Pairing Prolog and the Web in a Normalization and Denormalization Tool

Lule Ahmedi
Computer Engineering
University of Prishtina
Prishtina, Kosova
lule.ahmedi@uni-pr.edu

Naxhije Jakupi
Computer Science
South East European University
Tetovo, FYR Macedonia
naxhijejakupi@yahoo.com

Edmond Jajaga
Information Technoogies
State University of Tetovo
Tetovo, FYR Macedonia
e.jajaga@unite.edu.mk

Besmir Sejdiu
Computer Engineering
University of Prishtina
Prishtina, Kosova
besmir.sejdiu@gmail.com

Abstract—**The concept of database normalization emerges as very important since the need of redundant-free storages. Its hard understanding by students requires a flexible learning environment and an intelligent behavior by the machine. Our tool NORMALDB integrates both of them in a PHP and Prolog implemented platform providing a complex architecture internally and user friendly interface. Students will effectively learn by working with their own defined examples and at the same time can consult the theory behind each normalization step. On the other side, database teachers can track the tool usage of their students, by getting the information of their actions. This paper provides a detailed description of NORMALDB pointing out the communication between HTML pages and Prolog predicates and spread the challenges faced during the application deployment. Instead of normalization, our tool is the first in the series to support the teaching of denormalization procedure.**

Keywords-E-learning systems; database normalization and denormalization; logic programming; prolog server; web programming;

I. INTRODUCTION

Learning technologies is one of the fields that highlighted the potential of the web for education [1]. 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 [2]. 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. Passing to this level of authomatizm requires introducing the latest programming technologies, through which the machines will behave more like human. Algorithms that simulate thought processes and reasoning that produce behavior similar to humans belongs to a special field called Artificial Intelligence (AI). These software portions of applications are known as intelligent agents, which can learn from the interaction of the human and the machine.

The AI based e-Learning applications have the potential of producing realistic environments in which students can perform learning. In this simulated environment the student interacts with the intelligent agents, which in turn perceive changes and take appropriate actions. Such a feature is very important in the domain of e-Learning, especially in subject-based e-Learning. These applications are an example of providing support for even more complex subjects to learn [3], 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" [4]. 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 (ICs), 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. It is designated to provide theoretical background on the subject (normalization and denormalization of database schema): it explains stepwise every single detail of the process as a whole. It also provides an interactive interface of learning the subject, driven by the student's given examples. The content organization of the tool is a pure reflection of how the teacher organizes the subject.

The rest of the paper is organized as follows. Section II describes the current state-of-the-art on e-Learning tools for database normalization. A brief review on the main concepts of database normalization and denormalization is given in Section III, Section IV introduces the NORMALDB's system architectural layers and its features. Further, Section V provides the main challenges faced during the system design, whereas Section VI describes the learning methodology using NORMALDB. Section VII includes deployment of NORMALDB in real accessible platforms for testing purposes. Finally, the paper ends with the conclusion and future works section.

II. RELATED WORK

There are already few tools that cover executing normalization, like JMathNorm [5], NORMIT [6], RDBNorma [7] and Micro [8]. To the best of our knowledge, NORMALDB is the first which supports the denormalization step.

JMathNorm [5] is an interactive tool for relational database (RDB) normalization implemented by using Mathematica. The tool includes normalization till Boyce-Codd Normal Form (BCNF) and calculates closure and cover of functional dependencies (FDs). The design approach of our system is similar to this tool. JMathNorm integrates Mathematica for writing normalization modules and Java language for programming the user interface, while we use Prolog and PHP, respectively. Our advantage over JMathNorm is the advantage of using web-based systems over the standalone ones in terms of software distribution, compatibility issues and immediate software updates.

Another tool dealing with the database normalization, NORMIT [6], is a web-enabled Intelligent Tutoring Systems, which is also the first in the series of constraint-based tutors developed at ICTG (University of Canterbury). The emphasis of this system is on problem solving thus complementing the students' class work. Its domain knowledge is modeled using Constraint-Based Modeling (CBM), which enables a student to work on with normalization problems as long as he/she reaches a state known to be true pre-defined by 53 constraints. With NORMALDB our emphasis was on establishing an e-Learning tool which shall not only be complement to the class work, but also to be an independent database normalization "teacher". Additionally, our tool is more complete w. r. t. the inclusion of the normalization concepts like finding minimal cover, making the decomposition lossless and projecting FDs.

Micro [8] and RDBNorma [7] are tools that employ linked lists to model the relation entity along with its FDs. Micro uses two linked lists to store the relation, the first one stores all the relation attributes while the other one stores FDs. RDBNorma focuses on system performance in terms of space and time consumption by using only one linked list to represent a relation along with FDs holding on it. In our tool the relation schema and FDs are modeled using two separate Prolog lists. Our emphasis during the system design was on providing a more user friendly GUI and a flexible learning environment, while authors of RDBNorma does not deal with user interface issues.

Generally, 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 that implements the normalization and denormalization of a given database, 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 roll backing 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 content blocks across individual steps / subtopics is another strong reason to leverage NORMALDB. The level of interactivity, the ease of use, its logic-base of rules and the Web interface make NORMALDB unique among existing tools, which support normalization.

III. 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 [9] 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 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 FDs [10].

Together with the input schema, FDs provide the initial information from which it is produced normalized relations i.e., anomalies-free relations. Namely, FDs holding over a relation influence a relation to be split in two or more relations. This technique is known as relation decomposition. A decomposition of a relation schema R consists of replacing the relation schema by two (or more) relation schemas that each contain a subset of the attributes of R and together include all the attributes in R [10]. Whether to decompose a relation or not, it depends on the desired level of redundancy. Additionally, in order to not losing any information when performing relation decomposition we need to be aware of two issues:

- Lossless-join decomposition, which enables to recover any instance of the decomposed relation from the corresponding instances of the smaller relations.
- Dependency-preserving decomposition, which prevents from expensive joins of derived relations by enforcing original relation's FDs on each of the derived relations.

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 (NFs) have been [10] introduced in terms of FDs as follows:

- 1NF – First Normal Form: A relation R is in first normal form if and only if all underlying domains contain atomic values only.
- 2NF – Second Normal Form: A relation R is in second normal form 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 if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key.
- BCNF – A relation R is in 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 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.

Regarding the normalization theory it is necessary to mention the concept of attribute closure, since every normalization algorithm, including 3NF and BCNF algorithms, within the Prolog system for normalization [4] uses it. Closure of a set of attributes X with respect to a set of FDs as the set of all attributes A such that X → A can be deduced by Armstrong's axioms [4, 10].

B. Denormalization

Normalization of a relational schema with the given 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, known as 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 [11].

The two most common types of denormalization are:
- **Two Entities in a One-to-One Relationship:** The tables for these entities could be implemented as a single table, thus avoiding frequent joins required by certain applications.
- **Two Entities in a One-to-many Relationship:** Sometimes logical design results in very simple tables with very few attributes, where the primary key is a foreign key in another table you want to join with. In such cases, when a query wants data from both tables, it may be more efficient to implement them as individually named columns as an extension of the parent entity (table).

The procedure of denormalization need to follow these steps:

First of all there is a need to make sure that normalization process was done by correctly applying NF rules. Then will be selected the dominant queries and updates based on the criteria such as high frequency of execution, high volume of data accessed, response time constraints or explicit high priority. This analysis will result on definition of the tables that require extra columns, when appropriate to reduce the number of joins required for dominant queries. After the recomposition of the tables with extra colums there is a need to consider also the data integrity due to denormalization [9].

Denormalizing databases is a critical issue because of the important trade-offs between system performance, ease of use, and data integrity [12]. Thus, a database designer should have a good reason when deciding to perform denormalization.

IV. THE ARCHITECTURE OF 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 student's 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 is given a description of NORMALDB to reflect its implementation in two layers, the logical and the interface layer.

A. Logical Layer

1) Normalization: At the data tier and business logic layer of NORMALDB, we adopted the Ceri and Gottlob's script [4] implemented in Prolog. The Prolog programming language is known for its contributions to problem solving in AI [13]. 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 logical representation of normalization in Prolog [4].

The adopted Prolog script [4] 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 [4], a relation schema `rel` with the set `[a, b, c]` of attributes, and with set of FDs `[a → b, b → c]` 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 [4]:

- `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 FDs defined for REL
- `thirdnf(REL)` - Decomposes REL into 3NF
- `haslj(REL)` - Tests for losslessness of the decomposition of REL
- `makelj(REL)` - Makes the decomposition of REL lossless
- `projectfds(REL, RELl)` - Projects FDs holding for the relation REL to the relation RELl
- `isinbcnf(REL)` - Tests whether REL is in BCNF
- `bcnf(REL)` - Decomposes REL into BCNF
- `formminimize(REL)` - Minimizes the decomposition of REL

2) Denormalization: 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.

Regarding the feature of re-composition of relations we have extended the Prolog code with a pair of rules.

The first one, rule `recompose`, gets the two relations schema to be merged (row 2 and 3), generates a new name for the newly created relation (row 4), ensures there is not any other relation schema in the base of facts with the one it is trying to create by deleting any `NEWREL` instance (row 5), merges two schemas in a new one `NEWSCHEMA` (row 6) and finally calls the rule `recomposerel`, which will build the new relation NEWREL with its schema `NEWSCHEMA`.

```
1.    recompose(REL, REL1, REL2)-->
2.    {schema(REL1, SCH1),
3.    schema(REL2, SCH2),
4.    makerecomposedname(REL1, NEWREL),
5.    retractall(schema(NEWREL, K)),
6.    append(SCH1, SCH2, NEWSCHEMA)},
7.    html(div(\recomposerel(REL,
      NEWREL, NEWSCHEMA))).
```

The second one, rule `recomposerel`, asserts the newly created relation schema to the base of facts, asserts the decomposition information between the original relation and the newly re-composed relation, projects the FDs from the original relation schema to the recomposed relation, minimizes the original relation i.e., if eventually the recomposed relation schema is a subset of another existing relation schema it is discarded from the database, and finally an output information is published in a HTML page created by HTML generators.

Another improvement to the Prolog script was the Prolog DCG parser. Semantic grammar is an engineering technique for constructing natural language understanding systems [14]. The denormalization algorithm of the tool is written with this notation of Prolog. SQL queries given by the user will be read using DCG grammar and then processing them will result in a statistical table of joins between relations. A detailed description follows in the next subsection.

3) Rendering Prolog results into HTML. In addition to denormalization, few more modifications to the Prolog script of Ceri and Gottlob [4] were applied to enable the integration with the web-based module, 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 [15], namely they generate HTML tags, which hold the results retrieved from Prolog predicates.

These generators are built-up using DCG notation rules. The idea behind this is to translate a Prolog term i.e., predicate into an HTML document. HTML generators are predicates from html_write.pl library. This library contains a wide range of DCG rules that enables easy generation of HTML pages.

The DCG non-terminal `html/3 (html(:Spec))` is the main predicate of this library. It translates the specification for an HTML page into a list of atoms that can be written to a stream using `print-html/1`. The expansion rules of this predicate may be extended by defining the multi-file DCG `html_write: expand/1`. Spec is either a single specification or a list of single specifications.

Nested with this predicate can be used another predicates that are build up as per request, which contain some Prolog rules and then the result from this rule is injected within any HTML tag by using any `html` predicate.

B. Interface Layer

Next we discuss the interface layer of NORMALDB, which is mainly developed in PHP language.

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 [16] and jQuery UI [17] help make the NORMALDB interface simple and easy to navigate through.

Figure 1 shows the organization of NORMALDB from 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 a normalization step is called. Connection to the knowledge base does not require any extra procedure from the user, except for a user to be signed in. Simply, the given

connection hyperlink makes a request to the Prolog server, which in turn displays the result.

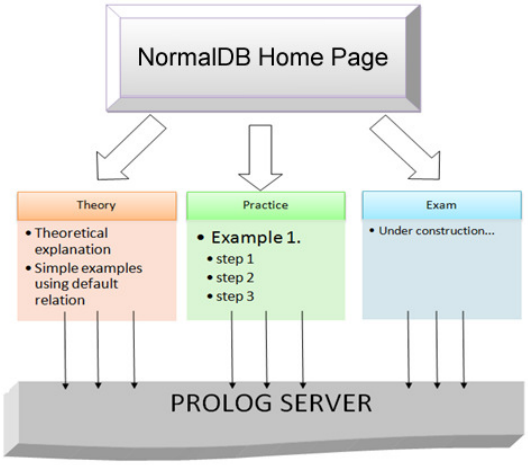


Figure 1. Organization of the interface layer in NormalDB.

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. Also the management of the Document Object Model (DOM) is carried out with jQuery routines. This is applied when the student gives initial relation schema attributes and FDs. The remove() and appendTo() methods are employed to add/remove attributes and FDs.

The whole interface of the tool (colored boxes in Figure 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 FDs 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.

V. IMPLEMENTATION

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 employed 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.

Figure 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 system design and implementation..

A. *Prolog Server*

Prolog is an excellent tool for representing and manipulating data written in formal languages as well as natural languages. Its safe semantics and automatic memory management make it a prime candidate for programming robust Web services [15].

There are two views on deploying Prolog for Web related tasks [15]:

- **Prolog like an embedded component**: Prolog acts as an embedded component in a general Web processing environment. In this view it is a component that can be part of any of the layers of the popular three-tier architecture for Web application. Components generally exchange XML if used as part of the backend or middleware services and HTML if used in the presentation layer.
- **Stand-alone application:** HTTP allows Prolog to be deployed in any part of the service architecture, including the realization of complete Web applications in one or more Prolog processes.

The later one is the view that is used in our project. The reason why this separation is done is based on the way how Prolog reacts to the inter-platform communication. There are web applications, which require the execution of the Prolog logic only once. Thus, the best suited solution is to follow the application of Prolog as embedded component. Other applications require the Prolog knowledge base to remain active during the user session. In this case the second view is recommended.

In NORMALDB the knowledge base changes several times through normalization steps. Because of this we followed the second approach by employing Prolog server features for handling the communication between HTML and Prolog.

Figure 2 describes how the communication flows between servers and client browser.

Whenever an HTTP request for PHP processing is sent, the Apache server responds with its results.

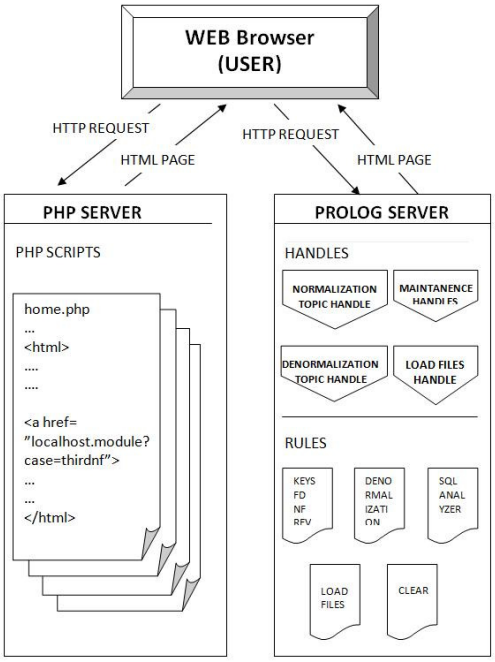


Figure 2. Workflow and request chains of NormalDB tool.

On the other side, when an HTTP request dedicated for Prolog processing is sent to the Prolog server, the later one responds with returning its results. The requests to the servers aredone from the user via clicking to the links appeard in HTML pages.

This way NORMALDB responds to the user requests by exchanging data on two servers at the same time, which illustrates the flexibility of the tool.

B. *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 affects 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 users' future request. 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 Prolog server [15] as the most appropriate solution for surpassing this problem, which will be discussed in detail in the following subsections.

C. *Communication with Prolog server*

The Prolog logic programming language supports a number of libraries for accessing data on HTTP (Hypertext 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 [17].

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.

In a traditional web environment user actions are captured by clicking the links. The links on the pages are formulated based on the desired action. For example, if the user lands in a page for finding the minimal cover of the FDs then the link containing the appropriate handle together with the input variables is constructed by using PHP string manipulation functions and is rendered to the anchor tag. The input variables in most of the cases consist of the name of the relation instance in which the minimal cover will be computed (`bank`) and the name of the predicate in the Prolog script for performing the calculation (`findmincover`).

```
<a href="http://localhost:5000
       /module?name=findmincover
               &relation=bank">
```

Library `library(http/http_dispatch)`, which is defined in the beginning of the script, dispatches the request from HTTP request, where it gets the location of the predicate that will serve the page from the URL and find the required handler.

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. The 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, a thread is employed to search for a predicate supported by the handler, which is defined in the code.

```
home(Request) :-
http_parameters(Request,
    [name(Name,[length >= 1]),
    relation(Rel,[length >= 2])]),
reply_html_page(title('Example'),

    [\html_requires(css('style.css')),
       \case(Name, Rel)]).
```

The HTTP request can include 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:

```
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 the 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 Apache 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.

D. Request processing inside Prolog server

After the appropriate handle is activated, the traversing inside the Prolog server rules and predicates starts in hierarchical manner.

The built-in rule `http_parameters` starts the procedure with extracting the parameters that are required from the following predicates. This predicate fetch and type-check parameters transparently for both GET and POST requests and converts them to atoms.

Parameters that were catch from the HTTP request, after converting to atoms, are passed to the main predicates that will render the HTML page.

```
reply_html_page(title('Example'),
  [\html_requires(css('style.css')),
  \case(Name, Rel)]).
```

The body content of the output pages is build using HTML tag generators written in DCG notation. The main tag generator that is used here is `html/1` predicate. Within this generator can be used other tag generators, like div, table, h2 and so on.

```
html(div([table(class(stats),
   [\theader([ 'Name',
               'Attribute',
               'Relation Key'])
   | \rowssimple(L)])])).
```

E. 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.

F. SQL Parser for denormalization

In the denormalization phase, there is no rule that indicates how to recompose resulted tables from

normalization. Instead, this phase is more intuitive and based on the queries that could be used by the end-user or data warehouse transactions..

To give this logic to our tool, it is required to have the module that will "understand" the queries posed by the user and to calculate how many times each table is joined with another one.

This knowledge is represented in Prolog using DCG notation, which is specific for natural language processing tasks.

The script starts with the rule:

```
sql(s(S,F,K)) -->  select(S),
                   from(F),
                   where(K).
```

which seperates the given query in three parts and continues with analyzing the content of each part.

In the body part of these parsing rules are placed predicates that count the joined relations after detecting the names of tables that are joined within query. This rationale has been implemented with the following predicate:

```
joining(T1, T2) :-
          (joined(T1,T2,N)),M is N+1,
          retractall(joined(T1,T2,N)),
          assertz(joined(T1,T2,M)).
```

VI. CASE STUDY: LEARNING NORMALIZATION IN NORMALDB

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

The *theoretical module* (opens when clicking the green box in Figure 3) 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 (Figure 4). 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.

Figure 3. Home Page

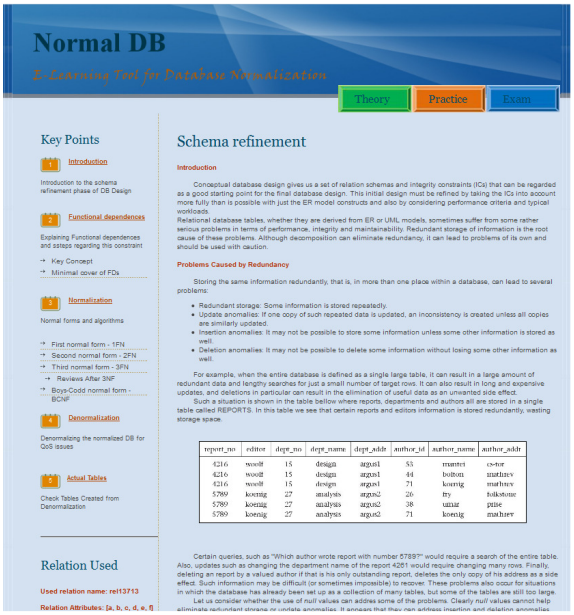


Figure 4. Theoretical part

The result representation in the application is done by using jQuery [16] controls. These controls are activated by clicking the buttons, which send appropriate requests to Prolog server.

In theoretical part, these functionalities are placed in the end of each introduced section, whereas in the practical part these are the main functionalities of the pages.

Navigation to the *practical module* is possible from any page (by clicking the orange box placed at the top of the page), not just while residing at the NORMALDB homepage. Once opened, this module (see the Web page in Figure 5) will first retract all facts belonging to the sample example of the theoretical part, and afterwards eventually load a new relational schema entered interactively by the user through a Web form. If the user provides also the relation attribute types, then the tool will be able to generate a ready-to-deploy SQL script consisting of procedures for creating the database and its tables.

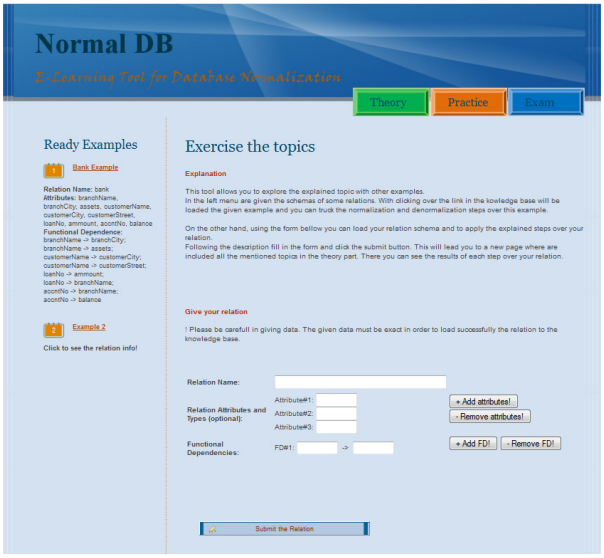


Figure 5. Practicall Part: Form for giving new relation

The relation schema, its attributes and FDs entered through the Web form are processed by PHP string manipulation built-in functions, and are written as new facts in a separate Prolog file. For instance, considering the example of Section III, 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 Figure 5.

In the left menu of this page, are listed different ready examples. By clicking to one of them, the relation that is described will be loaded and then the user can continue exercising with the given topics. Figure 6 shows new loaded ready-given exercise named "bank"

Exercising panel for both, ready-given examples and examples given by users, is the same page. This shows also the dynamicity of NORMALDB.

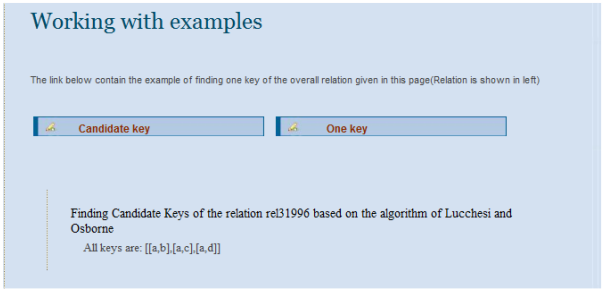
Figure 6. Practicall Part: Ready given example

After uploading the new relation (given by user or ready given example), all links that will be used for further explanation of the topic will use this relation as reference and will apply their rules to this relation.

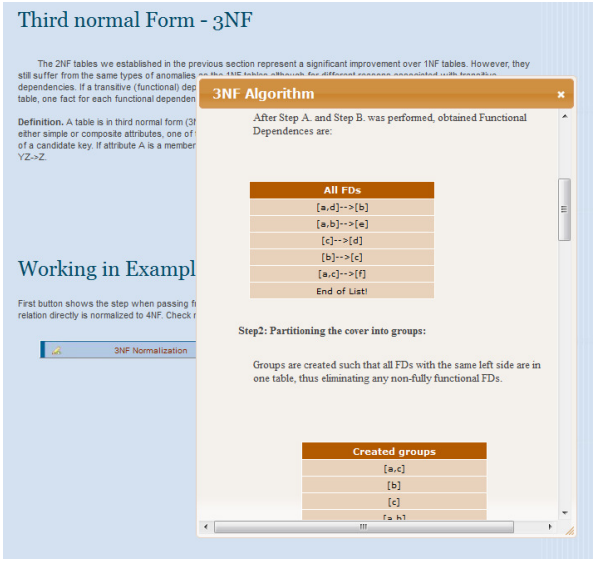
Whenever the user wants to change the relation instance he may move to the last tab and give new relation instance data. When the new relation will be uploaded, the data about the old one will be deleted and removed from the knowledge base.

The upper mentioned initial facts will be stored in the server in a separate Prolog file and will be eventually consulted during the lifetime of the user session (Figure 5).

After uploading the initial data the first thing a user can do is to find all relation keys by clicking the *Assert Key* button, which actually triggers the evaluation of the rule `assertallkeys(bank)` at the Prolog server. As illustrated in Figure 7a the result of our example will be "`All keys are [[branchcame, branchcity, assets, customername, customerstreet, loanno, accntno]]`", rendered at the bottom of the page.



a) Result rendered within the page



b) Result appears in new message box

Figure 7. Result rendering in practical part

The other display format of results is by using jQuery dialog box. This format is used when the resulting display contains more than one Prolog rule execution. Figure 7b displays the 3NF decomposition of our running example.

As one may experience NORMALDB has met the requirement of simplicity in its interface and usage. The organization of the page, its rich set of features 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.

VII. APPLICATION DEPLOYMENT

In order for users to be able to use this tool, we have published it on the Internet and have established an authentication module for accessing the tool. Because the platform of NORMALDB includes the scripting language PHP, JavaScript and the logic programming language Prolog, the PHP scripting files have been published on Apache Web Server port 80, which was installed on a Linux platform and the Prolog server was activated within SWI-Prolog server for handling Prolog requests. The HTTP Prolog server library consists of two mandatory parts and one optional part [15]. The first mandatory one deals with connection management, while the second mandatory one implements a generic wrapper for decoding the HTTP request calling for user code to handle it and encode the answer. Dealing with hosting of the Prolog server, three different approaches were considered:

- One is to run it on a dedicated machine [15] on port 80, the standard HTTP port. The machine may be a running virtual one. The (virtual) machine approach isolates security threads and allows for using a standard port.
- The server can also be hosted on a non-standard port such as 8000, or 8080. However, using non-standard ports may experience issues from intermediate proxy- and/or firewall policies.
- Another approach is to use Apache reverse proxies. This causes the main web-server to relay requests below a given URL location to our Prolog server.

In our case the Prolog server, has been hosted on a dedicated machine on port 80, a standard HTTP port, on Windows platform. Installation of the SWI-Prolog on a Windows system contains the following important features [15]:

- A folder called pl containing the executables, libraries, etc. of the system. No files are installed outside this folder.
- A program swipl-win.exe, providing a window for interaction with Prolog. Another program, swipl.exe, is a version of SWI-Prolog that runs in a DOS-box.
- The file-extension .pl is associated with the program swipl-win.exe. Opening a .pl file will cause swipl-win.exe to start, will switch the directory to the one in which the file-to-open resides and load this file. Additionally, other Prolog files may be consulted within the SWI-Prolog.

A. *Security, Authentication, and Event Logging*

Web security is a set of procedures, practices, and technologies for assuring a reliable, predictable operation on web servers, web browsers, other programs that communicate with web-servers, and the surrounding Internet infrastructure. Unfortunately, the sheer scale and complexity of the Web makes the problem of web security dramatically [18] more complex than the problem of Internet security in general. Today's web security problem has three primary facts:

- Securing the web-server and the data that is on it;
- Securing information that travels between the web-server and the user;
- Securing the end user's computer and other devices that people use to access the Internet;

NORMALDB authentication module offers three different user groups: students, instructors and administrators, where each of the user groups has different permissions. A one's access is enabled through traditional login data with username and password, where the password is encrypted with SHA-1 (Secure Hash Algorithm).

In each page of NORMALDB authentication control is performed, and in the case there is unauthorized attempt for access, the session automatically will be redirected to the login page. For security reasons, during the logging process some of the user's data are recorded, like: Username, Hostname – the computer name from which the user has tried to log in, Ipaddress – user's IP address, AttemptStatus – status of the attempt, which may take value Successful or Failed, and EntryDate – date and time when the attempt was made.

NormalDB also provides the feature of event logging where every user's action gets evidence. The logging event includes the following data: Username – the user who approaches the tool, SessionId – a unique ID that identifies each session to the application, EntryDate – date and time when the attempt was made, Actions – describes the actions taken by the user e.g., New Example means the user has practiced a new normalization example, and Details – contains information about the input relational schema and the input FDs. Figure 8 shows a view of event logging.

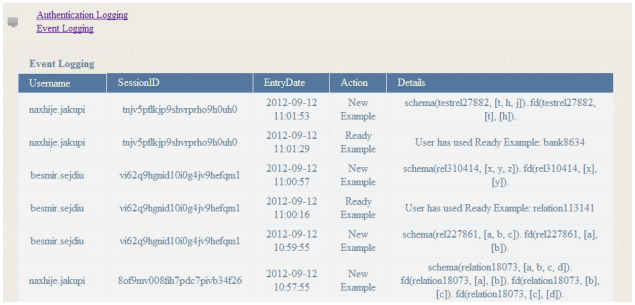


Figure 8. Event logging in NormalDB

VIII. CONCLUSION AND FUTURE WORK

This paper introduced NORMALDB, an e-Learning tool, which offers 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 features including experiments with self-chosen examples are some of the main advantageous features of NORMALDB. For the purposes of friendlier UI and efficient client-server interaction several jQuery and jQuery UI routines were applied. Additionally, some Prolog rules were added to the Prolog script for database normalization [4], mainly for supporting denormalization feature of our tool.

During the development phase different challenges were encountered. The main one was to enable an efficient communication between PHP and Prolog. Among several alternatives 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. Syntactically, the communication between HTML and Prolog predicates has been carried out through DCG notation. The HTML rendering is produced through HTML tag generators.

The tool was deployed in two separate servers, namely one for handling PHP requests – Apache server, and another for handling Prolog reasoning – Prolog server. While deploying Prolog server we eventually encountered difficulties, which were solved with deploying it into a dedicated Windows-based hosting server.

By placing NORMALDB in real environments, it was possible to check and test the communication between Apache and Prolog server. We plan to evaluate NORMALDB by placing it for use by database students among our future plans around the NORMALDB.

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 NFs is one of our future plans, besides adding new visual effects that will enable graphical interpretation of relations' decomposition. Adding higher NFs is easy to implement, since the interface is flexible for adding new features and as stated in [4] the knowledge base can be expanded.

REFERENCES

[1] L. Ahmedi, N. Jakupi, and E. Jajaga, "*NORMALDB – A logic-based interactive e-Learning tool for database normalization and denormalization*", The Fourth Intl. Conf. on Mobile, Hybrid, and On-line Learning (eL&mL), Jan. 30- Feb. 4, 2012, pp. 44-50.

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

[3] 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.

[4] 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.

[5] 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.

[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] P. S. Dhabe, Y. V. Dongare, and S. V. Deshmukh, "*RDBNorma: - A semi-automated tool for relational database schema normalization up to third normal form*", International Journal of Database Management Systems (IJDMS), vol. 3, no.1, Feb. 2011.

[8] Du H and Wery L, "*Micro: A normalization tool for relational database designers*" Journal of Network and Computer application, vol. 22, Oct. 1999, pp. 215-232, doi: 10.1006/jnca.1999.0096.

[9] R. Stephens, "*Beginning Database Design Solutions*" Wiley Publishing, 2008.

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

[11] 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.

[12] G. L. Sanders and S. K. Shin, "*Denormalization effects on performance of RDBMS*", Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34), Jan. 2001.

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

[14] R. R. Burton and J. S. Brown, "*Semantic grammar: A technique for constructing Natural Language interfaces to instructional systems*", 1977.

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

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

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

[18] S. Garfinkel, "*Web Security, Privacy and Commerce",* 2nd ed., O'Reilly, November 2001.

[19] 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.

[20] 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.

[21] 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 Zeland, 2008, pp. 327-330.