Implementing an Automated Normalization System for Functional Independent Normal Form in Relational Databases

Tihomir Orehvacki, Markus Schatten, and Alen Lovrencic
Faculty of Organization and Informatics, University of Zagreb
Pavlinska 2, 42000 Varazdin, Croatia
E-mail: {tihomir.orehvacki, markus.schatten, alen lovrencic}@foi.hr

Abstract. A deductive system for automated database normalization is implemented by using a HiLog reasoning engine. The system allows for checking a relational schema up to Boyce-Codd normal form (BCNF) and functional independency normal form (FINF). Examples of system usage are presented and discussed.

Keywords. automated database normalization, BCNF, FINF, HiLog

1 Introduction

The relational database life cycle is an iterative process that consists of four basic steps [19]: requirements analysis, logical design, physical design, and database implementation, monitoring, and modification. The problem of automated database normalization is an important aspect of CASE tools. Requirements analysis incorporates gathering information from both users and producers of database and results with a formal specification of requirements, including data, their interrelationships, a platform for database implementation, etc. Logical design starts with conceptual database modeling where the relationships among the data are graphically displayed with the Entity-Relationship (ER) or Unified Modeling Language (UML) diagram. In cases when database design includes a larger number of participants, and thus different views of the conceptual model, through the identification of synonyms, aggregation and generalization an unified view (i.e. global scheme) is obtained. The next step is mapping the conceptual scheme into a set of non-normalized relations. In order to eliminate redundancy and anomalies that occur during the mapping, the last phase of logical design comprises the normalization of database relations. Physical design or denormalization is a process of tuning the database to improve its performance in terms of efficiency. The last step of database life cycle includes implementation of database formal scheme within the selected database management system (DBMS). Using the Data Definition Language (DDL) and Data Manipulation Language (DML) on the database, limits can be defined, queries execute, errors eliminated and work of database monitored.

This paper deals with normalization of relational scheme as a part of the logical database design. Normalization is the progressive decomposition of the relational database model in order to minimize data redundancy and eliminate insert, delete, and update anomalies. It is most commonly implemented as a set of tests to determine whether a relation satisfies or violates the requirements of a particular normal form. Up to Boyce-Codd normal form (BCNF) decomposition and testing of relations is based on functional dependencies between attributes. Among the most important ones are Armstrong’s axioms of reflexivity, augmentation and transitivity [3]. Based on the functional dependencies, it is possible to define different levels of normal form:

- relation $R$ is in first normal form (1NF) if and only if the value of each attribute is atomic (i.e. single and indivisible);
- relation $R$ is in second normal form (2NF) if and only if is in 1NF and if every non-key attribute is fully dependent on the primary key (no Partial Dependency);
- relation $R$ is in the third normal form (3NF) if and only if is in 2NF and every non-key attribute is non-transitively dependent on the primary key (no Transitive Dependency);
- relation $R$ is in the Boyce-Codd normal form (BCNF) if and only if is in 3NF and
if each of its determinant is also a candidate key (all functional dependencies of relation are arising from its key).

In most cases, it is enough to normalize relation up to and including BCNF [4]. However, in practice it is possible to find trivial examples when functional dependencies cannot eliminate the basic data redundancy. Namely, if two attributes are not mutually functionally dependent, but the set of instances of one attribute is functionally dependent on the set of instances of other attributes, then among the attributes a Sub-Domain Dependency exists [7]. Since BCNF does not have implemented mechanisms for removing this type of dependency, data anomalies will surely occur in high level normal forms. As a solution of mentioned problem, Chen Et Al. [7] have proposed the implementation of Functional Independency and Functional Independent Normal Form (FINF):

- relation $R$ is in Functional Independent Normal Form (FINF) if and only if is in BCNF and $\forall X, Y \in R, X \rightarrow Y$ or $Y \rightarrow X$ or $X \gg Y$ holds.

Through the use of Functional Independency, FINF largely removes constraints that cannot be defined by keys and domains and thus addresses major requirements for reaching Domain Key Normal Form (DKNF) [13]. Accordingly, FINF is placed after 5NF, but before DKNF in the normalization process.

The rest of the paper is structured as follows. Section 2 briefly reviews the related work. Section 3 introduces the implementation of normalization algorithms in HiLog. In Section 4 examples of use and testing the Automated Normalization System are given. Remarks about implemented algorithms and points to future work are provided in last section.

2 Related work

Normalization is the most important process of logical database design. If performed manually, it is subject to errors and requires a tremendous amount of time and money. Mentioned drawbacks can be removed by automating the process of normalization. For more than 30 years automated normalization is a hot topic, and thus it’s no wonder that a large number of specialized tools and approaches have been proposed. Based on their theoretical and implementation features, these tools can be classified into several different categories.

The first category is composed of web and desktop applications that through interaction with users facilitate understanding of theoretical concepts of relational database normalization. NORMIT [18] is web enabled and constraint based Intelligent Tutoring System (ITS), which leads students through the procedural process of normalization up to BCNF. Fundamental part of the NORMIT system is a problem solver which by using the knowledge base of 53 restrictions checks the syntax of students’ submissions, and compares their semantics with an automatically generated ideal solution. A similar web environment that incorporates ten functional dependencies and allows direct and step-by-step normalization up to 3NF was presented by Kung and Tung [15].

Micro [12] is a desktop tool where users using a windowed graphical user interface (GUI) can decompose a database to BCNF and create tables in an MS Access DBMS environment. Normalizer [2] works in a similar manner. It is a tool which by means of two algorithms based on functional dependencies demonstrates automatic normalization up to 3NF. Finally, JMathNorm [21], whose algorithms are implemented in the Mathematica, and a graphical interface in Java, automates and validates normalization process up to BCNF. Although all mentioned tools from this category have great educational value, most of them are ineffective when complex databases are examined and it would be difficult to deploy them within different DBMSs.

The second group consists of graphical normalization algorithms. Based on the type of graphical representation that is used as an implementation framework, this group can be further decomposed to Entity-Relationship (ER), Unified Modeling Language (UML) and matrix based algorithms. As aforementioned, ER diagram graphically displays relationships between entities in the phase of conceptual database design. In order to avoid problems and anomalies that occur when converting ER diagram into a relational mo-
del, the bubble diagram [8] was introduced. Sets of entities from ER do not exist in the bubble diagram, but are replaced with bubbles that graphically display the primary keys and the mutual linkages between attributes. The main advantage of bubble diagrams are implemented functional and multivalued dependencies which enabled normalization up to 4NF. However, the problems that subsist in traditional design methods for decomposition relations to 3NF and 4NF are not fully solved with bubble diagrams, and there is still a possibility that unenforceable dependencies will occur.

As an alternative to the previous solution, the use of Articulated Entity Relationship (AER) diagrams [9] was proposed. AER is an extension of the ER diagram with functional dependencies which allows for automated normalization through minimal interaction with the user. However, the use of normalization algorithm that is based only on functional dependencies does not solve the problem of Sub-Domain Dependencies. Automated normalization systems based on two declarative specifications over the UML meta-model were introduced by Akehurst Et Al. [1]. The first specification contains Object Constraint Language (OCL) declarations to encode the normal form, while the second one consists of the graph rewriting rule that specifies transformation steps from a lower to a higher normal form. Finally, matrix based approaches automates the normalization up to BCNF through the use of three structures (Dependency Graph, Dependency Directed Graph Matrix and Matrix) and allows for algorithm implementation for both the manipulation of dependencies and path finding [4]. This category of tools can be used in the step of conceptual database design and thus minimize the anomalies in the relational model. Nevertheless, in cases when model contains a large number of relationships, diagrams due to their complexity become unusable.

The last category refers to normalization algorithms implemented in logic programming languages. The first consolidated system for database normalization whose algorithms were written in Prolog was implemented by Ceri and Gottlob [5]. Its main purpose was the design of small database and teaching the theory of normalization. Since it was based on functional dependencies, the normalization up to BCNF was possible. However its main disadvantage was lack of efficiency. As an extension of their work, a case tool that converts an ER conceptual model into a normalized relational scheme with algorithms implemented in Prolog [14]. This system is based on functional and inclusion dependencies and allows database normalization up to the Inclusion Normal Form (IN-NF) which guarantees that every database relation is in 3NF [16].

The above discussed approaches have one thing in common - they were not designed for automatic database normalization but for automated reasoning about sets of dependencies or automation of standard procedures defined in the theory of dependence (generating a cover set or closure of a set of functional dependencies, etc.). Therefore Lovrenčić Et Al. [17] presented functional dependencies and normal forms by formulae of predicate calculus and thereby created a backbone of a logical system which will automatically generate relational database scheme in BCNF.

In the context of an automated system we have implemented HiLog predicates for database normalization up to Functional Independent Normal Form (FINF) and thus extended the proposals in [7] and [17]. The reason why we have chosen HiLog among other programming languages is that its combination of advantages of higher-order syntax with the simplicity of first-order semantics facilitates manipulation of the relational database [6].

3 Implementation

We implemented the algorithm for automated database normalization using the FLORA-2 reasoning engine [20]. The implementation follows the mathematical formalization in [17].

The first predicate $g/2$ defines an ordering relation to which is later used in the predicate subset/2 to avoid equivalent subset. The predicate is implemented as follows:

\[
g(\ ?n1, \ ?n2) :-
g(\ ?n1, \ ?n2) @. plg,
\]

The mentioned predicate subset/2 which
succeeds if the second supplied argument is a subset of the first one. The following listing shows the implementation of the predicate.

```
succeeds( _, [] ).
succeeds( ?x, [ ?h | ?t ] ) :-
    member( ?h, ?x ) @ plg( list ),
    subset( ?x, ?t ),
    not( member( ?h, ?t ) @ plg( list ) ),
    if ?t = [ ?th | _ ]
    then succeeds( ?th, ?h ).
```

The set union predicate (union/3) succeeds if the last argument is the set union of the first two supplied arguments. The predicate is easily implemented using FLORA-2's collectset aggregate.

```
union( ?x, ?y, ?z ) :-
    ?z = collectset{ ?e | member( ?e, ?x ) @ plg(lists); member( ?e, ?y ) @ plg(lists) }.
```

Having the basic operations implemented, we can now advance further to the implementation of axioms. Reflexivity is defined for a functional dependency (FD/3) in which the first argument is the relational scheme, the second the left hand side of the dependency and the third the right hand side of the dependency. The predicate is implemented as follows:

```
FD( ?r, ?x, ?y ) :-
    subset( ?r, ?x ),
    subset( ?r, ?y ),
    subset( ?x, ?y ),
```

The transitivity axiom is even simpler in FLORA-2. Using the FD/2 predicate it is implemented as shown in the following listing:

```
FD( ?r, ?x, ?y ) :-
    FD( ?r, ?x, ?z ),
    FD( ?r, ?z, ?y ).
```

Augmentation is a bit trickier. To implement it we need to use the subset/2 and union/3 predicates. This is of course computationally expensive since all subsets of a given relational scheme have to be computed. The predicate is implemented as follows:

```
FD( ?r, ?xz, ?yz ) :-
    subset( ?r, ?x ),
    subset( ?r, ?y ),
    FD( ?r, ?x, ?y ),
    subset( ?r, ?z ),
    union( ?x, ?z, ?xz ),
    union( ?y, ?z, ?yz ).
```

Having functional dependencies implemented, we can now advance to the implementation of the super key predicate SK/2. A super key is any set of attributes $X$ for which it holds that $X \rightarrow R$, whereby $R$ is the scheme. This statement is implemented as follows:

```
SK( ?r, ?x ) :-
```

The super key but not key over some relational scheme $R$ is any set of attributes $X$ for which there is a set of attributes $Y \subset X$ for which it holds that $Y \rightarrow R$. This statement is implemented as the predicate NK/2:

```
NK( ?r, ?x ) :-
    subset( ?x, ?y ),
    ?x \supseteq ?y,
    SK( ?r, ?y ).
```

Having the super key and super key but not key predicates implemented we can now define the key predicate K/2. A key for relational scheme $R$ is a set of attributes $X$ for which if it is a super key, but not a super key but not key.

The following listing implements this statement:

```
K( ?r, ?x ) :-
    SK( ?r, ?x ),
    not( NK( ?r, ?x ) ).
```

Now we can define the predicate NBC-NF/1 which success if the supplied argument (the relational scheme) is not in BCNF. Again this is a computational expensive predicate, and is implemented as follows:

```
NBCNF( ?r ) :-
    subset( ?r, ?x ),
    subset( ?r, ?y ),
    not( subset( ?r, ?z ) ),
    not( SK( ?r, ?y ) ).
```

In order to model functional independencies we need to include another predicate FI/3. A call to FI( ?r, ?x, ?y ) will succeed iff $X \\supseteq Y$ in $R$. Now we are able to define the NFINF/1 predicate as follows:

```
NFINF( ?r ) :-
    NBCNF( ?r ).
```

K( ?r, ?x ) :-
    SK( ?r, ?x ),
    not( NK( ?r, ?x ) ).

Now we can define the predicate NBC-NF/1 which success if the supplied argument (the relational scheme) is not in BCNF.

Again this is a computational expensive predicate, and is implemented as follows:

```
NBCNF( ?r ) :-
    subset( ?r, ?x ),
    subset( ?r, ?y ),
    not( subset( ?r, ?z ) ),
    not( SK( ?r, ?x ) ).
```

In order to model functional independencies we need to include another predicate FI/3. A call to FI( ?r, ?x, ?y ) will succeed iff $X \\supseteq Y$ in $R$. Now we are able to define the NFINF/1 predicate as follows:

```
NFINF( ?r ) :-
    NBCNF( ?r ).
```

Finally, we can define the predicate F1/3 which success if the supplied argument (the relational scheme) is not in BCNF.
The first part of the predicate makes sure that $R$ is in BCNF. The second part verifies $\forall X, Y \in R$ that none of $X \rightarrow Y$, $Y \rightarrow X$ and $X \gg Y$ holds.

4 Examples
Consider the following relational schema $(R, F)$:

$$R : ABCD \quad F : AC \rightarrow B, A \rightarrow D$$

To acquire the superkeys of $(R, F)$ we can now issue the query:

$$\text{flora2 } ?- \text{SK( [ A, B, C, D ], ?x )}.$$

?x = [A, B, C]
?x = [A, B, C, D]
?x = [A, C]
?x = [A, C, D]

4 solution(s) in 1.2360 seconds
Yes

To get the key of $(R, F)$ we just need to call the K/2 predicate as in:

$$\text{flora2 } ?- \text{K( [ A, B, C, D ], ?k )}.$$

?k = [A, C]

1 solution(s) in 0.0080 seconds
Yes

To check if $(R, F)$ is in BCNF we issue the query:

$$\text{flora2 } ?- \text{NBCNF( [ A, B, C, D ] )}.$$

Elapsed time 0.0120 seconds
Yes

Now consider the following relational schema:

$$R : ABCD \quad F : A \rightarrow BC, A \rightarrow D$$

We can easily check if $(R, F)$ is in BCNF:

$$\text{flora2 } ?- \text{NBCNF( [ A, B, C, D ] )}.$$

No

But, $(R, F)$ isn’t in FINF, as the following query shows:

$$\text{flora2 } ?- \text{NFINF( [ A, B, C, D ] )}.$$

Elapsed time 0.0000 seconds
Yes

By adding the following functional independencies:

$$C \gg D \quad B \gg D \quad C \gg B$$

The relational schema is now in FINF, as we can see from the following result:

$$\text{flora2 } ?- \text{NFINF( [ A, B, C, D ] )}.$$

No

5 Conclusion

The main aim of this paper was to introduce a logic system which will be able to check if given relational scheme is in BCNF. With the implementation of Functional Independence a problem of Sub-Domain Dependency among two sets of attributes was solved as well and thus the feature of testing relational scheme almost to the Domain Key Normal Form was enabled. The test results have shown that in cases of small relational schemes the system works efficiently. The next step of our research will be focused on the optimization of implemented algorithms in order to improve performance when testing complex relational schemes, in order to reach the ultimate goal, a system which will automatically generate relational schemes in FINF from unnormalized or semi-normalized database scheme.

6 References


