NORMALDB – A Logic-Based Interactive e-Learning Tool for Database Normalization and Denormalization

Lule Ahmedi
Computer Engineering
University of Prishtina
Kodra e diellit pn
Prishtina, Kosova
e-mail: lule.ahmedi@uni-pr.edu

Naxhije Jakupi
Computer Science
South East European University
Ilindenska pn
Tetovo, FYR Macedonia
e-mail: naxhijejakupi@yahoo.com

Edmond Jajaga
e-Learning Center
South East European University
Ilindenska pn
Tetovo, FYR Macedonia
e-mail: e.jajaga@seeu.edu.mk

***Abstract*—This paper introduces the design and development of an e-learning tool, NORMALDB, that teaches normalization and denormalization phases of database design. What characterizes and makes unique this tool is that it smoothly integrates two entirely diverse paradigms: at the internal level, an intelligent layer based on logic rules in Prolog implements the normalization and denormalization of a given database, whereas at the external level, the user may friendly interact with the tool through a common Web interface, and not concerned with the complexity of the tool internally. Using this tool, students may stepwise exercise and explore the whole process of normalization and denormalization, which otherwise constitute an important but troublesome phase of database design.**

Keywords-E-learning tool; database normalization and denormalization; logic rules; client-server programming.

I. INTRODUCTION

Learning technologies is one of the fields that highlighted the potential of the web for education. Easy access, location and time independent resources, unlimited design space, flexibility, and a wide range of functionalities of the web prompted the development of a new branch of learning named e-Learning [1]. The advantages of the web were by default inherited by e-Learning technologies built upon them, enabling thus more sophisticated teaching and learning environments. That way, the utilization of e-Learning technologies by universities has led them to transform from didactic teaching methods to flexible and independent learning. Subject-based e-Learning is an example of providing support for even more complex subjects to learn [2], as is database normalization and denormalization, which is a subject-to-learn through our e-Learning tool we will here introduce.

Database design is the art and science of improving the structure of database relations that are most suited to represent a small portion of the world called the "universe of discourse" [3]. The relational schema that results at the end of the design phase must consist of normalized relations accordant with the semantics of given entities and their integrity constraints, avoiding at the same time as more as possible data manipulation anomalies. Hence, normalization is very important in practice, but also crucial to get familiar with for every student studying databases. Unfortunately, this subject is often dry and troublesome to learn, making it not well received by students.

To ease learning of database normalization and denormalization, we developed NORMALDB, a web-based e-learning tool that will be introduced here. The tool is designed to provide theoretical background on the subject (normalization and denormalization): it explains stepwise every single detail of the process as a whole. It also provides an interactive interface of learning the subject driven by own examples as given by the user (e.g., a student). The organization of the tool is a pure reflection of how the teacher organizes the subject.

There are already few tools that cover executing normalization, like JMathNorm [4], a Web-based tool of learning normalization [5], NORMIT [6], or a normalization tool that bases on UML [7]. NORMALDB is unique in that it smoothly integrates two entirely diverse paradigms, namely:

- at the internal level, an intelligent layer based on logic rules in Prolog implements the normalization of a given database as introduced by Ceri and Gottlob [3], whereas,
- at the external level, the user may friendly and stepwise interact with the tool through a common Web interface, kept thereby not concerned with the complexity of the tool at its internal level.

Moreover, using NORMALDB, students may step-by-step experience the whole life-cycle of database relations up to their normalized forms, or in a reverse process of denormalization which means rollbacking relations into their original form whenever deemed necessary for the sake of efficiency of join operations. Driven by examples, each step is in addition accompanied with comprehensive explanation of the theories applied in that given step. Navigation through theoretical blocks across individual steps / subtopics is another strong reason to leverage NORMALDB. The level of interactivity, the ease of use, and its logic-base of rules and the Web interface make NORMALDB unique among existing tools which support normalization.

 ISBN: 978-1-61208-180-9

This paper is organized as follows. A brief review on the main concepts of database normalization and denormalization in general is given in Section 2, while Section 3 introduces the NORMALDB, our normalization and denormalization tool. Further, Section 4 provides the software architecture of NORMALDB, with its features then summarized in Section 5. Finally, the conclusion and future work are discussed in Section 6.

II. DATABASE NORMALIZATION AND DENORMALIZATION

It has been estimated that more than 80% of all computer programs are database-oriented. This is easy to believe since databases allow the applications to meet all their requirements for storing, manipulating and displaying data [8] at once.

For years now, the relational data model remains the most used data model in databases. The central data description construct in this model is a relation which can be thought of as a set of records. The description of a data in terms of data model is called a schema. In relational model, the schema for the relation specifies its name, the name of each field, and the type of each field. User requirements may in addition result into certain integrity constraints (ICs) within the schema. ICs may in turn cause redundancy-related problems like: redundant storage, update anomalies, insertion anomalies, and deletion anomalies. Special group of ICs that plays the major role in the schema refinement are called functional dependences (FDs) [9].

A. Normalization

Following the FDs that hold over a relation, one may understand what redundancy problems, if any, might arise from the current schema. To provide such guidance, several normal forms have been [9] introduced in terms of FDs as follows:

- 1NF – First normal form: A relation R is in first normal form (1NF) if and only if all underlying domains contain atomic values only;
- 2NF-Second Normal Form: A relation R is in second normal form (2NF) if and only if it is in 1NF and every nonkey attribute is fully dependent on the primary key.
- 3NF – Third Normal Form: A relation R is in third normal form (3NF) if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key.
- BCNF- Boyce-Codd Normal Form: A relation R is in Boyce/Codd normal form (BCNF) if and only if every determinant is a candidate key.
- 4NF and 5NF are rarely achieved, and hence not implemented in our tool at this stage.

The procedure itself of transforming a relation, given its FDs, into any of the abovementioned normal forms (NFs) is known as normalization, and it:

- leaves the relation unchanged if it already satisfies the NF sought after, or
- decomposes the relation in two or more smaller relations, i.e., relations with less number of columns, each satisfying the NF sought after.

B. Denormalization

Normalization of a relational schema given a set of FDs results into a relational schema which is free of redundancy-derived anomalies, but might yet suffer from eventual performance-derived problems. To address that kind of problems, a reverse process to normalization, namely denormalization, has been introduced. Denormalization is the process of adding columns to the table to reduce joins in favor of performance, and is considered only if the integrity of data is not seriously compromised [10].

III. NORMALDB

As stated in [11], students find it difficult to understand the concept of FDs and normalize data in order to obtain smaller well-structured relations. NORMALDB is a web-based e-learning tool that we developed to aid students understand and experience the most complex tasks of database design, i.e., normalization and denormalization. The organization of NORMALDB resembles the way how a teacher schedules his / her class while teaching normalization and denormalization. Further, the ability to explore every single step / subtopic of normalization by running own examples and breaking them down to elementary details makes NORMALDB far more advantageous vs. traditional in-class teaching of the subject.

In the following subsections, a description of NORMALDB to reflect its implementation in two layers, the logical and the interface layer, is given.

A. Logical Layer

At the data tier and business logic layer of NORMALDB, we adopted the Ceri and Gottlob's script [3] implemented in Prolog. The Prolog programming language is known for its contributions to problem solving in artificial intelligence [12]. A common integrated framework for describing both data structures ("facts") and algorithms ("rules"), and the facilitated interaction with the code through the "trial and error" interface are few among several advantages readily provided by the Ceri and Gottlob code due to the representation in Prolog of the logic of normalization to the machine [3].

The adopted Prolog script [3] of normalization consists of the following:

- the facts which provide data about relations (the relation name, attributes, and FDs), and
- the rules which relate facts, and implement all algorithms throughout normalization.

For example, according to [3], a relation schema `rel` with the set `[a,b,c]` of attributes, and a set `[a→b,b→c]` of FDs is represented with the following facts at our logical layer:

```
schema(rel, [a, b, c]).
fd(rel, [a], [b]).
fd(rel, [b], [c]).
```

For each normalization step, there is a rule or a set of rules that may be invoked in any order. This way a user may observe results incrementally by executing certain rules step by step over the input base of facts. Some of the normalization rules provided in the script are as follows [3]:

- findonekey(REL, K) - Determines one key K of relation REL
- assertallkeys(REL) - Determines and asserts all keys of REL
- findmincover(REL) - Finds a minimal cover of the functional dependencies defined for REL
- thirdnf(REL) - Decomposes REL into third normal form
- haslj(REL) - Tests for losslessness of the decomposition of REL
- makelj(REL) - Makes the decomposition of REL lossless
- projectfds(REL, RELl) - Projects functional dependencies holding for the relation REL to the relation RELl
- isinbcnf(REL) - Tests whether REL is in Boyce-Codd normal form
- bcnf(REL) - Decomposes REL into Boyce-Codd normal formminimize(REL) - Minimizes the decomposition of REL

Denormalization also supported in NORMALDB is a rather intuitive task driven primarily by queries which are frequently invoked in a database and involve expensive joins. The implementation in NORMALDB of denormalization extends the existing Prolog knowledge base of normalization with the following:

- rules for the direct re-composition of tables, and
- a parser written in Definite Clause Grammar (DCG) notation of Prolog, which is able to read queries against the database and reason upon them to infer which tables need re-composition in favor of performance.

In addition to denormalization, few more modifications of the Prolog script of Ceri and Gottlob [3] were applied to make that script work in our web-based tool, i.e., adding new rules and facts for rendering Prolog results into the web page. Most of new rules are HTML generators which convert the adopted Prolog script to the Prolog server, namely they generate HTML tags which hold the results of retrieved predicates. Section 4 will reveal more details about HTML generators.

B. Interface Layer

Next we discuss the interface layer of NORMALDB which is mainly developed in PHP language. The HTML tag rendering is provided through PHP scripts, whereas JavaScript, especially its libraries jQuery [13] and jQuery UI [14] help make the interface of NORMALDB simple and easy to navigate.

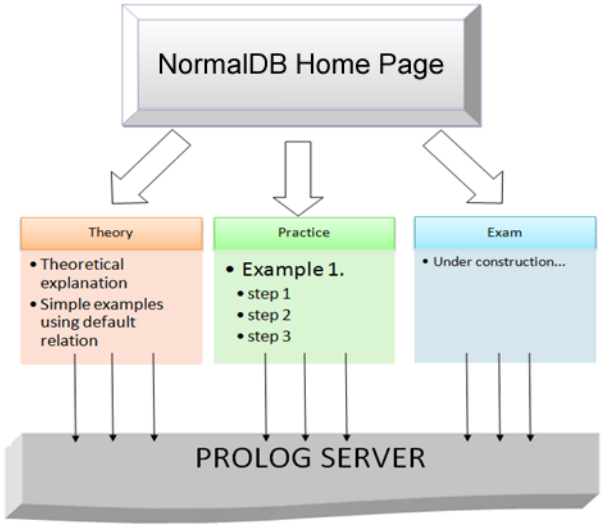

Figure 1. Organization of the interface layer in NormalDB.

Fig. 1 shows the organization of NORMALDB from a user perspective. The colored boxes represent web pages, whereas the grey box represents the knowledge base of the application supplied by the Prolog server which runs whenever examples are carried out. Connection to the knowledge base does not require any extra procedure from the user. Simply, the given connection hyperlink makes a request to the Prolog server which in turn displays the result.

The interconnection between topics and examples is done via hyperlinks, while results are represented in accordance to the actual web page template. Depending on the content, the result appears within a dedicated tag of the page, or in a message box that shows after clicking the link. The JavaScript language and its libraries jQuery and jQuery UI are employed for user-friendly purposes of the interface. jQuery tabs and message boxes helped us prevent the overload of the interface.

The whole interface of the tool (colored boxes in Fig. 1) is organized in three main parts:

Theoretical part: Consists of theoretical explanations of the topics covering the normalization phase of the database design. Each topic explanation appears as a hyperlink, and may further contain links which illustrate the application of the given theory to a given example. That example is referred to as a default example in our tool, and is designed to serve the demonstration of each of the theories over a given set of (default) relations and their FDs.

Practical part: This is the core part of NORMALDB, and is aimed to serve exercises. The user (e.g., a student) shall input the relation name, attributes, as well as functional dependencies of the relation that he / she wants to examine in terms of normalization and denormalization, and the tool will then start exploring the topics upon the given relation.

Exam part (Self-assessment part): This part is planned for future work. It is designed to provide a testing environment where users (e.g., students) may themselves examine their knowledge gained in the field concerning both exercises and the theory.

 ISBN: 978-1-61208-180-9

IV. THE SYSTEM ARCHITECTURE

The development environment in building NORMALDB consisted of the scripting language PHP, JavaScript, and the logic programming language Prolog.

The logic of the NORMALDB relays in the server side of the application. The client side shares just the functionalities given by JavaScript codes.

The server side of the application is comprised of two distinct servers. Apache server is needed for processing PHP scripts and generating HTML pages for the client while Prolog server provides the knowledge base of the system. The later one is accessed from the client side through links generated from the PHP scripts.

Fig. 2 describes the used architecture in NORMALDB with the pursued workflow of the functionalities. In the following subsections we will explain the challenges that appeared during the development of our tool.

A. Preserving the state

The process of normalization and denormalization flows over a step by step evaluation which requires keeping and following the active state of the script execution. This happens because of the modifications that are done to the knowledge base after evaluating a particular step. Traversing through the Prolog script to find the predicates that will implicate the required rule, results in asserting new facts that affect the next step. For example, if a user wants to test whether a particular decomposition has the lossless join property, a clause of the form `haslj(rel)` is searched in the base of facts. This kind of facts is provided after the execution of the 3NF algorithm. This way it was necessary to have the knowledge base updated with the information required by the future user requests. Hence, it was needed somehow the Prolog application to be active as long as a user session is active.

When addressing this concern, we ended up with three alternative solutions, each applying distinct techniques originating from different fields.

One solution was to run the Prolog script using the `system()`, `exec()`, and `shell_exec()` built-in PHP functions. These functions are easy to implement, but the troubles arise after running a required predicate since the script then closes up such that the newly asserted predicates cannot be saved.

Another solution was to use a relational database at the backend of the application which will track every inference deduced by the Prolog script, and accordingly modify respective tables in the database. Yet for the sake of the simplicity of the tool, and to avoid difficulties that might appear while tracking Prolog inferences, this solution was omitted.

Finally, we experienced the use of the Prolog server [15] as the most appropriate solution for surpassing this problem, which will be discussed in detail in the following subsections.Communication with Prolog server

The Prolog logic programming language supports a number of libraries for accessing data on HTTP (Hypertext

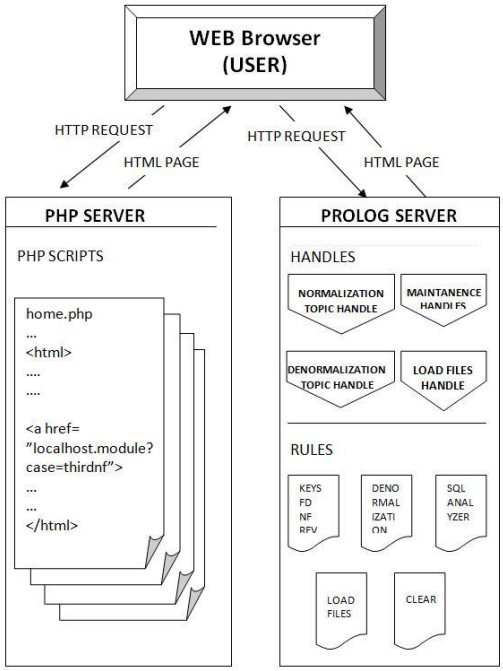

Figure 2. Workflow and request chains of NORMALDB tool.

Transfer Protocol) servers, as well as for providing HTTP server capabilities from SWI-Prolog. Both server and client are modular libraries. The server can be operated from the Unix `inetd` super-daemon, as well as a stand-alone server that runs on all platforms supported by SWI-Prolog [15].

In order to use these libraries, certain modifications to the actual normalization script in Prolog of Ceri and Gottlob were required. Thereby, the mere logic of the script is kept unmodified, extending it with built-in predicates to deal with HTTP requests to configure the HTML pages.

Request handlers are built-in predicates that handle the HTTP requests made from instantiated HTML pages using hyperlinks. When an HTTP request arrives at the server, then Prolog starts traversing through the predicate tree for finding the handle that matches the path came with the HTTP request. This required path is noted as first parameter of the predicate. The second parameter defines the main predicate that handles the handler. As the third parameter of this handler is the list of the options related with the handler. A code that creates a common handler is written below:

```
:- http_handler(root(module),home,[]).
```

After calling a handler, one thread is employed to search for the predicate supported by the handler which is defined in the code. Within the HTTP request can be sent variables which can be extracted from the request and inherited to other predicates that build up the rule. The base predicate that is retrieved from the handler renders the HTML page with enclosed html, title and body tags, by the following rule:

 ISBN: 978-1-61208-180-9

```
home(Request):-http_parameters(Request,
[name(Name,[length >= 2]),
relation(Rel,[length >= 2])]),
reply_html_page(title('Example'),
[\html_requires(css('style.css')),
\case(Name, Rel)]).
```

The body part of this new arranged HTML page is filled with other HTML tags that are derived from the next HTML generators used from other predicates.

Weaving of the HTML tags with the appropriate results is done through HTML tag generators included in the special defined rules which are called with DCG notation of Prolog. These rules get the results inferred from the base predicates and render them into HTML tags that can be read from every web browser.

The formulated HTML reply of the Prolog server is injected into the HTML page rend from the PHP server within a `<iframes>` tag.

A sample Prolog rule that makes the rendering of the result of 3NF normalization in a special `div` tag looks like follows:

```
case(thirdnf,Rel)-->
 {cleandecomp(Rel)},
 html(div([\step1_html(Rel),
 \step2_html(Rel),
 \step3_html(Rel),
 \step4_html(Rel),
 \step5_html(Rel)])).
```

Some of the links are processed at runtime during the execution of the PHP page, while others are simple links and are invoked when a user clicks them over.

B. *Preserving Consistency*

The consistency of the Prolog server will be intimidated if the same predicate is queried two or more times without rolling back to the actual state of the server. Multiple queries are posed in case the user clicks the same link multiple times. To avoid unwanted results from the repeated clicks, there are specific rules asserted in Prolog server that take care to clear the server from the previous results.

Additionally, using the same relation name within the same knowledge base by different users will raise the problem of interference among results of different users. To avoid this, we used the unique session number functionality of PHP. The opened index page automatically creates a new session with unique number which will uniquely identify the relation schema used by that session, concretely by one user. The task of Prolog server in this case is to create a copy of the default relation schema and to rename it with a combination of its original name and the session number of the user.

In this way, normalization rules consider every user session as unique, enabling thus every user work with exercises separately at the same time on the same server.

V. LEARNING NORMALIZATION IN NORMALDB

Accessing NORMALDB through a URL will initially open its homepage as is common for web applications (Fig. 3a). As mentioned earlier in Section 3, the tool is organized in three individual modules: theoretical, practical, and the testing module.

The *theoretical module* (opens when clicking the green box in Fig. 3a) loads a sample example which illustrates all steps of normalization one after another running in the Prolog server. The relational schema used in the example is named *rel* and is given a set of predefined FDs.

The normalization steps or subtopics are each provided on a left menu in the page (Fig. 3b). Changing along the menu links which represent subtopics does not affect the Prolog server unless a button involving the example is clicked. Such buttons contain links to the Prolog server, invoking thus the corresponding predicate to run for the given subtopic.

The left menu contains also the denormalization module of the tool. This page, beside the theoretical explanation of the topic, includes a form where the user may textually input queries supposed to be executed over time against the derived database schema. These queries are then analyzed with a SQL (Structured Query Language) analyzer script written in Prolog, which yields the statistics about which tables need to be joined. This way, the user may decide which tables need to be recomposed.

Navigation to the *practical module* is possible from any page (by clicking the orange box placed at the top of the page), not just when residing at the homepage of the NORMALDB. Once opened, this module (see the Web page in Fig. 3c) will first retract all facts belonging to the

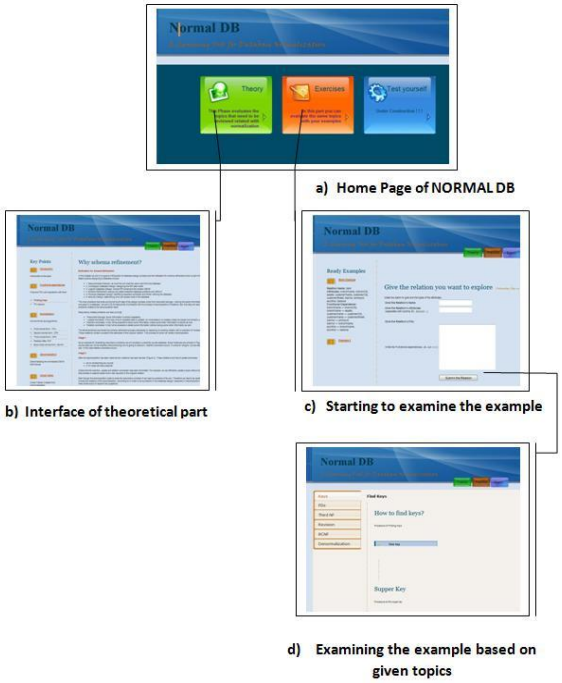

Figure 3. Web interfaces of NORMALDB.

sample example of the theoretical part, and afterwards eventually load a new relational schema entered interactively by the user through a Web form. At this input, if the user provides also the types of attributes of the relation, then the tool might generate a ready-to-deploy SQL script consisting of procedures for creating the database and its tables.

The relation schema, and its attributes and FDs entered through the Web form are then processed by PHP string manipulation built-in functions, and are written as new facts in a separate Prolog file. For instance, back to the example of Section 3, let us assume that the user enters the same input data, i.e., the `rel` relation schema with attributes `[a,b,c]` and a set `[a→b,b→c]` of FDs through the Web form of the Practice page as depicted in Fig. 4. It is the PHP script that would translate these Web form data into the Prolog facts listed earlier in Section 3.

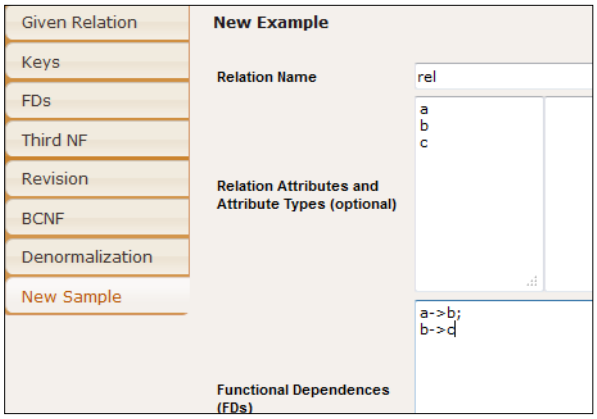

Figure 4. Web form to enter own examples.

These facts are further consulted by the main Prolog server starting from the next opened page up to the last page that requires evaluation of normalization rules, this time over a given set of facts residing at an external Prolog file (Fig. 3d).

Continuing our example with the `rel` relation at the input, Fig. 5 illustrates how finding all keys of the relation works by clicking the `Assert Key` button which actually triggers the evaluation of the `assertallkeys(rel)` rule at the Prolog server. See at the "All keys are [[a]]" row rendered at the bottom of the Web page in Fig. 5 which reflects the result inferred by this rule.

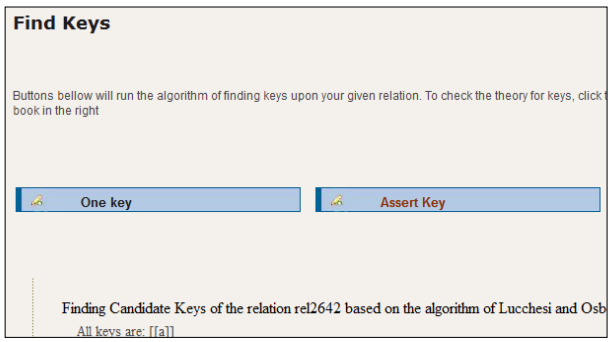

Figure 5. Finding all keys of a relation with NORMALDB.

As one may experience while using NORMALDB, the requirement to keep its interface of teaching normalization process of database design as simple as possible is met. The organization of the page, its rich set of functionalities, and the simple layout contribute altogether towards bringing closer to students the normalization theory which has otherwise proved to be troublesome to capture in a traditional teaching classroom.

VI. CONCLUSION AND FUTURE WORK

This paper introduces the NORMALDB, an e-Learning tool which provides to students a user-friendly environment for learning and experimenting the normalization phase of the database design. Simple interface with an internal complex system and a wide range of functionalities including experiments with self-chosen examples are some of the main advantageous features of NORMALDB. The main challenge was to enable an efficient communication between PHP and Prolog. Among several alternatives we considered, the interaction through a Prolog server [15] proved to be the most appropriate solution for our application since it already provides a number of Prolog libraries for accessing data on HTTP.

NORMALDB is further planned to expand its capabilities of a typical e-Learning tool by incorporating a test module where the student may test himself / herself by solving a given normalization problem, and then comparing his / her solution with the one generated by the tool. Moreover, the support for higher normal forms is also in view under future plans, and easy to implement, since the same rationale applies as when implementing the interface and its links to the Prolog facts and rules for, say, 3NF.

REFERENCES

[1] Namahn, "E-learning: A research note by Namahn," www.namahn.com/resources/documents/note-e-learning.pdf, 22.11.2011.

[2] J. O. Uhomoibhi, "Implementing e-learning in Northern Ireland: prospects and challenges," Campus-Wide Information Systems. vol. 23 no.1, 2006, pp. 4-14, doi:10.1108/10650740610639697.

[3] S. Ceri and G. Gottlob, "Normalization of relations and PROLOG." Commun. ACM , vol. 29 no. 6, 1986, pp. 524-544, doi:10.1145/5948.5952.

[4] A. Yazici and Z. Karakaya, "JMathNorm: A Database Normalization Tool Using Mathematica," ICCS (2), Lecture Notes in Computer Science , vol. 4488, 2007, pp. 186-193, doi:10.1007/978-3-540-72586-2_27.

[5] K. Hsiang-Jui and T. Hui-Lien, "A Web-based tool to enhance teaching/learning database normalization," Proc. of the 2006 Southern Association for Information Systems Conf., 2006, pp. 251-258.

[6] A. Mitrovic, "NORMIT: A Web-Enabled Tutor for Database Normalization," Intl. Conf. on Computers in Education (ICCE), Auckland, New Zealand, 3-6 Dec. 2002, pp. 1276-1280, doi:10.1109/CIE.2002.1186210.

[7] D. Akehurst, B. Bordbar, P. Rodgers, and N. Dalgliesh, "Automatic Normalisation via Metamodelling," ASE 2002 Workshop on Declarative Meta Programming to Support Software Development, Sept. 2002.

 ISBN: 978-1-61208-180-9

[8] R. Stephens, Beginning Database Design Solutions. Wiley Publishing, 2008.

[9] R. Ramakrishnan and J. Gehrke, Database Management Systems. 2nd ed., McGraw-Hill, 2002.

[10] S. S. Lightstone, T. J. Teorey, and T. Nadeau. Physical Database Design: the database professional's guide to exploiting indexes, views, storage, and more. 4th ed., Morgan Kaufmann, 2007.

[11] S. Ram, “Teaching data normalisation: Traditional classroom methods versus online visual methods - A literature review, “ Proc. of the 21st Annual Conf. of the National Advisory Committee on Computing Qualifications, Auckland, New Zealand, 2008, pp. 327-330.

[12] G. F. Luger, Artificial Intelligence: Structures and Strategies for Complex Problem Solving, 6th ed., Addison Wesley, 2008.

[13] jQuery Documentation. Sept. 2009. The jQuery Project: http://jquery.org/, 22.11.2011.

[14] jQuery User Inerface Library. Sept. 2009. The jQuery UI library: http://jqueryui.com/, 22.11.2011.

[15] J. Wielemaker, SWI-Prolog HTTP support. SWI-Prolog's home page: http://www.swi-prolog.org/pldoc/package/http.html, 22.11.2011.

 ISBN: 978-1-61208-180-9