STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 2, 2008

A SOFTWARE TOOL FOR DATA ANALYSIS BASED ON FORMAL CONCEPT ANALYSIS

KATALIN TUNDE JANOSI RANCZ, VIORICA VARGA, AND JANOS PUSKAS

Abstract. Formal Concept Analysis is a useful tool to represent logical implications in datasets, to analyze the underground knowledge that lies behind large amounts of data. A database relation can be seen as a many-valued context [3]. J. Hereth in [4] introduces the formal context of functional dependencies. In this context, implications hold for functional dependencies. We develop a software application that analyzes an existing relational data table and detect functional dependencies in it. The user can choose to analyze a table from a MS SQL Server, Oracle or MySQL database and the software will build the formal context of functional dependencies. We use Conexp [6] to build the concept lattice and implications in this context. These implications will be the functional dependencies for the analyzed table. Having the functional dependencies, we can detect candidate keys and we can decide if the table is in 2NF or 3NF or BCNF. To our knowledge, this method was not implemented yet.

1. Introduction

Formal Concept Analysis (FCA) appeared in 1980s ([7]) as a mathematical theory which formalises the notion of *concept* and is nowadays considered as an AI theory. It is used as a technique for data analysis, information retrieval and knowledge representation with various successful applications ([3]).

Functional dependencies (FDs shortly) are the most common integrity constraints encountered in databases. FDs are very important in relational database design to avoid data redundancy. Extracting FDs from a relational

Received by the editors: August 9, 2008.

2000 *Mathematics Subject Classification.* 68P15 Database theory, 03G10 Lattices and related structures.

1998 *CR Categories and Descriptors.* H2 Database Management [**Topic**]: Subtopic – H2.1 Logical design *Normal forms*.

Key words and phrases. Formal concept analysis, Normal forms.

This paper has been presented at the 7th Joint Conference on Mathematics and Computer Science (7th MaCS), Cluj-Napoca, Romania, July 3-6, 2008.

table is a crucial task to understand data semantics useful in many database applications. Priss in [5] presents the visualization of normal forms using concept lattices, where the notion of functional dependencies is life-line.

The subject of detecting functional dependencies in relational tables was studied for a long time and recently addressed with a data mining viewpoint. Baixeries in [2] gives an interesting framework to mine functional dependencies using Formal Context Analysis. Detecting functional dependencies seems to be an actual theme, [5].

Hereth [4] presents how some basic concepts from database theory translate into the language of Formal Concept Analysis. The definition of the formal context of functional dependencies for a relational table can also be found in [4]. Regarding to this definition the context's attributes are the columns (named attributes) of the table, the tuple pairs of the table will be the objects of the context. [4] gives the proposition which asserts that in the formal context of functional dependencies for a relational table, implications are essentially functional dependencies between the columns of the relational database table.

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 the formal context of functional dependencies for table m, which is notated $FD\left(m, \overrightarrow{K}(\mathcal{D})\right)$.

Informally, normal forms are defined in traditional database theory as a means of reducing redundancy and avoiding update anomalies in relational databases. Functional dependency means that some attributes' values can be reconstructed unambiguously by the others [1].

In this paper we intend to extend a previous research presented in [8]. We implemented the method presented in [8] for database design and completed it with a software tool, which analyzes an existing relational database table. Our software named FCAFuncDepMine constructs the formal context of functional dependencies. It uses Conexp [10] to build the concept lattice and to determine the implications in this context. The implications obtained correspond to functional dependencies in the analyzed table. The software can be used in relational database design and for detecting functional dependencies in existing tables, respectively.

2. Software description

This section presents how our software constructs the context of functional dependencies of an existing relational database table. The method used in relational database design was described in [8].

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 which can be selected for possible functional dependency examination.

The formal context of functional dependencies for the selected table has to be constructed. The attributes of the context will be the attributes of the studied table and the context's objects will be the tuple pairs of the table. A table may have a lot of tuples and much more tuple pairs. We optimize the construction of the context in both approaches.

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.

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. In order to omit this failure, the user can set a limit for the number of the selected tuples.

Let T be this table having attributes $A_1, \ldots, A_n$. 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 tested concept lattices omitting tuple pairs in the top and the bottom of the lattice. During this test we did 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 have 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 NOT (T1.A1=T2.A1 AND ... AND T1.An=T2.An)
```

This statement leads to a Cartesian-product of table T with itself, which is a very time consuming operation. The statement is transformed by eliminated NOT from it.

```
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. Let $P1, P2, ..., Pk (k \geq 1)$ be the primary key of table T. The definition of a relational table's primary key can be found in [6]. 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.

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

With `Selected Columns` button the user can choose a list of attributes of the selected table, otherwise all attributes will be selected. In order to create the cex file for Conexp the `Select Tuple Pairs` button have to be pressed, which selects tuple pairs which have at least one of its attribute value in common. Tuple pairs in the top and in the bottom of the concept lattice can be generated optionally, checking the `Add Extra Tuple Pairs` option in the File menu. Tuple pairs being generated we have to save the cex file, then it can be used as input for Conexp, which will build the concept lattice and implications. In the following we will examine the concept lattice of the context of functional dependencies which was constructed for a relational table.

3. The structure of the application

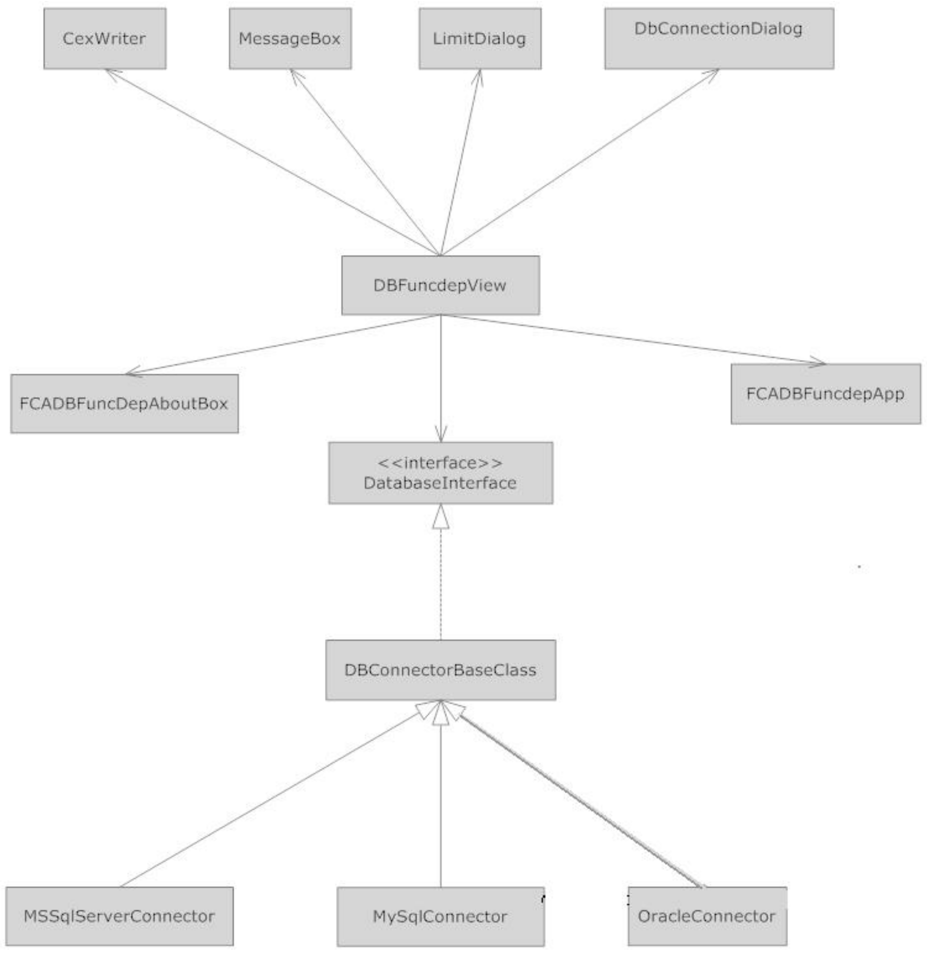


Figure 1. The class diagram of the application

The application was developed in Java programming language conforming to object-oriented programming paradigms. The class diagram of the application can be seen in Figure 1.

The class `DBFuncDepView` is the main class of the application. This class executes the graphical interface and also has a reference to the other classes, this means that every functionality can be reached trough this class.

In the interface `DatabaseInterface` are defined those functionalities with which we can execute the SELECT statement described in previous section against different Database Management Systems. It contains functionalities for selecting the list of the existing tables in the studied database and methods for selecting the rows and columns of the tables.

The class `DBConnectorBaseClass` implements the functions described in the earlier presented interface which can be implemented for each of the three DBMS.

The classes `MySQLConnector`, `MSSqlServerConnector`, `OracleConnector` contain the specific functions and drivers needed for the connection to the different databases.

The role of the `CexWriter` class is to export the tuple pairs of the context table in .cex format which is in fact in XML format. An example is in Figure 2.

4. Data Analysis

FD lattices can be used to visualise the normalforms [5]. The lattice visualizations can help to convey an understanding of what the different normalforms mean. All attributes of a database table must depend functionally on the table's key, by definition. Therefore for each set of formal concepts of a formal context there exists always a unique greatest subconcept (meet) which must be equal the bottom node of the FD lattice.

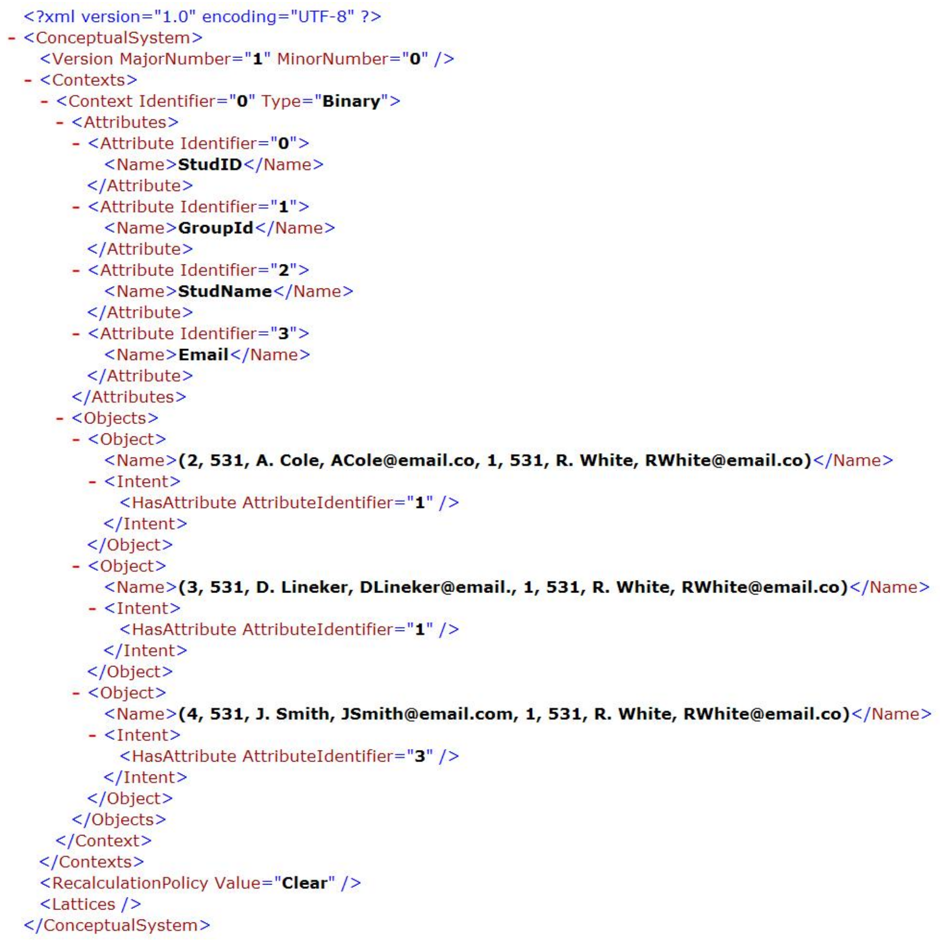
Let us begin with a simple example.

Example 1. Let be the next relational database table scheme:

`Students [StudID,StudName,GroupID,Email]`

We have analyzed this table with our software. The FD lattice and functional dependencies obtained for this table are shown in the Figure 3. The FD lattice interpretation is: the concept `StudID, StudName, Email` is a subconcept of concept `GroupID`. This means there is an implication from the concept `StudID, StudName, Email` to the concept `GroupID`. Accordingly, in every tuple pair where the `StudID` field has the same value, the value of the `GroupID` will remain the same.

Because all keys meet in the bottom node, for a determinant to be a candidate key means that the unique greatest subconcept of its attributes equals the bottom node. Consequently every attribute depends on `StudID`, therefore it is a candidate key. The same case is for `StudName` and `Email`.



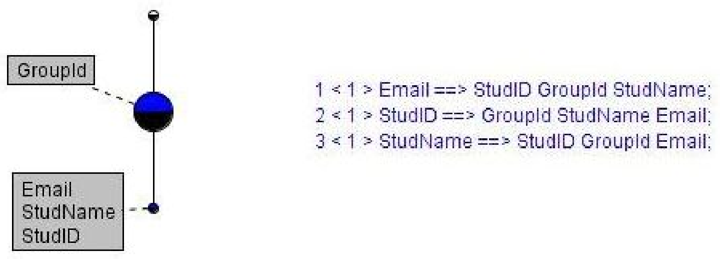
```
  <?xml version="1.0" encoding="UTF-8" ?>
- <ConceptualSystem>
    <Version MajorNumber="1" MinorNumber="0" />
  - <Contexts>
    - <Context Identifier="0" Type="Binary">
      - <Attributes>
        - <Attribute Identifier="0">
            <Name>StudID</Name>
          </Attribute>
        - <Attribute Identifier="1">
            <Name>GroupId</Name>
          </Attribute>
        - <Attribute Identifier="2">
            <Name>StudName</Name>
          </Attribute>
        - <Attribute Identifier="3">
            <Name>Email</Name>
          </Attribute>
        </Attributes>
      - <Objects>
        - <Object>
            <Name>(2, 531, A. Cole, ACole@email.co, 1, 531, R. White, RWhite@email.co)</Name>
          - <Intent>
              <HasAttribute AttributeIdentifier="1" />
            </Intent>
          </Object>
        - <Object>
            <Name>(3, 531, D. Lineker, DLineker@email., 1, 531, R. White, RWhite@email.co)</Name>
          - <Intent>
              <HasAttribute AttributeIdentifier="1" />
            </Intent>
          </Object>
        - <Object>
            <Name>(4, 531, J. Smith, JSmith@email.com, 1, 531, R. White, RWhite@email.co)</Name>
          - <Intent>
              <HasAttribute AttributeIdentifier="3" />
            </Intent>
          </Object>
        </Objects>
      </Context>
    </Contexts>
    <RecalculationPolicy Value="Clear" />
    <Lattices />
  </ConceptualSystem>
```

FIGURE 2. Cex file example for Students table

These attributes appear in the bottom of the FD lattice. `StudName` appears as candidate key, because all student name values were different in the analyzed table. We know that students may have the same name, but not the same ID, not the same Email address.

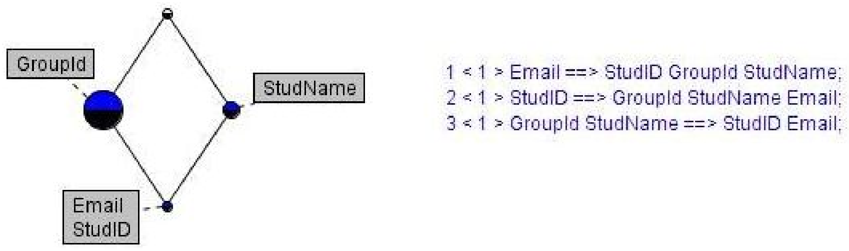
In Figure 4 we can see the results when in every group all students has different names than the `GroupID` and the `StudName` together form a composite key.

Figure 5 shows the FD lattice and functional dependencies obtained in case when we have same value for some student names in the same group and



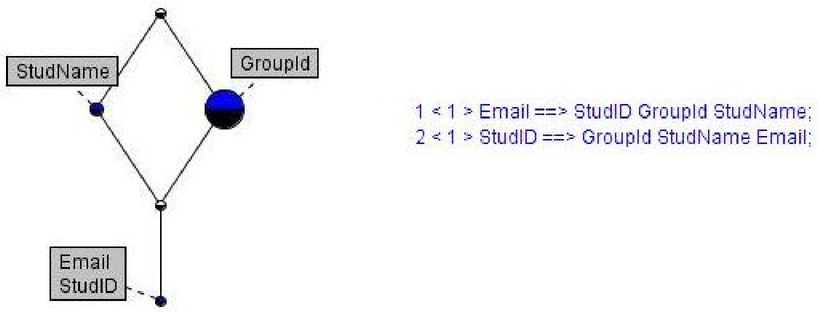
```
1 < 1 > Email ==> StudID GroupId StudName;
2 < 1 > StudID ==> GroupId StudName Email;
3 < 1 > StudName ==> StudID GroupId Email;
```

FIGURE 3. FD lattice and implications for the Students table with different values for student names



```
1 < 1 > Email ==> StudID GroupId StudName;
2 < 1 > StudID ==> GroupId StudName Email;
3 < 1 > GroupId StudName ==> StudID Email;
```

FIGURE 4. FD lattice and implications for the Students table when in every group all students has different names, but there are some students with same value for name



```
1 < 1 > Email ==> StudID GroupId StudName;
2 < 1 > StudID ==> GroupId StudName Email;
```

FIGURE 5. FD lattice and implications for the Students table with same value for some student names in the same group and in different groups too

in different groups too. This is the real case. We can see, that `StudName` doesn't appear in the bottom of the lattice, it isn't in the left hand side of any functional dependency, therefore it can't be a candidate key.

For determining whether an FD lattice is in BCNF, all non-trivial implications other than the ones whose left-hand side meets in the bottom node need to be checked. We can see, that every nontrivial functional dependency in the `Students` table has in its left hand side a superkey, therefore the table is in BCNF.

Example 2. Let `StudAdvisor` be a wrongly designed database table of a university database.

```
StudAdvisor [StudID,StudName,GroupID,StudEmail,SpecID,SpecName,
    Language,AdvisorId,TeacherName,TeacherEmail,TeacherPhone]
```

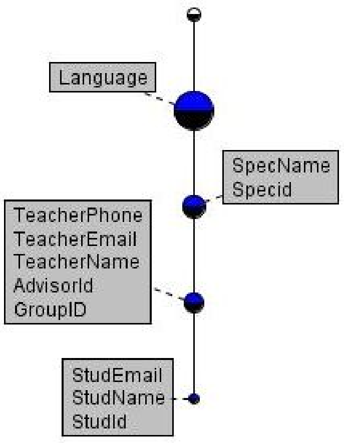
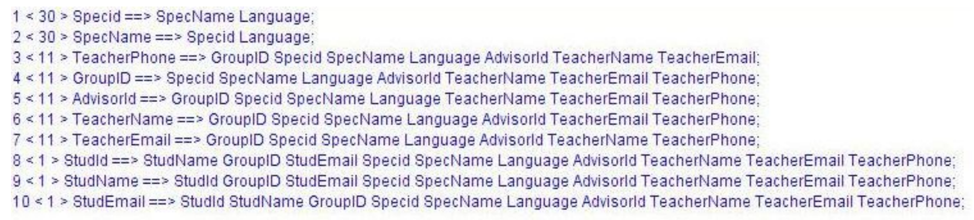


FIGURE 6. FD lattice for the StudAdvisor table

The FD lattice obtained for this table with software **FCAFuncDepMine** is in the Figure 6 and functional dependencies are in Figure 7. The concept `StudID, StudName, Email` is a subconcept of the concept `GroupID, AdvisorID, TeacherID, TeacherName, TeacherEmail,TeacherPhone,` which is the subconcept of the concept `SpecID, SpecName` and so on. The analysed data is not enough diversified, because every advisor has different name, every student has different name. The candidate keys of the table StudAdvisor are in the bottom of the FD lattice. But there are other functional dependencies, that has in its left hand side attributes, that are not in the bottom of the lattice: `SpecID, SpecName,TeacherPhone,`etc., therefore the table is not in BCNF.



```
1 < 30 > Specid ==> SpecName Language;
2 < 30 > SpecName ==> Specid Language;
3 < 11 > TeacherPhone ==> GroupID Specid SpecName Language Advisorld TeacherName TeacherEmail;
4 < 11 > GroupID ==> Specid SpecName Language Advisorld TeacherName TeacherEmail TeacherPhone;
5 < 11 > Advisorld ==> GroupID Specid SpecName Language TeacherName TeacherEmail TeacherPhone;
6 < 11 > TeacherName ==> GroupID Specid SpecName Language Advisorld TeacherEmail TeacherPhone;
7 < 11 > TeacherEmail ==> GroupID Specid SpecName Language Advisorld TeacherName TeacherPhone;
8 < 1 > Studld ==> StudName GroupID StudEmail Specid SpecName Language Advisorld TeacherName TeacherEmail TeacherPhone;
9 < 1 > StudName ==> Studld GroupID StudEmail Specid SpecName Language Advisorld TeacherName TeacherEmail TeacherPhone;
10 < 1 > StudEmail ==> Studld StudName GroupID Specid SpecName Language Advisorld TeacherName TeacherEmail TeacherPhone;
```

FIGURE 7. Functional dependencies in the StudAdvisor table

Introducing more varied data we get the FD lattice from the Figure 8 and functional dependencies from Figure 9.

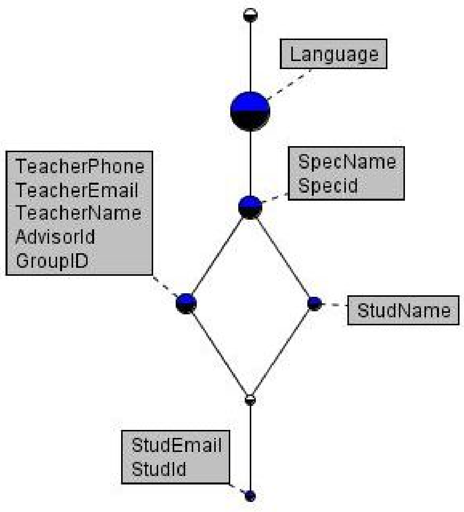
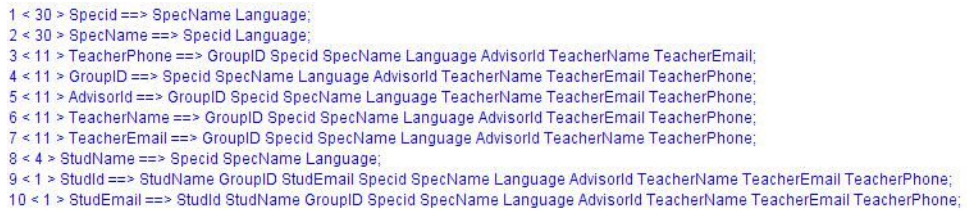


FIGURE 8. FD lattice for the StudAdvisor table with varied data

Having the functional dependencies, the candidate keys of the table can be seen and we can propose a decomposition of the table. From the last two FD's results, that every attribute is functionally dependent on `StudID,` as well as on `StudEmail,` therefore these two attributes are candidate keys. From the first two functional dependencies we can propose the next table:



```
1 < 30 > Specid ==> SpecName Language;
2 < 30 > SpecName ==> Specid Language;
3 < 11 > TeacherPhone ==> GroupID Specid SpecName Language AdvisorId TeacherName TeacherEmail;
4 < 11 > GroupID ==> Specid SpecName Language AdvisorId TeacherName TeacherEmail TeacherPhone;
5 < 11 > AdvisorId ==> GroupID Specid SpecName Language TeacherName TeacherEmail TeacherPhone;
6 < 11 > TeacherName ==> GroupID Specid SpecName Language AdvisorId TeacherEmail TeacherPhone;
7 < 11 > TeacherEmail ==> GroupID Specid SpecName Language AdvisorId TeacherName TeacherPhone;
8 < 4 > StudName ==> Specid SpecName Language;
9 < 1 > StudId ==> StudName GroupID StudEmail Specid SpecName Language AdvisorId TeacherName TeacherEmail TeacherPhone;
10 < 1 > StudEmail ==> StudId StudName GroupID Specid SpecName Language AdvisorId TeacherName TeacherEmail TeacherPhone;
```

FIGURE 9. Functional dependencies in the StudAdvisor table with varied data

`Specializations [SpecId,SpecName,Language]`

The FD's with number between 3 and 7 suggest the next table:

```
Advisors [GroupID,SpecId,AdvisorId,TeacherName,TeacherEmail,
                                   TeacherPhone]
```

The remaining attributes form the studied relation forms the next table:

`Students [StudID,StudName,GroupID,StudEmail]`

5. Conclusions and further research

We have proposed a software tool to detect functional dependencies in relational database tables. Our software constructs the power context family of the functional dependencies for a table, then Conexp gives the conceptual lattice and implications. 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, respectively.

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, Proceedings of Mathematical Methods for Learning, (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] 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

[6] Silberschatz, A., Korth, H. F.,Sudarshan, S.: Database System Concepts, McGraw-Hill, Fifth Edition, (2005)

[7] Wille, R. : Restructuring lattice theory: an approach based on hierarchies of concepts. In: I.Rival (ed.): Ordered sets. Reidel, Dordrecht-Boston, (1982) 445–470

[8] 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.

[9] Yao, H., Hamilton, H. J.: Mining functional dependencies from data, Data Mining and Knowledge Discovery, Springer Netherlands, (2007)

[10] Serhiy A. Yevtushenko: System of data analysis "Concept Explorer". (In Russian). Proceedings of the 7th National Conference on Artificial Intelligence KII-2000, p. 127-134, Russia, 2000.

Sapientia University, Tg-Mures, Romania
E-mail address: tsuto@ms.sapientia.ro

Babes-Bolyai University, Cluj, Romania
E-mail address: ivarga@cs.ubbcluj.ro

Tg-Mures, Romania
E-mail address: puskasj@gmail.com