A Software Tool to Transform Relational Databases in Order to Mine Functional Dependencies in it Using Formal Concept Analysis

Viorica Varga[1] and Katalin Tünde Jánosi Rancz[2]

[1] Babeş-Bolyai University, Department of Computer Science, Cluj-Napoca, Romania
[2] Sapientia University, Department of Mathematics and Informatics, Târgu-Mures, Romania

Abstract. The theory of Formal Concept Analysis offers an algebraic approach to data analysis and knowledge processing. The notion of dependencies between attributes in a many-valued context has been introduced in [3], by Ganter and Wille. J. Hereth (2002) introduces the power context family resulting from the canonical translation of a relational database. Regarding to this power context family, he defines the formal context of functional dependencies. In this context, implications hold for functional dependencies. We propose a software tool, which constructs the formal context of functional dependencies, and it builds the concept lattice and determines the implications in the context, which syntactically are the same as functional dependencies in the analyzed table. The software can be used in relational database design and for detecting functional dependencies in existing tables, respectively.

1 Introduction

The goal of relational database design is to generate a set of relation schemas that allows us to store information without unnecessary redundancy. Conceptual database design gives us a set of relation schemas and integrity constraints that can be regarded as a good starting point for the final database design. In order to ensure the integrity constraints more fully than in the case of Entity Relationship model, the initial design must be refined. Functional dependencies form an important class of integrity constraints. The relation scheme obtained by translating the Entity-Relationship model is a good starting point, but we still need to develop new techniques to detect possible redundancies in the preliminary relation scheme. The normal form satisfied by a relation is a measure of the redundancy in the relation. In order to analyze the normal form of a relation we need to detect the functional dependencies that are present in the relation.

The theory of Formal Concept Analysis offers an algebraic approach to data analysis and knowledge processing. The notion of dependencies between attributes in a many-valued context has been introduced in [3], by Ganter and Wille. J. Hereth (2002) investigates how some basic concepts from database

theory translate into the language of Formal Concept Analysis. He introduces the power context family resulting from the canonical translation of a relational database. Regarding this power context family, he defines the formal context of functional dependencies. In this context, implications hold for functional dependencies. Priss (2005) presents the visualization of normal forms using concept lattices, where the notion of functional dependencies is life-line. Baixeries (2004) gives an interesting framework to mine functional dependencies using Formal Context Analysis. Detection of functional dependencies seems to be an actual theme and a challenging problem, see [8].

This paper proposes a software named FCAFuncDepMine, which constructs the formal context of functional dependencies, uses Conexp to build the concept lattice and to determine the implications in this context, which are syntactically the same as functional dependencies in the analyzed table. The software can be used in relational database design and for detecting functional dependencies in existing tables, respectively.

In the next section we introduce the necessary theoretical notions for the translation from a relational database to a power context family and the formal context of functional dependencies using [4], [1].

2 Preliminaries

In relational databases the data is stored in data tables. The structure of a relational data table can be given in unnamed or in named perspective. In the case of the named perspective the columns of the table which are called attributes are given by names, while in unnamed perspective they are given only by position.

Let $\mathbf{D}$ be a set, called the domain of the database, that may include the set of integers, strings, and Boolean values. A data table of the relational database with i attributes is a subset of $\underbrace{\mathbf{D} \times \ldots \times \mathbf{D}}_{i-times}$, it can be seen as a set of i-tuples, beeing in fact the rows of the table. i is called the arity of the table. The following definition is very basic, e.g. we do not address the question of different value domains for different attributes, we use the unnamed perspective.

Definition 1. *We define a relational database to be a tuple* $\mathcal{D} := (\mathbf{D}, \mathcal{T})$ *with* $\mathbf{D}$ *being the domain of the database and* $\mathcal{T}$ *being the set of data tables in the database. A data table is any element* $T \in \cup_{i \in \mathbb{N}_0} \mathcal{P}\left(\mathbf{D}^i\right)$*. The arity of* T *is* $i \in \mathbb{N}_0$ *such that* $T \in \mathcal{P}\left(\mathbf{D}^i\right)$ *and is denoted by* $arity(T)$*.*

The operations on the relational model are based on algebra or on logic. The data integrity constraints of the relational model appear as functional dependencies. In relational database design the normalization theory is used to avoid redundancy. Normal forms use the notion of functional dependencies. The following definition uses the projection relational operator, see any database theory book [1], [7].

Definition 2. *Let T be a data table and $X, Y \subseteq \mathbb{N}_0$ its columns given by position. Then T fulfills the functional dependency $X \to Y$, if for all tuples $s, t \in T$, condition $\pi_X(s) = \pi_X(t)$ implies that also $\pi_Y(s) = \pi_Y(t)$.*

In order to define the functional dependency in a formal context, we need the notion of Power Context Family.

Definition 3. *A power context family $\overrightarrow{\mathbb{K}} := (\mathbb{K}_n)_{n \in \mathbb{N}_0}$ is a family of formal contexts $\mathbb{K}_k := (G_k, M_k, I_k)$ such that $G_k \subseteq (G_0)^k$ for $k = 1, 2, ...$ The formal contexts $\mathbb{K}_k$ with $k \geq 1$ are called relational contexts. The power context family $\overrightarrow{\mathbb{K}}$ is said to be limited of type $n \in \mathbb{N}_0$ if $\overrightarrow{\mathbb{K}} = (\mathbb{K}_0, \mathbb{K}_1, ..., \mathbb{K}_n)$, otherwise, it is called unlimited.*

The following definition from [4] gives the method to construct the power context family resulting from a relational database.

Definition 4. *The power context family $\overrightarrow{\mathbb{K}}(\mathcal{D})$ resulting from the canonical database translation of the relational database $\mathcal{D} := (\mathbf{D}, \mathcal{T})$ is constructed in the following way: we set $\mathbb{K}_0 := (\mathbf{D}, \emptyset, \emptyset)$ and, for $k \geq 1$, let G_k be the set of all k-ary tuples and $M_k \subseteq \mathcal{T}$ be the set of all data tables of arity k. The relation I_k is defined by $(g, m) \in I_k :\Leftrightarrow g \in m$.*

The formal context of functional dependencies is defined in the following way in [4].

Definition 5. *Let $\overrightarrow{\mathbb{K}}(\mathcal{D})$ be a power context family, and let $m \in M_k$ be an attribute of the kth context. Then the formal context of functional dependencies of m with regard to $\overrightarrow{\mathbb{K}}(\mathcal{D})$ is defined as*

$$FD\left(m, \overrightarrow{\mathbb{K}}(\mathcal{D})\right) := \left(m^{I_k} \times m^{I_k}, \{\ 1, 2, ..., k\}, J\right)$$

with $((g, h), i) \in J :\Leftrightarrow \pi_i(g) = \pi_i(h)$ with $g, h \in m^{I_k}$ and $i \in \{\ 1, 2, ..., k\}$.

In order to mine functional dependencies in the context defined in definition 5, we need the following proposition (see [4]).

Proposition 1. *Let $\mathcal{D}$ be a relational database and m a k-ary table in $\mathcal{D}$. For two sets $X, Y \subseteq \{\ 1, ..., k\}$ we have the following assertion: The columns Y are functionally dependent from the columns X if and only if $X \to Y$ is an implication in $FD\left(m, \overrightarrow{K}(\mathcal{D})\right)$.*

More detailed explanation and examples can be read in [5].

3 Software proposal

We propose a software named FCAFuncDepMine to mine functional dependencies in a relational table. It can be used for

– relational database design;
– mining functional dependencies in an existing table.

The software constructs the context of functional dependencies of a table. Regarding Definition 5 the columns (named attributes) of the table will be the attributes of the context, the context's objects are the tuple pairs of the table. The top of the concept lattice corresponds to tuple pairs in which there are no common values of the corresponding table attributes. Pairs of form (t, t), where t is a tuple of the table, have all attributes in common, these objects will arrive in the bottom of the lattice. In order to detect functional dependencies, we use the software Conexp [9], to build the concept lattice from the constructed context and to generate the implication base.

3.1 Table design and significant tuple introduction

In case of first approach - when the user introduces the structure of the table to be designed and provides some significant tuples - the input file *.cex interpretable by Conexp is generated as follows:

Let T be a table with columns corresponding to attributes: $a_1, a_2, ..., a_n$. We insert rows in the table, which will be stored in an xml file, then we build the formal context of functional dependencies to find existing functional dependencies as implications in the constructed table. In order to optimize the construction of the formal context of functional dependencies, we build inverted index files for every attribute.

We use the following notations for the jth inverted file, which contains the different values of the attribute a_j: v_{1j}, v_{2j}, ..., v_{mj} and the row numbers associated to each value where the corresponding attribute value appears, see Table 1.

Value	Row numbers
v_{1j}	rnr^1_{1j}, rnr^2_{1j}, ...
v_{2j}	rnr^1_{2j}, rnr^2_{2j}, ...
...	
v_{mj}	rnr^1_{mj}, rnr^2_{mj}, ...

Table 1. Inverted index InvInd_j

By using the previous considerations and parallel generation of inverted index files, Algorithm 1 builds the context of functional dependencies. More examples and explanations about the method can be found in [5].

Algorithm 1
for each inserted row in table T **do**
begin
let k be the number of row
let $e_{k1}, e_{k2}, ..., e_{kn}$ be the attribute values of row k
for j:=1 to n **do** // for every attribute value
begin
search e_{kj} in the j-th inverted index file //search the attribute value
// in the corresponding inverted file
if find, let v_{lj} be the value in the inverted file
such that $e_{kj} = v_{lj}$ **then**
begin // k is added to the list of row numbers for value v_{lj}
build the array list $al_{kj} = \{rnr^1_{lj}, rnr^2_{lj}, ...\}$
add k in al_{kj}
add k in the j-th inverted index in the list of row numbers
for value v_{lj}
end
else //value e_{kj} doesn't exist in the corresponding inverted index
//we insert it, k is the first row with value e_{kj} of attribute j
insert new line in the j-th inverted index file with values (e_{kj}, k)
end
// In order to insert tuple pairs as rows in the cex file:
build $al_k = \bigcup_{j=1}^{n} al_{kj}$ //al_k contains the row numbers,
// which have attributes in common with row k
//if al_k is empty, no row will be inserted in cex file
if $al_k \neq \emptyset$ **then**
for $s = 1$ to count(al_k) **do**
insert in cex file tuple (k, $al_k(s)$)
end

3.2 Mining functional dependencies in existing databases

In real world applications many databases are already designed. The developer may realize that the database is not well designed and tends to analyze it. The project manager can verify if the database designer works correctly. The aim of our software tool is to connect to an existing database by giving the type and the name of the database, a login name and password, then the software offers a list of identified table names that can be selected for possible functional dependencies examination. In the actual version we can connect to Oracle, MySQL and MS SQL Server.

Let T be this table having attributes $A1, \ldots, An$. The top of the concept lattice corresponds to tuple pairs in which there are no common values of the corresponding attributes. A lot of pairs of this kind may be present. Pairs which have all attributes in common, will arrive in the bottom of the lattice.

We test concept lattices omitting tuple pairs in the top and the bottom of the lattice. During this test we do not find the same lattice as that obtained with these special tuple pairs. In order not to alter the implications, we generate only a few (but not all) of these pairs. On the other hand, we need pairs of tuples of table T, where at least one (but not all) of the attributes has the same value.

The connection being established and table T selected to analyze the existing functional dependencies, the program has to execute the next SELECT - SQL statement:

```
SELECT T1.A1,...,T1.An,T2.A1,...,T2.An
FROM T T1, T T2
WHERE (T1.A1=T2.A1 OR ... OR T1.An=T2.An)
  AND (T1.A1<>T2.A1 OR ... OR T1.An<>T2.An)
```

Both (s, u) and (u, s) pairs of tuples will appear in the result, but we need only one of these.

If the table's primary key $P1, P2, ..., Pk (k \geq 1)$ is declared (see [7] for the definition of primary key), then in order to include only one of these pairs, we complete the statement's WHERE condition in case of $k = 1$ with:

```
AND (T1.P1 < T2.P1)
```

or if $k > 1$ with

```
AND (T1.P1k < T2.P1k)
```

where $P1k$ denotes the string concatenation of the primary key's component attributes, respectively.

An existing table may have a very large number of tuples. In this version of our software we use Conexp, which is not able to handle very large context tables. An input set for Conexp that consists of 15 000 selected tuple pairs is processed in a reasonable time (some seconds), but if the size of the input set is larger than 20 000, Conexp will fail. To solve this problem, in the construction of these tuple pairs, in our software the user can set the limit of the selected tuples.

Constructing a clustered index on one of the attributes can speed up the execution of the SELECT statement. The advantage of using this SELECT command is that every Database Management System (DBMS) will generate an optimized execution plan.

Example 1. Let us consider an example. The table:

```
OrderDetail [CustomerID, CompanyName, City, Address, Phone,
      OrderID, OrderDate, ProductID, UnitPrice, Quantity]
```

stores orders of different customers with detail information such as product ID, price and quantity, too. `CompanyName` is the name of customer, `Address` is the customer's address and the `City` is his city. In Fig. 1 is represented the concept lattice for $FD\left(OrderDetail, \overrightarrow{\mathbb{IK}}(Engross)\right)$.

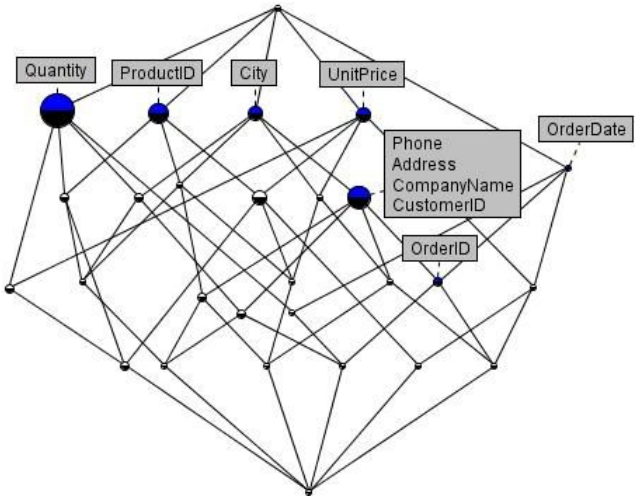

Fig. 1. Concept lattice for the context of functional dependencies $FD\left(OrderDetail, \overrightarrow{\mathbb{K}}(Engross)\right)$ with 19000 rows in the context

The implications in this lattice, which correspond to functional dependencies in the table can be seen as follows: the concept `Phone`, `Address`, `CompanyName`, `CustomerID` is a subconcept of concept `City`. This means that in every tuple pair where the `CustomerID` field has the same value, the name of the `City` is the same. The same reason for attributes `Phone`, `Address`, `CompanyName`. So we have the following implications which correspond to functional dependencies in the table:

$$CustomerID \rightarrow City$$
$$Phone \rightarrow City$$
$$Address \rightarrow City$$
$$CompanyName \rightarrow City$$

Concept with `OrderID` is a subconcept of concept `Phone, Address, CompanyName, CustomerID` and of concept `OrderDate` too. So we can read the following functional dependencies.

$$OrderID \rightarrow CustomerID$$
$$OrderID \rightarrow CompanyName$$
$$OrderID \rightarrow Address$$
$$OrderID \rightarrow Phone$$
$$OrderID \rightarrow OrderDate$$

The implication base given by software Conexp is illustrated in Fig.2. The user can make attribute exploration to decide which implications are valid. The number before the implication can help us. The implications labeled with larger numbers usually are best used in detecting functional dependencies.

Fig. 2. Implications which correspond to functional dependencies in the table OrderDetail given by the software

4 Further research

Further we tend to analyze the functional dependencies obtained, to construct the closure of these implications and to give a correct database scheme by using an upgraded version of the proposed software .

5 Acknowledgments

We thank Uta Priss, Joachim Hereth and Cristian Sacarea for their precious guidance and Andras Benczur for the idea of using inverted index files.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of databases. Addison-Wesley, Reading - Menlo - New York (1995)
2. Baixeries, J.: A formal concept analysis framework to mine functional dependencies, Workshop on Mathematical Methods for Learning, Villa Geno, Como, Italy, (2004).
3. Ganter, B., Wille, R.: Formal Concept Analysis. Mathematical Foundations. Springer, Berlin-Heidelberg-New York. (1999)
4. Hereth, J.: Relational Scaling and Databases. Proceedings of the 10th International Conference on Conceptual Structures: Integration and Interfaces LNCS 2393, Springer Verlag (2002) 62–76
5. Janosi Rancz, K. T., Varga, V.: A method for mining functional dependencies in relational database design using FCA. Studia Universitatis "Babes-Bolyai" Cluj-Napoca, Informatica, vol. LIII, No. 1, (2008) 17–28.
6. Priss, U.: Establishing connections between Formal Concept Analysis and Relational Databases. Dau; Mugnier; Stumme (eds.), Common Semantics for Sharing Knowledge: Contributions to ICCS, (2005) 132–145
7. Silberschatz, A., Korth, H. F.,Sudarshan, S.: Database System Concepts, McGraw-Hill, Fifth Edition, (2005)
8. Yao, H., Hamilton, H. J.: Mining functional dependencies from data, Data Mining and Knowledge Discovery, Vol. 16, Nr. 2, Springer Netherlands, (2008) 197–219
9. Serhiy A. Yevtushenko: System of data analysis "Concept Explorer". (In Russian). Proceedings of the 7th national conference on Artificial Intelligence KII-2000, Russia, (2000), 127–134.