema change	Yes

TABLE 2. Cases when forgetting is useful

then the system uses $n * (k + 1)$ records to store the dependencies. If we accept m CFDs/ARs, then the number of generated constraints will be m, those constraints need to be stored. We conclude that the space complexity of DependencyManager is relatively low.

3.4. **Forgetting.** After CFDs/ARs are learned, they are stored as SD-candidates. When they are accepted, they will be stored as SDs and removed from the SD-candidates. When SD-candidates are rejected, they will be stored as ADs, but not as SD-candidates. When we completely delete an SD, an AD or an SD-candidate, then we *forget* them and delete it along with its constraint. Table 2. describes the possible cases when an SD-candidate, and AD or an SD should be forgotten. Forgetting is useful to simplify dependency maintenance and it is included into DependencyManager.

4. Experimental results

We have evaluated efficiency and effectiveness of our algorithm on four datasets. In datasets named Numbers1 and Numbers2 the rows were generated numbers, using formulas to guarantee the occurrence of CFDs and ARs. We also conducted experiments used real datasets from the UCI machine learning repository http://archive.ics.uci.edu/ml/, namely, the Wisconsin breast cancer (WBC). The Exceptions dataset was also generated by us using real data.

Dataset	Arity	No. Rows	MOF	MOR	SD	AD
Numbers1	6	20000	200	0.15	95	13
Numbers2	5	200000	650	0.2	118	11
WBC	11	699	37	0.06	242	21
Exceptions	5	11192	1000	0.1	43	8

TABLE 3. Experimental results (MOF = MinOccurrenceFrequency, MOR = MinOccurrenceRate)

Our algorithm doesn't represent a new approach for finding CFDs, therefore there is no point to compare it to the CFD-finders of previous works. Our algorithm represents a new approach to CFD-handling.

Table 3. describes the parameters of the used datasets, the MinOccurrenceFrequency and MinOccurrenceRate used in our experiments and the number of accepted/rejected dependencies: SDs and ADs for each dataset. We use MOF and MOR to set the sensitivity of the system. If we choose low occurrence frequency then it is very sensible even to low numbered occurrences of dependencies than we will store a lot of dependency candidates from which many might be invalid, or if we set the sensitivity to be too high then we might not discover all the valid dependencies. The occurrence rate is the rate specified to be the minimum ratio between the number of records where the dependencies are applicable and the total number of records. We have to choose the sensitivity which potentially enabled us to find all the valid dependencies. Table 3. shows these settings chosen in our cases, but these numbers can vary from database to database or preferences.

If an SD pattern is accepted (validated), then a constraint prevents inserts and updates inconsistent with the accepted pattern. We quantified the amount of data protected by our system against inconsistency violation. In our dataset Numbers1 8620, Numbers2 102034, BCW 384, Exceptions 7083 presents the records complied to at list an SD and an arbitrary insert or update in the future has a probability of 43.1 %, 51.01 %, 54.93 % and 63.28% to be protected by our system against inconsistencies violating our SDs. If an error occurs because of mishandling data where an SD is applicable, then in the error logs the problem can be identified and potentially bugs can be found and fixed.

SD management is simplified by the More Useful relation, showing only the most useful SDs to the system/users, therefore we drastically reduce the number of dependencies to be taken into account and we only store the Most Useful SDs because all the other dependencies are implied by them.. We estimated the benefit of using the MU relation. If a table has n columns and a dependency has a column for the condition, s columns in the determinant column set and d columns in the dependent column set, then there are $2^{(n-s-d-1)}+2^d-1$ possible dependencies which are less useful. If we *accept* the given dependency, we automatically $refuse$ all the less useful dependencies, which optimizes by helping the system/user with the automatic reduction of the dependencies to be taken into account by a maximum of $2^{(n-s-d-1)}+2^d-1$ dependencies. This method helps a lot in the validation and the usage of the dependencies. This is an exponential optimization.

As a result of our experiments we have seen that the constraints were successfully generated and prevented any attempt to breach their CFDs, we have easily identified the SDs which were ready for our FCA module for further analysation.

5. Studying the set of SDs with FCA

We have used FCA (Formal Concept Analysis) methods to analyze SDs and draw conclusions from them. Our approach was agnostic towards the nature of the content of the database; we have implemented an FCA applicable to accepted dependencies of any database using our system.

FCA [10] is a useful algorithm that can be used to draw conceptual conclusions based on a set of data; its steps are as follows: 1. Creating a context which will contain objects and attributes. 2. Creating concepts by grouping objects which have the same set of attributes; objects with different attribute sets will be grouped into different concepts. 3. Each formal context is transformed into a concept lattice, which is the basis for further data analysis.

We have a smallest and a biggest possible value in the lattice and we know that not any concepts C_1, C_2 from C are comparable. We use the **Join** and the **Meet** operations between concepts, knowing that

$$\begin{array}{rcl} (Meet(C_1, C_2) = C_3) & \Leftrightarrow & (((C_1.A \cap C_2.A = \emptyset) \wedge (C_3 = C_{Min})) \vee ((C_1.A \cap C_2.A \neq \emptyset) \\ & \wedge & (C_1.O \cup C_2.O = C_3.O) \wedge (C_1.A \cap C_2.A = C_3.A))), and \\ (Join(C_1, C_2) = C_3) & \Leftrightarrow & (((C_1.O \cap C_2.O = \emptyset) \wedge (C_3 = C_{Max})) \vee ((C_1.O \cap C_2.O \neq \emptyset) \\ & \wedge & (C_1.O \cap C_2.O = C_3.O) \wedge (C_1.A \cup C_2.A = C_3.A))), \end{array}$$

the result of which is a concept from C, because $\emptyset \subseteq C_1.O \subseteq All(O, C)$ and $\emptyset \subseteq C_1.A \subseteq All(A, C)$ for any $C_1 \in C$.

Using FCA we can draw interesting conclusions. In DependencyManager we have implemented an FCA module, which studies the SDs. In this module the objects are the SDs and the attributes are possible properties for the objects. The attributes are Weakness, Determinant Column Cluster's Size, Column Cluster's Size, Frequency of Occurrence and Almost Symmetrical.

Weakness measures the smallness of the dependencies in the more useful relation. The less useful dependency the more weaker it is. We consider D_1 to be weaker than D_2 if D_2 MU D_1. If an SD is fairly weak, then it is normal to assign it lower priority in our researches than the priority assigned to its more useful counterparts.

Definition 1. *(Weakness) The formula of*

$$Weakness(C_1, C_2, CT) = \frac{1}{2} + \frac{C_1 - C_2}{2CT}$$

defines the attribute weakness of concepts C_1 and C_2, where C_1 is the number of determinant columns, C_2 is the number of dependent columns and CT is the total number of columns in the table.

Proposition 1. *The numerical value of Weakness is between 0 and 1.*

We have defined the Fuzzy sets of None, Very Low, Low, Medium, High, Very High and Total for this attribute. Based on this formula we believe that a dependency having weakness of zero can be considered to having no weakness or an other dependency having a value between 0 and 0.2 having a very low weakness rate and so on.

Determinant Column Cluster's Size is the number of SDs having the same set of determinant columns. We have defined the Fuzzy sets of Tiny, Small, Medium, Big and Huge for this attribute. If we have many dependencies having the same determinant column set then the given determinant column set is the source of multiple potential implication schema which is the starting point of many learning algorithm.

Column Cluster's Size is the number of SDs having the same dependent columns and the same determinant columns. This type of attribute shows concepts which have the same schema different only in condition. If we have many such schemas then we have to think about the scheme of the database because we might learn FDs with dirty data. We have defined the Fuzzy sets of Tiny, Small, Medium, Big and Huge for this attribute. We consider cluster size of 1 being tiny, cluster size of 6 being big.

Frequency of Occurrence is the percentage of occurrence of an FD pattern. This number is the sum of the number of all records matching any SDs with the same FD-pattern. For instance, if a CFD is fulfilling $R.A \rightarrow R.B$, where $R.A$ and $R.B$ are column sets from the R table with the condition of $T.C = somevalue$, then the CFD is matching the FD pattern of $R.A \rightarrow R.B$. Of course it is not an FD because of the condition, but still it matches an FD-pattern. We have defined the Fuzzy sets of Nonexistent, Very Rare, Rare, Medium, Frequent, Very Frequent and FD.

If D_1 and D_2 are SDs such as $DependentCols_{D_1} = DeterminantCols_{D_2}$ and $DependentCols_{D_2} = DeterminantCols_{D_1}$, then D_1 is **Almost Symmetrical** with D_2. We have used Fuzzy values for these attributes, Table 4. shows the quantitative metrics concerning our point of view of the meaning of these values. Example 3. illustrates the power of the combination of SDs and FCA.

Example 3. *Let's consider a table R, which holds many records, with a lot of white noise due to dirty data. Also, let's consider that there is a pattern of R.A → R.B where R.A and R.B are sets of columns. Because of the white noise there is no tool able to find the FD of R.A → R.B. However, Column Cluster's Size of CFD's matching the pattern of R.A → R.B is Big, all CFDs matching the pattern are not almost symmetrical, their weakness is the same, the Determinant Column Cluster's Size of the pattern is Big, the Frequency of Occurrence of all CFDs matching the pattern is either frequent or very*

W	None	Very Low	Low	Medium	High	Very High	Total
	0	<0.2	<0.4	<0.6	<0.8	<1	1
D	Tiny	Small	Medium	Big	Huge		
	1	<= 3	<= 5	<=7	>7		
C	Tiny	Small	Medium	Big	Huge		
	1	<= 3	<= 5	<=7	>7		
O	Nonexistent	Very Rare	Rare	Medium	Frequent	Very Frequent	FD
	0%	<0.15%	<0.4%	<0.6%	<0.75%	<1%	1%
A	Yes	No					

Table 4. Quantitative metrics used for Fuzzy Attribute sets(W=Weakness, D=Determinant Col. Clusters Size, C= Col. Clusters Size, O=Occurrence, A=Almost Symmetrical)

frequent (because of the table has many records and many dirty data). Because of these facts all the SDs meeting the FD pattern of $R.A \rightarrow R.B$ will be objects of $Concept_1$. If a user studies the results of the FCA algorithm, he will find many objects with the FD pattern of $R.A \rightarrow R.B$ in $Concept_1$. He will study the records of the table and will discover that the FD pattern is generating so many CFDs because in fact it is an FD obfuscated by dirty data, so he will know a useful information for certain which is a great starting point of fixing inconsistency problems in this case. Thanks to the analysis provided by FCA the user got a clue which has led him to the important conclusion that the FD pattern of $R.A \rightarrow R.B$ is essentially an FD instead of many CFDs.

	Determinant	Weakness	Fd	Occurrence	Symmetrical	Universal
ID:7	small	low	true	rare	false	tiny
ID:8	small	medium	false	medium	false	tiny
ID:3	medium	very low	false	rare	false	tiny
ID:6	medium	none	false	very rare	false	small
ID:5	medium	none	false	very rare	false	small
ID:2	medium	none	false	very rare	false	small
ID:4	medium	none	false	very rare	false	tiny

Table 5. FCA Context of SDs

Example 4. *We have created the FCA context, shown in Table 5, by pairing the fuzzy and boolean attributes to objects which are SDs in our Database about Exceptions. The IDs in the table represents the ID of an SD, see Table 3.1. From the context we built the concepts and generate the lattice from*

Level 3:
C_{68} = ({6, 5, 2, 4}, {W:None, D:Medium, O:Very rare});
C_{69} = ({4}, {W:None, D:Medium, U:Tiny});
C_{70} =({6, 5, 2}, {W:None, D:Medium, U:Small});
C_{91} = ({4}, {W:None, O:Very rare, U:Tiny});
C_{154} = ({3}, {W:Very low, D:Medium, O:Rare});
C_{155} = ({3}, {W:Very low, D:Medium, U:Tiny});
C_{173} = ({3}, {W:Very low, O:Rare, U:Tiny});
...
Level 4:
C_{344} =({4}, {W:None, D:Medium, O:Very rare, U:Tiny});
C_{345} = ({6, 5, 2}, {W:None, D:Medium, O:Very rare, U:Small});
C_{441} = ({3}, {W:Very low, D:Medium, O:Rare, U:Tiny});
C_{484} = ({7}, {W:Low, D:Small, O:Rare, U:Tiny});
C_{515} = ({8}, {W:Medium, D:Small, O:Medium, U:Tiny})

TABLE 6. Part of the generated concepts

which we can read useful conclusions. Some of the generated concepts can be found in Table 6. For example at level 3 we have the concept (C_{70}, Objects = {6, 5, 2}, Attributes = {Weakness:None, Determinant:Medium, Universal:Small}); and from this concept we can read that maybe we should think more about this dependencies because the determinant cluster is medium and is paired with virtually no weakness and this might lead to useful conclusion about our exception. At level 4 we can see another concept containing the same object (C_{345}, Objects = {6, 5, 2}, Attributes = {Weakness:None, Determinant:Medium, Occurrence:Very rare, Universal:Small}); and by the extra information of the occurrence being very rare we can be assured that this might not be the highest priority for further analysis because the occurrence rate discouraged us from focusing on this three objects otherwise if we were not using FCA then we might have been allocating a lot of time to analyze this FDs. The human mind can comprehend object-attribute pairs but in the lattice generated with our FCA the higher the level is the more difficult is for the human mind to comprehend it without FCA analysis. If a human observes that the size of the determinant cluster is big and sometimes even the dependent column cluster size is raised. If this is paired with virtually no weakness then we might allocate a lot of time to get more information about this object set = {6,5,2}, however at the fourth level we can see that they are very rare which lowers the priority of further investigations.

As we can see, FCA is relevant in this study, as it provides conceptual information as the result of SD-analysis with the attributes described in Section 5. To our knowledge no previous work contained FCA implementation where the input objects were SDs and the input attributes were properties of these dependencies.

6. Conclusions and Future Works

In this paper we have focused on inconsistency prevention instead of fixing them. Our method prevents inconsistencies by learning CFD and AR patterns. If such a pattern is accepted (validated), then a constraint prevents inserts and updates inconsistent with the accepted pattern. DependencyManager has a novel strategy for consistency protection with the additional benefit of preventing inconsistencies before they appear in the database, is scalable, easy to use and powerful in preventing inconsistencies. We have used FCA to analyze the SDs and draw useful conclusions; this way the users of the database can understand the db-schema deeper. We have introduced a novelty for FCA analysis by using FCA for dependency; analyzing abstract database patterns with it.

In the future we intend to make this strategy even more useful, we intend to find and analyze cross-table SD-candidates. We also want to generalize the set of conditions by using more columns in the boolean functions instead of only one and using more operator types, (our current implementation considers equality as the only conditional operator). Automatization of the validation process would be useful, this feature will probably be based on user-defined rules which will help the learner module to automatically determine whether an SD-candidate is an AD or an SD. The system stores the errors originating from unsuccessful inserts and updates which were prevented by the constraints created for SDs; this will generate knowledge which can be used to determine the cause of failure of inserts and updates. If the cause of failure is an incorrectly accepted pattern, the pattern should be dismissed; otherwise the incorrect user action or incorrect functionality can be detected.

Known SDs can be used to increase application performances. This can easily be understood by knowing that these rules are enforced by constraints, so it is enough to load all possible combinations for determinant column sets along with their corresponding values in the dependent column set where the condition of the SD is met. In this way it will be possible to determine the values of the dependent columns of other records. If there are too many possible combinations, the knowledge base can still be filled with only the most frequent or most recent combinations. This method for caching SDs can enhance performance, especially in applications where data is periodically loaded, such as applications responsible for the creation of backups.

7. Acknowledgment

We would like to thank Arpad Lajos for his advice and help during this research.

References

[1] J. Baixeries. A formal context for symmetric dependencies. In R. Medina and S. A. Obiedkov, editors, *ICFCA*, volume 4933 of *LNCS*, pages 90–105. Springer, 2008.

[2] G. Beskales, I. F. Ilyas, and L. Golab. Sampling the repairs of functional dependency violations under hard constraints. *PVLDB*, 3(1):197–207, 2010.

[3] F. Chiang and R. J. Miller. Discovering data quality rules. *PVLDB*, 1(1):1166–1177, 2008.

[4] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. *VLDB*, pages 315–326. ACM, 2007.

[5] G. Cormode, L. Golab, F. Korn, A. McGregor, D. Srivastava, and X. Zhang. Estimating the confidence of conditional functional dependencies. *SIGMOD Conference*, pages 469–482. ACM, 2009.

[6] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.*, 33(2), 2008a.

[7] W. Fan, F. Geerts, L. V. S. Lakshmanan, and M. Xiong. Discovering conditional functional dependencies. In *ICDE*, pages 1231–1234. IEEE, 2009.

[8] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *PVLDB*, 3(1):173–184, 2010.

[9] W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *IEEE Trans. Knowl. Data Eng.*, 23(5):683–698, 2011.

[10] B. Ganter and R. Wille. *Formal concept analysis - mathematical foundations*. Springer, 1999.

[11] L. Golab, H. J. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB*, 1(1):376–390, 2008.

[12] K. T. Janosi-Rancz, V. Varga and T. Nagy Detecting XML Functional Dependencies through Formal Concept Analysis, 14th ADBIS Conference, LNCS 6295, pp 595–598, 2010

[13] S. Kolahi and L. V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. *ICDT*, vol. 361 of *ACM ICPS*, pages 53–62, 2009.

[14] S. Lopes, J.-M. Petit, and L. Lakhal. Functional and approximate dependency mining: database and fca points of view. *J. Exp. Theor. Artif. Intell.*, 14(2-3):93–114, 2002.

[15] R. Medina and L. Nourine. Conditional functional dependencies: An fca point of view. In L. Kwuida and B. Sertkaya, editors, *ICFCA*, volume 5986 of *LNCS*, pages 161–176. Springer, 2010.

[16] R. Medina and L. Nourine. A unified hierarchy for functional dependencies, conditional functional dependencies and association rules. In *ICFCA*, pages 98–113, 2009.

[17] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, 4(5):279–289, 2011.

Sapientia Hungarian University of Transylvania, Tirgu Mures, Romania
E-mail address: tsuto@ms.sapientia.ro