Received: 5 July 2016 | Accepted: 9 April 2017
DOI: 10.1002/cae.21838

RESEARCH ARTICLE

WILEY

An educational software for teaching database normalization

Hugo I. Piza-Dávila | **Luis F. Gutiérrez-Preciado** | **Víctor H. Ortega-Guzmán**

Department of Electronics, Systems and Informatics, ITESO Jesuit University of Guadalajara, Tlaquepaque, Mexico

Correspondence
Luis F. Gutiérrez-Preciado, Department of Electronics, Systems and Informatics, ITESO Jesuit University of Guadalajara, Tlaquepaque, Mexico 45604.
Email: lgutierrez@iteso.mx

Abstract

Database normalization is a key process for designing databases. However, it is one of the most complex topics for students in IT undergraduate programs. This work proposes a new software tool that supports students in the process of normalization. This learning tool allows the students to see step by step how to carry out each normal form up to the 3NF. This tool is based on data analysis of the input table as it is explained in class, and not on analyzing the scheme or initial dependencies defined by business rules. Thus, novel algorithms were developed to process the input table (user-defined) as well as the tables created during the normalization process. The software was tested in two Database courses, obtaining an important increase in the performance of students who used the tool.

KEYWORDS

database normalization, functional dependency, learning support tool

1 | INTRODUCTION

Normalization is a process that takes an existing database design with some anomalies and redundancies, and improves it by decomposing relations, based on a known set of dependencies. When the normalization is not applied, database systems are usually inefficient and inaccurate. According to a questionnaire survey among engineering students [4], the most complex topics for students of careers related to Computer Science have been database normalization and relational algebra. The performance of most of the students in our Database Systems courses confirms the results of this survey.

In order to ease the students the comprehension of databases concepts and improve their skills in database design, several software tools have been developed on this direction. For instance, to learn the entity-relationship model [3], relational algebra for SQL [5], and to observe the process of database normalization with predefined examples [12], among other tools.

In this paper, we present a learning tool that explains gradually the normalization process from an input table (a csv file) with anomalies. The process implemented works with existing data instead of a given schema for the following reasons:

1. We consider that students can understand normalization better if they notice how redundancies are reducing by visualizing the new arrangements of the original data.
2. According to 1NF, the domain of each field must contain atomic values, but some tuples can have a set of these values. We can detect this anomaly only by analyzing data.
3. The students of any course related to database design must clear the concepts of *primary key* and *functional dependencies*. They can prove this knowledge if they are capable to identify them from any given dataset.

1.1 | Database normalization learning tools

Several existing educational tools perform database normalization. To mention a few, JMathNorm [14] is a tool developed over "Mathematica" and allows students to normalize a database from the description of a scheme and its functional dependencies given by the business rules. The tool generates the maximum set of functional dependencies, known as *closure*, and deletes the redundant dependencies obtaining the minimal cover. Besides, the tool was employed with students of a databases course, who used it to validate the normalization process. However, this tool works from a description and not from the initial table information. Another tool is called Normit [11], which is a tutoring tool that is based on self-explanation, that is, the student justifies each step of normalization and the systems support the students when the justification is wrong, which is performed by a comparison between the correct answers and the ones given by the student. However, the number of case studies are limited. Another kind of tutor-based normalization tool has been created using expert systems [8]. This tool attempts to create a learning environment by showing the normalization process adapted to each single student. However, the student introduces the initial scheme. A web-based tool [10] allows the student to introduce a scheme and observe step-by-step the normalization process. However, the student must introduce each functional dependency to the system and, thus, making difficult to test more complex examples.

All these tools are based on the database scheme, required as input, to start the normalization process, that is, these tools do not consider the initial data, but only the fields and the relationships between them. However, students might learn easily with a different approach: by applying the rules of each normal form as they visualize the new arrangements of data. The proposed learning tool considers as much as possible the way the students learn normalization in class. The learning tool intends to help students validating the results obtained at each step of the normalization process, as well as understanding the reasons for each result. Since, the tool needs to carry out the normalization process, some related algorithms must be studied in order to see if some of them fulfill our needs.

1.2 | Normalization algorithms

Automating the normalization of databases has been of interest to researchers in recent years. The studies are intended to facilitate the representation of functional dependencies, the identification of candidate keys, the selection of primary keys, and the efficiency of the algorithms to normalize the tables. There are algorithms capable to find a primary key among all the candidate keys efficiently since they are based on dependency graph; for instance [1] uses the dependency matrix and the matrix of a directed graph to generate the matrix of transitive dependencies. Other proposals are usually semi-automatic, receiving as input the descriptions of the basic functional units, and using a structure of linked lists in order to link functional dependencies [7]. This strategy reduces memory and increases processing speed. Another proposal that allows users normalize up to 3NF considers all possible candidate keys [6]. In a first stage, it eliminates functional dependencies and then normalizes relations using fields as input. Bernstein's algorithm synthesizes 3NF relations from functional dependencies [2], by finding a minimal cover for a set of functional dependencies [9]. Finally, [13] introduces a PROLOG-based normalization tool that records data obtained from a relation in the form of facts. Most of state-of-the-art normalization algorithms require a database schema to carry out efficiently the normalization process. However, they do not provide step-by-step feedback of the process and views of new data arrangements, which we consider key elements for learning normalization.

The contributions of the proposed tool are the following:

1. It helps the students identify data redundancies and anomalies by providing views of the resulting tables as each normal form is applied.
2. It provides a description of the violations found at each normal form, as well as how to address each one of them. These violations include the following: duplicate records, multivalued fields, partial dependencies to the primary key, and transitive dependencies.
3. It supports the students to discover the primary key and functional dependencies allowing them to reinforcing or proving their knowledge in these main aspects.
4. It is not limited to certain predefined tables, it is possible to create a new input table following a comma-separated format file and uploading it to the application.
5. It keeps a log of activities per student; therefore, it is possible to monitor the usage of the application.

2 | DATABASE NORMALIZATION

The normal forms make possible to find a set of fields that uniquely identify tuples at a table; such set of fields is known as the *primary key* of the table. Normal forms include the following: First normal form (1NF), Second Normal form (2NF), Third normal form (3NF), Boyce-Codd normal form (BCNF), Fourth normal form (4NF), Fifth normal form (5NF), and Domain-Key normal form (DKNF). In most of the cases, transforming a database into 3NF would be enough to avoid insertion, deletion, and update anomalies, since they contemplate the most common issues related to data redundancy.

The 1NF makes sure that a table holds the properties of a *relation*. Thus, a table in 1NF has the following properties:

1. No cell has more than one value.
2. All the values at the same column belong to the same domain (integers, dates, strings, ...).
3. There are no duplicate groups of data or rows.
4. Each column has a unique title denoting field names. If a column is untitled, we consider it as part of a multivalued field whose name is on the first titled column to the left.
5. The table has a primary key that might be composed by one or more columns. The value of the primary key on each row is unique.

PLATFORM				CONDITION	GENRE	PRICE	TITLE	RATING CATEGORIES	PUBLISHER
						X			
X360	PS3	WII	PS4	NEW	Adventure	799	THE LEGO MOVIE VIDEOGAME	Everyone	WARNER BROS GAMES
X360				USED	Estrategy	260	MINECRAFT	Everyone	MOJANG
X360	PS4			NEW	Estrategy	600	MINECRAFT	Everyone	MOJANG
PS4				NEW	Shooter	999	CALL OF DUTY GHOSTS	Mature	ACTIVISION
XONE	PS4			NEW	Shooter	999	CALL OF DUTY GHOSTS	Mature	ACTIVISION
XONE	X360	PS3		NEW	Vehicles	999	NEED FOR SPEED RIVALS	Teen	ELECTRONIC ARTS
3DS				USED	Party	680	MARIO PARTY ISLAND TOUR	Everyone	NINTENDO
PSV	PS3	X360		USED	Fights	680	INJUSTICE GODS AMONG US GOTY	Teen	WARNER BROS GAMES

FIGURE 1 Tabular view of the input table Videogames

In this work, we assume that the input dataset holds the first three properties, since they represent no difficulty for students to solve. Therefore, the algorithm proposed focuses on achieving properties 4 and 5, that is, handling multi-valued fields and finding the primary key.

The focus of the 2NF is to find functional dependencies between non-key fields and the primary key. Thus, in order to consider a table T in 2NF, it has to follow two rules:

1. T is in 1NF.
2. Every non-key field is fully functionally dependent on the primary key of T.

For every partial key dependency found $<k, f>$, the non-key field f is removed from T and added to a new (or existing) table T' where k is the primary key.

A table T is in 3NF if the following holds

1. T is in 2NF.
2. T does not have transitive dependencies, that is, there are not non-key fields functionally dependent on another non-key field.

For every functional dependency found $<f_1, f_2>$, the non-key field f_2 is removed from T and added to a new (or existing) table T' where f_1 is the primary key.

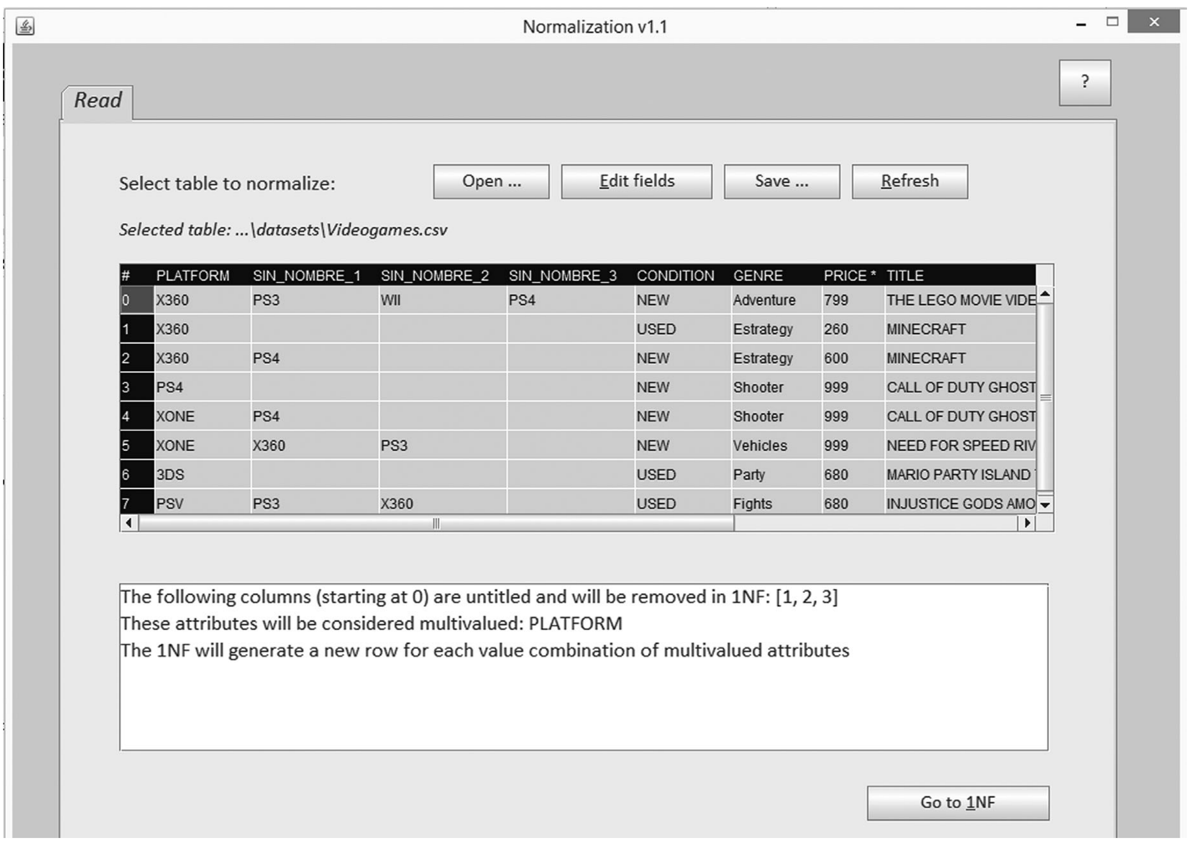


FIGURE 2 Read-table window

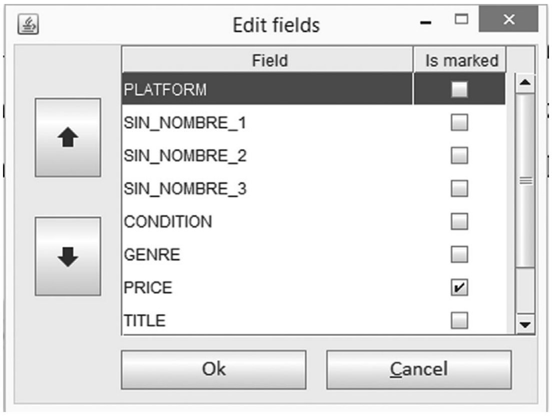


FIGURE 3 Edit-fields window

3 | NORMALIZATION ALGORITHM

The present section introduces the algorithms proposed to handle every step of the normalization process up to 3NF, as well as some definitions and notations employed.

3.1 | Definitions

A table T has fields, rows of data, a primary key, and functional dependencies. Prior to the first normal form, fields are referred as columns. Some of the fields may be *marked* which means that they are not candidate to be part of a primary key.

$T\,[r, f]$ refers to the value stored at row r, field f in table T. The primary key of a table T is a set of fields defined by no other field at T. A functional dependency is a pair $<\varphi, f>$, such that f is a field defined by a nonempty set of fields φ. Finally, a database Db is a set of tables, such that there is no two tables with the same primary key. Algorithm Main enlists the steps required to normalize a given table up to the Third Normal Form. Each step is explained in detail along the present section.

Algorithm Main. Main algorithm.

Input: a table T.

Output: the corresponding database in Third Normal Form.

1. $T \leftarrow$ Load from file
2. $T_{1NF} \leftarrow To1NF(T)$
3. $Db_{2NF} \leftarrow To2NF(T_{1NF})$
4. $Db_{3NF} \leftarrow \{\}$
5. For each table T in Db_{2NF} do:
   a. $Db_{3NF} \leftarrow Db_{3NF} \leftarrow To3NF(T)$
6. Return Db_{3NF}

3.2 | Functional dependencies

Finding functional dependencies is a common task for all the steps enlisted before. Thus, prior to the algorithms, we provide the following definitions and functions.

Definition 1. Let f_1, f_2 be two fields from a table T. We say that f_2 is **functionally dependent upon** f_1, or simply f_1 **defines** f_2 if no two tuples exist in T with the same value at f_1 but different value at f_2.

The following function determines if f_1 defines f_2. The key idea is to find a contradiction, that is, two rows with the same value at f_1 but different value at f_2.

DefinesFF $(\boldsymbol{T}, \boldsymbol{f_1}, \boldsymbol{f_2})$ returns:

false, there exist two rows r_1, r_2 such that: $T\,[r_1, f_1] = T\,[r_2, f_1]$ and $T\,[r_1, f_2] \neq T\,[r_2, f_2]$.

true, otherwise.

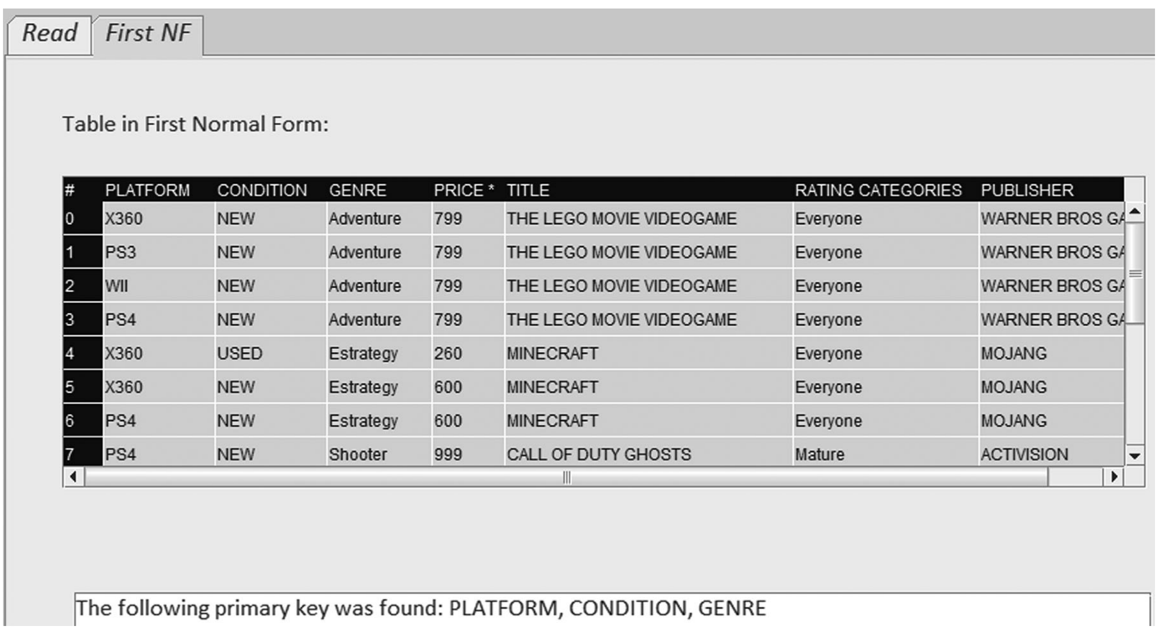


FIGURE 4 Table Videogames in 1NF. The fields in red make up the primary key

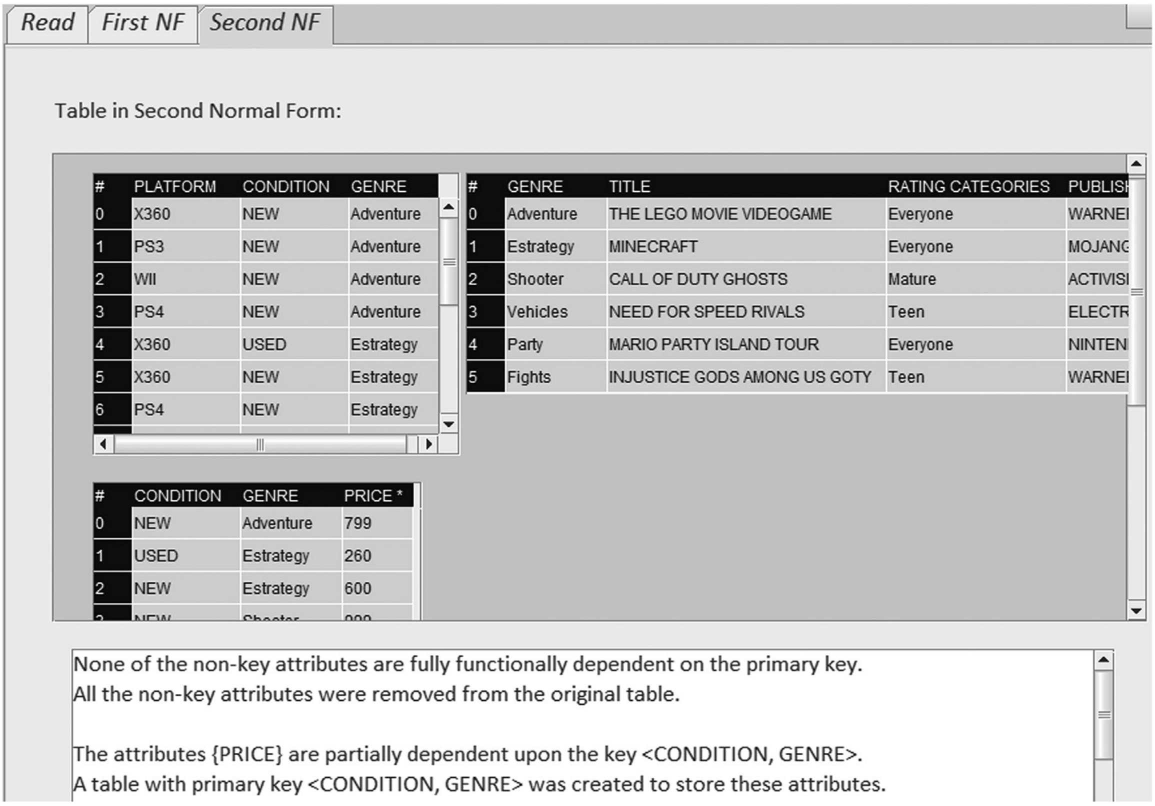


FIGURE 5 Tables resulting from converting Videogames into 2NF

Definition 2. Let f and F be, respectively, a field and a set of fields from a table T. We say that f is **functionally dependent upon** F, or simply F **defines** f if no two tuples exist in T with the same value at each field from F but different value at f.

The following function determines if a set of fields F_1 defines a field f_2. The key idea here is to find two rows with the same values at all the fields in F_1, but different value at f_2.

DefinesSF $(\boldsymbol{T}, \boldsymbol{F_1}, \boldsymbol{f_2})$ returns:

false, there exist two rows r_1, r_2 such that: $\forall f \in F_1$, $T[r_1, f] = T[r_2, f]$ and $T[r_1, f_2] \neq T[r_2, f_2]$.

true, otherwise.

3.3 | Primary key discovery

A primary contribution of this work is the fact that the input dataset does not include information about the primary key: an algorithm has to be developed. Algorithm FindPK finds a set of non-marked fields from a table T such that none of them is functionally dependent upon another field in T. It defines three sets of field indexes:

1. *Andf* contains fields that are not functionally dependent upon other(s) at each step of the algorithm. Initially, this set contains all the fields.
2. *Adef* is the complement of *Andf*.
3. *Ap* contains fields candidate to be the primary key. Unlike *Andf*, *Ap* does not include marked fields.

The algorithm iteratively builds field subsets of length 1 to *field-count*. Subsets containing fields already in *Adef* are discarded. For each field f in *Andf* functionally dependent to the current subset is removed from *Andf* and *Ap*, and added to *Adef*. Such dependency is added to table T since it will be used in further steps. At the end, *Ap* contains only non-marked fields not defined by any other, that is, the primary key.

Algorithm FindPK find the primary key of a table and store it inside the table.

Input: a table T.

1. Let *Andf* be a set containing the fields at T.
2. Let *Ap* be a set containing all the non-marked fields at T.
3. Let *Adef* ← Ø
4. Let $l \leftarrow 1$
5. For each $\varphi \subset Ap$ such that: $|\varphi| = l$, $\neg\exists f \in (\varphi \cup Adef)$:
    a. For each $f \in Andf \setminus \varphi$ and *DefinesSF* (T, φ, f):
        i. $Adef \leftarrow Adef \cup \{f\}$
        ii. Add the pair $<\varphi, f>$ to the functional dependencies in T.
    b. For each $f \in Adef$:
        i. $Andf \leftarrow Andf \setminus \{f\}$
        ii. $Ap \leftarrow Ap \setminus \{f\}$
6. $l \leftarrow l + 1$
7. If $l < |Andf|$ goto step 5
8. Set *Ap* as the primary key of T.

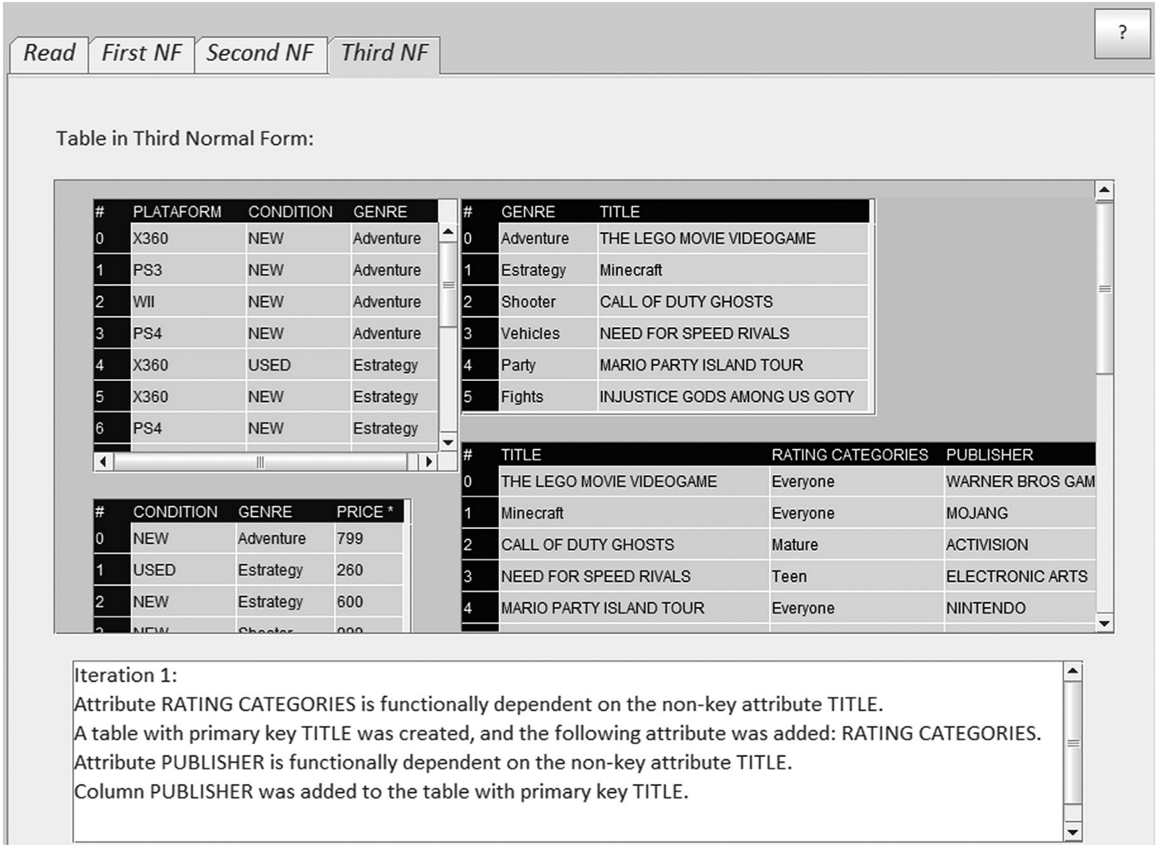


FIGURE 6 Database normalization up to 3NF. The tables are free from transitive dependencies

3.4 | First normal form

Algorithm 1NF converts a given table into the First Normal Form. The main purpose of this algorithm is to handle the multivalued fields. For a better understanding of the algorithm, consider the 4-column table $T = <a, ?, b, ?>$ containing two multivalued fields. The corresponding table in first normal form T' should have only two fields and more rows. Thus, for each row $<r_1, r_2, r_3, r_4>$ in T, the following rows are added to T': $<r_1, r_3>$, $<r_2, r_3>$, $<r_1, r_4>$, $<r_2, r_4>$, corresponding to all the two-tuples that can be built with all the different values at columns corresponding to a and all the different values at columns corresponding to field b. At the end of the algorithm, the primary key of the table is found.

The following sets and functions are employed in the algorithm:

Let $C = \{1, 2, \ldots, M\}$ be an ordered set of index columns at table T.

Let $D = \{d_1 = 1, d_2, \ldots, d_N\}$ be an ordered set of titled columns, such that each $d_i \in C$.

Let $F: D \to 2^C$ be a function that maps titled columns with one or more consecutive columns, such that: $F(d_k) = \{d_k, d_k + 1, \ldots, d_{k+1} - 1\}$

Algorithm 1NF convert a given table into the First Normal Form

Input: a table T.

Output: the corresponding table in First Normal Form

1. Let T_{1NF} be a new empty table with N columns
2. For each row r in T, do:
   a. For each $<c_{i1}, c_{i2}, \ldots, c_{iN}>$ such that: $c_{ik} \in F(d_k)$, $d_k \in D$, do:
      i. Let $r' = <T[r, c_{i1}], T[r, c_{i2}], \ldots, T[r, c_{iN}]>$
      ii. Add r' to T_{1NF}
3. *FindPK* (T_{1NF})
4. Return T_{1NF}

3.5 | Second normal form

Algorithm 2NF converts a given table into the Second Normal Form. The focus of this algorithm is to handle partial dependencies to the primary key. The result of this algorithm is not a table but a database, that is, a set of tables, because partial dependencies has to be removed from the original table and added to new ones.

Algorithm 2NF convert a given table into the Second Normal Form

Input: a table *T*.

Output: the corresponding database in Second Normal Form

1. Let *k* be the primary key of *T*
2. Let D_T be the set of functional dependencies in *T*
3. Let D_k be an initially empty set of partial dependencies to *k*
4. For each dependency <φ, *f*> in D_T, do:
   a. If $\varphi \subset k$ then $D_k \leftarrow D_k \cup \{<\varphi, f>\}$
5. Let *Db* be an initially empty database
6. For each dependency <φ, *f*> in D_k, do:
   a. Get table T_φ from *Db* such that φ is the primary key
   b. If there is no such table, create it and add it to *Db*
   c. Add field *f* to T_φ
   d. Remove column *f* from *T*
7. For each table *T*' in *Db*, do:
   a. Copy data from *T* to *T*' considering only the fields at *T*'
   b. Remove repeating rows from *T*'
8. Add *T* to *Db*
9. Return *Db*

3.6 | Third normal form

Algorithm 3NF converts a given table into the Third Normal Form. The focus of this algorithm is to find functional dependencies between non-key fields such that the definer field is not marked. For each dependency found, The defined field is removed from the table and moved to another table which primary key is the definer field. This process is run iteratively over all the tables created until no functional dependencies between non-key fields are found.

Algorithm 3NF Convert a given table into the Third Normal Form

Input: a table *T*.

Output: the corresponding database in Third Normal Form

1. Let *Db* be an initially empty database
2. For each pair of fields *f*, *g* of *T* not in the primary key ($f < g$), do:
   a. If *f* is not marked and *DefinesFF* (*T*, *f*, *g*) then *key* ← *f*, *defined* ← *g*
   b. Else if *g* is not marked and *DefinesFF* (*T*, *g*, *f*) then *key* ← *g*, *defined* ← *f*
   c. Else goto step 2
   d. Get table T_k from *Db* such that *key* is the primary key
   e. If there is no such table then
      i. Create T_k and add it to *Db*
      ii. Copy values at field *key* from *T* to T_k
   f. Add field *defined* to T_k
   g. Copy values at field *defined* from *T* to T_k
   h. Remove *defined* from *T*
3. Add *T* to *Db*
4. Remove repeating rows from each table *T'* in *Db*
5. If *Db* has more than one table then
   a. Let *Db*' be a new database
   b. For each table *T*' in *Db*, copy each table in *To3NF*' (*T*') to *Db*'
   c. Return *Db*'
6. Else return *Db*

4 | USING THE TOOL FOR LEARNING NORMALIZATION

In the classroom, the normalization process is explained to students considering the input table data instead of its scheme. Each student must be able to identify the rules that have to be applied step by step in order to translate the table into 3NF. Later, the instructor gives the students several exercises with different level of complexity allowing them to observe and understand how to apply each normal form. The presented tool seeks to support this process.

The tool was developed as a desktop application using Java SE 7. Each student and teacher uses his/her ID in order to have access to the tool. Previously, all of them were registered in a MySQL database that also stores the usage of the tool.

The tool allows students to load a table given in CSV format; immediately the tool, displays the initial table and identifies the multivalued fields and duplicate rows. We assume the initial table has enough data diversity to identify functional dependencies. Figure 1 depicts a tabular view of an input table: Videogames. The first row corresponds to the name of the fields; the second row indicates the marked fields using character X; the remaining rows store the table data. Multivalued fields cover more than one column, but only the first one is titled.

The student can select one or more fields which, according to a specific context, should not be considered part of a primary key of any table eventually, since no other field is functionally dependent upon them even though the available data suggest otherwise. Typical cases include money, date, and time values, but also description fields such that there exist another field that identifies them. Figure 2 shows how the student can mark and unmark fields using the Edit-fields window. This selection can be done every time the user desires, but the normalization process is restarted.

In addition, this window allows the user to change the order of the fields. This is particularly important when the diversity of data is not enough for an algorithm to determine that a field should not define another. For instance, consider a table containing the fields: *CustomerId* and *CustomerName*; it is clear that *CustomerName* is functionally dependent upon *CustomerId*, and not vice versa; however, if all the given customer names are different, any of the fields can define the other. In such a case, *CustomerId* should be placed after *CustomerName* so that the functional dependence *CustomerName* –> *CustomerId* is not tested.

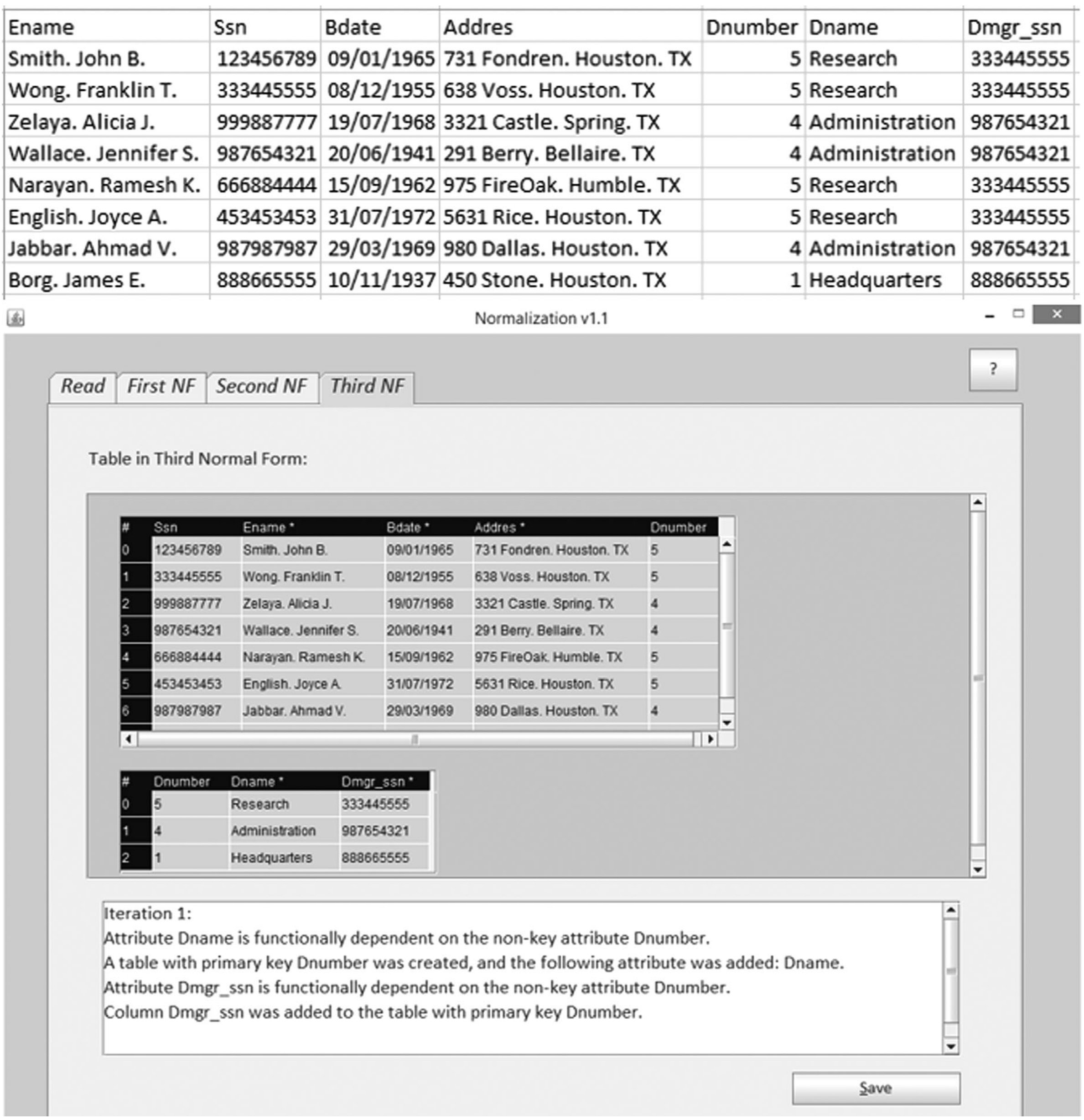
Figure 3 shows the feedback provided by the tool after loading the table Videogames. There we can see that field *Platform* is multivalued since it covers up to four columns.

Figure 4 shows the results of applying the 1NF to table Videogames. The primary key <Platform, Condition, Genre> was detected and there are no longer multivalued fields.

Figure 5 shows the results of applying the 2NF to table Videogames: all the non-key fields must be functionally dependent upon all the key fields. Since field Price depends only on key fields *Condition* and *Genre*, it was removed from the original table and added to a new table with primary key <*Condition, Genre*>. Similarly, the fields *Title, Rating Categories*, and *Publisher* are functionally dependent upon the key field Genre. Thus, they were removed from the original table and added to a new table with primary key <*Genre*>. At the end, the original table kept only the key fields.

Figure 6 shows the results of applying the 3NF to each of the tables created at 2NF. We can see that a new table with primary key <*Title*> was created. This is because the table with primary key <*Genre*> contains in 2NF two non-key fields, *Rating Categories*, and *Publisher*, which are functionally dependent upon another non-key field, *Title*.

The resulting tables might still have transitive dependencies, that is, a non-key field defines one or more non-key fields. Thus, the previous process must run as many times (or iterations) as necessary to remove all the transitive dependencies at the new tables created.



Ename	Ssn	Bdate	Addres	Dnumber	Dname	Dmgr_ssn
Smith. John B.	123456789	09/01/1965	731 Fondren. Houston. TX	5	Research	333445555
Wong. Franklin T.	333445555	08/12/1955	638 Voss. Houston. TX	5	Research	333445555
Zelaya. Alicia J.	999887777	19/07/1968	3321 Castle. Spring. TX	4	Administration	987654321
Wallace. Jennifer S.	987654321	20/06/1941	291 Berry. Bellaire. TX	4	Administration	987654321
Narayan. Ramesh K.	666884444	15/09/1962	975 FireOak. Humble. TX	5	Research	333445555
English. Joyce A.	453453453	31/07/1972	5631 Rice. Houston. TX	5	Research	333445555
Jabbar. Ahmad V.	987987987	29/03/1969	980 Dallas. Houston. TX	4	Administration	987654321
Borg. James E.	888665555	10/11/1937	450 Stone. Houston. TX	1	Headquarters	888665555

FIGURE 7 Test of using the normalization tool with an example taken from [12]

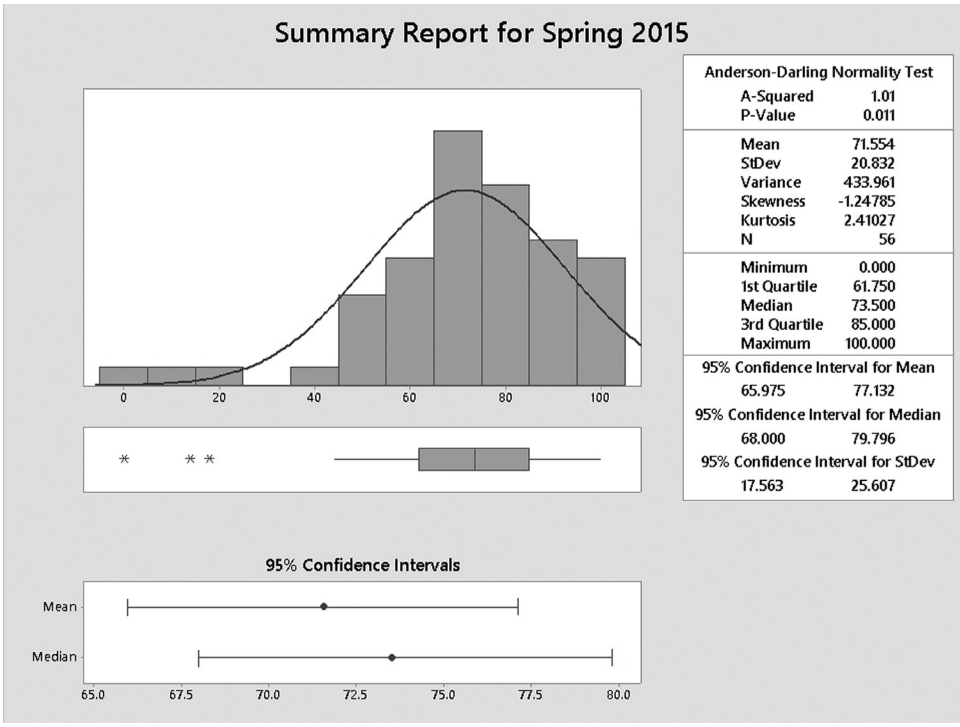


FIGURE 8 Grades obtained without the normalization tool

Once the initial table is converted into the Third Normal Form, the resulting database (set of tables) is now free from insertion, deletion, and/or update anomalies, and can be exported to a CSV file which contains all the tables that were created in the normalization process.

5 | RESULTS

Two criteria were employed to test the proposed tool:

1. The effectiveness of the algorithms proposed, and
2. The impact of the tool as a learning support.

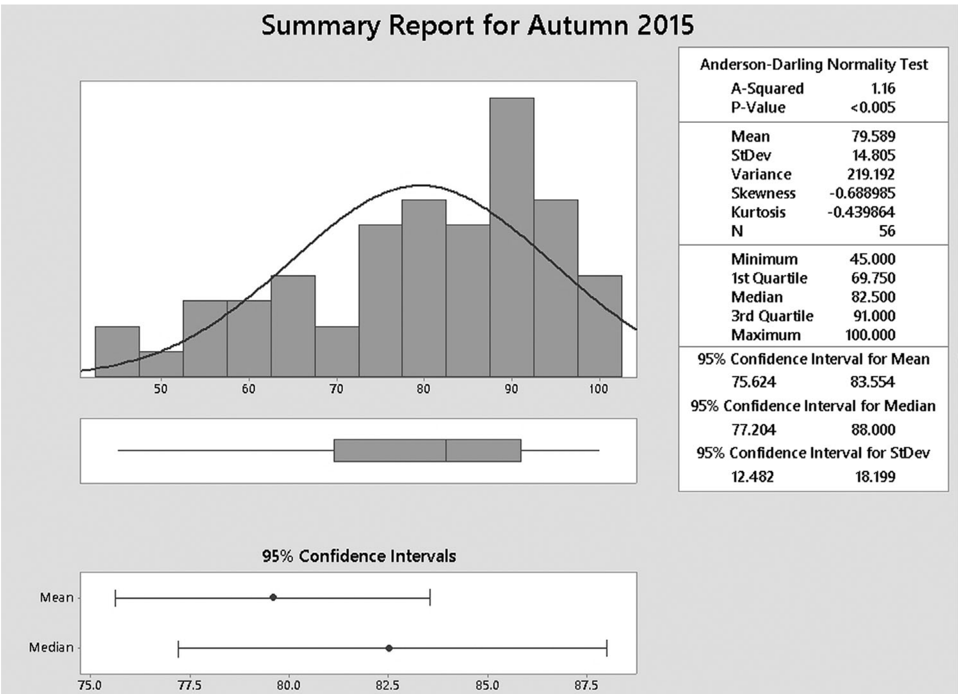


FIGURE 9 Grades obtained considering the normalization tool

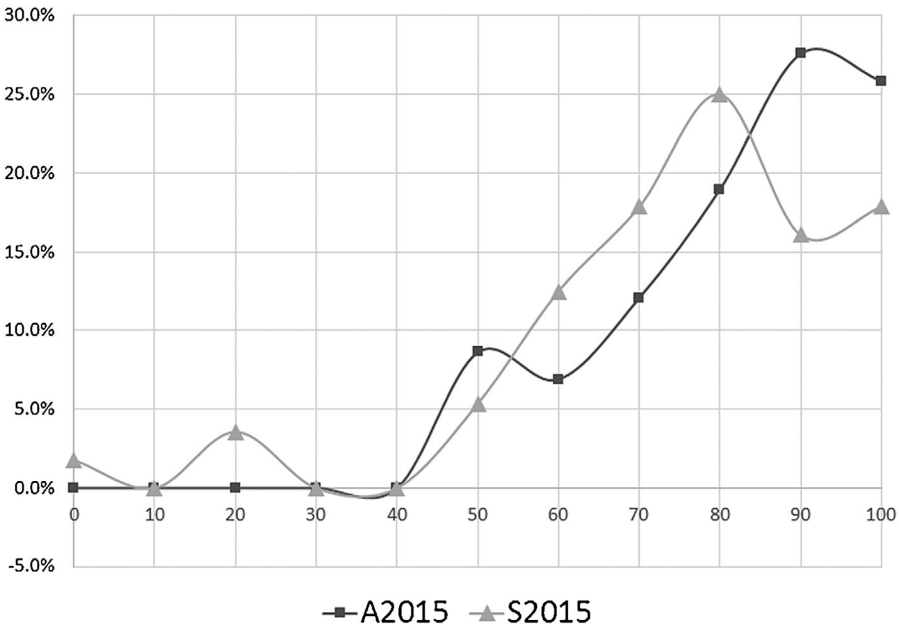


FIGURE 10 Comparing the grades of both periods, Spring 2015 and Autumn 2015

In order to prove that the developed algorithms produces the expected solution, some non-normalized tables taken from the book [9] were used. In all the test cases, the expected results were achieved, considering a minimal set of fields to mark. Figure 7 presents an example from this book as well as the resulting tables in 3NF produced by our tool.

During the term Spring 2015, a midterm test specific to evaluate the normalization process was applied to 56 students enrolled in the Databases Systems course, representing a sample of 90% of the total. The test consisted in normalizing a given table, including the results at each step up to the 3NF. The grade distribution of the students can be observed in Figure 8. During the term Fall 2015, the tool was delivered to the teachers and students of this course, and the same written test was applied to the non-repeating students considering the same sample size as in spring 2015. The grade distribution is shown in Figure 9.

It is notable how the grades have increased after endowing the students with the normalization tool which was employed during the teaching learning process (see Figure 10). According to the recorded logs, the tool was used 6,609 times, indicating an average of 103 times per student.

6 | CONCLUSIONS AND FUTURE WORK

Although database normalization is an essential skill for IT professionals, enhancing student performance on this topic is key for developing better software engineers. We developed a support-learning tool that allows students to observe step-by-step normalization process from the initial state to 3NF. The proposed tool differs from other tools because it applies the rules on an input table (CSV file) instead that on a defined structure. In addition, novel algorithms have been proposed that make possible the identification of functional, partial, or transitive dependencies by analyzing directly the input data; these algorithms are capable of finding the primary key at each normal form. The proposed tool helped the students to understand the normalization process obtaining a considerable improvement in grades with respect to the students that did not use this tool. Future work includes the incorporation to the tool of the following normal forms: Boyce-Codd, 4NF, and 5NF.

REFERENCES

1. A. H. Bahmani, M. Naghibzadeh, and B. Bahmani, Automatic database normalization and primary key generation, *Canadian Conference on Electrical and Computer Engineering 2008*, IEEE (2008), 11–16.
2. P. Bernstein and A. Philip, *Synthesizing third normal form relations from functional dependencies*, ACM Trans. Database Syst. (TODS), **1** (1976), 277–298.
3. P. Chen, *The entity-relationship model—toward a unified view of data*, ACM Trans. Database Syst. (TODS), **1** (1976), 9–36.
4. M. Czenky, *The efficiency examination of teaching of different normalization methods*, Int. J. Database Manag. Syst. **6** (2014).
5. C. Date and D. Hugh, *A Guide To Sql Standard*, 4th ed., Addison-Wesley, New York, 1997.
6. M. Demba, *Algorithm for relational database normalization up to 3NF*, Int. J. Database Manag. Syst. **5** (2013), 39–51.
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*, Int. J. Database Manag. Syst. **3** (2011), 133–154.

8. P. Douglas and S. Barker, *Dependency theory e-learning tool. Information Technology: Coding and Computing, Proceedings*. ITCC 2004, **1** (2004), 151–155.
9. R. Elmasri and S. B. Navathe, *Fundamentals of database systems*, 7th ed, Pearson Education, India, 2016.
10. H. Kung and H. L. Tung, A Web-based tool to enhance teaching/learning database normalization. *Proc. of the 2006 Southern Association for Information Systems Conference*, (2006), 251–258.
11. A. Mitrovic, *Scaffolding answer explanation in a data normalization tutor*, Facta Universitatis – Series: Electron. Energ. **18** (2005), 151–163.
12. M. Murray and M. Guimaraes, *Animated database courseware (ADbC): interactive instructional materials to support the teaching of database concepts*, J. Comput. Sci. Colleges, **24** (2009), 267–267.
13. T. Welzer, I. Rozman, and J. Györkös, *Automated normalization tool*, Microprocess. Microprogramming, **25** (1989), 375–380.
14. A. Yazici and Z. Karakaya, *JMathNorm: A database normalization tool using mathematica*. Computational Science–ICCS 2007, Springer Berlin Heidelberg, 2007.

H. I. Piza-Dávila received his BS in Computer Systems in 2000 from Instituto Tecnologico de Colima, Colima, Mexico, and his MS degree and PhD degree in Computer Science in 2002 and 2006, from Centro de Investigacion y de Estudios Avanzados del IPN, Mexico. He is currently full time professor-researcher in the Instituto Tecnologico y de Estudios Superiores de Occidente, Guadalajara, Mexico. His research interests include pattern recognition, machine learning, and parallel algorithms. He has published several papers in journals and conference proceedings.

L. F. Gutiérrez-Preciado received his BE degree in Computer Systems from Autonomous University of Aguascalientes, Mexico, in 2007. He obtained his MSc in Computer Science and his PhD from CINVESTAV Guadalajara in 2009 and 2013 respectively. Currently, he is a professor at ITESO University in the department of Electronics, Systems, and Informatics. His research interests include structured and unstructured data analytics and virtual reality applications.

V. H. Ortega-Guzmán has a Master's Degree in Computer from the UNIVA University. He also has a specialization in Business Intelligence: Data Warehouse y Data mining from the University Oberta of Cataluña (UOC) and another in Business Process Management from ITESO University. Currently, he is a student of the PhD in Engineering Sciences at ITESO and professor at the same University. His research area is focused on databases, business intelligence, and graph databases.

How to cite this article: Piza-Dávila HI, Gutiérrez-Preciado LF, Ortega-Guzmán VH. *An educational software for teaching database normalization*. *Comput Appl Eng Educ*. 2017;1–11. https://doi.org/10.1002/cae.21838