UMONS
Université de Mons

Faculté des Sciences

Mémoire

Data Repairing

Project made by : Maxime Van Herzeele
Academic Year : 2017-2018
dissertation director : Jeff Wijsen
Section : 2^{nd} Master Bloc in ComputerSciencesSciences

Faculté des Sciences • University of Mons • Place du Parc 20 • B-7000 Mons

Table des matières

1 Introduction **1**

2 Integrity constraints **2**
2.1 Database 2
2.2 Constraint on database 4
2.3 Integrity constraints variations 6

3 Data Repairing **7**
3.1 θ-tolerant model 9
3.2 Others 9
3.2.1 Holistic 9
3.2.2 9

4 Implementation and comparison with others models **10**

5 Conclusion **11**

Chapitre 1

Introduction

Many institutions and companies collect, store and use a lot of data. These data could be dirty which means they contain erroneous information. An Erroneous information may mislead anyone who want to use the database. To prevent this problem, data should respect integrity constraints which are rules in database. Any information who doesn't fit these constraints is considered as a dirty data. But these constraints may be imprecise as well, failing to identify good data and dirty data. For this reason, some data aren't identify as violation as they should be and others data are identify as violent and they shouldn't be. Both mistakes on data and integrity constraints are a problem for anyone using the database.

During my training, i had to work on a project related to a database with such problems. It has a huge impact for a part of my project. The repair of these data is a project for 2018 and i feel it would be interesting to study the data repairing concept.

Data repairing means recover erroneous data but also repair bad integrity constraints. It would be naive to think we can delete dirty data as we wish. The number of loss would be huge because sometimes there is only one error in a row. Furthermore, integrity constraint can also be dirty which means row deletion could erase some clean data. For this reason we need techniques to repair a data without loosing too much information and without failling to identifie dirty data.

In this thesis we are going to analyse the *θ-tolerant repair model* as explained in a scientifique article[3]. First of all, we have to re-explain some notions and definition to understand the θ-tolerant model and we also need to present some database we used to illustrate data repairing models. Next we'll present some data repairing models with among them the θ-model. We'll theorically compare them and identify pros and cons for all them.
TO CONTINUE.

Chapitre 2

Integrity constraints

In this chapter we'll remind some important notions that we are going to use to explain some data repairing models. We use database following the relational model which was introduced by E.F. Codd [2]. We'll also present some database we're going to use to illustrate different notions.

2.1 Database

In this section we'll present databasse we used as example in this thesis. These databases are used to ulistrate data repairing models and others notions we'll define.

The first database comes from the main article used as bibliography in this thesis[3].

	Name	BirthDay	Cellphone Number	Year	Income	Tax
t1	Ayres	8-8-1984	322-573	2007	21k	0
t2	Ayres	5-1-1960	***-389	2007	22k	0
t3	Ayres	5-1-1960	564-389	2007	22k	0
t4	Stanley	13-8-1987	868-701	2007	23k	3k
t5	Stanley	31-7-1983	***-198	2007	24k	0
t6	Stanley	31-7-1983	930-198	2008	24k	0
t7	Dustin	2-12-1985	179-924	2008	25k	0
t8	Dustin	5-9-1980	***-870	2008	100k	21k
t9	Dustin	5-9-1980	824-870	2009	100k	21k
t10	Dustin	9-4-1984	387-215	2009	150k	40k

TABLE 2.1 – Table of the main article[3]

.

The second database we are going to use comes from a personal experience. In a training, i had to work on a project related to a database with some dirty data. These data can't be used outside the company but we'll try to get the main idea. It's a table named person, who got several basic information on people from Belgium [1].

- **NISS :** The national number of the person. A national number is unique. Usually, a NISS is formated like this[1]
  - It start with the birthdate of the person in a YY-MM-DD format. Exception are made for stranger(People without Belgian nationality), but for ease we won't consider these cases.
  - Number 7 to 9 is even for men and odd for female.
  - Remaining number are the modulo 97 of the 9 first number.
- **LN :** Person's lastname.
- **FN :** Person's firstname.
- **Birth_Date :** birthDate in DD-MM-YYYY format.
- **Decease_Date :** Person's date decease.
- **Civil_State :** Person's current civil state(example : single, married, divorced, decease, widow,...)

	Niss	LM	FN	Birth_date	Decease_date	civil_state	district	post code
t1	14050250845	Dupont	Jean	14-05-1902	18-05-1962	decease	Hainaut	7822
t2	08042910402	Brel	Jacques	08-04-1929	09-10-1978	decease	Schaerbeek	1030
t3	08042910402	Merckx	Eddy	07-06-1945	null	decease	Schaerbeek	1030

TABLE 2.2 – Table Person

.

1. Every data in our database are fictional person.

2.2 Constraint on database

When you want to add rows in a database, you can't put what you want. It would be a problem if it was possible to add non-logical value on some columns of a table. To avoid this kind of problems, we can add rules on a database. Basically a rule works on this way : if the entry row t_α respects some conditions, we can accept the value. Otherwise t_α is not correct and something is wrong with the value of this entry row.

On the relational database model which is used in most of the database, the notion of *functional dependency(FD)* is used.

Definition 1. *Given a relation R and a set of attributes $X \in R$, a **functional dependency** determine another set $Y \in R$ (written $X \to Y$) if and only if each X value is associated with one Y value.*

In other words, for a dependency $X \to Y$ means that for a specific value X there's only one possible value for Y. If the DF is respected on a relation R, we say that R satisfy the DF. Let's take some examples on the table 2.2.

1. A NISS identify a person. This constraint can be describe by a key constraint $KeyNiss$
2. Two persons with the same post code lived in the same district. The functional dependency for this constraint is $post\_code \to district$
3. If some got a decease date, his civil status should be equal to decease. In this case we need a conditional functional dependency(CFD) which is typically a functional dependency with equality operator on some columns. A functional dependency should work for all records on the table, CFD can hold some conditions on collumns. $[Decease\_date =$'18-05-1962'$] \to [civil\_status = decease]$

Definition 2. *A set Σ of DF on the relation schema A. The relation R satisfy Σ noted $R \models \Sigma$ if for each DF in Σ, R satisfy the DF.*

However we can't express everything as functional dependencies. Sometimes it fails. For example, if we want to express "Nobody can die before his own birth", in this case we need to compare the birth date and the decease date. Functional dependencies can't work in this case. We can use the first-order logic.

$$\forall t_\alpha \in R,\ \neg(t_\alpha.decease\_date \leq t_\alpha.birth\_date)$$

We can express the others examples in first-order logic.

1. A NISS identify a person. $\forall t_\alpha, t_\beta \in R\ \neg(t_\alpha.NISS = t_\beta.NISS)$ (We suppose in this notation, we can't take $t_\alpha = t_\beta$)
2. Two persons with the same post code lived in the same district. $\forall t_\alpha, t_\beta \in R\ \neg(t_\alpha.post\_code = t_\beta.post\_code \wedge t_\alpha.district \neq t_\beta.district)$

3. If some got a decease date, his civil status should be equal to decease. $forall t_\alpha, t_\beta \in R$ $\neg(decease\_date = null \wedge civil\_state = decease)$

On the main paper which this thesis is based, they define a denial constraint as [3] :

Definition 3. *Consider a relation scheme R with attributes $attr(R)$. Let predicate space $\mathbb{P}$ be a set of predicate P in the form $v_1 \phi V_2$ or $v1\phi c$ with $v_1, v_2 \in t_x.A$, $x \in \{\alpha, \beta\}$, $t_\alpha, t_\beta \in R$, $A \in attr(R)$, c is a constant and $\phi \in \{=, <, >, \leq, \geq, \neq\}$ is a build-in operator. A* ***Denial Constraint(DC)*** *:*

$$\varphi : t_\alpha, t_\beta, ... \in R, \neg(P_1 \wedge ... \wedge P_m)$$

states that for any tuples $t_\alpha, t_\beta, ...$ from R, all the predicates $P_i \in pred(\varphi)$,i = 1,...,m should not be true at the same time.

In other words, a denial constraint is a first-order logic conjunction of predicates that shouldn't be true all in the same time. So if one of the predicates is false, the data is consider to be clean.

A DC can be oversimplified which means a correct data could consider as a violation. Let's take an example on the table 2.1. If we take the following constraint :

$$\varphi : t_\alpha, t_\beta \in R, \neg(t_\alpha.Name = t_\beta.Name \wedge t_\alpha.CP \neq t_\beta.CP)$$

This constraint means that any person with the same *Name* should get the same cellphone number (*CP*), which is incorrect because two person different person can get the same name and of course different cellphone number. For example t_1 and t_2 don't respect this denial constraint. So it's considered as a violation, but we can see it's two different person because they don't have the same age. If we want a correct DC we need to check their age, i.e their *Birthdate* :

$$\begin{gathered}\varphi : t_\alpha, t_\beta \in R\ \neg(t_\alpha.Name = t_\beta.Name \wedge t_\alpha.CP \neq t_\beta.CP \wedge \\ t_\alpha.Birthday = t_\beta.Birthday)\end{gathered}$$

In the opposite of oversimplified, a DC can be overrefined which means a dirty data could be consider as a clean data. An example could be :

$$\begin{aligned}\varphi : t_\alpha, t_\beta \in R\ \neg(&t_\alpha.Name = t_\beta.Name \wedge t_\alpha.CP \neq t_\beta.CP \wedge \\ &t_\alpha.Birthday = t_\beta.Birthday \wedge t_\alpha.Year = t_\beta.Year)\end{aligned}$$

In this case the Year information is not usefull. We don't recognize t_5 and t_8 as a violation with this constraint.

2.3 Integrity constraints variations

We saw earlier that a constraint can be overrefined failing to detect some error or in the opposite a constraint can be oversimplified leading to consider some good data as a error. Because constraints can be inaccurate we need to modify them. We'll considerer two type of constraint variance : predicate deletion and in the opposite predicate insertion.

When we perform a predicate insertion, some tuples no longer violate the DC. With this variation we can repair a oversimplified constraint but we need to be careful otherwise the DC can be useless. We need to avoid insertion which can lead to a trivial DC or insertion of predicates with constants.

An example of trivial DC is a DC φ with a predicate $P_i : x\phi_i y$ and we had another predicate $P_j : x\phi_j y$ in the DC with $\overline{\phi_j} \in Imp(\phi_i)$. In [3], they said for $Imp(\phi)$: For any two values a and b, if $a\phi_2 b$ always implies $a\phi_2 b$, it means $\phi_2 \in Imp(\phi_1)$. With this definition, we see that if P_i is true, P_j is false and vice versa.

For example if we have $x = y$ as predicate and we add $x < y$, both the predicates can't be true at the same time. Table 2.3

ϕ	=	$\neq$	>	<	$\leq$	$\geq$
$\overline{\phi}$	$\neq$	=	$\leq$	$\geq$	<	>
$Imp(\phi)$	= $\geq$,$\leq$	$\neq$	>,$\geq$,$\neq$	<,$\leq$,$\neq$	$\geq$	$\leq$

TABLE 2.3 –

.

Chapitre 3

Data Repairing

Errors are frequent in database. Because these anomalies can make applications unreliable, some methods detect them but don't repair the detected anomalies. But if you simple filter the dirty data you've detected, applications could still be unreliable. [4] Instead of only detecting errors and delete them, it's better to repair the dirty data.

The goal of data repairing is to find a modification I' for I,an instance of R, in which all of violation in the constraints Σ are eliminated. In other words, we want $I' \models Sigma$ (I' satisfy Σ). Data repairing process follows the minimum change principle : the data repair I' have to minimize the data repair cost define as [3] :

Definition 4. *If I' is a repair for I instance of R by modifying attribute values without any deletion or assertion tuples, the data repair cost is :*

$$\Delta(I, I') = \sum_{t \in I, A \in attr(R)} w(t.A).dist(I(t.A), I'(t.A)$$

where :

— $dist(I(t.A), I'(t.A))$ *is the distance between two values on cell* $t.A$ *in* I *and* I'.

— $w(t.A)$ *is the weight of cell* $t.A$.

We can see that the cost can be the number of cell we changed if we put :

$$dist(I(t.A), I'(t.A)) = \begin{cases} 1 \; if I(t.A) \neq I'(t.A) \; (the\ value\ changed) \\ 0 \; otherwhise \; (no\ changes\ were\ made) \end{cases}$$

We can put the distance for numerical values on the difference of the two values. For string values we can use the edit distance.

The weight $w(t.A)$ can show the trust of the original value in cell which is subjective or simply be a constant if we don't have a lot of knowledge about the data.

It's important to notice it's not impossible to don't find any repair I' that can eliminates all the violations. It's possible any values in $dom(A)$ can fit the constraint. In that case, we can use a *fresh variable (fv)* out of the current domain $dom(A)$ in order to extend the domain. This fresh variable is a value that does not satisfy any predicate, we are sure that we can satisfy the DC. (it's satisfy if at least one of the predicates is false).

Let's take an example on the table 2.1. Let's say our Denial Constraint is the following one :

$$\varphi : t_\alpha, t_\beta \in R, \neg(t_\alpha.Income > t_\beta.Income \wedge t_\alpha.tax \leq t_\beta.CP)$$

In other words, we supposed that if someone get a higher income than another person then he should paid an higher tax every year. We have $\langle t_1, t_2 \rangle \not\models \varphi$ because t_1.Income < $t2$.Income and $t2$.Tax $\leq$ $t1$.Tax. Same problem with $\langle t_1, t_3 \rangle$, $\langle t_1, t_5 \rangle$, ect... A repair I' could be the following one :

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_18	t_9	t_10
Tax	0	fv_1	fv_2	3k	fv_3	fv_4	fv_5	21k	21k	40k

TABLE 3.1 – Example of repair with Tax

.

The reason we put fv_1 as Tax value for t_2 is because we knew the following things :

1. $I(t_1.Tax) = 0$ so $I(t_2.Tax) > 0$ because $I(t_1.Income) < I(t_2.Income)$
2. $I(t_3.Tax) = 3$ so $I(t_2.Tax) < 3$ because $I(t_2.Income) < I(t_3.Income)$
3. $dom(Tax) = \{0, 3k, 21k, 40k\}$

Because we had no value in the $dom(Tax)$ that would respect 1 and 2, we need to use a fresh variable fv_1 out of the $dom(Tax)$.

We can compute the repair cost for Tax in this table. Let's say that :

$$\forall a, b \in dom(A)\ with\ a \neq b. \begin{cases} dist(a,a) = 0 \\ dist(a,b) = 1 \\ dist(a,fv) = 1.5 \end{cases}$$

When we don't change anything, the distance is obviously equal to zero. $dist(a, fv)$ have to be higher than $dist(a, b)$ otherwise the cost for a non-domain value will be lower than a domain value and we want to avoid fresh variable as much as possible. In our example, with the value said just before, we can compute a $\Delta(I, I')$ = 7.5

3.1 θ-tolerant model

3.2 Others

3.2.1 Holistic

3.2.2 ...

Chapitre 4

Implementation and comparison with others models

Chapitre 5

Conclusion

Bibliographie

[1] Description des données du registre national et du registre bcss. https://www.ksz-bcss.fgov.be/sites/default/files/assets/services_et_support/cbss_manual_fr.pdf. accessed : 2018-02-15.

[2] ics relational database model. http://databasemanagement.wikia.com/wiki/Relational_Database_Model. Accessed : 2018-02-13.

[3] Shaoxu Song, Han Zhu, and Jianmin Wang. Constraint-variance tolerant data repairing. Technical report, Tsinghua National Laboratory of Information Science and Technology, 2016.

[4] Aoqian Zhang, Shaoxu Song, Jianmin Wang, and Philip S. Yu. Time series data cleaning : From anomaly detection to anomaly repairing. Technical report, Tsinghua National Laboratory of Information Science and Technology, 2017.