EDNA, a Software Tool for Verifying the Normalisation of Relations during the Logical Database Design Process

Conference Paper · July 2014

CITATIONS
2

READS
348

2 authors:

Adam Floyd
The University of Buckingham
1 PUBLICATION **2** CITATIONS
SEE PROFILE

Hongbo Du
The University of Buckingham
35 PUBLICATIONS **101** CITATIONS
SEE PROFILE

EDNA, a Software Tool for Verifying the Normalisation of Relations during the Logical Database Design Process

Adam Floyd
Department of Applied Computing
University of Buckingham
Buckingham MK18 1EG
adam.floyd@buckingham.ac.uk

Hongbo Du
Department of Applied Computing
University of Buckingham
Buckingham MK18 1EG
hongbo.du@buckingham.ac.uk

Abstract

Logical design of relational databases is an important part of a typical undergraduate database module in a range of computing curricula such as software engineering, computer science, computing, business computing, information systems, etc. A crucial part of logical design is normalisation, which is often perceived as a difficult topic to study by students and not so easy to teach by lecturers. This paper presents a software tool named EDNA that aims to automate the normalisation of relation schemes to 2NF, 3NF or BCNF. Upon user request, EDNA can generate standard SQL statements necessary for the creation of tables of the normalised schemes for a number of designated database management systems. EDNA is equipped with a simple interface that allows the user to enter a list of attributes and specify or edit functional dependencies among the attributes with ease. The software can be used as a teaching tool for verifying the validity of manually normalised relation schemes, aiming to enhance the student learning experience of the topic.

Keywords

Relational database, relational schemes, logical database design, normalisation.

1. INTRODUCTION

The relational data model is still the dominant form of data model used in current database systems. The relational database therefore remains a central theme of a typical undergraduate database module at the introductory level, and is covered by most texts on databases, such as Date (1995) and Elmasri & Navathe (2011). One important aspect of relational database theory is database design through normalisation.

Normalisation is a formal process of decomposing loosely organised and often large relation schemes into smaller and more tightly coupled structures according to a set of constraints on data such as functional dependencies between attributes (Elmasri & Navathe 2011). The higher the normal form relation schemes are, the more constraints are imposed. The process of normalisation aims to reduce the amount of data redundancy and avoid the associated update anomalies in the practical use of the final database.

Many students consider normalisation a hard topic to learn, and many lecturers find it not so easy to teach. Although attempts have been made in the past to dispense with normalisation by adopting the Object Role Modelling (ORM) methodology (Byrne et al. 2003) or enhancing "good" conceptual designs (Byrne 2003), such an approach of database

design without normalisation has not been endorsed by the mainstream (Nelson & Jayakumar 2012). Normalisation remains a fundamental topic within the relational database design that is being taught in classrooms across universities, and a practice exercised widely by many database designers in the industry. A software tool that can assist the process of normalisation and/or the teaching of normalisation would be therefore desirable.

In this paper, we report a software tool known as EDNA (an Enhanced Database Normalisation Automator). The software automates the process of normalisation of relational schemes to second (2NF), third (3NF) or Boyce-Codd (BCNF) normal forms. Although the software was primarily intended for assisting database designers in the field, it can also be used as a tool to validate the result of manual normalisation while students are learning this topic or practising normalisation in their database design projects. The purpose of introducing EDNA to the TLAD community is to enhance the student learning experience of this useful topic.

The rest of the paper is organised as follows. In section two, a brief review of database design approaches and the position of normalisation in the process of database design is given. Existing works in the literature about automatic normalisation are also presented and discussed. Difficulties in teaching and learning this topic are summarised. Section 3 describes the main functions and the interactive user interface of EDNA together with issues relating to the organisation and implementation of the software. Section 4 presents a brief and qualitative evaluation about the effectiveness of the software, its strengths and limitations in practical use for teaching and learning before section 5 concludes the paper with future works.

2. RELATIONAL DATABASE DESIGN AND NORMALISATION

2.1 Normalisation Process and Relational Database Design

Relational database design tends to follow one of two established approaches. In the bottom-up approach (Date 1995), all attributes describing data objects are collected into a single universal relation scheme where various dependencies between the attributes such as functional dependencies are globally studied and specified. A step-by-step stringent normalisation process is then followed to decompose the universal scheme into higher normal form schemes. For instance, 2NF schemes are obtained to ensure total dependencies from non-key attributes to the primary key, followed by 3NF normalisation to ensure no transitive dependencies exist from non-key to key attributes. The process may eventually continue up to 5NF normal form schemes, but 1NF, 2NF and 3NF schemes are arguably the most common. The process of normalisation is well established, having been proposed by Codd as part of his wider work on relational database theory in the late 1960s (Codd 1970).

Alternatively, a top-down approach of design can be followed (Elmasri & Navathe 2011). In this approach, a high level conceptual model such as the Entity-Relationship (ER) model is first used to describe a database in terms of high level concepts like entities (data objects), relationships (associations) and attributes (properties). The conceptual design result, in the form of an Entity-Relationship Diagram (ERD) for example, is then transformed or mapped into a set of relation schemes according to a set of explicit rules (Elmasri & Navathe 2011). However, the ER model has its own limitations: it does not explicitly express the transitive functional dependencies between attributes of an entity, and hence the resulting schemes are in second normal form but not always in fourth normal form as suggested in Byrne (2003), unless the designers explicitly describe the concerned attributes as separate entity

types. It is therefore safe to state that the resulting schemes from the transformation may still not be final and need further normalisation to decompose into higher normal forms.

In summary, it appears that no matter which approach one takes, a normalisation process of relation schemes is normally needed. In fact, Connolly & Begg (2010) suggest using normalisation at least for validating the resulting relational schemes from the process of translating ERD into table structures.

2.2 Practical Problems in Teaching and Learning Normalisation

Based on the first author's recent learning experience and the second author's past teaching experiences over many years, students tend to face a number of problems when studying normalisation. First, students often consider the concepts of functional dependency and normalisation to be abstract and have difficulty relating them to real-world problems, an issue that is further compounded by the use in many textbooks of abstract terms such as 'dependency' that separate students from the data reality. Second, students often find it non-trivial to effectively apply the concept of the functional dependency in stating facts through non-key attributes about entities and relationships expressed by the keys. This is less of a problem when only a small number of attributes are involved, but the amount of difficulty increases significantly when the number of attributes gets large and the dependencies tend to involve multiple key and non-key attributes. Third and more seriously, despite textbook guidelines on how to normalise, students tend to conduct the normalisation process in a much less disciplined manner with a great chance of trial and error. This may not be entirely the fault of the students as some tutors may be more interested in formally defining normal form schemes accurately than explaining how to achieve them practically. Last but not least, students often find the manual normalisation process boring and time-consuming, and hence the risk of human error is constant. As a result, the conduct of normalisation is often unsatisfactory, causing loss and bad decomposition of schemes and ultimately a collection of poorly organised tables that are difficult to work with.

2.3 Normalisation Algorithms

Algorithms for decomposing relation schemes up to 3NF and BCNF existed in early literature on relational database design in the late 1970s and early 1980s, and can be found in a number of reputable textbooks on databases today, for instance Elmasri & Navathe (2011) and Ullman & Widom (2008). However, these high level abstract algorithms have been used in those texts to demonstrate the correctness of a formal manual decomposition process for normalisation, rather than as a basis for the development of a fully automated normalisation system. In fact, these algorithms in their abstract forms are more useful in understanding the theoretical aspects of relational database design. As indicated in (Ullman & Widom 2008), some of these algorithms are potentially computationally expensive and hence were prohibitive at that time of limited computing resources.

Nevertheless, attempts at automating the normalisation process have been continuing for more than three decades. The earliest work can be traced back to 1983 when a prototype tool based on Ullman's algorithms (Travis 1983) was developed for the United States Air Force, but little detail regarding implementation and performance of the final software is known. Du and Wery (1999) developed Micro, the first automated system that used efficient data structures to implement their own 2NF algorithm and the 3NF and BCNF algorithms from Elmasri & Navathe (2011). However, not only does the system still lack the

functionality of table creation for serious practical use, but it is also now outdated in terms of software modularity and portability.

Since then, several follow-ups have been reported. More recent is a software tool known as RDBNorma that claims more efficient solutions than Micro using simpler data structures (Dongare et al. 2011), but this claim cannot be substantiated beyond a simple timing comparison that is lacking in technical details of the test environment. Bahmani et al. (2008) proposed graphical representations of functional dependencies known as directed graph dependency matrices to visually illustrate dependencies between attributes and provide a means of manipulating the dependencies to achieve normalised relation schemes. Akehurst et al. (2002) proposed a class-based method for modelling database schemas using the Unified Modelling Language (UML) extension known as Object Constraint Language. Attaran and Hosseinian Far (2011) developed a tree-based visualization tool for producing a normalised database schema by connecting trees and avoiding the use of functional dependencies entirely.

The majority of these systems have however been developed purely as academic research projects, with an emphasis on efficiently delivering accurate results over explaining the process involved. Their authors have made little or no attempt to build in features that are desirable for supporting the teaching and learning of database design, and as such these systems are poorly suited to use in an educational environment. The emphasis of teaching and learning strategies therefore remains very much on the explanation of normalisation as a manual process, with little practical support and the attendant drawbacks as discussed above.

3. EDNA, AN ENHANCED DATABASE NORMALISATION AUTOMATOR

3.1 Overview of the Main Functions

The primary function of EDNA is to take a user-defined set of atomic attributes and functional dependencies, and perform normalisation up to and including Boyce-Codd normal form. The normalisation procedure is carried out in a fully automated fashion using the user-defined dependencies, with no further user intervention required. When complete, the user is then able to specify data types for each attribute of the resulting decomposed relation schemes and generate SQL commands to create the physical table structures in a number of popular database management systems. The latter facility provides an opportunity to introduce a practical element into the teaching of database technologies; in that students can actually create the databases they have designed and enter some sample data, giving them a greater understanding of the differences between normal forms and an appreciation of the problems caused by poor design.

EDNA includes a facility to save design data to disk in the form of a project file at any stage of the process. This allows a set of attributes and functional dependencies to be defined in advance of a lesson and loaded into EDNA on demand. The same set of dependencies can be given to students to normalise manually as a practical task, and then loaded into EDNA for automatic normalisation so the results can be compared to validate their understanding of the subject. It is to be hoped that this comparison will provoke a discussion, especially if EDNA's results differ from the students' own, which will help to further their knowledge of the subject.

3.2 The Interactive User Interface

EDNA is designed to be easy to use and presents a Windows Forms-based interface that should be familiar in style to most computer users. The user is guided through the stages of the database design process by the use of tabs that follow the logical order of operations from definition of attributes to database creation.

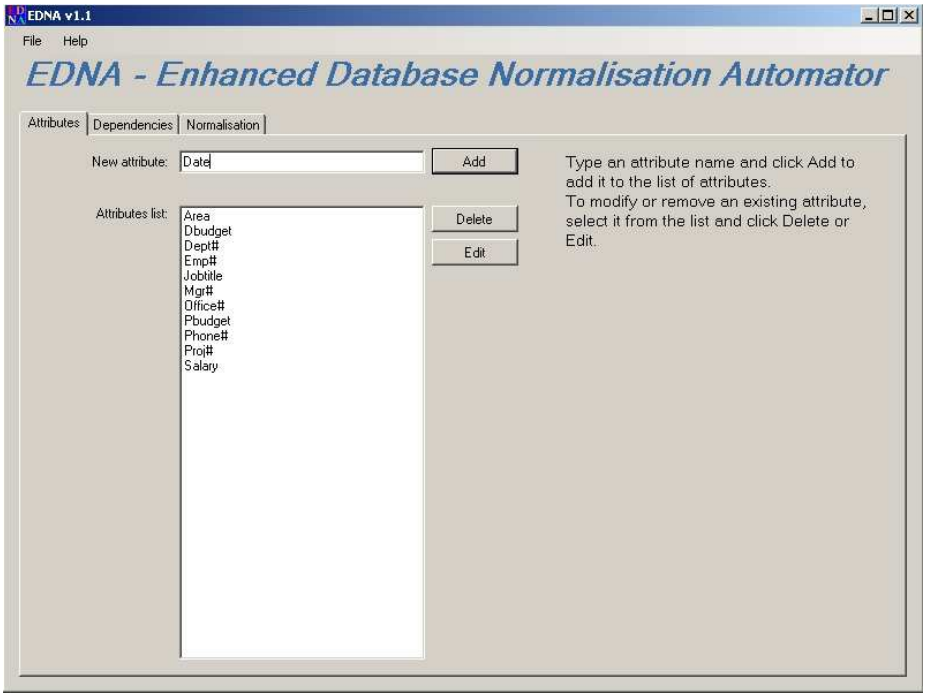

Figure 1. Entry of attributes

As shown in figure 1, the first tab presents a simple interface to define the attributes required within the database. New attributes are entered simply by typing them into the text entry field and clicking a button or pressing the enter key. Existing attributes can be renamed or deleted, and changes made to attribute definitions will automatically propagate through to any functional dependencies that exist. EDNA enforces the uniqueness of every attribute name, and as such any attempt to create a duplicate name will be rejected.

Functional dependencies are defined on the second tab, shown in figure 2. The user first selects one or more attributes that form the determinants from the left-hand list, and similarly selects those that form the dependents from the right-hand list, before clicking the 'Determines' button. This will create a new functional dependency, which is added to the list of existing dependencies on the right hand side. It is possible to select multiple dependents within a single functional dependency, but these will automatically be split into multiple dependencies meeting the requirement of minimal cover that each dependency must have exactly one dependent (Date 1995) (Elmasri & Navathe 2011). EDNA does permit the user to enter trivial dependencies, but these will be automatically removed as part of the normalisation process.

Once the definition of attributes and functional dependencies has been completed, normalisation is performed using the third tab, as shown in figure 3. The desired normal form is selected from the available options of 2NF, 3NF and BCNF, and the 'GO' button clicked to carry out the normalisation process.

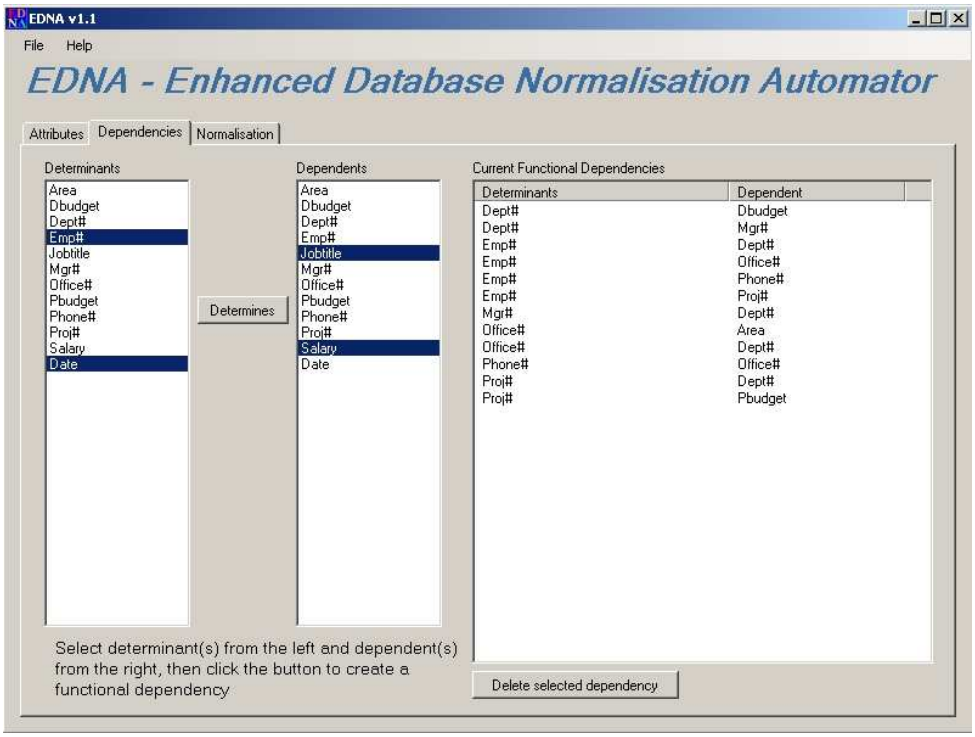

Figure 2. Specification of functional dependencies

No further user intervention is required during the process and the resulting decomposed relation scheme is displayed in a tree view form with primary key attributes highlighted in red. The resulting relations are named using a default system, but can be renamed by editing the default names, as partially seen in figure 3, where the first three relations have been given meaningful names but the remaining ones retain their default names.

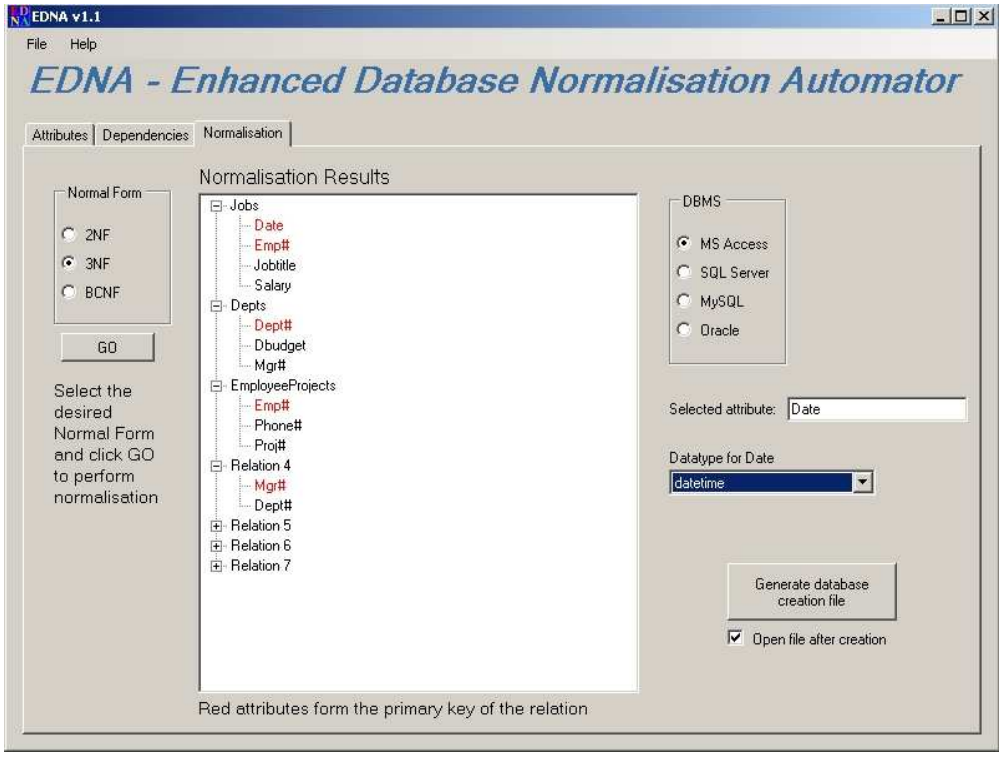

Figure 3. A resulting decomposed database schema

A relation scheme defined by EDNA can be created as a physical table structure in one of the popular database management systems by first selecting the desired DBMS from the options. An appropriate data type is then specified for each attribute by selecting a suitable option from the list of data types available in that particular DBMS in the combo box on the right. When multiple instances of an attribute exist, all are given the same data type to prevent join issues caused by mismatches between primary key and foreign key pairs. Finally, clicking the 'Generate SQL' button will generate a set of SQL commands that can be executed within the DBMS to create a physical database structure that matches the relation scheme produced by EDNA, an example of which can be seen in figure 4.

3.3 The Implementation Issues

The major challenge of producing an automated normalisation tool lies in the difficulty of converting the high-level abstract algorithms presented in the literature into functioning and fully tested program codes. These algorithms are not intended for this purpose and hence are written in a manner close to natural language, often using a single sentence to describe a task that must be performed by a complex function. Detailed analysis of the intended function and results of these algorithms was therefore necessary in order to translate them into the code written in Visual C#.

```
CREATE TABLE `Jobs` (`Date` datetime, `Emp#` int UNSIGNED, `Jobtitle` varchar(20), `Salary` decimal(6,2) UNSIGNED, CONSTRAINT PK_Jobs PRIMARY KEY (`Date`, `Emp#`));
CREATE TABLE `Depts` (`Dept#` tinyint UNSIGNED, `Dbudget` decimal, `Mgr#` tinyint UNSIGNED,
CONSTRAINT PK_Depts PRIMARY KEY (`Dept#`));
CREATE TABLE `EmployeeProjects` (`Emp#` int UNSIGNED, `Phone#` text(10), `Proj#` int, CONSTRAINT PK_EmployeeProjects PRIMARY KEY (`Emp#`));
CREATE TABLE `Managers` (`Mgr#` tinyint UNSIGNED, `Dept#` tinyint UNSIGNED, CONSTRAINT PK_Managers PRIMARY KEY (`Mgr#`));
CREATE TABLE `Offices` (`Office#` bigint UNSIGNED, `Area` tinytext, `Dept#` tinyint UNSIGNED, CONSTRAINT PK_Offices PRIMARY KEY (`Office#`));
CREATE TABLE `Phones` (`Phone#` text(10), `Office#` bigint UNSIGNED, CONSTRAINT PK_Phones PRIMARY KEY (`Phone#`));
CREATE TABLE `Projects` (`Proj#` int, `Dept#` tinyint UNSIGNED, `Pbudget` decimal(6,2),
CONSTRAINT PK_Projects PRIMARY KEY (`Proj#`));
```

Figure 4. MySQL table creation commands for the schema shown in Figure 3

EDNA has been written using the object-oriented style of programming, as the database design procedure involves a number of structured entities that can be considered "objects" in the programming sense of the word. It was necessary to carefully design data structures capable of storing these objects and preserving their format, while allowing efficient in-memory processing by the computationally-intensive normalisation algorithms. This process was further complicated by the need to compose single objects into sets; for example, each individual functional dependency can itself be represented as a single object, but in order to represent the complete schema it is also necessary to store an ordered set of multiple instances of this object together with implementing methods to manage this set.

The design of EDNA is such that the functionality has been split between two separate but linked components for greater flexibility, but it remains a stand-alone program that requires no third party software to be installed. As shown in figure 5, the user interface is provided by a Windows Forms application, which is linked to a DLL (dynamic link library) file that

contains the normalisation algorithms themselves, both parts being written in Visual C# to ensure compatibility. This approach widens the appeal of the program as potential users are not limited to using the interface provided in the complete EDNA package, but able to integrate the normalisation library into their own software, for instance to produce an alternative version that provides greater visualisation of the intermediate stages of the process to further enhance the learning experience.

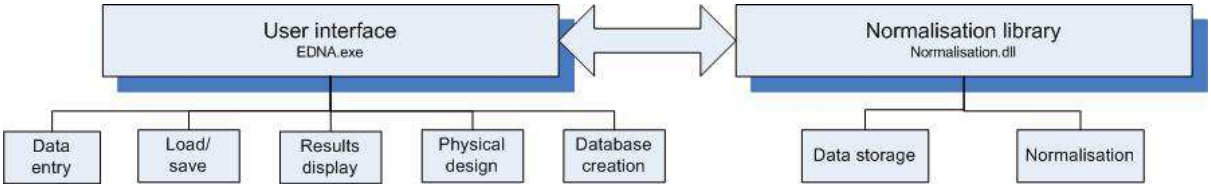

Figure 5. Separation of functionality between the user interface and library program

A significant property of the normalisation procedure that adds complexity to any implementation of an automated system is its non-deterministic nature. For any given set of functional dependencies, there is not usually a single resulting schema, but a set of multiple schemas, each of which decomposes the relations differently but is equally valid according to the rules of decomposition. Which of the possible schemas is obtained can depend on the order in which the attributes and dependencies are entered, stored and processed, making it difficult to produce consistent and repeatable output. To overcome this issue, EDNA imposes an alphabetical sort order on the set of functional dependencies, meaning that the same result will always be obtained for a particular input set, but this may differ from those published elsewhere.

A specific concern pertinent to the use of EDNA in an educational context is that the published works on the subject (Date 1995) (Elmasri & Navathe 2011) generally do not present an algorithm for achieving second normal form. The reason for this omission is that this normal form does not fully address the issue of redundancy and is therefore not generally used in real-world databases, and it is possible to normalise directly to the more useful 3NF without passing through 2NF as an intermediate stage, so such an algorithm would be redundant in a commercial system. However, when teaching the theory of normalisation, 2NF is an important concept and it is therefore desirable for EDNA to allow normalisation only to this stage in order to compare the resulting database schemas with those of 3NF. As such, EDNA includes an additional algorithm that was developed in (Du & Wery 1999) to achieve second normal form, alongside improved versions of the 3NF and BCNF algorithms based on those from the same source.

4. EVALUATION AND DISCUSSION

EDNA has already been used in a teaching and learning role by students studying the Principles of Database Systems module at the University of Buckingham, as part of their module project. This involved the design and implementation of a working database to meet a specific need defined by a case study, and the assessed project report was required to include a comparison of the students' own manual schema designs with those produced by EDNA. In general, EDNA has received positive feedback from the students concerned, who felt it was a useful tool that enhanced their understanding of what is perceived to be a difficult topic.

However, it must be remembered that class sizes at Buckingham are generally much smaller than those of other institutions, and the undergraduate programme is completed in just two years, thus limiting the amount of resources available for testing. The Principles of Database Systems class of 2013 consisted of only 18 students, who carried out the module project in groups of two or three, and this module is taught only once per year; at the time of writing this paper, it has yet to be taught again to the first-year students of the 2014 intake. Hence, although it is intended to continue testing the software as part of this module in future years, it has not yet been possible to increase the test sample size, and with such a small sample detailed quantitative analysis of the test results cannot be performed meaningfully so only a limited qualitative evaluation can be provided. One of the main reasons for presenting EDNA to the TLAD community is the opportunity for obtaining a much larger set of testers, from whom some more meaningful conclusions regarding the effectiveness of the tool in assisting the teaching can be drawn in future.

The first two of the problems detailed in section 2.2 are difficult, if not impossible, for any software product to solve as they are reliant on human powers of explanation and comprehension to ensure that students gain a correct understanding of the subject. However, in this context, EDNA does provide an opportunity for learning by experimentation as it allows different combinations of functional dependencies to be tried and the results observed, hopefully providing a justification to the students of the importance of learning these concepts. EDNA has concentrated on addressing the third and fourth problems as the rigid method of entering attributes and dependencies imposes the discipline that is often lacking in manual normalisation, and the decomposed relation schemes are produced in a matter of seconds rather than hours or days, thus preventing boredom from setting in.

The performance of EDNA appears more than satisfactory in terms of speed, even when dealing with large sets of complex dependencies. The program was developed on a 2007 Fujitsu Siemens laptop running Windows XP with a 1.6GHz processor and 1GB of memory, which is an extremely low specification by current standards, and even on this machine, the decomposed relations were generated virtually instantly, so there should be no significant performance issues on any modern system. It has also been tested on the Windows 7 and 8 operating systems and no issues were encountered.

All three of the desirable properties of a normalised database are fully supported by EDNA. The system performs a complete decomposition, in which no extra attributes are added and none of the user-specified attributes are removed, so the set of all attributes of the decomposed schema remains equal to that of the original universal relation. Wherever possible, all non-redundant functional dependencies are preserved, except in those rare cases where producing a valid BCNF schema makes dependency loss unavoidable as reported in most textbooks. EDNA decomposes the relation scheme using a lossless join method, such that all of the information in the original universal relation can be recovered by using natural join operations on the decomposed relations.

In certain circumstances involving multi-value dependencies considered by 4NF normalisation, a many-to-many relationship exists between multiple relations and they must be connected by a link table with no additional attributes, but there is no valid functional dependency that can describe this relationship. For these cases, a special ‘null’ dependent is provided, which can be used to specify that two or more attributes of different relations are related but do not determine any other attributes, such that an all-key relation is created to link these relations.

The database schemas generated by EDNA have been validated for correctness up to and including third normal form, and are at least partially compliant with fourth normal form. The resulting schemas occasionally vary from those published as examples in the literature, but this is due to the non-deterministic nature of the decomposition process and they have been verified to be equally compliant with Codd's rules of decomposition (Codd 1970).

While it provides similar functionality to certain of the existing works discussed in section 2.1, EDNA is distinguished from these by its method of implementation. The object-oriented programming approach is a unique feature of the software and the data structures have been carefully designed for the most efficient storage and processing of large and complex datasets. EDNA's modularity is another significant difference that lends it far more flexibility than existing solutions, which are typically implemented as homogenous 'black box' programs; the separation of the normalisation functionality from the user interface provides many opportunities for independently developing each component to suit specific needs.

5. CONCLUDING REMARKS

This paper presented a software tool EDNA for automatic normalisation for supporting teaching and learning of relational database design. By automating the process of normalisation, the system allows students and teachers to verify their own manual normalisation results, and hence improve their learning and teaching experience of this difficult but practically useful topic. The tool is intended to be freely available for teaching purposes across computing departments in the UK (download site: http://www.buckingham.ac.uk/appliedcomputing/researchstudents#databases). Through the widespread use of the tool, we aim to improve its performance and usability further with future new versions.

In its present form, EDNA is by no means a finished product and the emphasis of its development has been on ensuring correct functionality rather than providing an elegant user interface. It is acknowledged that the interface as it stands, while functional, is somewhat basic and user-unfriendly, so immediate future work involves providing documentation and generally improving the appearance and usability of the user interface. It is also felt desirable to add a facility to print sets of functional dependencies or export them to another program, so they can easily be given to students for normalisation as a manual task. In the longer term, the proposed work naturally includes more algorithms for 4NF and 5NF normalisation so that a complete set of normal forms can be obtained upon user request. Another future work will be to exploit the possible use of touch screen technology with visual gestures such as drag-and-drop, pinching etc. in specifying data dependencies to further improve the usability of the software for teaching and learning purposes.

In its current form, the system still requires the user to understand and correctly apply the concept of functional dependencies in order to manually enter them, but there have been a number of suggestions that it would be desirable to automate this process by providing a first normal form dataset and letting the software automatically determine the functional dependencies within. While this is a sound concept in theory, there is a major practical difficulty to overcome. Accurate results would depend on the provision of potentially large datasets that include all possible domain values for every attribute, as the data contained in the provided subset may imply functional dependencies that hold upon this subset but are violated by legal data entered at a later date; this incorrect specification of functional dependencies could result in the very design issues that EDNA initially set out to solve, and this functionality could easily become a separate research area in its own right.

REFERENCES

Akehurst, D., Bordbar, B., Rodgers, P., & Dalgliesh, N. (2002). Automatic Normalisation via Metamodelling. *SE 2002 Workshop on Declarative Meta Programming to Support Software Development.*

Attaran, H., & Hosseinian Far, A. (2011). A Novel Technique for Object Oriented Relational Database Design. *Proceedings of the 2011 10th IEEE International Conference On Cybernetic Intelligent Systems*, (pp. 128-132).

Bahmani, A. H., Naghibzadeh, M., & Bahmani, B. (2008). Automatic database normalisation and primary key generation. *Canadian Conference on Electrical and Computer Engineering*, (pp. 11-16).

Byrne, B. (2003). Good Top-Down Design Methodologies Tend to Produce Fully Normalised Designs Anyway. *1st LTSN Workshop on Teaching, Learning and Assessment of Databases.* Coventry.

Byrne, B., Garvey, M., & Jackson, M. (2003). Using Object-Role Modelling to Teach Database Design. *1st LTSN Workshop on Teaching, Learning and Assessment of Databases.* Coventry.

Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM , 13* (6).

Connolly, T., & Begg, C. (2010). *Database Systems, A Practical Approach to Design, Implementation and Management.* Addison-Wesley.

Date, C. J. (1995). *An Introduction to Database Systems.* Addison-Wesley.

Dongare, Y., Dhabe, P., & Deshmukh, S. (2011). RDBNorma: A semi-automated tool for relational database schema normalization up to third normal form. *International Journal of Database Management Systems , 3* (1), (pp. 133-153).

Du, H., & Wery, L. (1999). Micro: A normalization tool for relational database designers. *Journal of Network and Computer Applications , 22*, (pp. 215-232).

Elmasri, R., & Navathe, S. B. (2011). *Database Systems: Models, Languages, Design, and Application Programming.* Pearson.

Nelson, D., & Jayakumar, R. (2012). A Ten-Year Review of Workshops on the Teaching, Learning and Assessment of Databases. *10th HEA Workshop on Teaching, Learning and Assessment of Databases.* Hertfordshire.

Travis, C. T. (1983). *Interactive Automated System for Normalisation of Relations.* Defense Technical Information Center.

Ullman, J. D., & Widom, J. (2008). *A First Course in Database Systems.* Pearson.