



National University Of Computer and Emerging Sciences



PhpMyAdmin – Synchronizing Databases and Tables

Sehrish Abdul Malik
Zahra Naeem

Supervisor: Dr. Fakhar Lodhi

B.S. Computer Science
Final Year Project: March 2010

Department of Computer Science
FAST NU, Lahore, Pakistan

Anti-Plagiarism Declaration

This is to declare that the above publication produced under the:

Title: Synchronizing databases and tables

is the sole contribution of the authors and no part hereof has been reproduced on **as it is** basis (cut and paste) which can be considered as **Plagiarism**. All referenced parts have been used to argue the idea and have been cited properly. We will be responsible and liable for any consequence if violation of this declaration is determined.

Date: April 5, 2010

Student 1

Name: Sehrish Abdul Maalik

Signature: _____

Student 2

Name: Zahra Naeem

Signature: _____

Table of Contents

Abstract.....	4
1. Introduction.....	5
2. Goals and objectives.....	5
3. Scope of the project.....	6
4. Background.....	6
5. System Requirements.....	8
5.1 Functional Requirements.....	8
5.1.1 Synchronization.....	8
5.1.2 Structure Synchronization on database level.....	8
5.1.3 Structure Synchronization on table level.....	8
5.1.4 Data Synchronization on database level.....	9
5.1.5 Data Synchronization on table level.....	9
5.1.6 GUI Requirements.....	10
5.2 Non-Functional Requirements.....	10
5.2.1 Usability.....	10
5.2.2 Development considerations.....	10
5.3 Software Requirements.....	11
6. Detailed Design and Architecture.....	11
7. Testing.....	13
7.1 Scenarios.....	13
7.1.1 General.....	13
7.1.2 Back-end.....	13
7.1.3 GUI Testing.....	14
7.2 Test Results.....	14
8. References.....	16
9. Appendix-I.....	16
10. Appendix-II.....	17

Abstract

phpMyAdmin is an open source tool written in PHP intended to handle the administration of MySQL over the World Wide Web. It can perform various tasks such as creating, modifying or deleting databases, tables, fields or rows; executing SQL statements; or managing users and permissions.

Keeping in mind, the increasing need of updating and maintaining several copies of databases, we came up with the idea of developing the “Synchronization” feature. This feature intends to synchronize a target database with the source (latest modified master copy) database. The synchronization works one-way i.e. from source to the target.

Synchronization allows users to keep the structure and data of the databases in synch with each other. For this process, it is necessary to first track all the changes between the source and the target databases. And then apply them according to the user’s choice. But, the structure difference must always be applied before the data difference.

The project is part of a research initiative being conducted at SERC to study and understand the open-source development processes and incorporate them in the dissemination of Software Engineering Education.

1. Introduction

phpMyAdmin [1] is an open source web client for *MySQL* [2] – a very popular opensource relational database management system. *PhpMyAdmin* supports a wide range of operations available within *MySQL* ranging from simple data definition and manipulation operations (including creation, updation of tables and views, insertion/updation/deletion of records) to advanced database administration tasks. Table 1 below summarizes some of the most frequently used operations in *MySQL*.

Table 1

<ol style="list-style-type: none">1. browse and drop databases, tables, views, fields and indexes2. create, copy, drop, rename and alter databases, tables, fields and indexes3. maintenance server, databases and tables, with proposals on server configuration4. execute, edit and bookmark any SQL-statement, even batch-queries5. manage MySQL users and privileges6. manage stored procedures and triggers7. Import data from CSV and SQL8. Export data to various formats: CSV, SQL, XML, PDF, ISO/IEC 26300 - Open Document Text and Spreadsheet, Word, Excel and others9. Administering multiple servers10. Creating PDF graphics of your database layout11. Creating complex queries using Query-by-example (QBE)12. Searching globally in a database or a subset of it13. Transforming stored data into any format using a set of predefined functions, like displaying BLOB-data as image or download-link
--

However, a very important feature that was as yet missing from *phpMyAdmin* is of Synchronization, which can be used to synchronize two databases on the same or remote servers.

This project is about adding support for database synchronization in *phpMyAdmin*. The feature determines the difference between two relational databases – one acts as a source, while other as a target. Source database is used as the base database for finding the differences in the target database. Once, the difference has been determined, changes could also be made in the target database in order to synchronize source and target databases.

2. Goals and objectives

This project is about providing the synchronization support in *phpMyAdmin* for *MySQL* databases. Such functionality is particularly helpful in the scenario of replicated databases.

At high level the project aim is to add support in *phpMyAdmin* for determining difference between a source and a target database. The following broad functionalities shall be supported:

1. User can apply the difference to the target database so that the target database is completely synchronized with the source database
2. User can also apply only selected changes to the target database.
3. Remote database comparisons are also supported i.e. databases that reside on different host machines can also be synchronized.

3. Scope of the project

This project covers only one-way synchronization. One-way synchronization implies that one database acts as a base and differences are taken from it and applied to the target database. Synchronization includes structure and data synchronization.

Structure synchronization includes making the structure of the databases i.e. schema of the databases same. This requires comparing the two databases to be synchronized and finding the structural level differences. Then formulating the appropriate queries and applying them to finish these differences.

Data synchronization is about making the data of the two databases same. For this, insertion, updation and deletion could be done to remove the differences.

Requirements that are out of scope of this implementation but may be considered later, include:

1. Two-way synchronization
2. Optimized difference determination and application

4. Background

The problem of database synchronization and comparison has multiple levels of definition. This largely depends upon the domain in which the problem is being addressed but it can be broadly categorized into two dimensions – structure and data.

The data mining community is often interested in analyzing data to detect change over time and identify patterns of interest. For instance, a retail chain having multiple sales outlets may have different customer buying patterns at its various outlets or simply at different points in time. A data miner may be interested in comparing these databases maintained at different outlets (and/or their different snapshots) to identify the factors that drive the changes in these buying patterns.

The primary thing that helps in identifying these factors is the difference in the data. Thus, database difference is characterized as difference in data [3]. There are a number of studies that deal with this and there are numerous ways it is defined in literature. Some define it as a

database distance – the number of tuples contained in one database but not in the other [4]. Some define it as a sequence of insert, update and delete operations required to transform one database to other [5]. Yet, some define it as a minimal number of set-oriented insert, delete and update sequences required to do the task [6].

The basic premise behind this is that if you can find the difference between the two databases then you can also find a sequence of operations that can transform one database into the other, and this is equally true vice-versa (i.e. if sequence of transformations is identified then the difference is also determined). Thus, in other words, the comparison and synchronization problem are two different representations of the same problem.

Then, there are other applications of this problem, for instance, in the area of database load management. A database application having a huge user-base can significantly slow down the response time. A standard technique used to solve this problem is to create different replicas of the same database and route the incoming user traffic to multiple databases instead of just one. Thus, the load is distributed over several databases but for the end-user everything happens in a seamless fashion. The problem here is that as the users make changes to these separate databases, differences in the data emerge that leads to issues of data integrity. To correct these, the differences among these different replicas are determined and eventually applied to synchronize all the replicas.

This is such a common problem that all popular relational database management systems – both proprietary (such as Oracle, Microsoft Sql Server, Teradata, IBM DB/2, Informix, etc) and open source (such as MySQL, PostgreSQL, etc) provide support for replication and synchronization. Generally, all these products advertise these feature-sets as database replication/synchronization, where it actually deals with the data replication and synchronization only.

However, the database is not defined in terms of data only. Database is defined as a set of schema and data objects [7]. Schema defines the structure or information about data, which is in other words known as metadata. It captures essentially the data types, relationships and constraints that should hold for the data [7].

The metadata is as important as the data itself and it holds significant importance in the area of Data Modeling. Data Modeling is an activity concerned with creating a design of database. Designers use metadata information to analyze a database at an abstract level. Often, it involves comparing two or more different designs to identify and pick the best one. For this purpose, a number of data modeling tools provide facility to compare two designs or structure of two databases.

Moreover, when an application is in maintenance phase changes to database schema/structure may happen in order to add some new functionality or to fix some existing bug(s) or for query optimization and database tuning considerations [7]. These changes are tested in the development version of the application but once the changes are approved, they need to be ported to the production version where the application is live and running. Again, an often used strategy is to compare the two databases and apply all the schema related differences in the target database (i.e. production database). Data modeling related and/or other client-side tools help in this regard.

Then, there are also situations when database needs to be migrated from one platform (read RDBMS) to another and this includes both – structure and data. Tools that allow comparison and synchronization of heterogeneous databases prove extremely helpful in this regard and thus, a number of such client-side tools are available that can help determine and then apply both the structural and data differences between two databases.

However, almost all of these tools are proprietary and commercial and very few are available in the open source domain, a comprehensive listing of which, offering varied feature-set, is available on Wikipedia (please see Appendix-II).

The motivation of this project is to add this very important synchronization feature in *phpMyAdmin* that is a very popular client for *MySQL*, and currently lacks any support for synchronization.

5. System Requirements

This section lists all the requirements including functional and non-functional requirements for the system

5.1 Functional Requirements

Functional requirements of the system are detailed as under:

5.1.1 Synchronization

1. Given two databases, one being the source database and the other being the target should be enabled to synchronize in the following ways:
 - a. Structure Synchronization
 - b. Data Synchronization
2. Synchronization must be applied in the order mentioned above i.e. first Structure Synchronization then Data Synchronization must be applied.
3. The source and target databases can be either local or remote.

5.1.2 Structure Synchronization on database level

The structure at database level means that each table in the source database must have a matching table in the target database. If it does not exist then it should be created.

If it does exist then it should match the structure synchronization on table level as well.

5.1.3 Structure Synchronization on table level

The structure includes column structure, indexes and primary keys applied to each table of the databases being compared

Column Structure

Column structure means that each column of the source table must have a corresponding matching column existing in the matching table in target database. If it does not exist then it should be created. Otherwise, it should match the following criteria as well:

- Field
- Collation
- Null
- Default
- Type
- Comment

If there are certain non-matching columns present in the target matching table, they must be dropped as well.

Indexes

Each index must have a corresponding matching index existing in the matching table. If it does not exist then it should be created. Otherwise, it should match the following criteria as well:

- Key name
- Unique

If there are certain non-matching indexes present in the target matching table, they must be dropped as well.

Primary Keys

Each primary key must have a corresponding matching primary key existing in the matching table. If it does not exist then it should be created.

If there are certain non-matching primary keys present in the target matching table, they must be dropped as well.

5.1.4 Data Synchronization on database level

The source and target databases have all tables, matching. And the state of each table, (whether filled or not) must also match. If it does not match then data should be inserted or removed accordingly.

5.1.5 Data Synchronization on table level

Each matching table of source database must have each row matching with the target table's.

- **Updating:** If a row matches partially then it should be updated.
- **Insertion:** If a row does not match completely then it should be inserted.
- **Deletion:** And if some non-matching rows exist in the target table, they must be removed.

5.1.6 GUI Requirements

1. Firstly, the information regarding the databases should be entered in a form including
 1. Host
 2. Username
 3. Password
 4. Port
 5. Database Name
2. Once both the databases have been selected then their respective tables are presented in tabular form. The source tables being at the left and the target tables at the right.
3. The tables that have same names are placed in top adjacent columns. While rest of the non-matching table names are placed at the end.
4. The difference between the tables is represented by the red and green buttons placed in the column between source tables' and target tables' columns. The red button represents structure difference while green button represents data difference.
5. By clicking any button (red or green), the corresponding table gets selected and the details of the difference appear in a table below.
6. By clicking the "Apply Changes" button at the bottom of the screen, the selected changes are applied. The system then shows a similar screen but now the red or green buttons do not appear for the tables whose changes have been applied.
7. The system always displays a list of the queries (in a scrollable text area) executed as soon as "Apply Changes" or "Synchronize Databases" buttons were clicked.

5.2 Non-Functional Requirements

Non-functional requirements of the system are elaborated as follows:

5.2.1 Usability

The GUI look'n'feel shall conform to the existing GUIs. Each step involved in completely performing the new functionality had to ensure ease of use and understandability of the user.

5.2.2 Development considerations

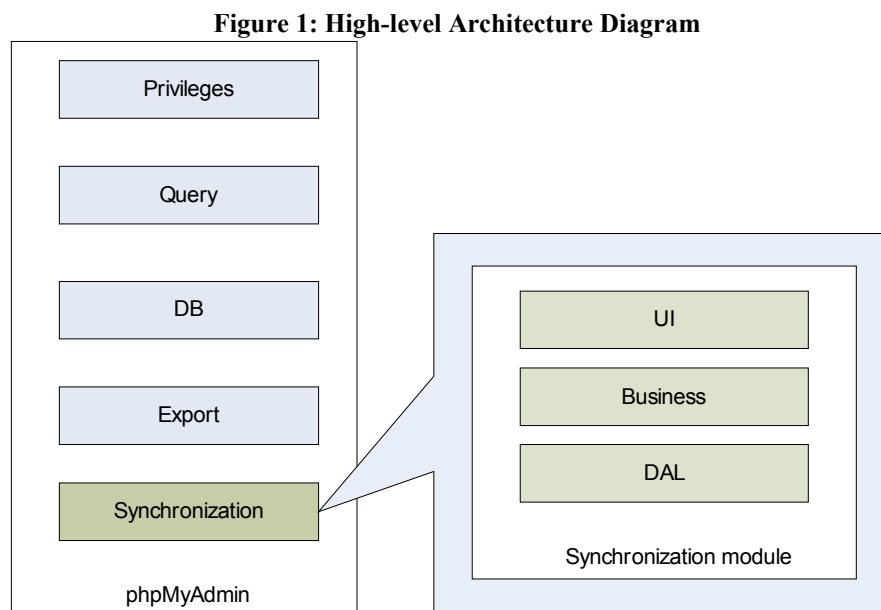
Coding conventions and standards for the PhpMyAdmin will be followed according to the pear coding conventions mentioned on GSoC 2009 coding.

5.3 Software Requirements

- XAMP
- SVN
- NuSphere

6. Detailed Design and Architecture

Figure 1 below presents a simplistic high-level architecture of the system. There are a number of modules already existing in phpMyAdmin that includes (but not limited to) Privileges, Query Manager, DB, Import/Export, etc. Synchronization module is another module that will be added to the system for the purpose of database synchronization. This module follows the 3-layered architecture; where the UI, Business and Data Access layers are kept separate in order to ensure easy maintainability.



The design of Business layer is presented in figures 2(a) and 2(b).

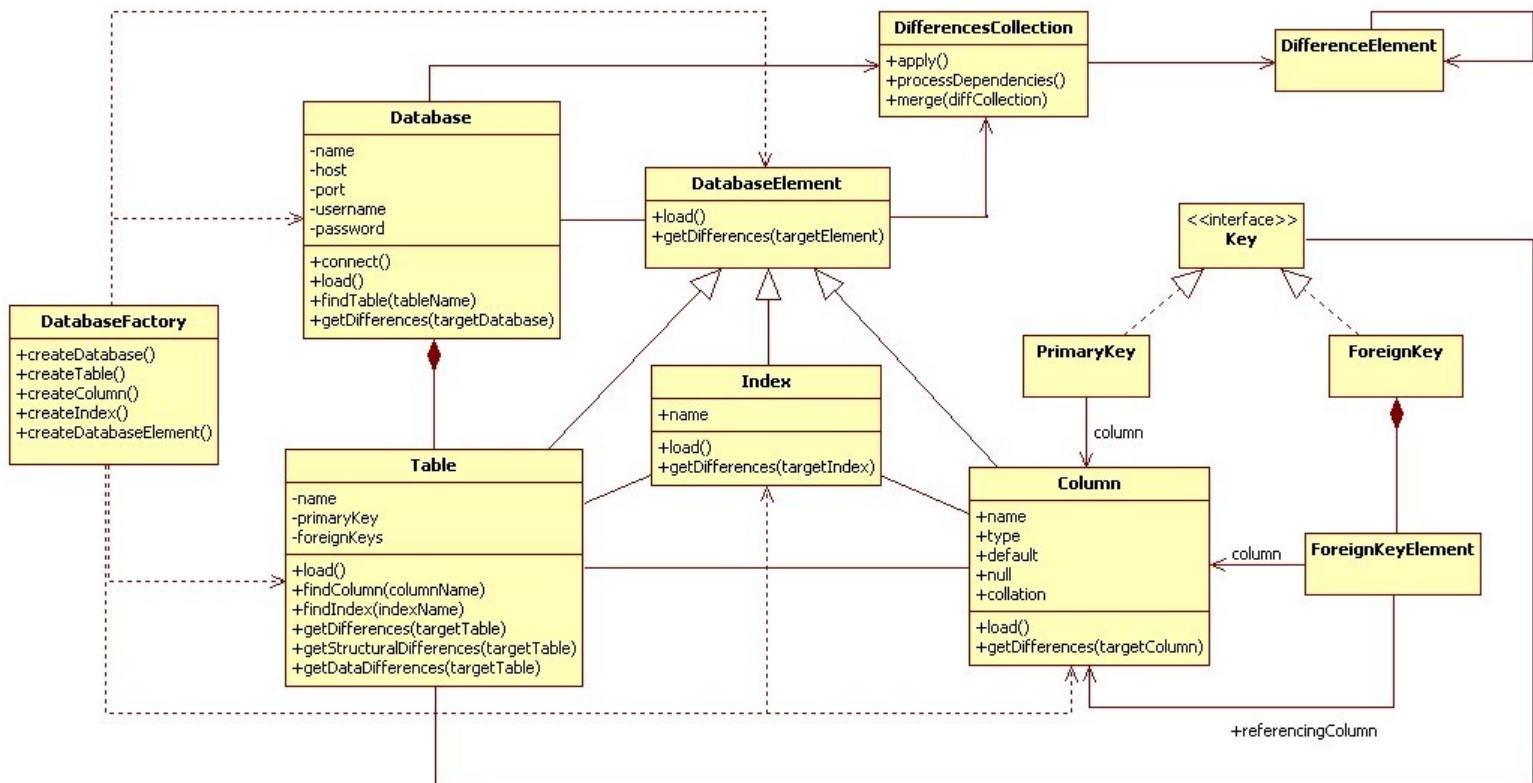
Database object represents a database in the system. It is identified by name, host (IP address of the database management system hosting the database) and port (port of the database management system hosting the database) member attributes. It also has username and password as attributes to establish connection with the database.

Database object is then composed of a set of *Table* objects – each object represents a table in that database – and a set of other *DatabaseElement* objects that may be added as and when required. *Table* is composed of *Column* objects. *Column* is associated with *Index*. Each *Index* may have one or more *Column* objects. The whole scheme follows a Composite approach of creating part-whole structures [9].

Creation of all these database element objects is managed by *DatabaseFactory* class. Apart from specific methods like *createDatabase*, *createTable*, *createIndex* and *createColumn* that are

respectively responsible for creating Database, Table, Index and Column objects; there is a method named *createDatabaseElement* that can be used to create additional database elements that may be added to the framework, for instance, View, Trigger, etc. To support creation of these objects, the DatabaseFactory can be sub-classed overriding the implementation of *createDatabaseElement* method.

Figure 2(a) Detail Design Diagram

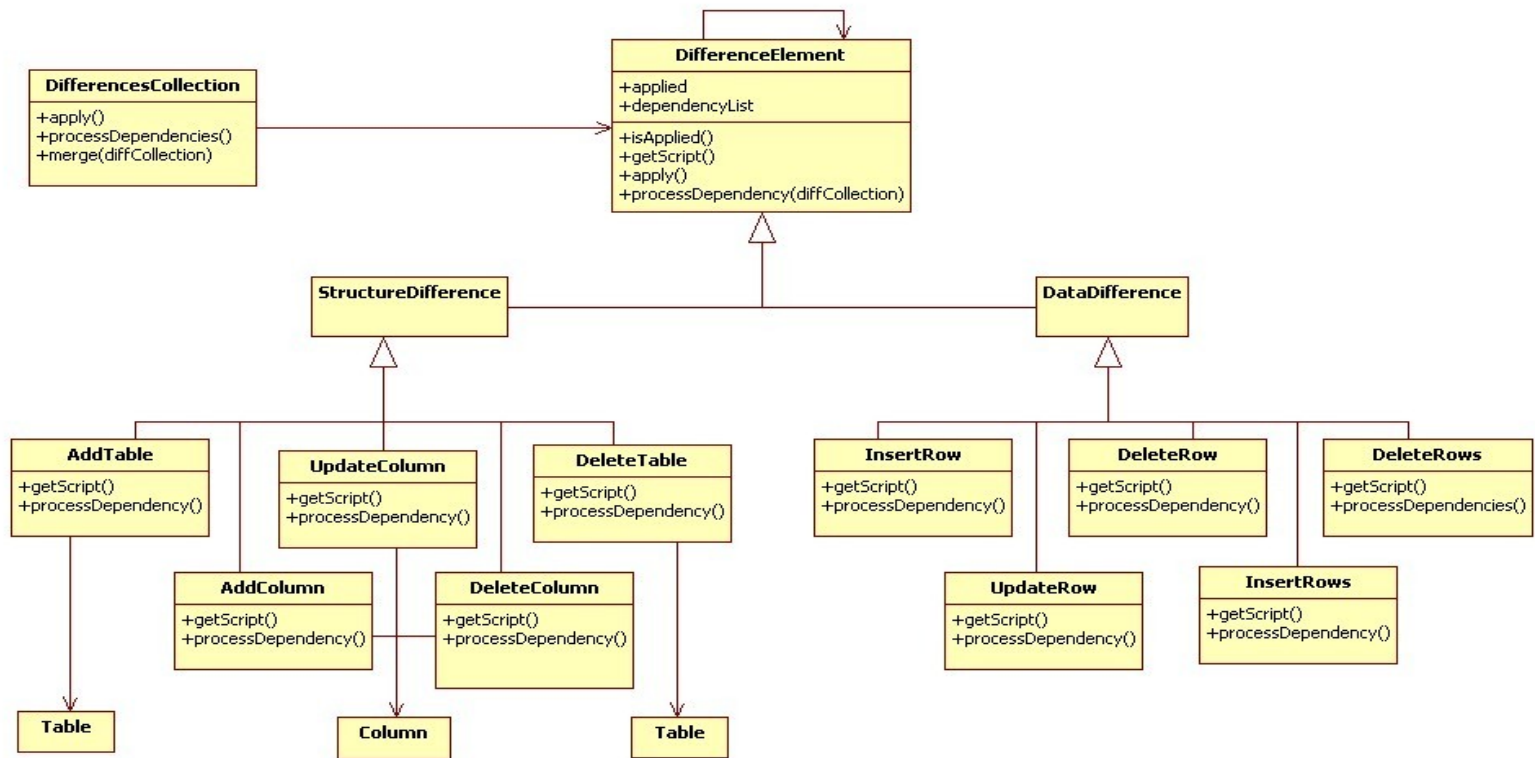


DifferenceElement is used to represent a difference in source and target databases. Broadly, they are categorized into two sets – *StructureDifference* as *AddTable*, *AddColumn*, etc. and *DataDifference* as *InsertRow*, *UpdateRow*, etc. To add a new difference into the framework (such as *AddView*, *UpdateFunction* etc.) one can simply sub-class *DifferenceElement*.

Differences are returned in the form of a *DifferenceCollection*. Two *DifferenceCollection* objects can be merged together through merge method in the *DifferenceCollection* class in order to make a larger *DifferenceCollection*.

Every *DatabaseElement* object has a method *getDifferences* that returns a *DifferenceCollection*. A *DifferenceCollection* contains a set of *DifferenceElement* objects where each *DifferenceElement* object represents some difference between two database elements of the same type (such as Table, Index, Column, etc) – one of the source and the other of the target database. It is the responsibility of each *DatabaseElement* to determine the difference between itself and another *DatabaseElement* if both are of same type.

Figure 2(b) Detail Design diagram



7. Testing

Testing approach primarily involved unit black-box testing, however no formal test cases were developed. General scenarios were tested that are elaborated in sections below

7.1 Scenarios

7.1.1 General

1. Form is submitted correctly.
2. The system displays correct error message on wrong data entry in the form.
3. The system displays the correct response screen on correct data entry in the form.

7.1.2 Back-end

Data synchronization is always performed after the structure has been synchronized. This should be done whenever the user selects the data difference of a table without caring about whether the user selected the structure difference of that table or not.

Structure Difference

1. Column structure synchronized
2. Indexes synchronized
3. Primary keys synchronized

Data Difference

1. Required rows have been updated
2. Required rows have been inserted
3. Required rows have been deleted

7.1.3 GUI Testing

1. Synchronization is being performed only when “Apply Changes” or “Synchronize Databases” button is clicked.
2. The correct queries are being displayed after either of “Apply Changes” or “Synchronize Databases” button is clicked.
3. All the selected tables’ differences are applied after “Apply Changes” button is clicked.
4. The difference (depicted by red and green buttons) appears only corresponding to those tables that bear structure or data difference.
5. Each red and green button when pressed becomes gray in color and the corresponding change’s details appear correctly in the table appearing in the lower part of the screen.

7.2 Test Results

1. When the user clicks the “Synchronize” tab the expected results are that the synchronization form should appear under the tabs. The actual result matches the expected result.
2. When the user fills the synchronization form, no field could be empty except that of password. If any field is empty an alert “Missing value in the form!” appears and the user could not proceed without filling the fields.
3. When the user enters wrong username or password in the source or target databases’ fields and clicks the “Go” button an error message is expected to be displayed saying “Could not connect to the source/target”. The actual result matches the expected result.

4. When the user enters the source or target database's name which does not exist in the server, an error message "xyz database does not exist" is expected to be displayed. The actual result matches the expected result.
5. When the user has correctly filled the form and pressed "Go" button the window showing the structure and data difference of each table of the database is expected to appear. The actual result matches the expected result.
6. If structure difference exists in the source and target table then a red button with "S" written on it should appear in the "Difference" column. The actual result matches the expected result.
7. If data difference exists in the source and target table then a green button with "D" written on it should appear in the "Difference" column. The actual result matches the expected result.
8. If a table exists in the source database but not in the target database then in the "Source Database" column with the table name a "+" plus sign should be prefixed and in the "Target Database" with the table name "(Not present)" should be appended. The actual result matches the expected result.
9. If a table exists in the target database and not in the source then with the table in the "Target Database" a "-" minus sign should be prefixed while "Source Database" column is expected to remain empty for this table. The actual result matches the expected result.
10. When the user clicks red (structure synchronization) or green (data synchronization) button its color should turn grey. The actual result matches the expected result.
11. If the data synchronization button is pressed then in the lower table correct table name should appear in the "Table name" column and in the columns "Update Row (s)" and "Insert Row (s)" correct number of rows that are to be inserted or updated should be there. Rest of the row must display "--" dashes. The actual result matches the expected result.
12. When the structure synchronization button is pressed then ,in the lower table, correct table name should appear in the "Table name" column and in the columns "Create Table", "Add Column (s)", "Remove Column (s)", "Alter Column (s)", "Remove Indexes (s)", "Apply Indexes (s)" correct number of fields that are to be altered should be there. Rest of the row must display "--" dashes. The actual result matches the expected result.
13. After the user has pressed some table's structure and data synchronization buttons and then presses "Apply Selected Changes" button those changes are applied and a success message "Selected target tables has been synchronized with source tables" appears. Then the difference table is expected to show the new state of the source and target database. The actual result matches the expected result.
14. When the user clicks the "Synchronize Databases" button the target and source databases should be synchronized and success message "Target database has been

synchronized with source database” is expected to be appear with the queries that have been executed. The actual result matches the expected result.

8. References

- [1] <http://www.phpMyAdmin.net>
- [2] <http://www.mysql.com/>
- [3] J. Vreeken, M. Leeuwen, A. Seibes, “Characterising the Difference”, International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, 2007
- [4] M. Arenas, L. Bertossi, J. Chomicki, “Consistent Query Answers in Inconsistent Databases”, ACM Symposium on Principles of Databases, Philadelphia, Pennsylvania, 1999
- [5] W.J. Labio, H. Garcia-Molina, “Efficient Snapshot Differential Algorithms for Data Warehousing”, International Conference on Very Large Databases, 1996
- [6] H. Muller, Johann. Freytag, U. Leser, “Describing Differences between Databases”, ACM 15th International Conference on Information and Knowledge Management, 2006
- [7] R. Elmasri, S. Navathe, “Fundamentals of Database Systems”, Fifth Edition, Pearson Education, New Dehli, 2008
- [8] E. Rahm, P. Bernstein, “A Survey of Approaches to Automatic Schema Matching”, The Very Large Databases Journal, Vol. 10, Springer Verlag, November 2001
- [9] E. Gamma, R. Helm, R. Johnson, J. Vlissides, “Design Patterns: Elements of Reusable Object-oriented Design”, Addison-Wesley, 1995

9. Appendix-I

This appendix lists all the important sources of information which have been consulted for this project for the purpose of requirements gathering.

Since the initial part of the project had been a part of Google Summer of Code 2009 therefore we gathered requirements from mentors sitting in Europe and U.S. The mentor assigned to us by Google was Michal Cihar. He blogs at <http://blog.cihar.com/> and can be reached at his email address michal@cihar.com

Following links provided help to understand the feature:

- http://sourceforge.net/tracker/index.php?func=detail&aid=631976&group_id=23067&atid=377411
- http://sourceforge.net/tracker/index.php?func=detail&aid=1741213&group_id=23067&atid=377411
- <http://www.techmixer.com/navicat-mysql-mysql-database-management-tools-for-windows/>
- <http://articles.sitepoint.com/article/mysql-data-sqlyog-job-agent>

10. Appendix-II

This appendix gives a list and comparative assessment of various database administration and modeling tools listed on Wikipedia (http://en.wikipedia.org/wiki/Comparison_of_database_tools) - last accessed on March 11, 2010.

The table below lists the product name and license and whether database comparison feature is supported or not. It also lists if the product is opensource or not.

This is not an exhaustive list of such tools available but covers a number of popular tools.

Product	License	Compare feature?	Opensource?
ACDB	Commercial		
Adminer	Apache license	✓	✓
Advanced Query Tool (AQT)	Commercial		
Altova DatabaseSpy	Commercial		
Aqua Data Studio	Commercial	✓	
Caché Monitor	Freeware		
CompareData	Commercial		
Database Master	Commercial	✓	
dbForge Studio for MySQL	Commercial	✓	
DbSchema	Commercial	✓	
DBTyp.NET	Commercial		
DbVisualizer	Commercial		
Devgems Data Modeler	Commercial		
Dezign for Database	Commercial	✓	
DreamCoder	Commercial	✓	
Embarcadero DBArtisan	Commercial		
EMS SQL Management Studio	Commercial	✓	
Epictetus	Commercial		
ESF Database Migration Toolkit	Commercial	✓	
Happy Fish	Commercial	✓	

HeidiSQL	GPL		✓
ifacedbc2	GPL		✓
Maatkit	GPL		✓
MicroOLAP Database Designer	Commercial		
Microsoft SQL Server Management Studio	Commercial		
ModelRight	Commercial / Free Community Edition	✓	✓
MySql Lite Administrator	GPL		✓
Navicat	Commercial	✓	
Nob Hill Database Compare	Commercial		
Oracle Enterprise Manager	Commercial		
Oracle SQL Developer	Commercial		
OraDeveloper Studio	Commercial	✓	
pgAdmin III	Artistic License		✓
phpMSAdmin	GPL		✓
phpMyAdmin	GPL		✓
phpPgAdmin	GPL		✓
PL/SQL Developer	Commercial	✓	
QuantumDB	GPL		✓
RazorSQL	Commercial		
SchemaBank	Commercial	✓	
SQL Developer	Commercial	✓	
SQL Edge	Commercial		
SQL Maestro	Commercial		
SQL Navigator	Commercial		
SQLPro SQL Client	Commercial		
SQLyog	GPLv2	✓	✓
Squirrel SQL	GPL		✓
Toad	Commercial	✓	
Toad Data Modeler	Commercial	✓	
WWW SQL Designer	GPL		✓