An interactive tool for teaching and learning database normalization

Christos Stefanidis
Applied Informatics Department
University of Macedonia
Thessaloniki, Greece
chstefanides@yahoo.gr

Georgia Koloniari
Applied Informatics Department
University of Macedonia
Thessaloniki, Greece
gkoloniari@uom.gr

ABSTRACT

This paper presents a software tool that automates the process of relational database normalization. In particular, based on a user-given table and a set of functional dependencies, it outputs a set of BCNF normalized tables. The design goal of our software tool is twofold. Besides providing an automated normalization process that can efficiently handle small applications, its primary goal is to act as a teaching aid assisting educators to demonstrate the normalization process, and also as a self-training tool enabling students to practice with normalization by themselves. To this end, the tool provides a step-by-step interactive normalization process allowing the user to try alternative decompositions by selecting at each step a functional dependency and the corresponding table to be decomposed. Furthermore, it provides detailed explanations at each step and sub-process supported to help students understand the different steps of the normalization process.

CCS Concepts

•Information systems → Database design and models; •Applied computing → *Education;*

Keywords

relational database, normalization, educational tool

1. INTRODUCTION

Normalization is a necessary step in relational database design to avoid data repetition and anomalies. When following a *top-down* approach, normalization is required to correct the inconsistencies of a poor conceptual schema that are otherwise inherited by the relational schema. On the other hand, following a *bottom-up* approach, normalization is the main mechanism that automates the design of a relational schema by gradually decomposing a universal relation into a set of smaller tables according to a specific normal form.

PCI '16, November 10-12, 2016, Patras, Greece

DOI: http://dx.doi.org/10.1145/3003733.3003790

No matter which approach is used, it is obvious that normalization is indispensable. However, it is often neglected by application developers as they consider it difficult and time consuming. Also, while there is a plethora of well-known CASE tools that automatically derive from the logical model the SQL code that implements the physical schema, the tools providing automated normalization are few and limited. Moreover, normalization is perceived as one of the most difficult topics by students in database courses that usually fail to understand it and consider it too theoretical.

In this paper, we present a software tool for automating the normalization process, and in particular a tool that, based on a user-given table and a set of functional dependencies, outputs a set of normalized tables following the Boyce-Codd Normal Form (BCNF) [6]. Our tool (available at http://users.uom.gr/~gkoloniari/normTool.html) can be used by database designers to produce automatically a normalized relational schema for their application, or check the validity of their own normalized schema. However, our main design goal is educational. As teaching normalization encounters difficulty, our tool was developed to assist both educators as a teaching aid to demonstrate normalization and also to enable students to practice and learn normalization on their own. To this end, our tool provides besides an automatic decomposition functionality, an interactive step-by-step decomposition process. In this process, a user is able to try alternative decompositions by selecting at each step a functional dependency and the corresponding table to be decomposed. Detailed explanations are provided, and the tool also provides a user-friendly and practical GUI.

The rest of the paper is structured as follows. Section 2 discusses related work and Section 3 outlines the basic concepts and algorithms for normalization. Section 4 describes the requirements of the software tool and demonstrates its main functionality, while Section 5 concludes our paper.

2. RELATED WORK

There are a number of tools for database normalization and we distinguish between two categories: tools that automate the normalization process designed for database developers [3], [7], [8], and tools that are designed for educational purposes [1], [2], [5], [9], [10], [11], [12].

In the first category, Micro [8] supports normalization to 2NF, 3NF and BCNF, using a novel algorithm for 2NF normalization. The focus of the system is on efficiency with the use of appropriate data structures and it provides a fully automated normalization process, enforcing specific ordering of the functional dependencies. It also generates the

code for implementing the decomposed tables in Microsoft Access. Normalizer [3] is another tool similar to Micro, that only supports 2NF and 3NF, and also focuses on a GUI that tries to limit human errors. Finally, RDBNorma [7] is a semi-automatic tool that outperforms Micro in terms of efficiency both with respect to space and time, but supports only 2NF and 3NF normalization, similarly to Normalizer. Our tool provides an automatic normalization process and can be used for small applications. Though it can deal with a large number of attributes and dependencies in satisfying time, our focus is on its educational use, similarly to the tools of the second category.

Two simple web-based tools are presented in [11] and [1]. Both support only 3NF normalization, while the second additionally supports the evaluation of the closure and minimal cover of the functional dependencies. The tool, in [11], presents the decomposition results gradually, but without allowing user interaction. Both tools do not provide extensive explanations of the decomposition process. In contrast, the web-based tool, presented in [5], offers detailed explanations. It normalizes a relation to BCNF but not directly, normalizing first to 3NF. All three web-based tools besides not allowing any user intervention, do not have particularly user-friendly GUIs, and no loading nor saving functionality. NormalDB [2] uses Prolog in its core and through a web-interface provides a user-friendly GUI that enables normalization to 3NF, BCNF and also denormalization. JMathNorm [12] is written in Mathematica with a GUI in Java. While its main purpose is educational it can also be used for other purposes as it is simple and efficient. EDNA [9] is another educational tool that fully automates the normalization process and additionally derives SQL statements for implementing the corresponding physical schema. Finally, LBDN [10] is a web-based environment that supports decomposition to 2NF, 3NF and BCNF. It allows users to provide their own solutions that are evaluated by the system for validity, and it also provides a sample solution when required. However compared to all educational tools, ours is the only one that allows the user to direct the decomposition process and provides alternative solutions when users select the functional dependencies in different order.

3. RELATIONAL DATABASE DESIGN

A fundamental concept in normalization is that of *functional dependency* (FD) which defines a constraint between subsets of attributes in a relation. For two sets of attributes X and Y belonging to relation R, we say that X functionally determines Y, $X \rightarrow Y$ if and only if, each value of X is associated strictly with a single value of Y.

A functional dependency is *trivial* if and only if $Y \subseteq X$. The non-trivial functional dependencies are divided into *completely non-trivial* and *partially non-trivial*. A functional dependency is completely non-trivial when $X \cap Y = \emptyset$, otherwise, it is considered as partially non-trivial. When normalizing, trivial FDs are ignored and partially non-trivial ones are transformed to completely non-trivial.

Given a relation R and a set F of functional dependencies, the closure of a set of attributes X of R, denoted as X^+ is the set of all attributes that are functionally dependent from X as they can be derived by using Armstrong's axioms [4] for dependency deduction.

The attributes closure is evaluated to determine the candidate keys of a relation R. A set of attributes X is a candidate key for R if $X^+ = R$ and $\nexists Y : Y \subset X$ such that $Y^+ = R$.

To evaluate the candidate keys for a relation R with n attributes, the trivial algorithm, in the literature, examines the closure of combinations of attributes, until all candidate keys are found. If a candidate key X is found, then all its supersets are omitted for further examination as they are determined as superkeys.

While our focus is not on efficiency, we applied simple optimizations to the trivial algorithm, by considering only candidate keys that contain all attributes that do not appear in the right side of any of the FDs in F, if such attributes exist. We also used an efficient binary scheme to enumerate all possible attribute combinations for consideration.

Data: R, F
Result: Set of relations derived from R in BCNF
for *each* $X \rightarrow Y \in F$ **do**
  if $X^+ \neq R$ **then**
    $R1 \leftarrow X \cup Y$;
    $R2 \leftarrow R - Y$;
    return R1,R2 ;
  end
end
return R ;

Algorithm 1: BCNF Decomposition

Edgar Codd, along with Raymond Boyce, in 1974, redefined the 3rd normal form, upgrading it into the "3.5" normal form [6]. This normal form is well known as *Boyce-Codd Normal Form* or simply BCNF. While the third normal form, 3NF, is also very popular, we selected to focus our tool on BCNF as it ensures that all redundancies based on functional dependencies have been removed from the relational schema. BCNF normalization constitutes a part of every database course curricula, being one of the fundamental concepts of relational database design.

Given a relation R and a set of FDs F, for R to conform to BCNF, every functional dependency in F must contain a superkey for R in its left side. If there is a dependency that violates this rule, then R is not in BCNF and must be decomposed. Algorithm 1 selects one of the FDs that violates BCNF and R is split into two new relations satisfying the lossless join property. The resulting relations are examined using Alg. 1 recursively and more decompositions may arise until all remaining tables are in BCNF.

4. NORMALIZATION SOFTWARE TOOL

The normalization tool we present is implemented in C# following the principles of object-oriented programming.

4.1 Design Goals and Requirements

We designed our software tool for educational purposes. We intend for it to be useful to database course instructors as a teaching aid to demonstrate the normalization process through multiple examples and case studies. Furthermore, we intend for the tool to be used by the students themselves to practice on examples and homework assignments.

To this end, we set the following non-functional requirements that our software tool satisfies. Firstly, the software is simple and easy to use, with a self-explanatory GUI. We selected to use a windowing environment with typical menus and command buttons so as to ensure that users will familiarize themselves quickly with its main operations. Secondly,

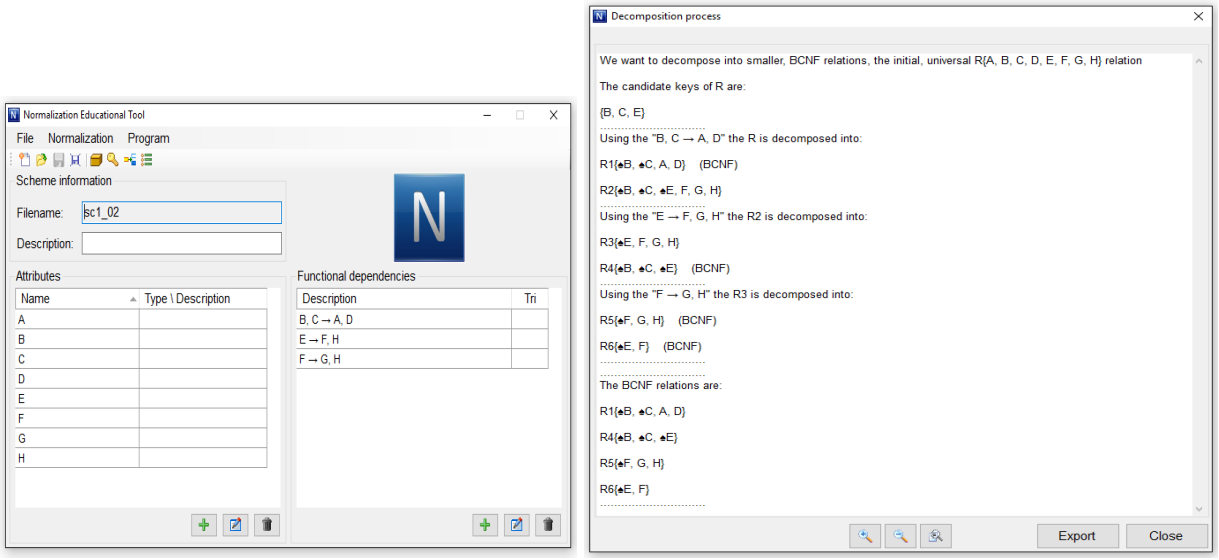

Figure 1: (left) Loading a database schema and (right) automatic decomposition.

it reduces the possibility of human errors by providing mechanisms that enforce basic checks to user input. Particularly when used by students for self-training, guiding them towards the right direction through corresponding error messages is very helpful. Thirdly, the source code is optimized with the use of appropriate data structures and it is also commented and structured so as to be easily extensible. Finally, the results of all operations are explained in detail.

4.2 Basic Functionalities

The basic use of our tool is: Given a relation R and a set of functional dependencies F to derive a set of appropriate normalized relations in BCNF. We are going to demonstrate the supported functionalities through a running example: consider a relation $R(A, B, C, D, E, F, G, H)$ and a set of FDs $F = \{BC \rightarrow AD,\ E \rightarrow FH,\ F \rightarrow GH\}$.

Initially, the user can create a new relation by enumerating its attributes and optionally specifying a type and a description for each. After defining at least two attributes, the user can define one by one the FDs in F. The specification of the dependencies is assisted by the GUI enabling the user to select the appropriate attributes for the left and right part of each functional dependency. A preview window enables users to check the formed dependency.

At any point the user may save the newly created schema. User-defined schemas are saved and can later be loaded, either to continue their editing or to perform normalization. Figure 1(left) is the main window of our tool and after creating or loading a schema, a user can now proceed to perform normalization.

As the first step of the normalization process is always determining the candidate keys, the tool provides the corresponding functionality producing a result screen with all candidate keys, i.e. (B, C, E) for our example.

While performing normalization, our tool selects a candidate key as a primary key for each relation and denotes all primary attributes in the result screens with a spade symbol. If there is more than one candidate key, the shortest one is chosen and between keys with the same length the choice is random.

Additionally, our tool provides an operation that evaluates the closure for any attribute combination the user selects. The results demonstrate the process in detail to assist the students in understanding how they are produced. With this functionality students can attempt to find the candidate keys for the relation without using the automatic option or better understand why a combination they consider a key is not. For this reason, messages that explain superkeys are also included as this is one of the subtleties many students have difficulty understanding.

The "automatic" decomposition presents the normalization process in detail but without giving the user any chance to intervene (Fig. 1(right)). In this case, the FDs are examined in the order they were defined by the user. All results can be exported in text or html format so that students and instructors can easily edit them, add notes or any other comment they want.

The most important functionality of our tool is the alternative, interactive step-by-step decomposition process in which the user guides the normalization process selecting at each step from the available relations and dependencies the relation to be decomposed and the functional dependency to be used each time.

When selecting step-by-step decomposition a new window pops up and we can see one table with the initial, universal relation and a list of all our functional dependencies. If a functional dependency is trivial, it is denoted as such with a mark in a corresponding column. To start the step-by-step process, the user needs to select one relation and one functional dependency. By default at the first step, the initial, universal relation is selected by the program, so the user needs to choose a functional dependency. The preview command enables us to check the effect of the selected FD before applying the decomposition.

We can also edit our dependencies during this process so as to derive alternative decompositions on the fly, and explore different case studies. This enables instructors to address different students' questions without permanently altering the stored schemas.

Whenever a relation is successfully decomposed, two new

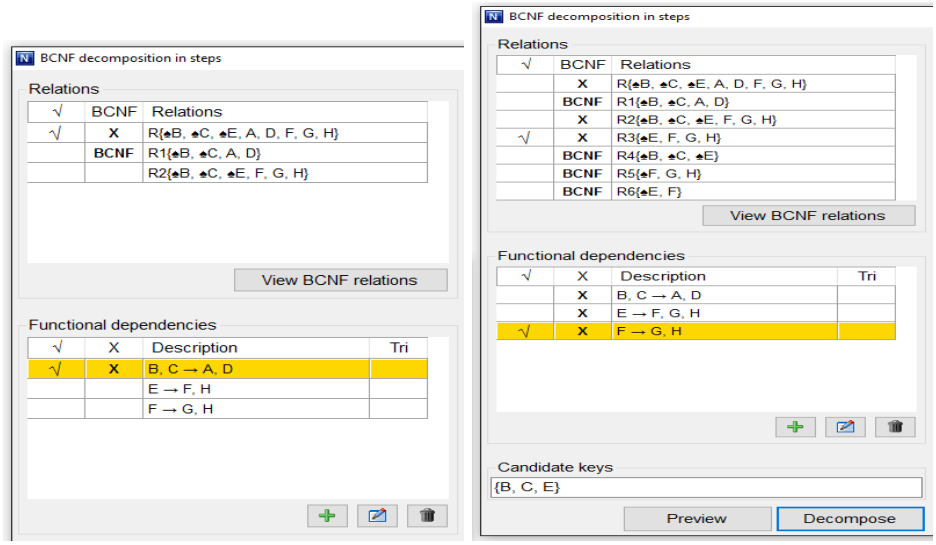

Figure 2: (left) Selection of second step and (right) final step for the decomposition.

relations are created and shown in the relations table. Figure 2(left) shows the result of decomposing R using $B, C \rightarrow A, D$ at the first step. In our example, the $R1$ relation is in BCNF, thus the BCNF mark appears and it can no longer be decomposed (though we can still preview a new decomposition that will create a corresponding error message). The $R2$ relation is not in BCNF and since there are two more functional dependencies available, we can try afterwards to decompose following the same process. In case a relation cannot be decomposed based on a chosen functional dependency, a message box appears explaining the reason. For the given example, we require three decompositions in total resulting in the six tables as detailed in Fig. 2(right), while the complete process explained is identical to the one derived from the automatic decomposition.

Note that if one wants to normalize an entire database schema and not just a single universal table she can do it by normalizing each table. Therefore, the tool can also be used to check whether a table or an entire database schema is already in BCNF. To check for alternative BCNF schemas one can provide our tool with the universal table and experiment with the interactive decomposition functionality so as to attain, for instance, BCNF schemas with a fewer number of tables if such exist.

5. CONCLUSIONS AND FUTURE WORK

We presented a software tool that automates the normalization of a relational database to the Boyce-Codd Normal Form. The tool is designed to both assist educators in explaining normalization, and enable self-training by students. It offers a simple interface, thorough explanations and instructions for each operation, and particularly an interactive step-by-step decomposition functionality.

As the tool is currently limited to BCNF decomposition, we plan to extend it to support other normal forms (i.e., 2NF, 3NF and 4NF). Also, we are going to deploy the tool in databases courses and monitor the results and feedback by the students to validate our tool's design goals. Finally, we will be providing the source code and software to anyone interested to either use or extend the tool.

6. REFERENCES

[1] Tool for database design. http://uisacad5.uis.edu/cgi-bin/mcrem2/database_design_tool.cgi.

[2] L. Ahmedi, N. Jakupi, and E. Jajaga. Normaldb - a logic-based interactive e-learning tool for database normalization and denormalization. In *eL&mL 2012*, pages 44–50, 2012.

[3] N. Arman. Normalizer: A case tool to normalize relational database schemas. *Information Technology Journal*, 5(2):329–331, 2006.

[4] W. Armstrong. Dependency structures of data base relationships. In *IFIP Congress*, 1974.

[5] R. Cho. Relational database tools. http://raymondcho.net/RelationalDatabaseTools/RelationalDatabaseTools.

[6] E. F. Codd. Recent investigations into relational data base systems. In *IFIP Congress*, 1974.

[7] Y. V. Dongare, P. S. Dhabe, and S. V. Deshmukh. Rdbnorma: - A semi-automated tool for relational database schema normalization up to third normal form. *CoRR*, abs/1103.0633, 2011.

[8] H. Du and L. Wery. Micro: A normalization tool for relational database designers. *Journal of Network and Computer Applications*, 22(4):215 – 232, 1999.

[9] A. Floyd and H. Du. Edna: a software tool for verifying normalisation of relations during logical database design process. In *12th HEA STEM Workshop on Teaching, Learning and Assessment of Databases*, pages 51–61, 2014.

[10] N. Georgiev. A web-based environment for learning normalization of relational database schemata. masters thesis, Umea university, Sweden, 2008.

[11] H. J. Kung and H. L. Tung. A web-based tool to enhance teaching/learning database normalization. In *SAIS 2006 Proceedings*, pages 251–258, 2006.

[12] A. Yazici and Z. Karakaya. Jmathnorm: A database normalization tool using mathematica. In *Computational Science – ICCS 2007, Part II*, pages 186–193, 2007.