2003

WIKI-style administration of online course content

Jianmin Wang

WIKI-STYLE ADMINISTRATION OF ONLINE COURSE CONTENT

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Chien-Min Wang

December 2003

WIKI-STYLE ADMINISTRATION OF ONLINE COURSE CONTENT

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Chien-Min Wang

December 2003

Approved by:

[redacted]

Dr. David Turner, Chair, Computer Science — 1-dec-2003 Date

[redacted]

Dr. Ernesto Gomez

[redacted]

Dr. Kerstin Voigt

ABSTRACT

What is WIKI? Simply, WIKI is "The Simplest online database that could possibly work," according to Ward's original description. In the other words, WIKI is a piece of server software that allows users to freely create and edit Web page content using any Web browser. The program, "Wiki-Style Administration of Online Course Content" (WAOCC), is a teaching tool which will be used as a teaching communication interface repository of student work. WAOCC can be used as a teaching assistant board. Students can edit their works online and the instructor can review the works and correct the work directly if necessary from the WAOCC. And furthermore, WAOCC may also support some kinds of simple HTML statements like adding a hyper-link or pasting pictures on the web.

ACKNOWLEDGMENTS

I thank the faculty of Computer Science department for giving me an opportunity to pursue my M.S. in Computer Science at California State University, San Bernardino. I express my sincere appreciation to my graduate advisor, Dr. David Turner who offered me this project and directed me through this entire effort. I also thank my other committee members, Dr. Kerstin Voigt and Dr. Ernesto Gomez for their valuable input.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

Wiki-Style Administration of Online Course Content (WAOCC) is a teaching tool that will be used as a teaching communication interface between students and the instructor. By using WAOCC, pages are created and modified easily. From the WAOCC, new pages are created by adding a wiki link to an existing page and the users can modify the same web page even though he/she is not the owner (creator) of the page. To edit the page, users need not know the HTML expressions. WAOCC uses a simplified markup language to specify formatting of pages. WAOCC can be used as a public or private repository of student work. Students can edit their works online and the instructor can review the works and correct the works directly from the WAOCC. Some of the announcements like syllabus or schedules may also be put on WAOCC as read-only pages. And furthermore, WAOCC will also support some kinds of simple HTML statements like adding a hyper-link or pasting pictures on the web pages.

1.1 Purpose of this Project

The purpose of this project is to design, build and implement a teaching assistant system for a normal class. All the pages and user information will be stored in a

PostgreSQL database and retrieved by JAVA Servlet and JDBC. The main purpose of this project is to provide an easy-to-edit and web-base communication environment for the students and the instructor in a class. Moreover, the system offers the authorization function to make sure that the pages are secure from malicious editing. In the system, all the users can manage their own account information such as changing passwords or e-mail addresses. And furthermore, in order to help users remember their passwords and login ids, the password query function which can e-mail users the login ids and passwords is also included in the system.

1.2 Project Products

This project would lead to the following products:

- Implementation of WAOCC: a working web site with JSP programs, Java programs and PostgreSQL database, which would achieve the needs of a communication board of a regular class. All the pages will be watched by the security system in order to keep the information on the pages correctly and even securely.
- Users manual: an implementation manual will be available for the user.
- Systems Manual: a project report (this report)

will be available with design details and specifications.

CHAPTER TWO

WAOCC ARCHITECTURE

This project, Wiki-Style Administration of Online Course Content (WAOCC), implements a web system to provide an environment for the students and the instructor in a class to share information. Thus, the components needed to implement WAOCC are a database server, a web server, graphical user interface components, and a database interface Application Programming Interface (API) to programmatically access the database. The following figure describes the interaction among the components used in WAOCC.

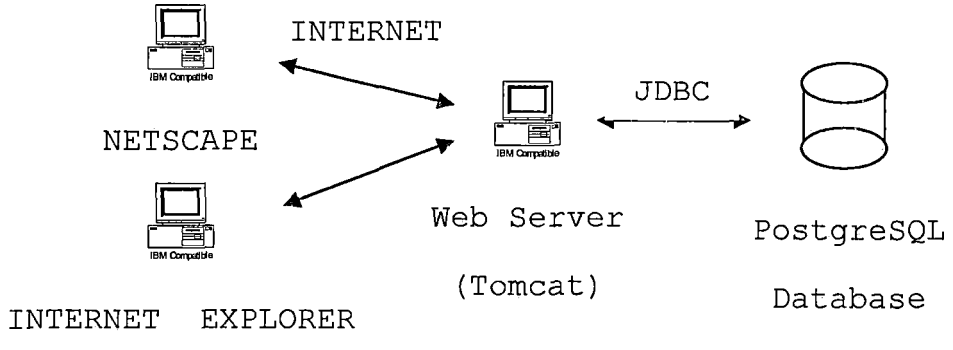

Figure 1. Wiki-Style Administration of Online Course Content Architecture

The components used to build WAOCC were chosen with the following criteria: (i) the components should be shareware, i.e., available freely for non-commercial purposes, (ii) be

part of a standard, i.e., the do not depend on a specific operating system and hence are easily portable across systems with ease, (iii) database server independent, so that new and different versions of the server can be plugged in easily.

The user interface components are built by using HTML 6.0 forms, frames and Javascript. And the applications are launched using the JavaServer Pages (JSP) and Java Servlet. JSP was used because it can use javabean, which provides a reusable way for all programs and java container, Tomcat can be installed under Windows or Linux. Also, it is easy to process whole user input from the HTML forms. The reason that Java Servlets was used is that it has the advantages of portability and efficiency. By using Java Servlets, the Servlet can be executed in any other web server which is the special property of Java Servlets, "write once, serve anywhere". Moreover, Java provides a convenient function, Java Database Connector (JDBC), to connect database.

The database used for WAOCC is PostgreSQL. PostgreSQL is a real multi-user database and is royalty-free open source software. To use it, simply activate PostgreSQL in Linux. Also, the availability of the JDBC driver for PostgreSQL is another important reason to choose it. Moreover, the same code could be used to link with another

database by changing proper JDBC driver, thereby making it database independent.

2.1 Software Interfaces

- Internet browser: Netscape or Internet Explorer.
- Operating system: Windows 98/Me/2000/XP or Unix/Linux.
- Database: PostgreSQL.
- Compiler: JDK 1.4.1.
- Language: HTML / JAVA / JavaScript / JSP.
- Database connector: JDBC.
- JSP Container/Web server: Jakarta Tomcat.

CHAPTER THREE

DATABASE DESIGN

3.1 Data Analysis

The data for designing and implementing the schema of the database depends on the properties of pages and users. The page data needed by the wiki page are title, body, publish setting, modify setting, the owner number, page lock indicator and page lock time. The user data needed by the system are the user number, login id, user name, password and e-mail address. The user number is auto increased and assigned at the time when a user is created. All the user data will be checked by using JavaScript when the user is created. The pages and users are connected by the relation of user number. In order to indicate the relation between pages, the table italics links is also created and maintained by the system. The data in links are connected to pages by the relation of page title. The pages which are accessed more often will be stored in a page pool in order to decrease the access frequency to the database.

3.2 Database Schema Conceptual Model - ER Diagram

In designing the schema for the WAOCC database, two distinct parts have been identified. The first includes page part, which includes page title, body, publish setting, modify

setting and owner number (no). The second includes user number (no), id, password and e-mail address. All the entities and attributes are detailed in Figure 2.

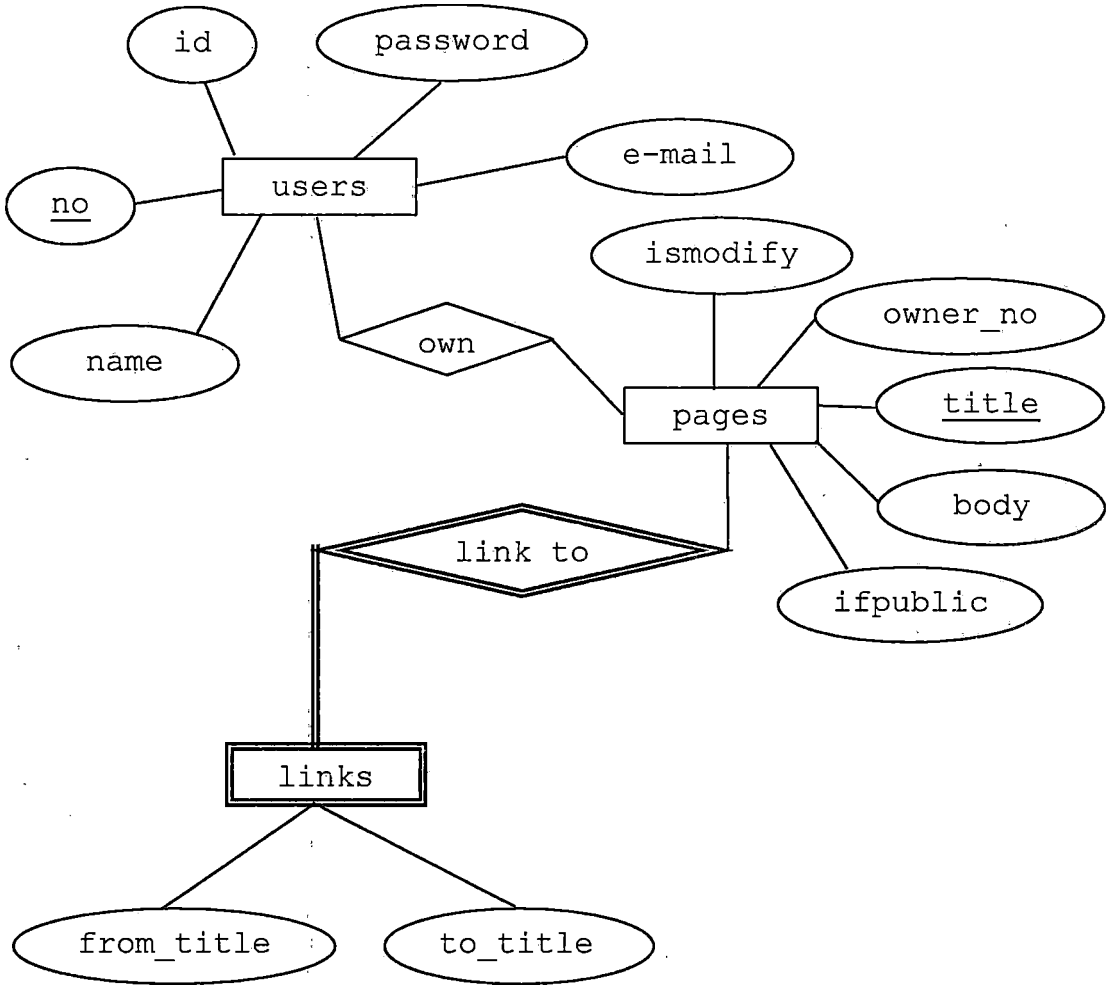

Figure 2. E-R Diagram

3.3 Database Schema Logical Model - Relational Schema

The conceptual model ER diagram maps into the following relational table design. In the following tables, underlined fields indicate the primary key.

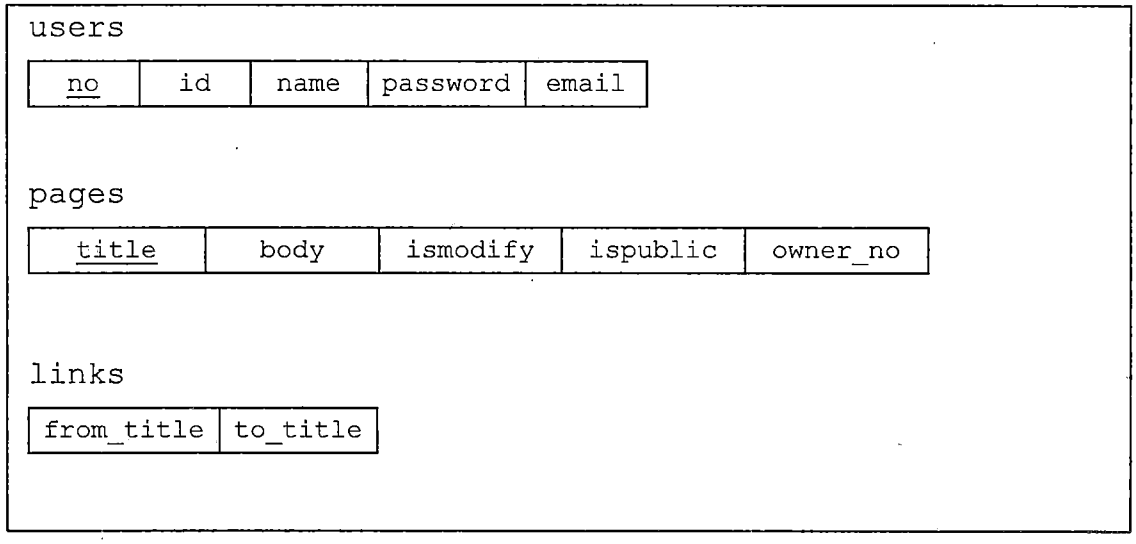

Figure 3. Wiki-Style Administration of Online Course Content Database Relational Schema

3.4 Data Type and Details

The logical model establishes the following detailed design in PostgreSQL database. The following tables describe data type, length, primary key, null or non-null keys, and extra information, such as auto_increment.

Table 1. Structure of Table users

field	type	null	key	default	extra
no	int		PRI		
id	varchar(25)				
name	varchar(50)	YES			
password	varchar(25)				
email	varchar(30)				

Table 2. Structure of Table pages

field	Type	null	key	default	extra
title	varchar(255)		PRI		
body	Text				
ismodify	varchar(1)			1	
ispublic	Varchar(1)			1	
owner_no	Int				

Table 3. Structure of Table links

field	Type	null	key	default	extra
from_title	varchar(255)				
To_title	varchar(255)				

There are 11 vocabularies in the markup language of our system, which allow content authors to incorporate special formatting in their pages. The html objects corresponding to these wiki markup functions include hyperlink, table, level-2 title, image, bulleted item, and code section. The vocabularies can be used in the edit page and will be transfer to the desired format that the author need. Table 4 contains a list showing the elements of our wiki markup vocabulary.

Table 4. Wiki Markup Vocabulary List

Style	Markup vocabulary
Internal_link	/Internal_link
External_link	http://<URL>|External_link
URL	http://<URL>
Add image	Img://<URL>
• Bullet Item	*Bullet Item
Insert a horizontal rule	----
Display an asterisk	*
Display a slash	\\
Insert a table	|| column || column ||
Start a code area	*start-code
End a code area	*end-code

4.1 WAOCC Graphical User Interface Design

WAOCC GUI is easy to use. The GUI is written using Hyper Text Markup Language (HTML). All the functions that the user has are placed in the menu part which is the upper region of the page. And, the wiki page is placed at the body

part which is lower region of the page. Also, it also uses JavaScript to check the user input accuracy. All the inputs which are not acceptable by the system will be reported by an error message box. Hence, the WAOCC GUI is executable with browsers that support javascript. The following sub sections explain the GUI work and details.

4.1.1 WAOCC Home Page

This page will be the first page that all the users will see when they enter WAOCC. The home pages will be owned by the administrator of WAOCC and the properties of "visible by others" and "modifiable by users" will be set to be true as default value, which means everyone except guest can view and edit the page. The default content of the homepage will be empty. All the users includeed guests can browse this page unless the administrator disables the property of "visible by others" for the page. In order to edit the page, simply double click on the page or click the "Edit" function in the menu at the top of the page then the Contorller Servlet will forward to edit page. The user then can use the markup language of WAOCC and edit the page.

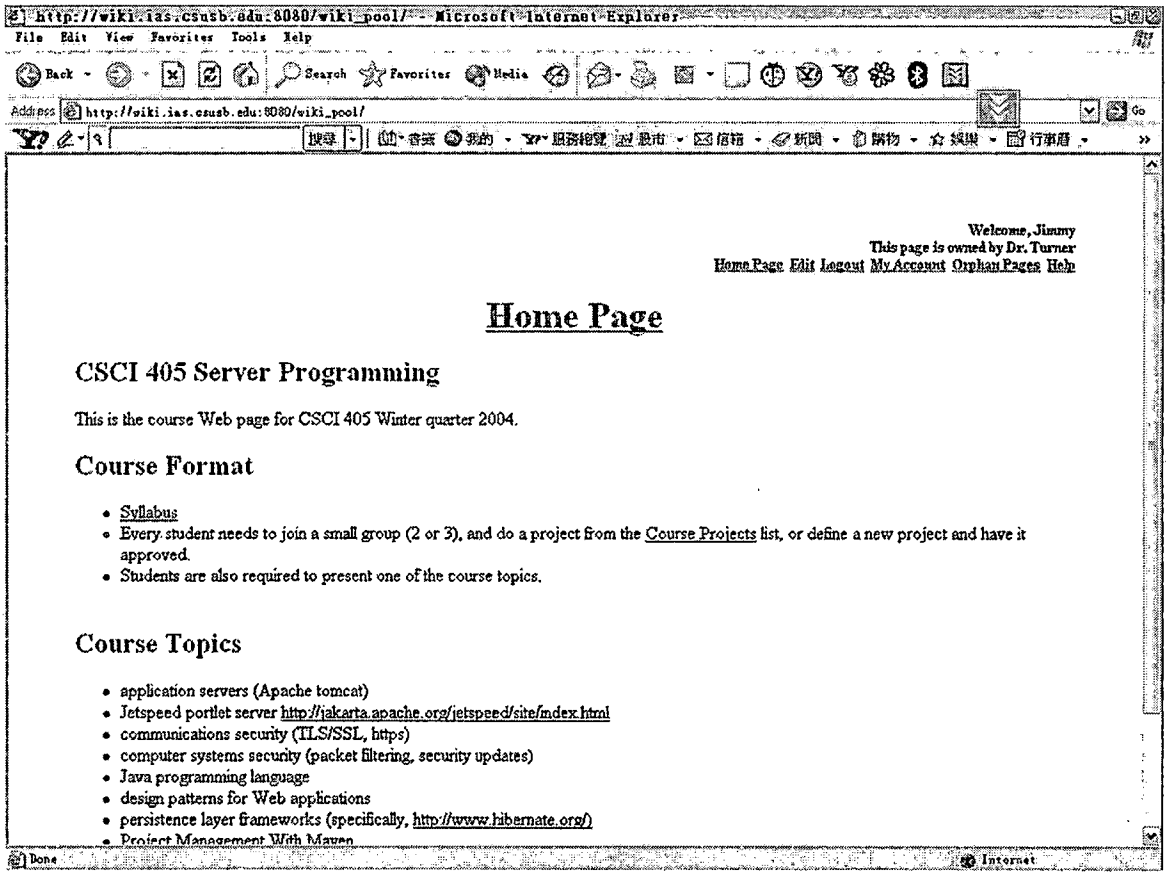

Figure 5. Home Page of WAOCC

To edit a page of the WAOCC, the user must not be a guest and must have the privilege to edit the page. Otherwise, the user will not be given an edit link, nor will the javascript double click edit function be enabled.

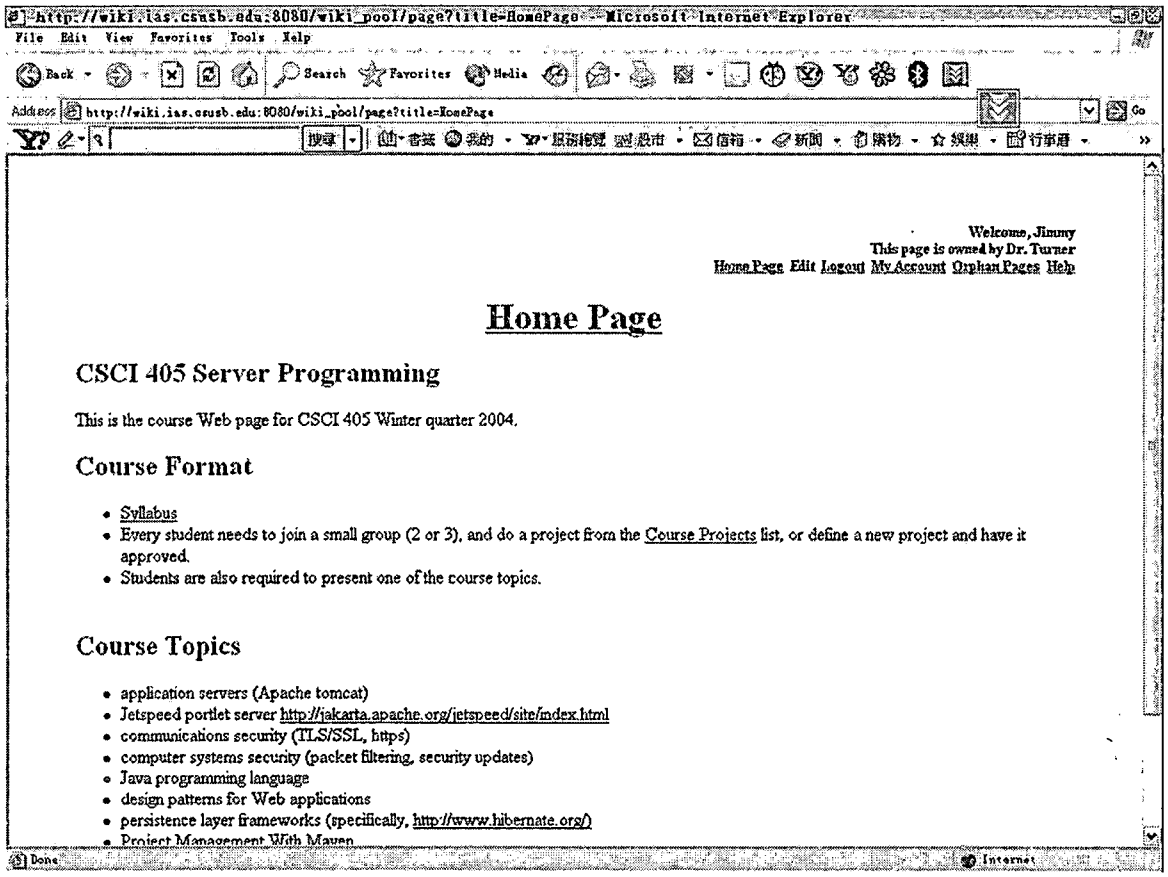

Figure 6. Disabled Edit Function for User Have No Privilege to Edit

4.1.2 WAOCC Login

The user logs in by providing a user id and a password that are created by the administrator who is the instructor of the class. After the servlet verifies the user id and password, it forwards to a jsp page, which will show the main page. Moreover, the user information will be saved in the session for later use, and the session will be killed when the browser is closed or when the user is idle for 1800 seconds (30 minutes) which is the default session life time

set up by Tomcat Server. The system will display different menu items based on the privileges granted to the user. If the user id or password is wrong, the program will show an error message and the user can re-login. For guests, there is no need to check the database. The default user who does not login will be treated as a guest who can only browse the pages which are published as public.

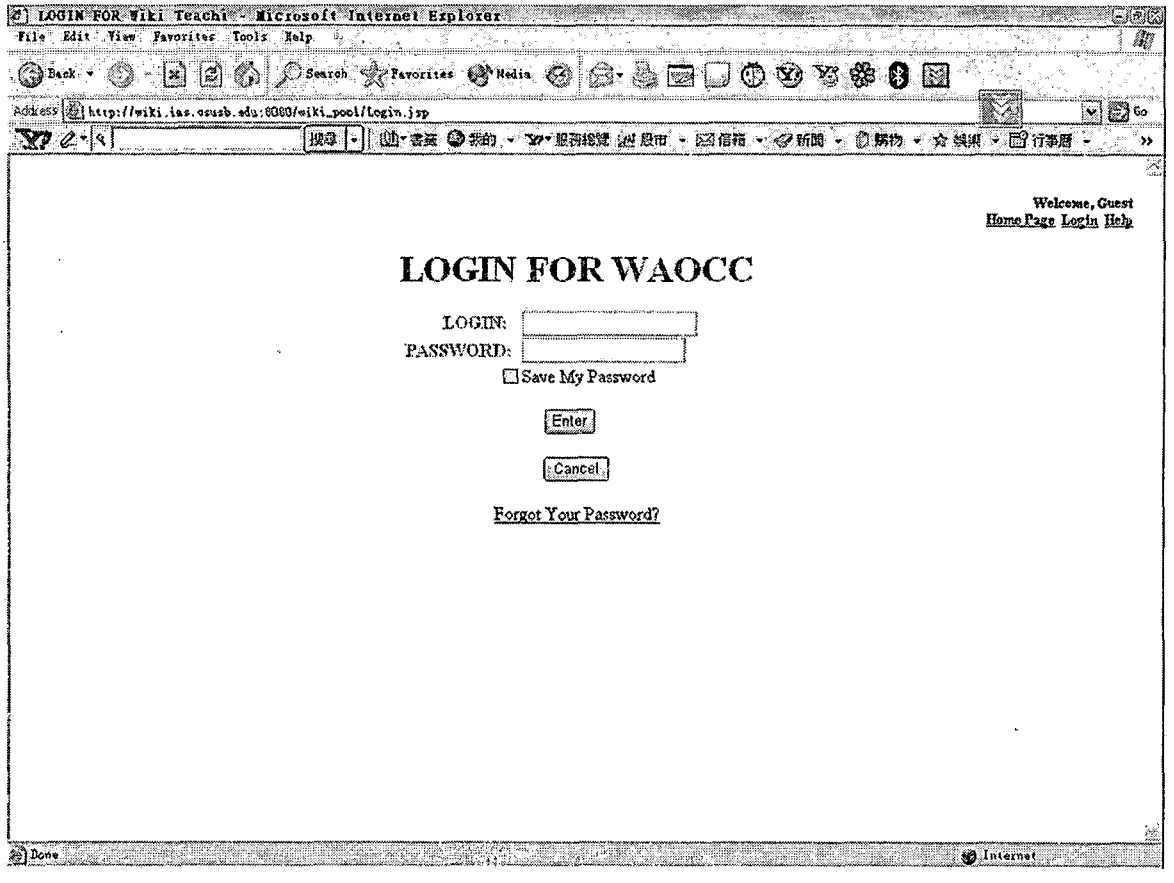

Figure 7. Login Page

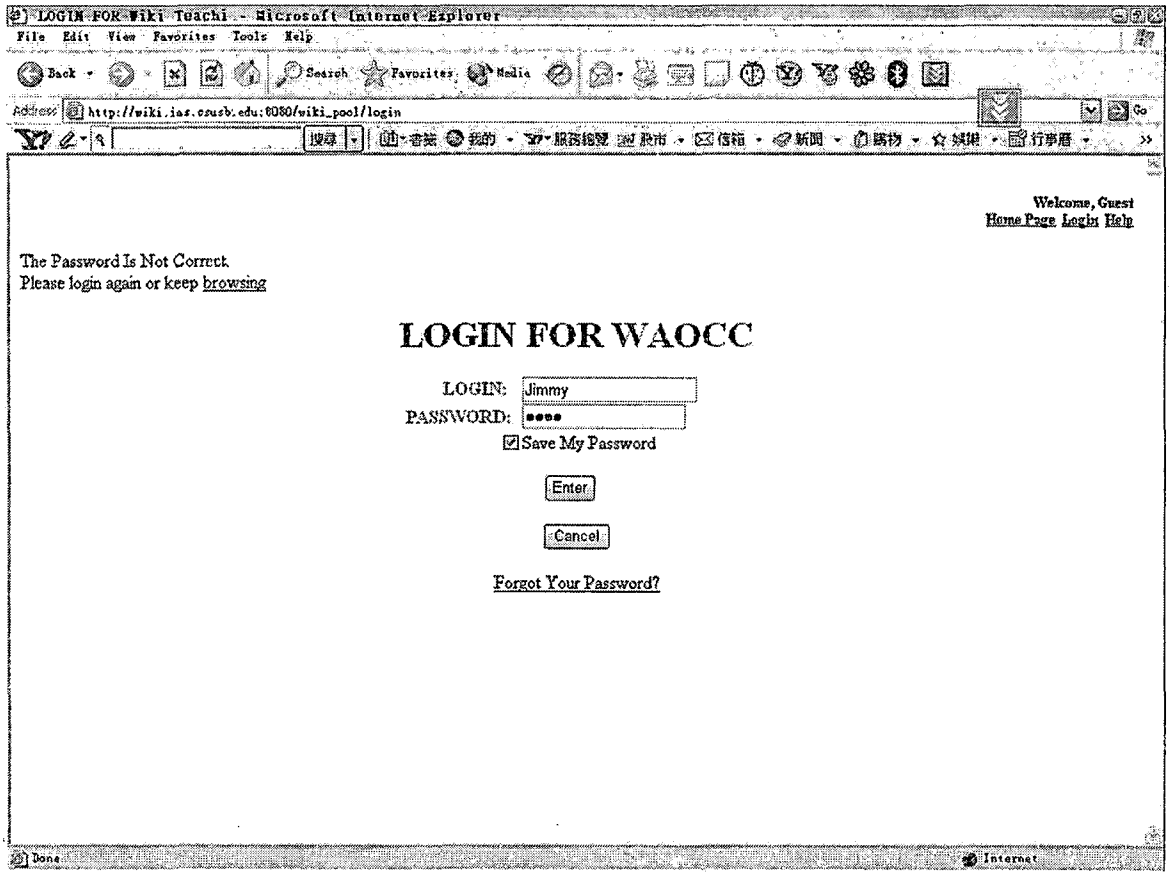

Figure 8. Login Error at Login Page

Moreover, the system offers the function to save the password in a cookie, so that the next time the user visits WAOCC, the system will automatically login the user. To save the password in the cookie, simply check "Save My Password" under the Password field, and press Enter; if the authorization passes, the password will be saved in the cookie for future use.

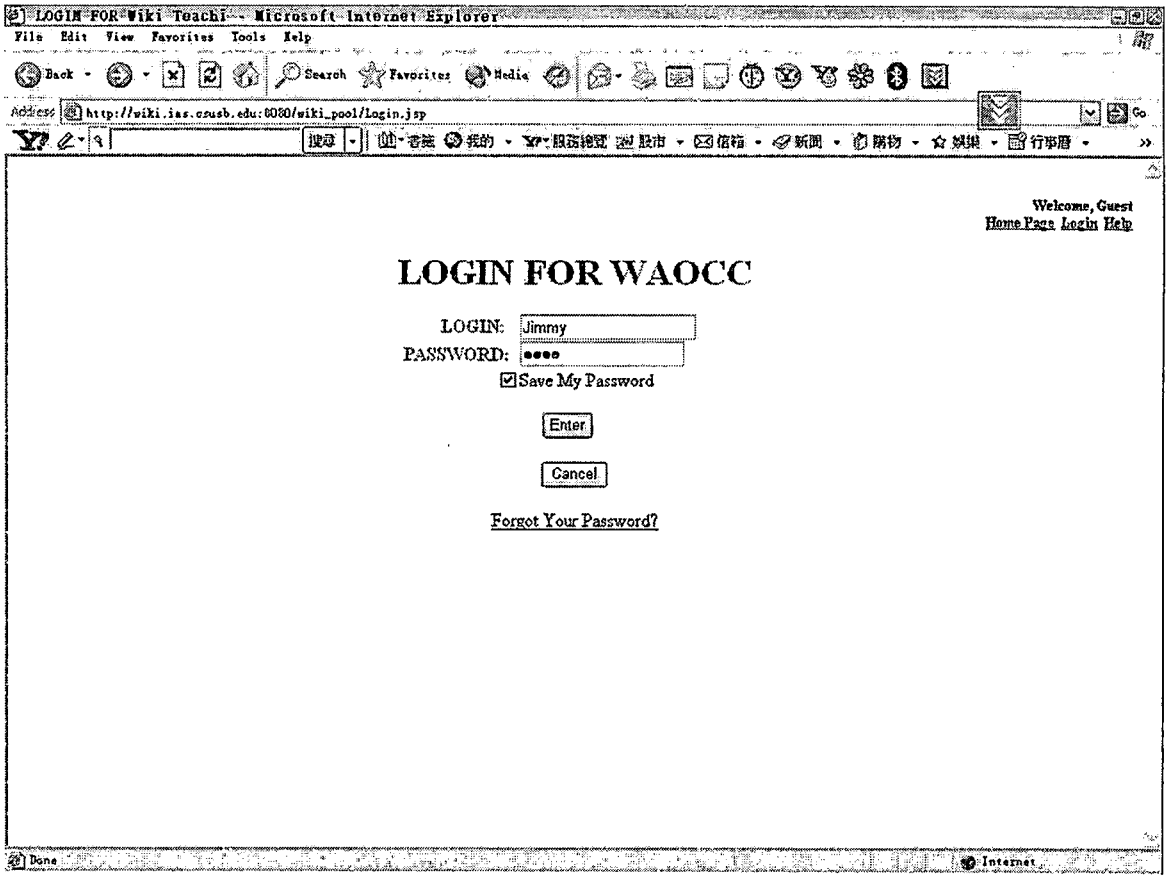

Figure 9. Save Password

When a user forgets his/her password, a password query function is also offered by the system. Simply click the link "Forget Your Password?" under the login page, then the jsp will redirect the page to the password query page. In this page, the user will need to offer his/her e-mail for authorization, and then the system will look for the e-mail address in the user information database. The login id and password will be e-mail to the e-mail address once the e-

mail is found in the database. Otherwise, the system will show an error message on the screen.

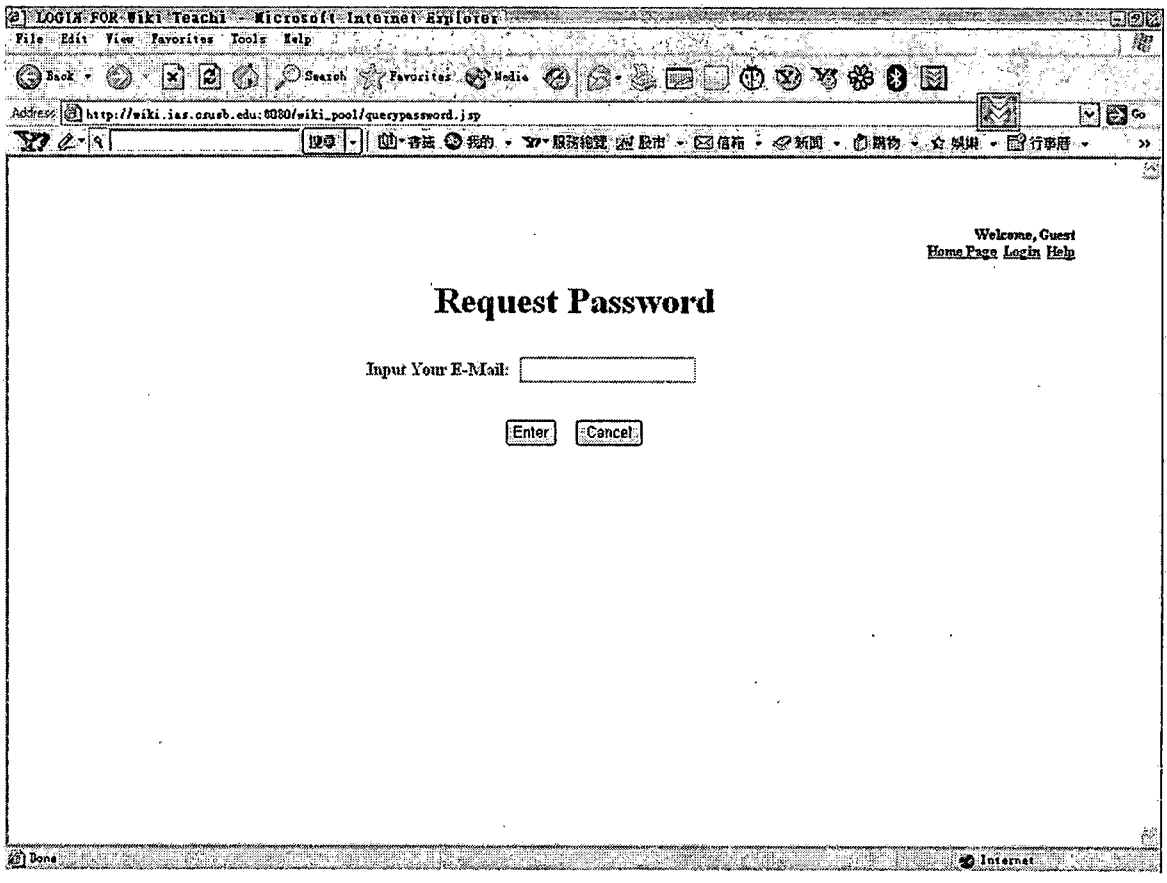

Figure 10. Query Password Page

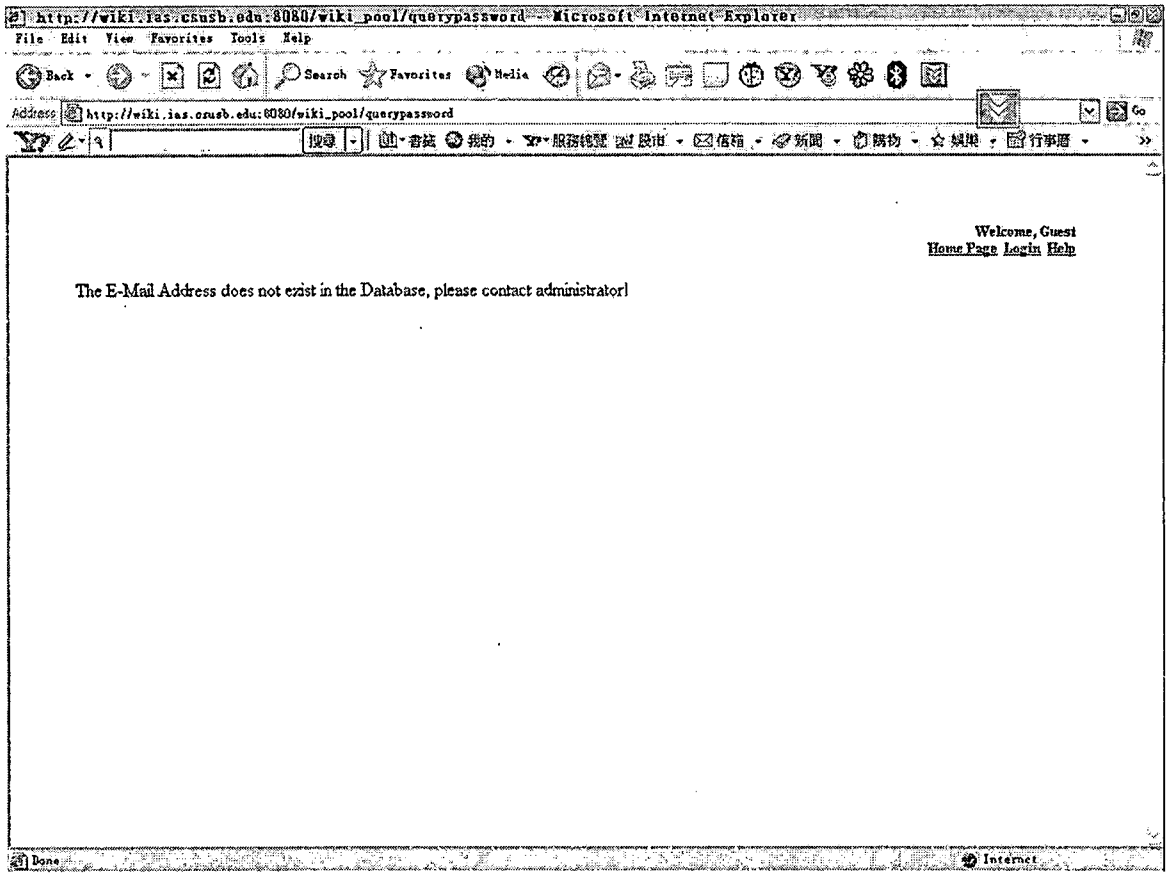

Figure 11. Query Password Error

4.1.3 WAOCC User Account

A WAOCC user account is necessary for anyone who wants to modify or add a wiki page. The user account must be created by the administrator before the first login of a user. The data that administrator provides to create a user account is the default data in the user account. When the user logins, the user should be able to find a link, "My Account", in the menu. The user can view his/her profile by

click the "My Account" link. There are four data items in each account, which are id, name, password and e-mail.

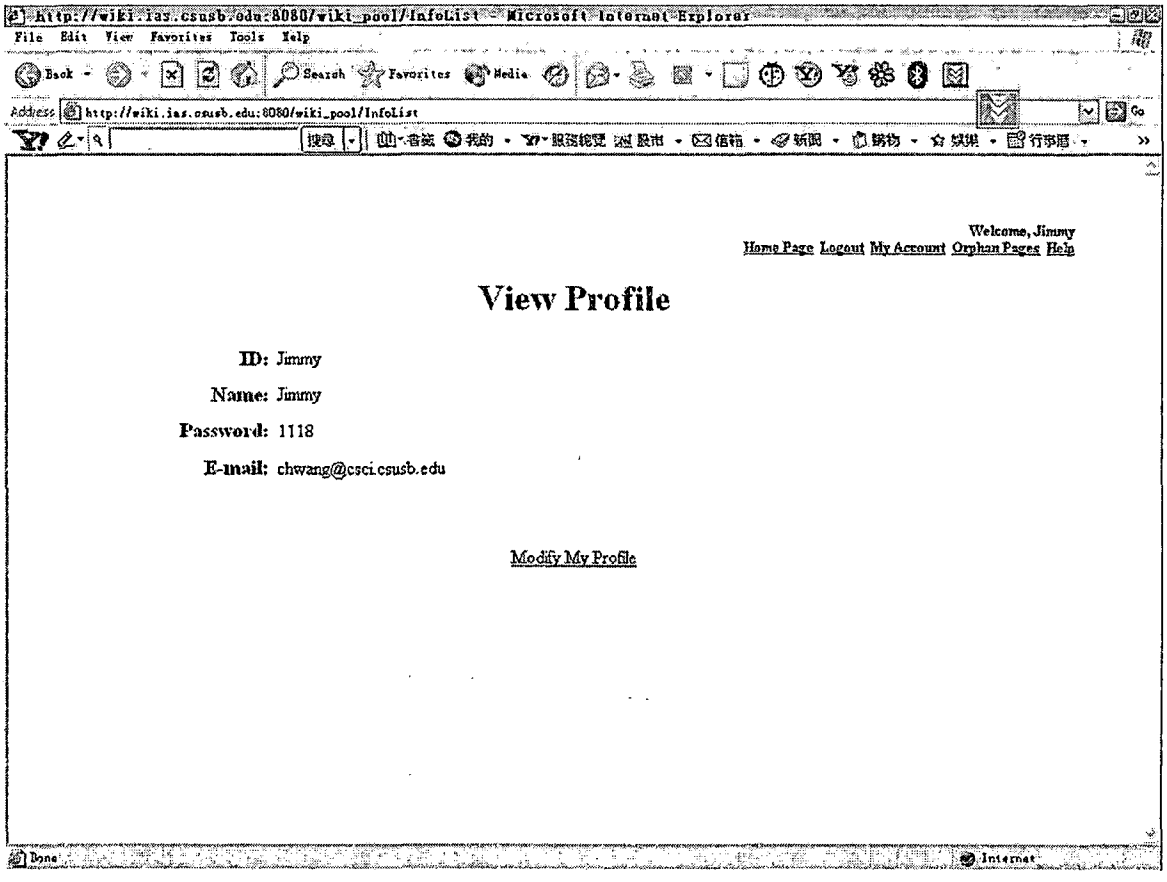

Figure 12. User Profile

Two of the data items in the profile can be changed by the user. One is the password of the user and the other one is the E-mail of the user. In order to change the data, click the "Modify My Profile" link at the bottom of the profile view page to request the modifying profile procedure. In the modify profile page, just fill in the correct data and click the update button; the profile will be updated,

and the new profile data will be shown again in the view profile page. When updating the profile, JavaScript will validate that the password and password confirm fields are the same and that the e-mail field is not empty.

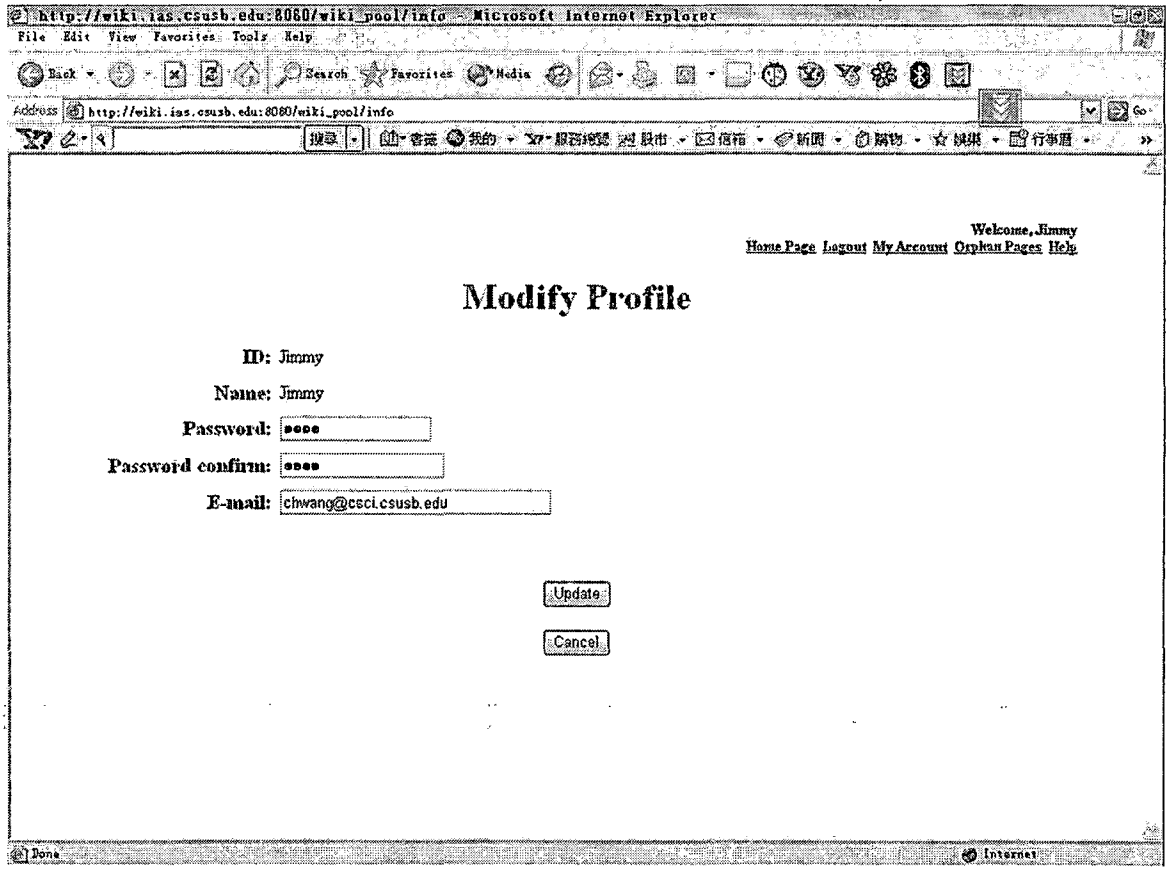

Figure 13. Modify Profile

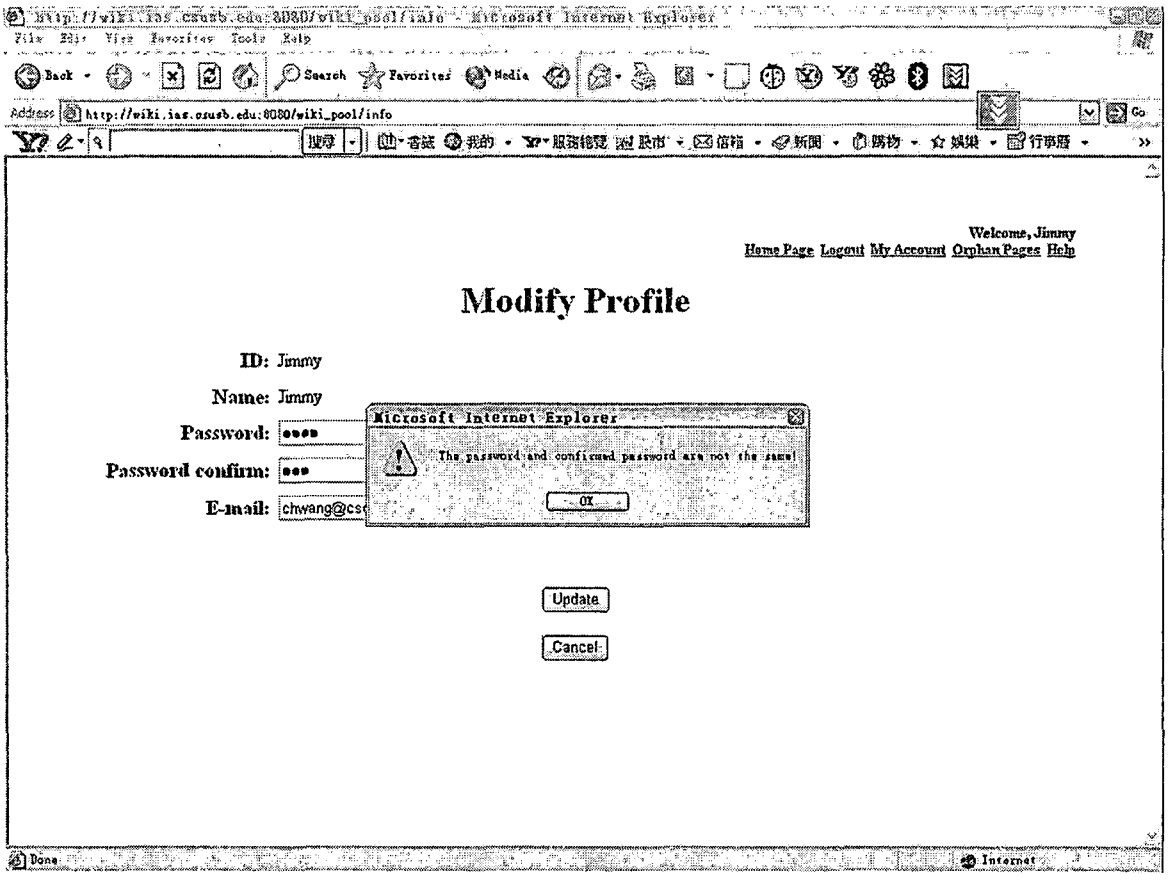

Figure 14. Error Message When Validating Data

4.1.4 Edit Page

This is the most important page that the system has. To edit the page, browse to the page you want to edit and click on the link "edit" in the menu frame or double-click anywhere in the page to access then the edit page. There is one thing that users need to know. To edit a page, the user needs to be logged in and has the privilege to edit the page. If the user who is editing the page is the owner of the page, there will be two more check boxes shown at the bottom of

the text field named "modifiable by users" and "visible by others". These are the page attributes that can be set by the owner of the page to grant or deny to other users the ability to view or modify the page. At the left of the text field, there are several buttons that are used to insert mark up language. The users also can get all the mark up language defined by the system in the help page, which will be talked about later in this chapter.

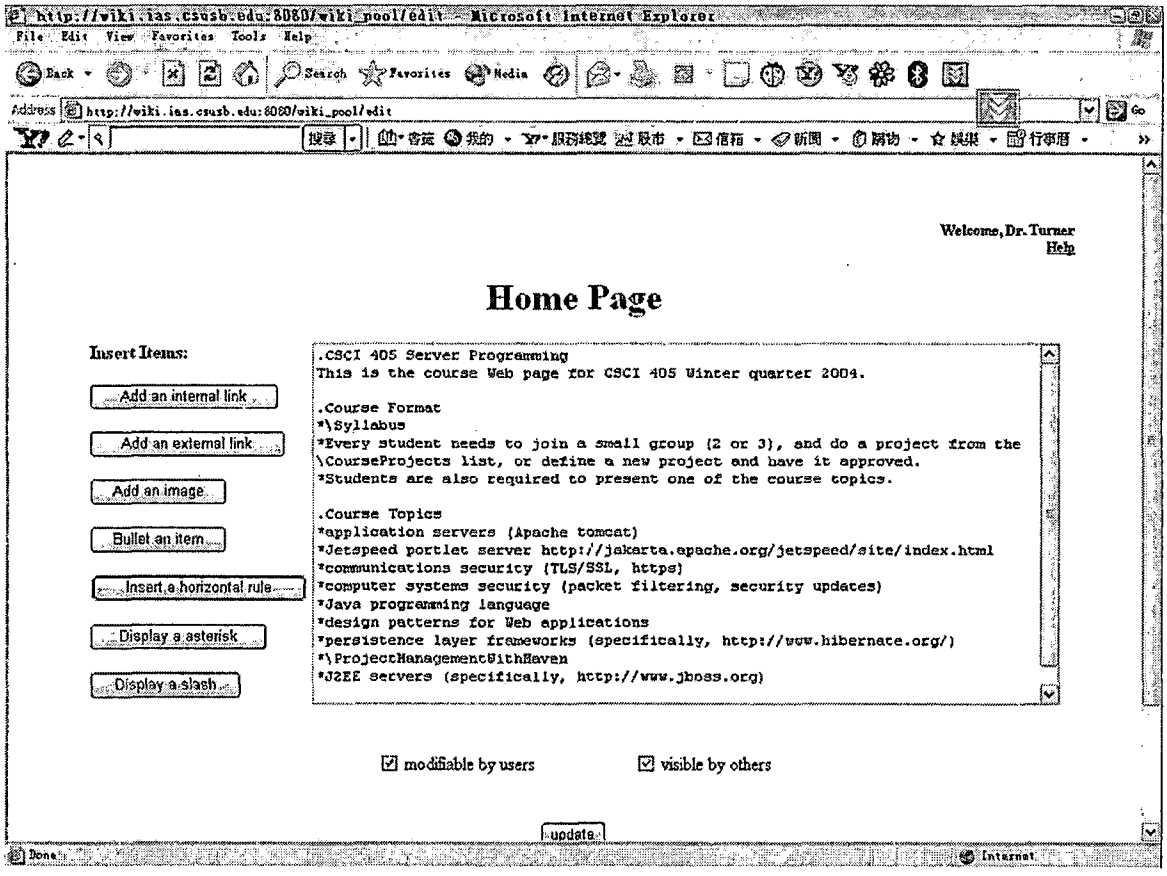

Figure 15. Edit Page

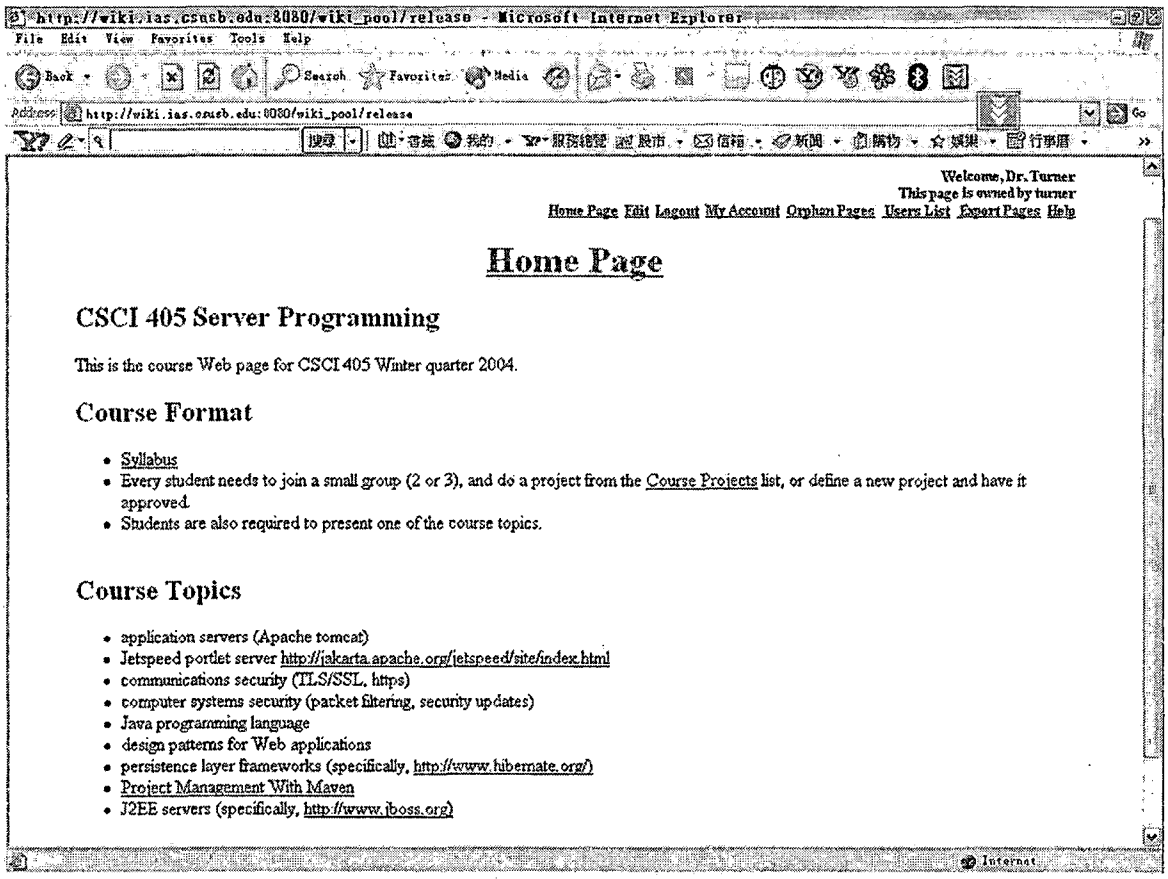

Figure 16. Page Content After the Edit in Figure 15

4.1.5 Orphan Pages

Generally speaking, all the pages in the system should have one or more parent links to it. For the pages other than the home page that have no links to them, we call them the orphan pages. To list the orphan pages, the user clicks the link "Orphan Pages" in the menu. The users can check the content of the page by clicking on the name of the orphan page and deciding if they want to reuse the page by re-link them again or not. In order to re-link the orphan page, add

an internal page link with the name of the orphan page in any non-orphaned pages then the orphan page will be removed from the orphan pages list. To delete the orphan pages from the system, simply click the link "Delete" next to the name of the orphan page title.

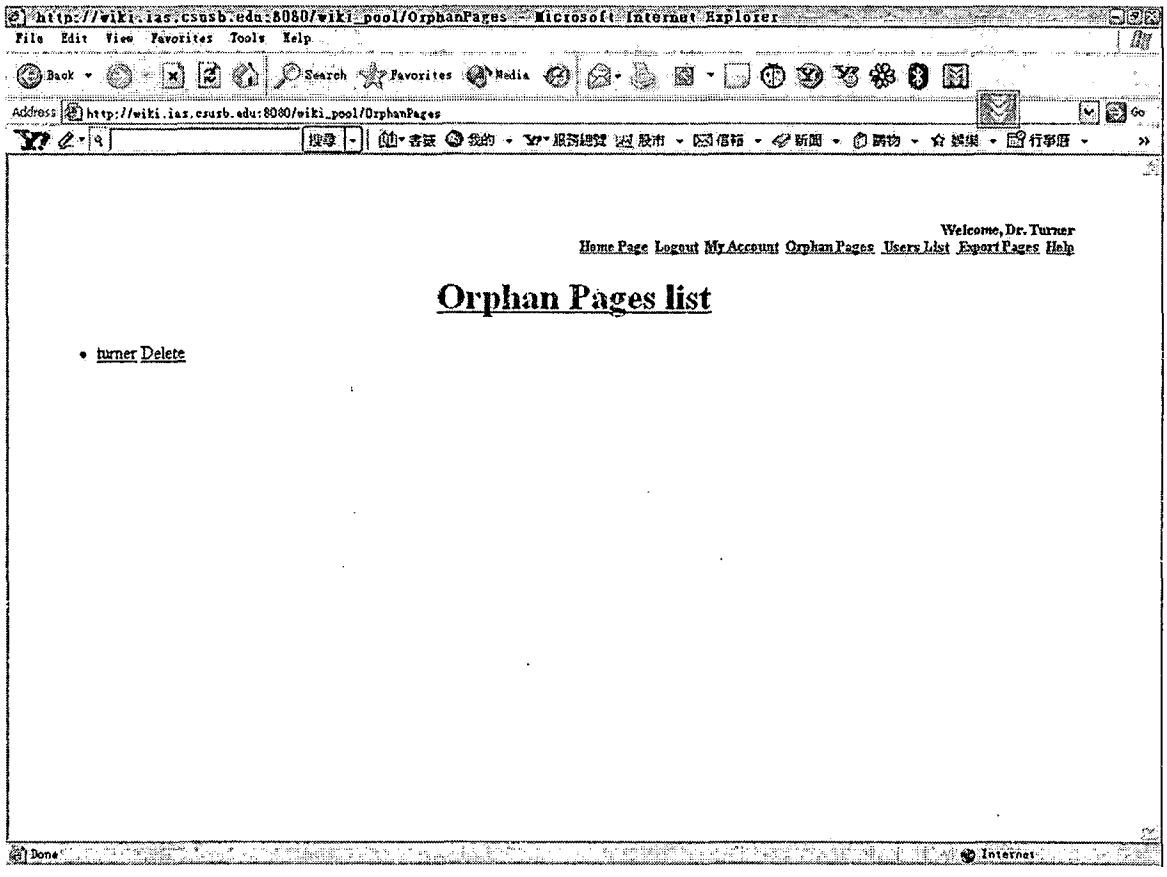

Figure 17. Orphan Pages List Page

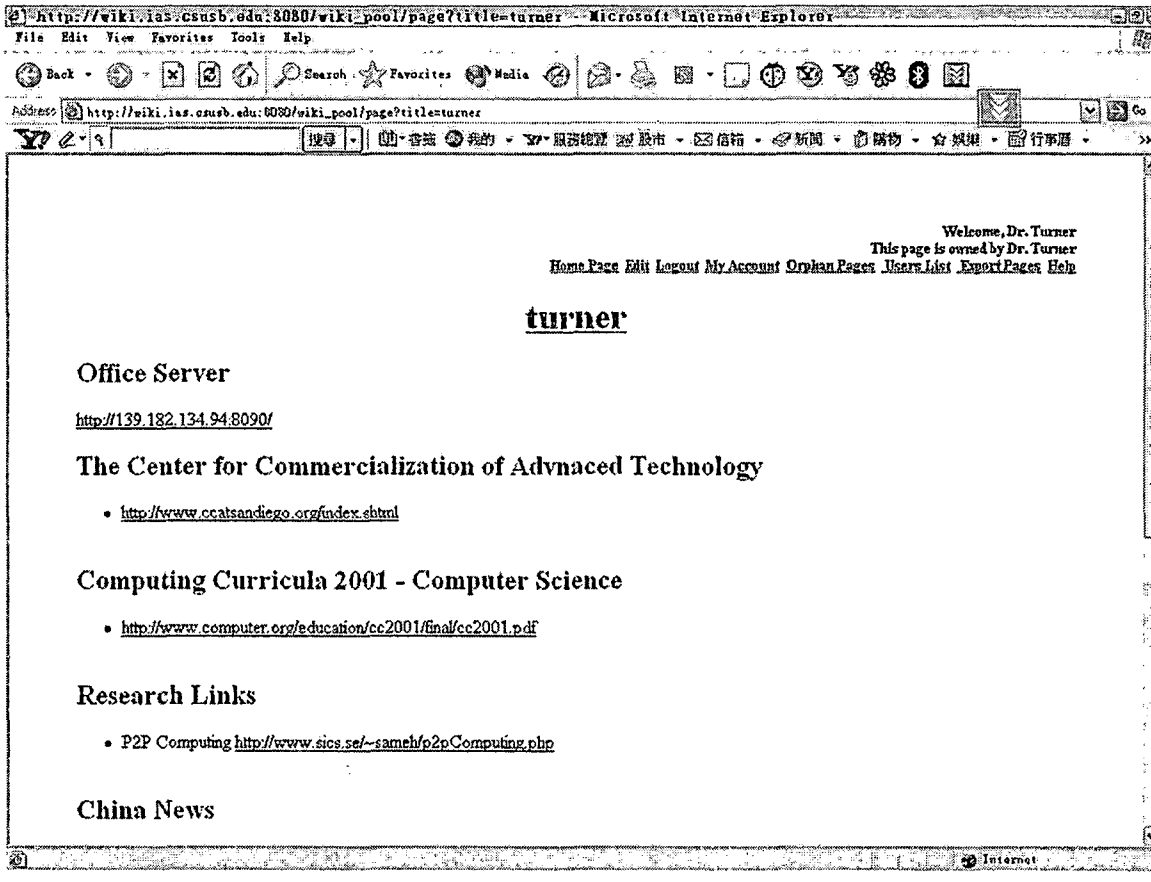

Figure 18. Check the Content of Orphan Page "turner"

4.1.6 Help Page

In order to help users know more about the system, the system also offers the help page for all users including the guest users. The help page tells the users how to use the system and also shows the users how to use the mark up language to edit the wiki page. To view the help page, click the link "Help" in the menu, then the help page will display.

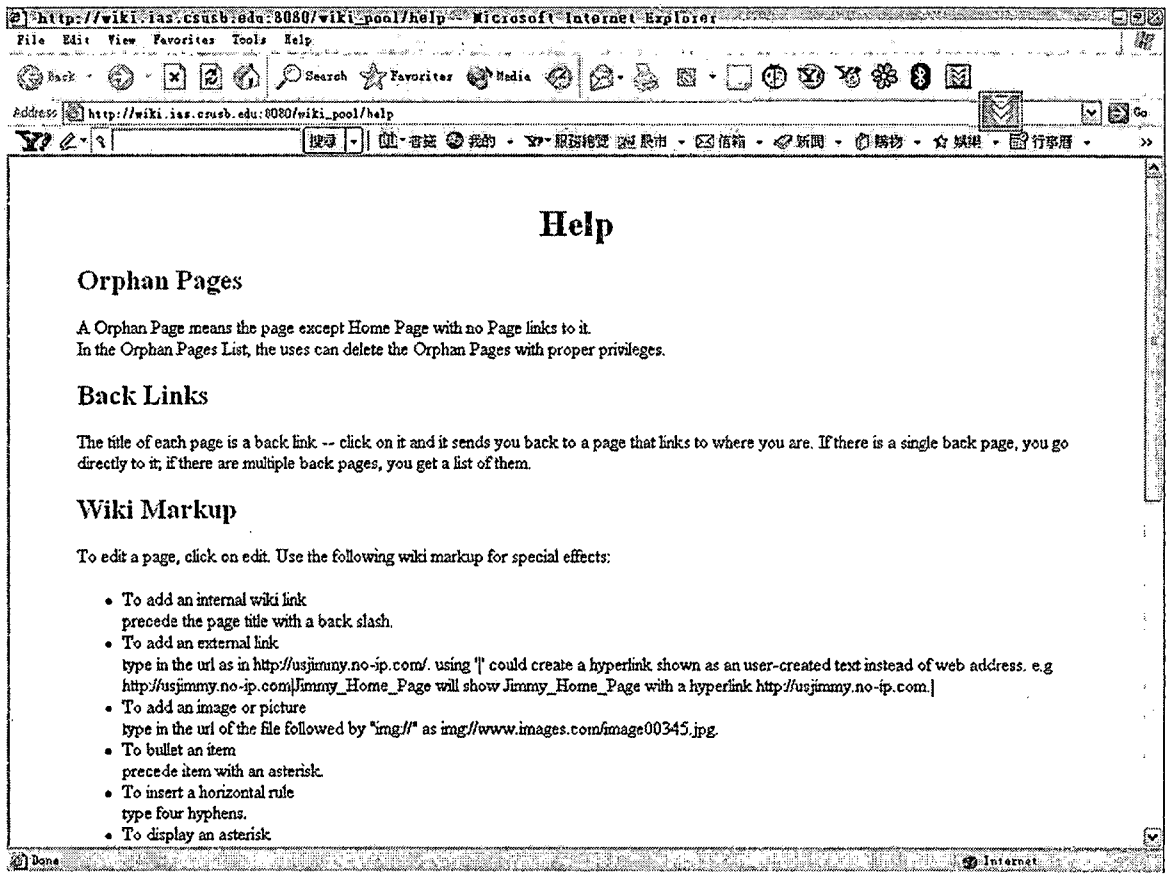

Figure 19. Help Page

4.1.7 Create User Account (Administrator's Function)

There are four itmes of data needed to create a user account: id, name, password and e-mail. When creating a user account, the administrator, who is the only user being able to create a user account, has to provide the system the four items with nonblank and legal data. The JavaScript will validate the data when the Add button is hit and if there is any illegal data the JavaScript will pop up a message to inform user to input correct data. To create a user account,

the user has to login as an administrator first. Then, on the menu frame, there will be a link "Users List" which will list all the users in the system. After clicking the link, the user list page will show up. This page allows the administrator to create or delete a user.

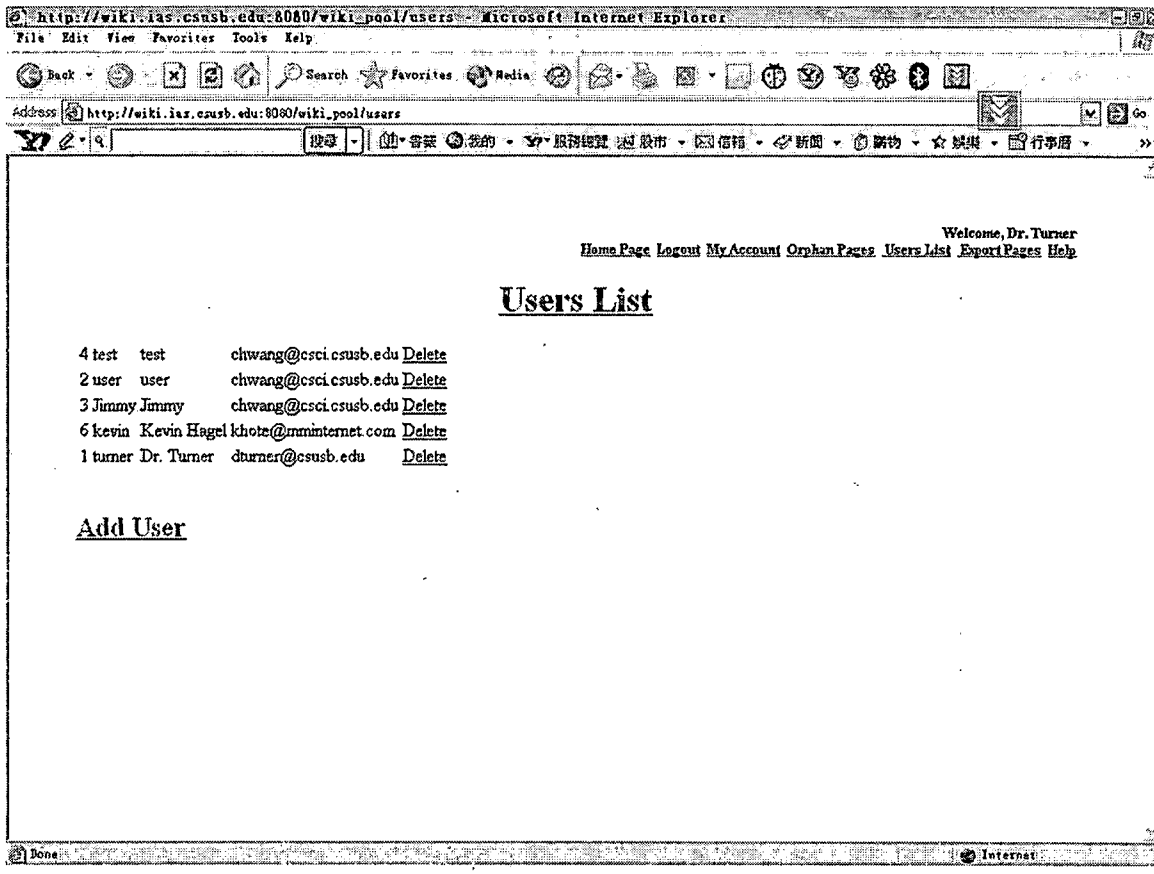

Figure 20. User List Page

At the end of user list, there will be a link "Add User". Click on the link "Add User", the page will forward to add the user page which allows the administrator to create a user account.

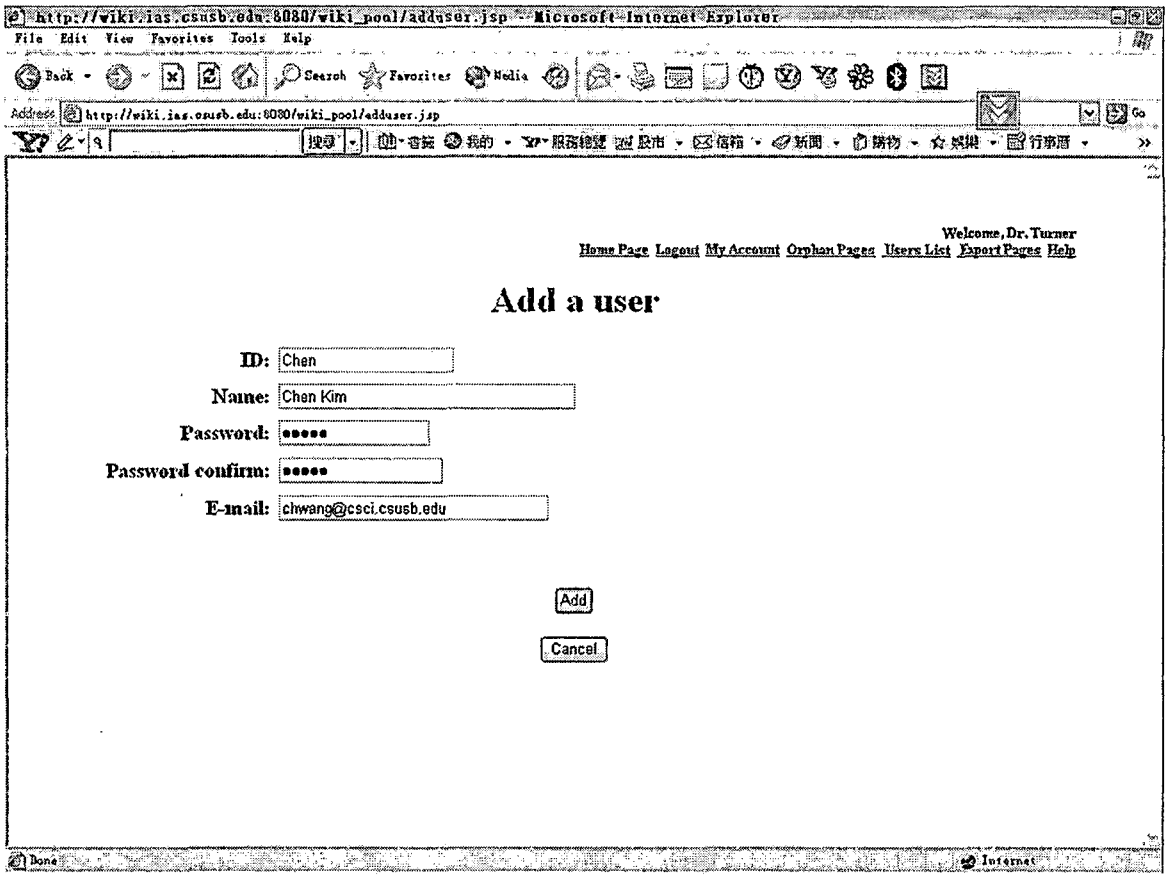

Figure 21. Create User Account Page

Fill in the user id, name, password and E-mail address and click the "add" button; the JavaScript will validate the data. If there is no mistake with the data, the system will create the user. After that, the page will forward to the user list page where you will find the user you just created in the list.

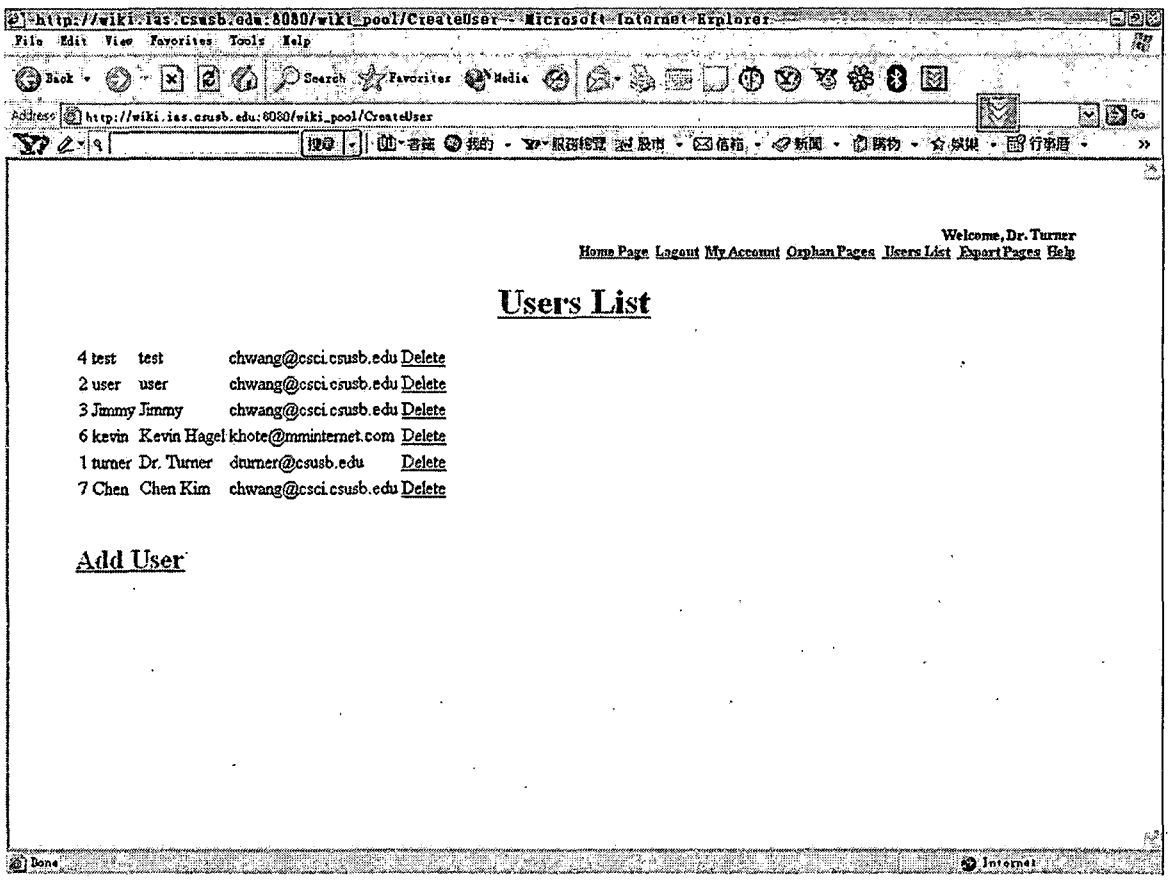

Figure 22. The User Just Created Appears in the List

CHAPTER FIVE

SYSTEM VALIDATION

The system validation test is a kind of test process which can ensure that our program meets the expectation of the user. The purpose of the system validation is to provide a high degree of assurance that a specific process will consistently produce a result which meets predetermined specifications and quality attributes. This can also guarantee the system performance and reliability.

5.1 Unit Test

Unit test is the basic level of testing where individual components are tested to ensure that they operate correctly. These individual components can be object, class, program, etc. The unit testing results of WAOCC are shown in Table 23.

Table 5. Unit Test Results (Forms)

Forms	Tests Performed	Results
Add User Page	• Verify handling valid data input. • Check all the buttons work properly.	Pass
Frame allocate page	• Verify the frame allocation is properly.	Pass

Forms	Tests Performed	Results
Title Page	• Check all the menus shown properly by the user. • Check all the links work as expected.	Pass
Showing Body Page	• Check JavaScript function work properly. • Verify the page get the correct page body from Servlet. • Check user privilege before loading page.	Pass
Edit Page	• Verify the timer work properly. • Check the insert function work properly. • Check all the button work properly.	Pass
Login Page	• Check all the button work properly. • Verify the page can get the error message and work properly by the message. • Verify the user save in session after login	Pass
Logout Page	• Check all the button work properly. • Verify the user remove from session after logout. • Check the page redirect to proper page after logout.	Pass
My Account Page	• Verify showing the correct user account information. • Verify the update link work properly.	Pass

Forms	Tests Performed	Results
Update My Account Page	• Check all the buttons work properly. • Verify the page get the correct user account information. • Verify handling valid data input. • Verify the data updated correctly.	Pass
Orphan Pages list Page	• Verify all the orphan pages are listed as expected. • Check the delete function for each orphan page shown by the user privilege. • Check if the delete orphan page function works properly. • Check if the link of the orphan page directs to correct page.	Pass
User List Page	• Verify the users except administrator can not view this page. • Verify all the user accounts information is correct. • Verify all the user account shown in the list. • Make sure the delete user function works properly. • Make sure the link of "Add User" works properly.	Pass

Forms	Tests Performed	Results
Help Page	• Verify the context in the help page is correct. • Make sure the links work properly.	Pass

Table 6. Unit Test Results (Class: DataBase)

Functions	Tests Performed	Results
getPage	• Check if get the correct page. • Verify it can handle the request of a non-exist page.	Pass
getAllPageTitle	• Verify it gets all the page titles.	Pass
getUser	• Check if get the correct user. • Verify it can handle the request of a non-exist user.	Pass
getUserByNo	• Check it get the correct user. • Verify it can handle the request of a non-exist user number.	Pass
getUserByemail	• Check if get the correct user. • Verify it can handle the request of a non-exist user e-mail address.	Pass

Functions	Tests Performed	Results
deletePage	• Check if the page is deleted properly and correctly. • Make sure it can handle the request of deleting a non-exist page.	Pass
deleteUser	• Check if the user is deleted properly and correctly. • Make sure it can handle the request of deleting a non-exist user.	Pass
updatePage	• Check if the page is updated correctly. • Make sure it can handle the request of updating a non-exist page.	Pass
getFromTitles	• Make sure it returns correct pages for all titles.	Pass
getOrphanPages	• Check if it returns all the orphan pages correctly.	Pass
getUsers	• Verify it returns all the users. • Check each user returned has the correct account information.	Pass
getNewUserNo	• Verify it returns the maximum user no existing in the database.	Pass
ifuserexist	• Verify if the return value is correct.	Pass
UpdateInfo	• Check if it updates the data before check the user privilege. • Verify if the updated value is correct.	Pass

Functions	Tests Performed	Results
CreateUser	• Verify if it gets a proper user no. • Verify the user account is created by the value that user offer.	Pass

Table 7. Unit Test Results (Class: PageTitle)

Forms	Tests Performed	Results
getTitle	• Make sure it returns the correct title.	Pass
isValidTitleCharacter	• Make sure it validates the title by the characters.	Pass
getSpaceTitle	• Verify the title is converted properly.	Pass
E-mail List Import	• Check all the links and buttons work properly. • Make sure selected data stored into database.	Pass

Table 8. Unit Test Results (Class: Page)

Functions	Tests Performed	Results
getTitle	• Make sure it returns the correct title of the page.	Pass
getBody	• Make sure it returns the correct body of the page.	Pass
getismodify	• Make sure it returns the correct ismodify property of the page.	Pass
getispublic	• Make sure it returns the correct ispublic property of the page.	Pass
getownerno	• Make sure it returns the correct owner's no of the page.	Pass
setBody	• Make sure it gets the updated body text from the update page. • Verify the updated body of the page is correct.	Pass
getLinks	• Make sure it gets all the links correctly.	Pass
isValidUrlCharacter	• Make sure it validates the Url by the characters.	pass
getFormattedBody	• Verify if it gets the correct body. • Make sure the body is transfer to HTML format by the mark up language rule.	Pass
readString	• Make sure it reads the forward string correctly.	Pass
isValidTitleCharacter	• Make sure it validates the title by the characters.	Pass
getSpacedTitle	• Make sure the title returned is proper spaced	Pass

Functions	Tests Performed	Results
getFromTitles	• Make sure the pages retuned is correct.	Pass

Table 9. Unit Test Results (Class: User)

Forms	Tests Performed	Results
getno	• Make sure the returned number is correct.	Pass
Getid	• Make sure the returned id is correct.	Pass
Getname	• Make sure the returned name is the correct user's name.	Pass
Getpassword	• Make sure the returned password is the user's password.	Pass
Getemail	• Make sure the email address returned is the correct user's e-mail address.	Pass

Table 10. Unit Test Results (Class: ControllerServlet)

Forms	Tests Performed	Results
doGet	• Verify all the attribute is gotten properly and correctly. • Make sure all the container is handled. • Verify the session of the login user is handle properly.	Pass
processCreateUser	• Make sure it checks if the user going to create is exist. • Make sure it calls the correct function in DataBase. (CreateUser()) • Make sure it redirect to the correct page after user is created.	Pass
processUpdateInfo	• Make sure it checks the users privilege before up date the information. • Verify the information is updated correctly.	Pass
processOP	• Make sure it gets all the orphan pages.	Pass
processUsers	• Make sure it checks the user privilege before query the users from database. • Make sure all the users are returned.	Pass
processQueryPassword	• Make sure it checks the user privilege before query the user's password. • Verify if the password. Is correct	Pass

Forms	Tests Performed	Results
processInfo	• Make sure it returns the correct user's information. • Make sure it redirects to the proper page.	Pass
processInfoList	• Make sure it returns the correct user's information. • Make sure it redirects to the proper page.	Pass
processAu	• Verify it can authorize the user id and password properly.	Pass
page_pool	• Make sure the page which is in the pool can be gotten without reading from database.	Pass
page_pool_update	• Check the page which is read most recently is placed at the top of the pool.	Pass
processPage	• Make sure the correct page is returned. • Check if the user's privilege is check before returning the page.	Pass
processEdit	• Check if the user's privilege is checked before redirect to the edit page. • Verify the page body is returned to the edit page properly and correctly.	Pass
processUpdate	• Check if the user's privilege is check before update the page data. • Verify the page data is updated correctly.	Pass
processLogout	• Verify the data in login session is cleared.	Pass

Forms	Tests Performed	Results
processDelete	• Check if the user has the privilege to delete the page. • Verify the page is removed from database and page pool.	Pass
processBack	• Verify it gets the correct back links. • Check if the page is redirected to the proper page.	Pass

5.2 Subsystem Testing

Subsystem testing is the next step up in the testing process where all related units from a subsystem do a certain task. Thus, the subsystem test process is useful for detecting interface errors and specific functions. Table 24 show subsystem test results in detail.

Table 11. Subsystem Test Results

Subsystem	Tests Performed	Results
Authorize subsystem	• Test if it can get the error message. • Make sure the result of authorizing user is correct. • Verify the login user information is store in session properly. • Check if the saving user login information function stores the user information in cookies for future login use. • Check if the login page can get the saved user login information saved before from cookies. • Verify the login page redirect to the correct browsing or editing page after the user logins in.	Pass
Accounts management subsystem	• Make sure all the existing users are list in the user list. • Check if the subsystem can detect the error of creating of the user that exists in the subsystem. • Check if the user can update his/her own account properly. • Verify the created user information is the same as the information provided. • Verify the subsystem can delete a user account properly.	Pass

Subsystem	Tests Performed	Results
Browsing subsystem	• Check if the subsystem checks for user privilege before showing pages. • Verify the page is showing properly after the user click on the page link.	Pass
Editing subsystem	• Make sure the subsystem checks the user privilege before forwarding to edit page. • Verify the subsystem check the user privilege before update the page information. • Verify if the subsystem shows the page properties is the users are the owner or the administrator.	Pass
Orphan pages subsystem	• Make sure all the orphan pages are shown on the list. • Verify the owner the page or the administrator can delete the specific orphan page.	Pass

5.3 System Testing

System testing is the testing process that uses real data, which the system is intended to manipulate, to test the system. First all subsystem will be integrated into one system. Then test the system by using a variety of data to see the overall result.

System testing of WAOCC system begins with the following steps:

Table 12. System Test Results

System Testing	Results
1. Install WAOCC system into server.	Pass
2. Start up all servers such as Tomcat server, PostgreSQL database server.	Pass
3. Running testing by using real data on all forms and reports.	Pass

CHAPTER SIX

MAINTENANCE MANUAL

It is very important to have a maintenance manual with a system no matter how small the system is. The maintenance manual records any information that can be used to setup the system or backup the system. In order to make sure the system works smoothly and meets the expectation of the users, it is very important to follow the maintenance manual step by step carefully. In WAOCC, there are 3 major issues: Software Installation, Variable Installation, and WAOCC Installation.

6.1 Software Installation

In WAOCC, it requires RedHat, PostgreSQL, JSDK, Ant, TOMCAT, and JDBC to run the programs. Following will detail the installation of those 6 softwares.

6.1.1 RedHat Installation

RedHat is a linux base operating system which is offerd freely and be downloaded from internet. The reason we choose RedHat is it offers better performance than a Microsoft operating system. Following are the steps to install RedHat onto your machine.

1. Download a latest version of the RedHat operating systems from

http://ftp.redhat.com/pub/redhat/linux/9/en/iso/i386/ and burn the iso files into CDs.

2. Install the operating system by inserting CD 1 into the CD-ROM and start up the machine which is going to install the operating system.
3. The machine will startup via CD-ROM and start to install RedHat.
4. Follow the install wizard and sets up the required information such as network setting and the hardware environment.
5. After all the necessary files are copied into the computer and install it, the machine will restart and Redhat is installed.

6.1.2 PostgreSQL Installation

PostgreSQL is the database system we use in WAOCC; it's free, and is included in RedHat by default. The reason that we choose PostgreSQL as WAOCC's database system is because it also provides a JDBC driver to easily connect from a JAVA program, thus it's a good choice for designing this project.

To install PostgreSQL, follow the following steps:

1. Because PostgreSQL may install on to RedHat when the operating system is installed, the first thing we have to do is to check if the

PostgreSQL is already in the operating system. Using the command to check if PostgreSQL exist in the operating system:

```
rpm -q postgresql
```

If PostgreSQL is not installd in the operating system, then use rpm to install it.

2. In order to create a database user, "wiki", and database, "wiki" for WAOCC use, have the following commands executed.

```
su postgres
initdb -D /var/bin/pgsql/data
createuser wiki
createdb wiki
```

where at the first command, the postgres is the default user for PostgresSQL. Starting the database by using the command "initdb" with the directory "/var/bin/pgsql/data/" which is the default database directory will start up the database system. Login postgres as the supervisor and create a database user, "wiki", and the database "wiki".

3. There are still some steps needed to setup the environment values.

In the user's environment setup file /etc/profile.d/*.sh, add the following line:

export PGDATA=/var/lib/pgsql/data

Open the file /var/lib/pgsql/data/postgresql.conf and uncomment the line:

tcpip_socket = true

In order to have the database system starup at the system start, have the command executed:

/sbin/chkconfig --level 3 postgresql on

And, the last step is to startup the database system immediately now without restart the system:

/sbin/service postgresql start

After having the steps above executed, the database system is ready to go and now we have to install JAVA platform, JAVA 2 Platform, Standard Edition (J2SE).

6.1.3 JAVA 2 Platform, Standard Edition (J2SE)

J2SE is the compiler program for JSP and JAVA Servlet programs and it's required in TOMCAT JAVA Container. Fist of all, we go to http://java.sun.com/j2se/1.4.1/download.html to download SDK Linux (all languages, including English) to the directory /usr/java, then execute the following commands:

chmod +x j2sdk-1_4_2_01-linux-i586-rpm.bin

./j2sdk-1_4_2_01-linux-i586-rpm.bin

rpm -ivh j2sdk-1_4_2_01-linux-i586.rpm

And, set the environment variables in the file /etc/profile.d/*.sh:

```
JAVA_HOME=/usr/java/j2sdk1.4.2_01
PATH=${PATH}:${JAVA_HOME}/bin
Export JAVA_HOME
```

6.1.4 Tomcat

TOMCAT is one of the apache jakarta projects, which is a web container to process JSP and JAVA Servlet programs, and to serve static web pages. First of all, we go to the tomcat's official download ftp server at http://ftp.epix.net/apache/jakarta/tomcat-5/v5.0.12-beta/bin/ to download the file of tomcat server for linux jakarta-tomcat-5.0.12.tar.gz to /usr/java/ and extract it to the hard drive.

tar -xzvf jarkata-tomcat-5.0.12.tar.gz

Also, we modify the file /usr/java/jarkata-tomcat-5.0.12/conf/server.xml by add the following setting in the file:

```
<Context
   path="/wiki"
   docBase="/pub/wiki"
   debug="0"
```

```
    swallowOutput="true" >
  <Logger
    className="org.apache.catalina.logger.FileLogger"
    prefix=""
    suffix=".log"
    directory="/pub/wiki/logs"
    timestamp="true" />
      <Parameter name="contextPath"
                                value="/wiki" />
      <Parameter name="homePage" value="HomePage" />
      <Resource name="jdbc/postgres" auth="Container"
                      type="javax.sql.DataSource" />
      <ResourceParams name="jdbc/postgres">
        <parameter>
          <name>factory</name>
          <value>org.apache.commons.dbcp.BasicDataSo
                                urceFactory</value>
        </parameter>
        <parameter>
          <name>driverClassName</name>
          <value>org.postgresql.Driver</value>
        </parameter>
        <parameter>
          <name>url</name>
```

```
            <value>jdbc:postgresql://127.0.0.1:5432/wi
                                    ki</value>
        </parameter>
        <parameter>
            <name>username</name>
            <value>Jimmy</value>
        </parameter>
        <parameter>
            <name>password</name>
            <value></value>
        </parameter>
        <parameter>
            <name>maxActive</name>
            <value>10</value>
        </parameter>
        <parameter>
            <name>maxIdle</name>
            <value>2</value>
        </parameter>
        <parameter>
            <name>maxWait</name>
            <value>-1</value>
        </parameter>
    </ResourceParams>
```

```
</Context>
```

And, set the environment variable by adding the following lines in the file /etc/profile.d/*.sh

```
CATALINA_HOME=/usr/java/jarkata-tomcat-4.1.27
PATH=${PATH}:${JAVA_HOME}/bin:${CATALINA_HOME}/bin
export CATALINA_HOME
```

Add the following lines in the file /etc/rc.local to have the tomcat run when the system boots:

```
Export JAVA_HOME=/usr/java/j2sdk1.4.2_01
export CATALINA_HOME=/usr/java/jarkata-tomcat-4.1.27
${CATALINA_HOME}/bin/startup.sh
```

6.1.5 JAVA Database Connectivity (JDBC)

The API used to execute SQL statement is different for each database engine. Java programmers, however, are lucky and are freed from such database portability issues. They have a single API, the Java Database Connectivity API (JDBC), that's portable between database engines. The JDBC library provides an interface for executing SQL statements. It provides the basic functionality for data access. A number of drivers are available for PostgreSQL, and information about this can be obtained at the PostgreSQL homepage at http://jdbc.postgresql.org/download/. Download

pg73jdbc3.jar and copy the file to /usr/java/jakarta-tomcat-4.1.18/common/lib/.

6.2 Variables Modification

In WAOCC, we have to change some environment variables in the linux system and server.xml in Tomcat server configuration directory.

6.2.1 System Variables

1. Open the file "server.xml" in the directory "/usr/java/jakarta-tomcat-4.1.18/conf" via "vi" or any other editor.
2. Scroll down until you see the context area we added in at chapter 7.4.1.
3. The variable "path" in Context indicates the context path of the web application. The default value would be "/wiki".
4. The variable "docBase" in Context is the files directory for the web application. The default value would be "/pub/wiki".
5. The variable "variable" in Logger is the absolute or relative pathname of a directory in which log files created by this logger will be placed. The default value would be "/pub/wiki/logs".

6. Now, lets look down at the parameter setting for WAOCC.

7. The parameter "contextPath" indicate the context path for WAOCC which would be the same as the value of path.

8. The parameter "homepage" sets the name of the home page of WAOCC. The default value would be "HomePage".

9. The parameter "username" is the user name who can access the database system. Usually, this value would be the administrator of WAOCC.

10. The parameter "password" is the password corresponding to the user name at 9. If there is no special setting in database system, leave the value to be empty.

6.3 Wiki-Style Administration of Online Course Content Installation/Migration

1. All the JSP programs and HTML programs are stored in

   \pub\wiki

2. All the classes are stored in

   \pub\wiki\WEB-INF\classes

3. Place the web.xml for WAOCC in

   \pub\wiki

6.4 Backup and Restore

Backup is a very important action needed for any system to prevent losing data. No one can say a system works very well and will never have a problem. So, now lets talk about the backup process for WOACC. There are two steps to back up WOACC. One is to backup the system files. The other step is to backup the database which is used by WAOCC.

6.4.1 System Backup

All the WAOCC system files are located in the directory "/pub/wiki" and all its subdirectory. Thus, in order to backup the system files, all we need to do is to backup the files in the directory. The method here I suggest is to compress the directory of "/pub/wiki" including its subdirectory to compress files for future use by the compress program "tar". Using the following command to backup the system files:

```
tar -cf WAOCC.tar /pub/wiki
```

6.4.2 Database Backup

To backup the database system, we use pg_dump command to backup the database used by WAOCC. The following command is used to backup the database:

```
pg_dump wiki | gzip > WAOCC.zip
```

After executing the backup command above, the file WAOCC.zip would be the backup file of the database.

6.4.3 System Restore

To restore the system file, simply extract the backup file by using the following command:

```
tar -xzvf WAOCC.tar /
```

By the command above, all the WAOCC system files will restore into the directory /pub/wiki and complete the restore system process.

6.4.4 Database Restore

To restore the database needed for the WAOCC, go to the directory where your databse backup file is in, and execute the following commands:

```
createdb wiki
gunzip -c WAOCC.zip | psql WAOCC
```

After the commands are executed, the database is restored to the database system. Then, restart tomcat, the WAOCC will be completely restored.

CHAPTER SEVEN

CONCLUSION AND FUTURE DIRECTIONS

7.1 Conclusion

WAOCC provides a very good communication environment for the instructor and the students. For the instructor, WAOCC offer a very good environment to post the announcement and the syllabus for a class. For the students, WAOCC provides a good environment that all the students can share any ideas or problems they have on WAOCC. The most important of all, the easy to use and learn markup language offered by WAOCC helps the users to edit and create html pages to share information.

WAOCC offers a page security function. Only the user who has the privilege to modify the page can edit the page. By this function, all the pages can prevent being edited by malicious editing. Thus, WAOCC is definitely a good system which will help the instructor and the students have a better communication environment.

7.2 Future Directions

WAOCC is a system offered to be used for any classes. The system is not created to be used for any specific class, so any class can use the system as its online class board. In the future, if there is any html function needed for the

system, the changes needed for the system is in the class called Page. Modify the parsing functions, and the new function can be added easily. If the change to the page style is needed, just edit the jsp file, then the page style will be changed simply.

Because the system is widely accepted, the system can be used not just in the education area. It can be used as a general purpose communication board. Simply speaking, this is a collaborative tool that can be used in many contexts.

APPENDIX A

CONTROLLERSERVLET CLASS PRINTOUT

```
package wiki;

import java.util.*;
import java.io.*;
import java.util.Vector;
import javax.servlet.*;
import javax.servlet.jsp.*;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.sql.*;

public class ControllerServlet extends HttpServlet
{
   protected static String homePage;
   private static String contextPath;
   private ServletContext context;

   /**
    *  This method is called by the init(ServletConfig), which was called
    *  by the servlet container just after it constructed the servlet
    *  from its default constructor.
    */
   public void init()
   throws ServletException
   {
      HttpSession session;
      context = getServletContext();
      homePage = context.getInitParameter("homePage");
      contextPath = context.getInitParameter("contextPath");
      Page.contextPath = contextPath;
   }

   /**
    *  This method is called when the servlet instance is going to be
    *  destroyed -- this is the place where resources are released,
    *  such as database connections, or other open socket connections.
    */
   public void destroy()
   {
   }

   protected void doPost(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException
   {
      doGet(req, resp);
   }

   protected void doGet(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException
   {
      String action = req.getServletPath();
      String title = req.getParameter("title");
      String body = req.getParameter("body");
      String user_info_no = req.getParameter("user_info_no");
      String user_no;
      HttpSession session;
      session = req.getSession();
      if (session.getAttribute("user_no") == null)
      {
         user_no = "0";
         session.setAttribute("user_no", "0");
         session.setAttribute("title", "");
      }
      else
         user_no = session.getAttribute("user_no").toString();
      if (title == null && !session.isNew()) title =
session.getAttribute("title").toString();
```

```
    RequestDispatcher dispatcher = null;
    try {
       if (action.equalsIgnoreCase("/page")) {
          dispatcher = processPage(req, resp, title, user_no);
       }
       else if (action.equalsIgnoreCase("/")) {
          dispatcher = processPage(req, resp, homePage, user_no);
       }
       else if (action.equalsIgnoreCase("/edit")) {
          dispatcher = processEdit(req, title);
       }
       else if (action.equalsIgnoreCase("/update")) {
          dispatcher = processUpdate(req, title, body, user_no);
       }
       else if (action.equalsIgnoreCase("/back")) {
          dispatcher = processBack(req, title);
       }
       else if (action.equalsIgnoreCase("/OrphanPages")) {
          dispatcher = processOP(req);
       }
       else if (action.equalsIgnoreCase("/CreateUser")) {
          dispatcher = processCreateUser(req);
       }
       else if (action.equalsIgnoreCase("/login")) {
          dispatcher = processAu(req, resp, action, title);
       }
       else if (action.equalsIgnoreCase("/logout")) {
          dispatcher = processLogout(req);
       }
       else if (action.equalsIgnoreCase("/delete")) {
          dispatcher = processDelete(req, title);
       }
       else if (action.equalsIgnoreCase("/users")) {
          dispatcher = processUsers(req);
       }
       else if (action.equalsIgnoreCase("/info")) {
          dispatcher = processInfo(req, user_no, user_info_no);
       }
       else if (action.equalsIgnoreCase("/InfoList")) {
          dispatcher = processInfoList(req, user_no, user_info_no);
       }
       else if (action.equalsIgnoreCase("/UpdateInfo")) {
          dispatcher = processUpdateInfo(req, user_no);
       }
       else if (action.equalsIgnoreCase("/querypassword")) {
          dispatcher = processQueryPassword(req);
       }
       else if (action.equalsIgnoreCase("/help")) {
          dispatcher = context.getRequestDispatcher("/help.jsp");
       }
       if (action.equalsIgnoreCase("/release")) {
          dispatcher = processRelease(req, resp, title, user_no);
       }
    } catch(SQLException e) {
       throw new ServletException(e);
    }

    if (dispatcher == null) {
       resp.sendRedirect(contextPath + "/");
    }
    else {
       dispatcher.forward(req, resp);
    }
}

private RequestDispatcher processCreateUser(HttpServletRequest req)
throws SQLException
{
    HttpSession session;
    session = req.getSession();
    String user_no = session.getAttribute("user_no").toString();
```

```
    if (user_no.toString().compareTo("1") != 0)
       return context.getRequestDispatcher("/help");

    String id = req.getParameter("id");
    String name = req.getParameter("name");
    String password = req.getParameter("password");
    String email = req.getParameter("email");

    int new_no=DataBase.getNewUserNo();
    User user = new User(new_no, id, name, password, email);
    // return value:
    //  0: id is used
    //  1: create successfully
    if(DataBase.CreateUser(user) == 1)
       return context.getRequestDispatcher("/users");
    else
       return context.getRequestDispatcher("/CreareUserError.jsp");
}

private RequestDispatcher processUpdateInfo(HttpServletRequest req, String user_no)
throws SQLException
{
    HttpSession session;
    if (user_no.compareTo("user_info_no") == 0)
       return context.getRequestDispatcher("/help");

    String password = req.getParameter("password");
    String email = req.getParameter("email");

    // return value:
    //  0: update fail
    //  1: update successfully
    if(DataBase.UpdateInfo(user_no, password, email) == 1)
    {
       User user = DataBase.getUserByNo(user_no);
       if (user == null) return null;
       req.setAttribute("User", user);
       return context.getRequestDispatcher("/InfoList.jsp");
    }
     else
    {
       User user = DataBase.getUserByNo(user_no);
       if (user == null) return null;
       req.setAttribute("User", user);
       return context.getRequestDispatcher("/InfoList.jsp");
    }
}

private RequestDispatcher processOP(HttpServletRequest req)
throws SQLException
{
    HttpSession session;
    session = req.getSession();
    String user_no = session.getAttribute("user_no").toString();

    Vector OrphanPages = DataBase.getOrphanPages();
    req.setAttribute("OP", OrphanPages);
    return context.getRequestDispatcher("/OrphanPages.jsp");
}

private RequestDispatcher processUsers(HttpServletRequest req)
throws SQLException
{
    HttpSession session;
    session = req.getSession();
    String user_no = session.getAttribute("user_no").toString();
    if (user_no.toString().compareTo("1") != 0)
       return context.getRequestDispatcher("/help");

    Vector Users = DataBase.getUsers();
    req.setAttribute("Users", Users);
```

```
      return context.getRequestDispatcher("/Users.jsp");
   }

   private RequestDispatcher processQueryPassword(HttpServletRequest req)
   throws SQLException
   {
      String email = req.getParameter("email");
      User user = DataBase.getUserByemail(email);

      if (user != null)
      {
         req.setAttribute("password", user.getpassword());
         req.setAttribute("email", email);  // the email address exist in the database
         req.setAttribute("id", user.getid());
      }
      else
      {
         req.setAttribute("email", null);   // the email address not exist
         req.setAttribute("password", null);
         req.setAttribute("id", null);
      }

      return context.getRequestDispatcher("/sendpassword.jsp");
   }

   private RequestDispatcher processInfo(HttpServletRequest req, String user_no, String
user_info_no)
   throws SQLException
   {
      if (user_info_no == null) user_info_no = user_no;
      if (Integer.valueOf(user_info_no).intValue() < 1) return null;
      if (Integer.valueOf(user_no).intValue() == 1 ||
Integer.valueOf(user_info_no).intValue() == Integer.valueOf(user_no).intValue())
      {
         User user = DataBase.getUserByNo(user_info_no);
         if (user == null) return null;
         req.setAttribute("User", user);
         return context.getRequestDispatcher("/modifyuserinfo.jsp");
      }
      else
         return null;
   }

   private RequestDispatcher processInfoList(HttpServletRequest req, String user_no,
String user_info_no)
   throws SQLException
   {
        User user = DataBase.getUserByNo(user_no);
        if (user == null) return null;
        req.setAttribute("User", user);
        return context.getRequestDispatcher("/InfoList.jsp");
   }

   private RequestDispatcher processAu(HttpServletRequest req, HttpServletResponse resp,
String action, String title)
   throws SQLException, IOException
   {
      String id = req.getParameter("id");
      String password = req.getParameter("password");
      String ErrMsg = req.getParameter("ErrMsg");

      User user = DataBase.getUser(id);
      if (user == null)
      {
         req.setAttribute("ErrMsg", "4");
         return context.getRequestDispatcher("/Login.jsp");
      }
      else if (password.compareTo(user.getpassword()) != 0)
      {
         req.setAttribute("ErrMsg", "5");
         return context.getRequestDispatcher("/Login.jsp");
```

```
      }
      else
      {
         // process cookies: save or unsave the user id and password

         Cookie[] cookies = req.getCookies();
         Cookie wiki_user = new Cookie("wiki_user", id);
         Cookie wiki_pwd = new Cookie("wiki_pwd", password);
         wiki_user.setPath("/");
         wiki_pwd.setPath("/");

         if (req.getParameter("saved") != null)
         {
             wiki_user.setMaxAge(60 * 60 * 24 * 30); // seconds in a month
             wiki_pwd.setMaxAge(60 * 60 * 24 * 30); // seconds in a month
         }
         else
         {
            wiki_user.setMaxAge(0); // expire immediately
            wiki_pwd.setMaxAge(0);
         }
         resp.addCookie(wiki_user);
         resp.addCookie(wiki_pwd);

         req.setAttribute("user_no", Integer.toString(user.getno()));
         if (title == null) title = homePage;
         req.setAttribute("title", title);
         req.setAttribute("user_name", user.getname());
         req.setAttribute("ErrMsg", ErrMsg);
         return context.getRequestDispatcher("/AuCookies.jsp");
      }
   }
   private RequestDispatcher processRelease(HttpServletRequest req, HttpServletResponse
resp, String title, String user_no)
   throws SQLException, IOException
   {
      Page page = new Page();
      page = page.page_pool(title, context, homePage);
      Page new_page = new Page(title, page.getBody(), page.getismodify(),
page.getispublic(), page.getownerno(), "0", page.getlocktime());
      new_page.page_pool_update("1", context);
      req.setAttribute("wpage", page);
      return context.getRequestDispatcher("/page.jsp");


   }

   private RequestDispatcher processPage(HttpServletRequest req, HttpServletResponse resp,
String title, String user_no)
   throws SQLException
   {
      Page page = new Page();
      if (title == null) title = homePage;
      page = page.page_pool(title, context, homePage);
      System.out.println(title == null);
      System.out.println(page == null);
      if (title.equals(homePage) || page.getispublic() || (page.getownerno() ==
Integer.valueOf(user_no).intValue()) || user_no.equals("1"))
      {
         req.setAttribute("wpage", page);
         req.setAttribute("wuser", DataBase.getUser(user_no));
         return context.getRequestDispatcher("/page.jsp");
      }
      else
      {
         req.setAttribute("title", title);

         req.setAttribute("ErrMsg", "2"); // 2: no privilage to read
         return context.getRequestDispatcher("/Login.jsp");
      }
   }
```

```
   private RequestDispatcher processEdit(HttpServletRequest req, String title)
   throws SQLException, IOException
   {
      HttpSession session;

      if (title == null) return null;
      Page page = new Page();
      page = page.page_pool(title, context, homePage);
      if (page == null) return null;
      session = req.getSession();
      String user_no = session.getAttribute("user_no").toString();
      if((page.getismodify() || Integer.toString(page.getownerno()).equals(user_no)) ||
user_no.equals("1"))
      {
         if (page.checkIflock(user_no, context))
         {
            req.setAttribute("Err", "6");   //1: the page is lock and time is not set
            return context.getRequestDispatcher("/page?title=" + title);
         }
         req.setAttribute("wpage", page);
         return context.getRequestDispatcher("/edit.jsp");
      }
      else
      {
         req.setAttribute("title", title);
         req.setAttribute("ErrMsg", "1"); // 1: no privilage to modify
         return context.getRequestDispatcher("/Login.jsp");
      }
   }

   private RequestDispatcher processUpdate(HttpServletRequest req, String title, String
body, String user_no)
   throws SQLException, IOException
   {
      if (user_no.compareTo("0") == 0)
      {
         req.setAttribute("ErrMsg", "3"); // 3: edit time out
         req.setAttribute("title", title);
         return context.getRequestDispatcher("/Login.jsp");
      }
      String ismodify = req.getParameter("ismodify");
      String ispublic = req.getParameter("ispublic");

      boolean setmodify, setpublic;
      if (title == null || body == null) return null;


      if (ismodify != null)
         setmodify = true;
      else
         setmodify = false;

      if(ispublic != null)
         setpublic = true;
      else
         setpublic = false;
      Page page = new Page();
      page = page.page_pool(title, context, homePage);

      page = new Page(title, body, setmodify, setpublic, page.getownerno(), "0", "");
      page.page_pool_update(user_no, context);
      req.setAttribute("wpage", page);
      return context.getRequestDispatcher("/page.jsp");
   }

   private RequestDispatcher processLogout(HttpServletRequest req)
   throws SQLException
   {
      HttpSession session = req.getSession();
      session.setAttribute("user_no", "-1");
```

```
      return context.getRequestDispatcher("/logout.jsp");
   }

   private RequestDispatcher processDelete(HttpServletRequest req, String title)
   throws SQLException
   {
      HttpSession session;
      session = req.getSession();
      String user_no = session.getAttribute("user_no").toString();
      if (user_no.toString().compareTo("0") == 0)
         return context.getRequestDispatcher("/help");

      String id = req.getParameter("id");

      if (id != null) {
         User user = DataBase.getUser(id);
         if (user == null) return null;
         if (DataBase.deleteUser(user) == 0) return null;
         return context.getRequestDispatcher("/users");
      } else {// if (title != null) {
         Page page = DataBase.getPage(title);
         if((!Integer.toString(page.getownerno()).equals(user_no)) &&
(!user_no.equals("1")))
            return context.getRequestDispatcher("/OrphanPages");
         if (page == null) return null;
         if (DataBase.deletePage(page) == 0) return null;
         page.page_pool_delete(page, context);
         return context.getRequestDispatcher("/OrphanPages");
      }
   }

   private RequestDispatcher processBack(HttpServletRequest req, String title)
   throws SQLException
   {
      if (title == null) return null;
      Page page = new Page();
      page = page.page_pool(title, context, homePage);
      if (page == null) return null;
      Vector fromTitles = page.getFromTitles();
      // If a single back link, then go there directly.
      if (fromTitles.size() == 1) {
         PageTitle fromTitle = (PageTitle) fromTitles.firstElement();
         page = DataBase.getPage(fromTitle.getTitle());
         req.setAttribute("wpage", page);
         return context.getRequestDispatcher("/page.jsp");
      }
      // If there is no back page, do following
      if (fromTitles.size() == 0) {
         // If click from Home Page, do not go anywhere
         if (title.compareTo("HomePage") == 0)
         {
            req.setAttribute("wpage", page);
            return context.getRequestDispatcher("/page.jsp");
         }
         else
         // If not click from Home Page, go back to Orphan Pages list
         {
            Vector OrphanPages = DataBase.getOrphanPages();
            req.setAttribute("OP", OrphanPages);
            return context.getRequestDispatcher("/OrphanPages.jsp");
         }
      }
      // Display two or more back links.
      req.setAttribute("wpage", page);
      return context.getRequestDispatcher("/back.jsp");
   }

   public ServletContext getContext() {return context; }
}
```

APPENDIX B

DATABASE CLASS PRINTOUT

```
package wiki;

import java.sql.*;
import javax.sql.*;
import javax.servlet.*;
import javax.servlet.ServletContext;
import java.util.*;
import javax.naming.*;

public class DataBase implements ServletContextListener
{
   private static DataSource ds;

   public DataBase() { };

   public void contextInitialized(ServletContextEvent event)
   {
      Connection conn = null;
      Connection conn1 = null;
      Statement statement = null;
      Statement statement1 = null;
      ResultSet rs = null;
      ResultSet rs1 = null;
      String query = null;
      String query1 = null;

      try {
         InitialContext ic = new InitialContext();
         if (ic == null) throw new Exception("No initial context.");
         ds = (DataSource) ic.lookup("java:comp/env/jdbc/postgres");
         if (ds == null) throw new Exception("No datasource.");
         conn = ds.getConnection();
         statement = conn.createStatement();
         conn1 = ds.getConnection();
         statement1 = conn1.createStatement();

         // Create tables if not present.
         try {
            query = "create table users (" +
               "no int primary key," +
               "id varchar(25) not null," +
               "name varchar(50)," +
               "password varchar(25) not null," +
               "email varchar(30) not null);";
            statement.executeUpdate(query);
            query = "create table pages (" +
               "title varchar(255) primary key," +
               "body text default ''," +
               "ismodify varchar(1) default '1'," +
               "ispublic varchar(1) default '1'," +
               "owner_no int not null references users (no)," +
               "iflock varchar(1) default '0'," +
               "locktime varchar(10) default '');";
            statement.executeUpdate(query);
            query = "create table links (" +
               "from_title varchar(255) not null references pages (title)," +
               "to_title varchar(255) not null references pages (title) );";
            statement.executeUpdate(query);
            query = "create index links_index_to on links(to_title);";
            statement.executeUpdate(query);
            query = "create index links_index_from on links(from_title);";
            statement.executeUpdate(query);
         } catch (SQLException sqle) { } // tables exist

         // Create default user if not exist
         query1 = "select * from users;";
         rs1 = statement1.executeQuery(query1);
         if (!rs1.next()) { // no user exists
            query1 = "insert into users values (1, 'wiki', 'wiki default user', 'wiki',
'');";
            statement1.executeUpdate(query1);
```

```
            }

            // Create homePage if not present.
            ServletContext context = event.getServletContext();
            String homePage = context.getInitParameter("homePage");
            query = "select title from pages where title='" + homePage + "';";
            rs = statement.executeQuery(query);
            if (!rs.next()) { // homePage missing
               query = "insert into pages values ('" + homePage + "', '', '1', '1', 1, '0',
'');";
               statement.executeUpdate(query);
            }


            rs.close();
            statement.close();
            conn.close();
            rs1.close();
            statement1.close();
            conn1.close();
         } catch (Exception e) {
            throw new RuntimeException(e);
         }
      }

      public void contextDestroyed(ServletContextEvent event)
      {
      }

      public static Page getPage(String title) throws SQLException
      {
         Connection conn = null;
         Statement statement = null;
         ResultSet rs = null;
         try {
            conn = ds.getConnection();
            statement = conn.createStatement();
            String query = "select * from pages where title='" + title + "';";
            rs = statement.executeQuery(query);
            if (!rs.first()) return null;
               return new Page(title, rs.getString("body"), rs.getBoolean("ismodify"),
rs.getBoolean("ispublic"), rs.getInt(5), rs.getString("iflock"),
rs.getString("locktime"));
         } finally {
            rs.close();
            statement.close();
            conn.close();
         }
      }

      public static Vector getAllPageTitle() throws SQLException
      {
         Vector AllPageTitle = new Vector();
         Connection conn = null;
         Statement statement = null;
         ResultSet rs = null;
         int count = 0;

         try {
            conn = ds.getConnection();
            statement = conn.createStatement();
            String query = "select title from pages";
            rs = statement.executeQuery(query);
            while(rs.next())
               AllPageTitle.add(rs.getString("title"));
            return AllPageTitle;
         } finally {
            rs.close();
            statement.close();
            conn.close();
         }
```

```
    }

    public static User getUser(String id) throws SQLException
    {
        Connection conn = null;
        Statement statement = null;
        ResultSet rs = null;
        try {
            conn = ds.getConnection();
            statement = conn.createStatement();
            String query = "select * from users where id='" + id + "';";
            rs = statement.executeQuery(query);
            if (!rs.first()) return null;
                return new User(rs.getInt(1), id, rs.getString("name"),
rs.getString("password"), rs.getString("email"));
        } finally {
            rs.close();
            statement.close();
            conn.close();
        }
    }

    public static User getUserByNo(String user_info_no) throws SQLException
    {
        Connection conn = null;
        Statement statement = null;
        ResultSet rs = null;
        try {
            conn = ds.getConnection();
            statement = conn.createStatement();
            String query = "select * from users where no='" + user_info_no + "';";
            rs = statement.executeQuery(query);
            if (!rs.first()) return null;
                return new User(rs.getInt(1), rs.getString("id"), rs.getString("name"),
rs.getString("password"), rs.getString("email"));
        } finally {
            rs.close();
            statement.close();
            conn.close();
        }
    }

    public static User getUserByemail(String email) throws SQLException
    {
        Connection conn = null;
        Statement statement = null;
        ResultSet rs = null;
        try {
            conn = ds.getConnection();
            statement = conn.createStatement();
            String query = "select * from users where email='" +email + "';";
            rs = statement.executeQuery(query);
            if (!rs.first()) return null;
                return new User(rs.getInt(1), rs.getString("id"), rs.getString("name"),
rs.getString("password"), rs.getString("email"));
        } finally {
            rs.close();
            statement.close();
            conn.close();
        }
    }

    public static int deletePage(Page page) throws SQLException
    {
        String title = page.getTitle();
        Connection conn = null;
        Statement statement = null;
        int rs;
        try {
            conn = ds.getConnection();
            statement = conn.createStatement();
```

```
            String query = "delete from links where from_title='" + page.getTitle() + "';";
            rs = statement.executeUpdate(query);
            query = "delete from pages where title='" + title + "';";
            rs = statement.executeUpdate(query);
            return rs;
        } finally {
            statement.close();
            conn.close();
        }
    }

    public static int deleteUser(User user) throws SQLException
    {
        String id = user.getid();
        Connection conn = null;
        Statement statement = null;
        int rs;
        try {
            conn = ds.getConnection();
            statement = conn.createStatement();
            String query = "delete from users where id='" + id + "';";
            rs = statement.executeUpdate(query);
            return rs;
        } finally {
            statement.close();
            conn.close();
        }
    }


    /**
     * Update body of a page in the database.
     * @param page the page with body to be updated
     * @returns false if page is not found.
     */
    public static boolean updatePage(Page page, String user_no) throws SQLException
    {
        Connection conn = null;
        Statement statement = null;
        ResultSet rs = null;
        String title = page.getTitle();
        try {
            conn = ds.getConnection();
            statement = conn.createStatement();

            // Get old version of page.
            String query = "select * from pages where title='" + title + "';";
            rs = statement.executeQuery(query);
            if (!rs.first()) return false; // Page not found.
            Page oldPage = new Page(title, rs.getString("body"), rs.getBoolean("ismodify"),
rs.getBoolean("ispublic"), rs.getInt("owner_no"), rs.getString("iflock"),
rs.getString("locktime"));

            // Write new body.
            int setmodify, setpublic;
            if(page.getismodify())
                setmodify = 1;
            else
                setmodify = 0;
            if(page.getispublic())
                setpublic = 1;
            else
                setpublic = 0;

            query = "update pages set body='" + escapeForSqlQuery(page.getBody()) +
                        "', ismodify='" + setmodify +
                        "', ispublic='" + setpublic +
                        "', iflock='" + page.getiflock() +
                        "', locktime='" + page.getlocktime() +
                        "' where title='" + title + "';";
            statement.executeUpdate(query);
```

```
        // Make lists of old links to be deleted, and new links to be added.
        HashSet oldLinks = oldPage.getLinks();
        HashSet newLinks = page.getLinks();
        Iterator it = oldLinks.iterator();
        while (it.hasNext()) {
           String oldTitle = (String) it.next();
           if (newLinks.contains(oldTitle)) {
              it.remove();
              newLinks.remove(oldTitle);
           }
        }

        // Delete old links not found in the new link set.
        it = oldLinks.iterator();
        while (it.hasNext()) {
           String oldTitle = (String) it.next();
           query = "delete from links where from_title='" + title +
                    "' and to_title ='" + oldTitle + "';";
           statement.executeUpdate(query);
        }

        // Create new empty pages for new links that don't exist in database.
        it = newLinks.iterator();
        int setismodify, setispublic;
        if(page.getismodify())
           setismodify = 1;
        else
           setismodify = 0;
        if(page.getispublic())
           setispublic = 1;
        else
           setispublic = 0;
        while (it.hasNext()) {
           String newTitle = (String) it.next();
           query = "select title from pages where title='" + newTitle + "';";
           rs = statement.executeQuery(query);
           if (rs.first()) continue; // page already exists
           query = "insert into pages values ('" + newTitle +
                   "', '', '" + setismodify +
                   "', '" + setispublic +
                   "', '" + user_no +
                   "', '0'" +
                   ", '');";
           statement.executeUpdate(query);
        }

        // Add new links not found in the old link set.
        it = newLinks.iterator();
        while (it.hasNext()) {
           String newTitle = (String) it.next();
           query = "insert into links (from_title, to_title) values ('" +
                    title + "', '" + newTitle + "');";
           statement.executeUpdate(query);
        }
     } finally {
        rs.close();
        statement.close();
        conn.close();
     }
     return true;
  }

  private static String escapeForSqlQuery(String s)
  {
     String r = s.replaceAll("\\\\", "\\\\\\\\");
     return r.replaceAll("\\'", "\\\\'");
  }

  /**
   * Get the titles of pages that link to the given title.
```

```
 * @returns Vector of Title
 */
public static Vector getFromTitles(String title) throws SQLException
{
   Connection conn = null;
   Statement statement = null;
   ResultSet rs = null;
   try {
      conn = ds.getConnection();
      statement = conn.createStatement();
      String query = "select from_title from links where to_title='" + title + "';";
      rs = statement.executeQuery(query);
      Vector fromTitles = new Vector();
      while (rs.next()) {
         PageTitle fromTitle = new PageTitle(rs.getString("from_title"));
         fromTitles.add(fromTitle);
      }
      return fromTitles;
   } finally {
      rs.close();
      statement.close();
      conn.close();
   }
}

 /**
  * Get a Orphan Pages from the Database.
  */
public static Vector getOrphanPages() throws SQLException
{
   Connection conn = null;
   Connection conn1 = null;
   Statement statement = null;
   Statement statement1 = null;
   ResultSet rs = null;
   ResultSet rs1 = null;
   String querypage;
   Vector OrphanPages = new Vector();
   try {
      conn = ds.getConnection();
      statement = conn.createStatement();
      String query = "select title from pages where title <> 'HomePage'";
      rs = statement.executeQuery(query);
      while(rs.next())
      {
         querypage = rs.getString("title");
         System.out.println(querypage);
         try {
            conn1 = ds.getConnection();
            statement1 = conn1.createStatement();
            String query1 = "select * from links where to_title = '" + querypage + "'";
            rs1 = statement1.executeQuery(query1);
            if (!rs1.next())
            {
               PageTitle fromTitle = new PageTitle(querypage);
               OrphanPages.add(fromTitle);
            }
         } finally {
            rs1.close();
            statement1.close();
            conn1.close();
         }
      }
         return OrphanPages;
   } finally {
      rs.close();
      statement.close();
      conn.close();
   }

}
```

```
    /**
     * Get a Users List from the Database.
     */
    public static Vector getUsers() throws SQLException
    {
       Connection conn = null;
       Statement statement = null;
       ResultSet rs = null;
       String querypage;
       Vector Users = new Vector();
       try {
          conn = ds.getConnection();
          statement = conn.createStatement();
          String query = "select id from users";
          rs = statement.executeQuery(query);
          while(rs.next())
             Users.add(rs.getString("id"));
          return Users;
       } finally {
          rs.close();
          statement.close();
          conn.close();
       }
    }

    public static int getNewUserNo() throws SQLException
    {
       Connection conn = null;
       Statement statement = null;
       ResultSet rs = null;
       try {
          conn = ds.getConnection();
          statement = conn.createStatement();
          String query = "select max(no) as userno from users";
          rs = statement.executeQuery(query);
          rs.next();
          int rv = rs.getInt(1) + 1;
          return rv;
       } finally {
          rs.close();
          statement.close();
          conn.close();
       }
    }

    public static boolean ifuserexist(User user) throws SQLException
    {
       Connection conn = null;
       Statement statement = null;
       ResultSet rs = null;
       try {
          conn = ds.getConnection();
          statement = conn.createStatement();
          String query = "select * from users where id = '" + user.getid() + "'";
          rs = statement.executeQuery(query);
          if(rs.next())
             return true;
          else
             return false;
       } finally {
          rs.close();
          statement.close();
          conn.close();
       }
    }

    public static int UpdateInfo(String user_info_no, String password, String email) throws
SQLException
    {
       Connection conn = null;
```

```
        Statement statement = null;
        try {
            conn = ds.getConnection();
            statement = conn.createStatement();
            String query = "update users set password = '" + password +
                           "', email='" + email + "' where no='" + user_info_no + "';";
            statement.executeUpdate(query);
            return 1;
        } finally {
            statement.close();
            conn.close();
        }
    }

    public static int CreateUser(User user) throws SQLException
    {
        if (ifuserexist(user))
            return 0;
        if (getUser("wiki") == null) {
            Connection conn = null;
            Statement statement = null;
            try {
                conn = ds.getConnection();
                statement = conn.createStatement();
                String query = "insert into users (no, id, name, password, email) values ('" +
                               user.getno() + "', '" + user.getid() + "', '" +
                               user.getname() + "', '" + user.getpassword() + "', '" +
                               user.getemail() + "');";
                statement.executeUpdate(query);
                return 1;
            } finally {
                statement.close();
                conn.close();
            }
        } else {
            Connection conn = null;
            Statement statement = null;
            try {
                conn = ds.getConnection();
                statement = conn.createStatement();
                String query = "update users set id ='" + user.getid() +
                               "', name ='" + user.getname() +
                               "', password ='" + user.getpassword() +
                               "', email='" + user.getemail() +
                               "' where id='wiki';";
                statement.executeUpdate(query);
                return 1;
            } finally {
                statement.close();
                conn.close();
            }
        }
    }
}
```

APPENDIX C

PAGE CLASS PRINTOUT

```
package wiki;

import java.sql.*;
import java.io.*;
import java.util.*;
import javax.servlet.*;

public class Page
{
   public static final int MAX_PAGE_NAME_LENGTH = 255;

   /** Initialized by init() in Controller servlet. */
   protected static String contextPath;

   private String title;
   private String body;
   private boolean ismodify;
   private boolean ispublic;
   private int owner_no;
   private String user_no;
   private String iflock;
   private String locktime;

   public Page()
   {}

   public Page(String title, String body, boolean ismodify, boolean ispublic, int
owner_no, String iflock, String locktime)
   // public Page(String title, String body, String ismodify, String ispublic)
   {
      this.title = title;
      this.body = body;
      this.ismodify = ismodify;
      this.ispublic = ispublic;
      this.owner_no = owner_no;
      this.iflock =  iflock;
      this.locktime = locktime;
   }

   public String getTitle() { return title; }
   public String getBody() { return body; }
   public boolean getismodify() { return ismodify; }
   public boolean getispublic() { return ispublic; }
   public int getownerno() { return owner_no; }
   public String getiflock() {return iflock; }
   public String getlocktime() { return locktime; }
   public void setBody(String body) { this.body = body; }
   public void setIflock(String iflcok) { this.iflock = iflock; }
   public void setlocktime(String locktime) { this.locktime = locktime; }

   /**
    * Parses body for links to other wiki pages.
    * @returns HashSet containing titles of wiki pages
    */
   public HashSet getLinks()
   {
      HashSet titles = new HashSet();
      try {
         StringReader sr = new StringReader(body);
         while (true) {
            int c = sr.read();
            if (c == -1) break;
            char ch = (char) c;
            if (ch == '\\') {
               sr.mark(2);
               c = sr.read();
               if (c == -1) break;
               ch = (char) c;
               if (!isValidTitleCharacter(ch)) continue;
               sr.reset();
               String title = extractTitle(sr);
```

```
            titles.add(title);
         }
      }
   } catch (IOException ioe) {
      throw new RuntimeException(ioe);
   }
   return titles;
}

private static String urlCharacters =
      "/~%#?$-_@.&+!*\"'(),:=" +
      "0123456789" +
      "abcdefghijklmnopqrstuvwxyz" +
      "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

private static boolean isValidUrlCharacter(char c)
{
   return urlCharacters.indexOf(c) >= 0;
}

public void setUserNo(String no)
{
   user_no = no;
}

/**
 * Transformation Rules:
 * 1. & converts to &amp;
 * 2. < converts to &lt;
 * 3. \\ converts to \
 * 3. \* converts to *
 * 4. \NewPage converts to <a href='/wiki/page?title=NewPage'>New Page</a>
 * 5. http://yahoo.com converts to <a href='http://yahoo.com'>http://yahoo.com</a>
 * 6. ---- converts to <hr>
 */
public String getFormattedBody() throws IOException, SQLException
{
   StringBuffer sb = new StringBuffer();
   StringReader sr = new StringReader(body);
   boolean inPre = false;
   boolean firstCharInLine = true;
   boolean inHead2 = false;
   boolean table = false;
   boolean pre_space = false;
   int  bullet = 0;
   boolean if_pre_bullet = false;
   while (true) {
      int c = sr.read();
      if (c == -1)
      {
         if (table)
         {
            table = false;
            sb.append("</table>");
         }
         break;
      }
      char ch = (char) c;
      if (firstCharInLine) {
         bullet = 0;
         firstCharInLine = false;
         if (ch == '.') {
            inHead2 = true;
            sb.append("<h2>");
            continue;
         }
         if ((ch != '|') && table)
         {
            table = false;
            sb.append("</table>");
         }
```

```
            if (ch == '|')   // table parse
            {
               sr.mark(128);
               c = sr.read();
               ch = (char) c;
               pre_space = false;
               if (ch != '|')
               {
                  sb.append('|');
                  sr.reset();
                  continue;
               }

               sr.mark(128);   // get a '||'
               StringBuffer s = new StringBuffer();
               while (true)
               {
                  c = sr.read();
                  if (c == -1) break;
                  ch = (char) c;
                  if (ch == '\r')
                     break;
                  s.append(ch);
               }
               if (((s.charAt(s.length() - 2) == '|') && (s.charAt(s.length() - 1)
==='|'))) // insert a table
               {
                  sr.reset();
                  sr.mark(128);
                  c = sr.read();
                  ch = (char) c;
                  int i = 0;
                  while (ch == '|')
                  {
                     i++;
                     sr.mark(128);
                     c = sr.read();
                     ch = (char) c;
                  }
                  sr.reset();
                  int colspan = (i / 2) + 1;
                  if (table)
                     sb.append("<tr><td colspan=" + colspan + ">");
                  else
                     sb.append("<table border = 1><tr><td colspan=" + colspan + ">");
                  if ((i % 2) == 1)
                     sb.append("|");
                  table = true;
               } else
               {
                  sb.append("||");
                  sr.reset();
               }
               continue;
            }
            if (ch != '*' && if_pre_bullet)
            {
               if_pre_bullet = false;
               sb.append("</ul>");
               bullet = 0;
            }
         }

         if (ch == '\n') {
            firstCharInLine = true;
            // if_pre_bullet = false;
            if (!inPre && !table) {
               for (int i = 1; i < bullet; i++)
                  sb.append("</li></ul>");
               if (bullet == 1)
```

```
                sb.append("</li>");
            bullet = 0;
            if (inHead2) { sb.append("</h2>"); inHead2 = false; }
            else sb.append("<br>");
            continue;
         }
         if (pre_space)
         {
            sb.append("</pre>");
            continue;
         }
      }
      if (ch == '&') { pre_space = false; sb.append("&amp;"); continue; }
      if (ch == '<') { pre_space = false; sb.append("&lt;"); continue; }
      if (ch == '\\') {
       . if (inPre) { sb.append("\\"); continue; }
         sr.mark(2);
         c = sr.read();
         if (c == -1) { sb.append('\\'); break; }
         ch = (char) c;
         if (ch == '\\') { sb.append('\\'); continue; }
         if (ch == '*') { sb.append('*'); continue; }
         if (!isValidTitleCharacter(ch)) continue;  // invalid syntax
         // We have a wiki link.
         sr.reset();
         String title = extractTitle(sr);
         // Page wpage = new Page(title, "", true, true, 1);
         Page wpage = DataBase.getPage(title);
         if (wpage.getispublic() || user_no.equals("1") || wpage.getownerno() ==
Integer.valueOf(user_no).intValue())
            sb.append("<a href='" + contextPath + "/page?title=" + title + "'>" +
wpage.getSpacedTitle() + "</a>");
         else
            sb.append(wpage.getSpacedTitle());
         continue;
      }

      if (ch == 'h') {
         sr.mark(128);
         StringBuffer s = new StringBuffer();
         for (int i = 0; i < 6; ++i) {
            c = sr.read();
            if (c == -1) break;
            ch = (char) c;
            s.append(ch);
         }
         if (!s.toString().equalsIgnoreCase("ttp://")) {
            sb.append('h');
            sr.reset();
            continue;
         }
         // We have a hyperlink.
         // Read until white space.
         StringBuffer url = new StringBuffer();
         StringBuffer url_mask = new StringBuffer();
         boolean if_url_mask = false;
         while (true) {
            c = sr.read();
            if (c == -1) break;
            ch = (char) c;
            if (ch == '|') if_url_mask = true;
            if (if_url_mask)
            {
               if((ch != '|') && isValidUrlCharacter(ch)) url_mask.append(ch);
               if((ch != '|') && !isValidUrlCharacter(ch)) break;
            }
            else
            {
               if (!isValidUrlCharacter(ch)) break;
               url.append(ch);
            }
```

```
            }
            String title = new String(url_mask);
            Page wpage = new Page(title, "", true, true, 1, "0", "");
            if (url.length() != 0)
               if (if_url_mask && (url_mask.length() != 0))
                  sb.append("<a href='http://" + url + "' target=_blank>" +
wpage.getSpacedTitle() + " </a>");
               else
                  sb.append("<a href='http://" + url + "' target=_blank>http://" + url +
"</a>");

            if (c == -1) break;
            continue;
         }

         // check if inserting a image
         if (ch == 'i') {
            sr.mark(128);
            StringBuffer s = new StringBuffer();
            for (int i = 0; i < 5; ++i) {
               c = sr.read();
               if (c == -1) break;
               ch = (char) c;
               s.append(ch);
            }
            if (!s.toString().equalsIgnoreCase("mg://")) {
               sb.append('i');
               sr.reset();
               continue;
            }
            // We have a hyperlink.
            // Read until white space.
            StringBuffer url = new StringBuffer();
            while (true) {
               c = sr.read();
               if (c == -1) break;
               ch = (char) c;
               if (!isValidUrlCharacter(ch)) break;
               url.append(ch);
            }
            if (url.length() != 0) {
               sb.append("<img src='http://" + url + "'>");
            }
            if (c == -1) break;
            continue;
         }

         if (ch == '-' && !inPre && readString(sr, "---")) {
            sb.append("<hr>");
            continue;
         }
         if (ch == '*') {
            if (!inPre && readString(sr, "start-code")) {
               sb.append("<pre>");
               inPre = true;
               continue;
            }
            if (inPre && readString(sr, "end-code")) {
               sb.append("</pre>");
               inPre = false;
               continue;
            }
            if (inPre) {
               sb.append("*");
               continue;
            }
            else {
               if (if_pre_bullet)
               {
                  sb.append("<li>");
               }
```

```
            else
            {
               sb.append("<ul><li>");
               if_pre_bullet = true;
               // bullet++;
            }
            bullet++;;
            continue;
         }
      }
      // in table area
      if (table)
      {
         if (ch == '|')
         {
            sr.mark(128);
            c = sr.read();
            ch = (char) c;
            if (ch != '|')
            {
               sb.append("|");
               sr.reset();
               continue;
            }
            int i = 0;  // get a '||'
            sr.mark(128);
            c = sr.read();
            ch = (char) c;
            while (ch == '|')
            {
               i++;
               if (ch == '|')
                  sr.mark(128);
               c = sr.read();
               ch = (char) c;
            }
            if (ch == '\r')
            {
               if ((i % 2) == 1)
                  sb.append("|");
               sb.append("</td></tr>");
               continue;
            } else
            {
               sr.mark(128);
               c = sr.read();
               if (c == -1)
               {
                  sb.append("</table>");
                  continue;
               }
               else
               {
                  sr.reset();
                  int colspan = (i / 2) + 1;
                  sb.append("</td><td colspan=" + colspan + ">");
                  if ((i % 2) == 1)
                     sb.append("|");
                  sr.reset();
                  continue;
               }
            }
         }
      }
      // It's an ordinary character.
      sb.append(ch);
   }
   if (if_pre_bullet)
      sb.append("</li/</ul>");
   return sb.toString();
}
```

```
   private boolean readString(Reader r, String s) throws IOException
   {
      r.mark(s.length());
      for (int i = 0; i < s.length(); ++i) {
         int c = r.read();
         if (c == -1 || c != (int) s.charAt(i)) {
            r.reset();
            return false;
         }
      }
      return true;
   }

   private static String validTitleCharacters =
         " " +
         "0123456789" +
         "abcdefghijklmnopqrstuvwxyz" +
         "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

   public static boolean isValidTitleCharacter(char ch)
   {
      return validTitleCharacters.indexOf(ch) >= 0;
   }

   private static String extractTitle(Reader r) throws IOException
   {
      StringBuffer sb = new StringBuffer();
      for (int i = 0; i < MAX_PAGE_NAME_LENGTH; ++i) {
         r.mark(2);
         int c = r.read();
         if (c == -1) break;
         char ch = (char) c;
         if (!isValidTitleCharacter(ch)) break;
         sb.append(ch);
      }
      r.reset();
      return sb.toString();
   }

   public String getSpacedTitle()
   {
      if (title.length() == 0) return "";
      StringBuffer sb = new StringBuffer();
      sb.append(title.charAt(0));
      for (int i = 1; i < title.length(); ++i) {
         if (Character.isUpperCase(title.charAt(i))) {
            sb.append(' ');
         }
         sb.append(title.charAt(i));
      }
      return sb.toString();
   }


   public Vector getFromTitles() throws SQLException
   {
      return DataBase.getFromTitles(title);
   }

   public synchronized boolean checkIflock(String user_no, ServletContext context)
   throws SQLException, IOException
   {
      int iflockedit = 0;
      int remain_min = 5;
      int remain_sec = 0;
      int remain = remain_min * 60 + remain_sec; // have time 5 mins after another user
request

      int Ehour = 0, Emin = 0, Esecond = 0, Etotalsecond = 0;
      int Chour = 0, Cmin = 0, Csecond = 0, Ctotalsecond = 0;
```

```
      java.util.Date current_date = new java.util.Date();
      String current = Integer.toString(current_date.getHours()) + ":" +
Integer.toString(current_date.getMinutes()) + ":" +
Integer.toString(current_date.getSeconds());
      int i;
      if (locktime != null)
      if (!locktime.equals(""))
      {
         Ehour = (Time.valueOf(locktime)).getHours();
         Emin = (Time.valueOf(locktime)).getMinutes();
         Esecond = (Time.valueOf(locktime)).getSeconds();
         Etotalsecond = (Ehour*60 + Emin) * 60 + Esecond;;
      }
      Chour = Time.valueOf(current).getHours();
      Cmin = Time.valueOf(current).getMinutes();
      Csecond = Time.valueOf(current).getSeconds();
      Ctotalsecond = (Chour * 60 + Cmin) * 60 + Csecond;

      if(iflock.equals("1") && (!locktime.equals("")) && ((Ctotalsecond > Etotalsecond) ||
((Etotalsecond-Ctotalsecond) > remain)))
      { // the page is edited by fail at the last time, so it's not editing.
         iflockedit = 0;
         locktime = "";
         iflock = "1";
      } else if(iflock.equals("1") && locktime.equals(""))
      { // the page is editing, and this is the first time to request editing.
         iflockedit = 1;
         int hours = Time.valueOf(current).getHours();
         int minutes = Time.valueOf(current).getMinutes() + remain_min;
         int seconds = Time.valueOf(current).getSeconds() + remain_sec;
         if (seconds > 59)
         {
            seconds = seconds - 60;
            minutes++;
         }
         if (minutes > 59)
         {
            minutes = minutes - 60;
            hours++;
         }
         setlocktime(Integer.toString(hours) + ":" + Integer.toString(minutes) + ":" +
Integer.toString(seconds));
         iflock = "1";
      } else if(iflock.equals("1"))
      { // the page is editing, and this is not the first time to request editing.
         iflockedit = 1;
      } else if(!(iflock.equals("1")))
      { // the page is not editing.
         iflockedit = 0;
         iflock = "1";
         locktime = "";
      }
      page_pool_update(user_no, context);
      return iflockedit == 1;
   }

   public Page page_pool(String title, ServletContext context, String homePage)
   throws SQLException
   {
      Page page = null;
      Pool page_pool = new Pool();

      if (title == null) title = homePage;
      if (context.getAttribute("page_pool") != null)  // check if page_pool exist
      {
         int i;
         page_pool = (Pool)context.getAttribute("page_pool");
         System.out.println("ControllerServlet.java " + page_pool.size());
         page = (Page) page_pool.get(title);
         if (page == null)
         {  // the page is not in the pool
```

```
            page = DataBase.getPage(title);
            if (page == null) return null;

            // move the page into pool
            page_pool.put(title, page);
            context.setAttribute("page_pool", page_pool);
         }
      }
      else
      {  // the page pool does not exits
         System.out.println("Page_Pool not exist");
         page_pool = new Pool();
         page = DataBase.getPage(title);
         page_pool.put(title, page);
            context.setAttribute("page_pool", page_pool);
      }
      return page;
   }

   public void page_pool_update(String user_no, ServletContext context)
   throws SQLException, IOException
   {
      Pool page_pool = (Pool) context.getAttribute("page_pool");
      Page old_page = (Page) page_pool.get(getTitle());
      if (old_page != null)
      {
         page_pool.remove(old_page.getTitle());
         page_pool.put(getTitle(), this);
         context.setAttribute("page_pool", page_pool);
      }
      boolean foo = DataBase.updatePage(this, user_no);
   }

   public void page_pool_delete(Page page, ServletContext context)
   {
      Pool page_pool = (Pool) context.getAttribute("page_pool");
      page_pool.remove(page.getTitle());
      context.setAttribute("page_pool", page_pool);
   }


}
```

APPENDIX D

PAGETITLE CLASS PRINTOUT

```
package wiki;

import java.sql.*;
import java.io.*;
import java.util.*;

public class PageTitle
{
   public static final int MAX_PAGE_NAME_LENGTH = 255;

   private String title;

   public PageTitle(String title)
   {
      this.title = title;
   }

   public String getTitle() { return title; }

   private final static String validTitleCharacters =
         " " +
         "0123456789" +
         "abcdefghijklmnopqrstuvwxyz" +
         "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

   public static boolean isValidTitleCharacter(char ch)
   {
      return validTitleCharacters.indexOf(ch) >= 0;
   }

   /**
    * Initialize by extracting title from the reader.
    */
   public PageTitle(Reader r) throws IOException
   {
      StringBuffer sb = new StringBuffer();
      for (int i = 0; i < MAX_PAGE_NAME_LENGTH; ++i) {
         r.mark(2);
         int c = r.read();
         if (c == -1) break;
         char ch = (char) c;
         if (!isValidTitleCharacter(ch)) break;
         sb.append(ch);
      }
      r.reset();
      title = sb.toString();
   }

   public String getSpacedTitle()
   {
      if (title.length() == 0) return "";
      StringBuffer sb = new StringBuffer();
      sb.append(title.charAt(0));
      for (int i = 1; i < title.length(); ++i) {
         if (Character.isUpperCase(title.charAt(i))) {
            sb.append(' ');
         }
         sb.append(title.charAt(i));
      }
      return sb.toString();
   }

   /**
    * @returns Vector of PageTitle
    */
   public Vector getFromTitles() throws SQLException
   {
      return DataBase.getFromTitles(title);
   }
}
```

APPENDIX E

USER CLASS SOURCE CODE

```
package wiki;

import java.sql.*;
import java.io.*;
import java.util.*;

public class User
{
   public static final int MAX_PAGE_NAME_LENGTH = 255;

   /** Initialized by init() in Controller servlet. */
   // protected static String contextPath;

   private int no;
   private String id;
   private String name;
   private String password;
   private String email;

   public User (int no, String id, String name, String password, String email)
   {
      this.no = no;
      this.id = id;
      this.name = name;
      this.password = password;
      this.email = email;
   }

   public int getno() { return no; }
   public String getid() { return id; }
   public String getname() { return name; }
   public String getpassword() { return password; }
   public String getemail() { return email; }

}
```

APPENDIX F

POOL CLASS SOURCE CODE

```
package wiki;

import java.sql.*;
import java.io.*;
import java.util.*;

public class Pool extends LinkedHashMap
{
   private static final int MAX_ENTRIES = 100;

   protected boolean removeEldestEntry(Map.Entry eldest)
   {
      return  size() > MAX_ENTRIES;
   }
}
```

REFERENCES

1. Martin Fowler with Kendall Scott. UML Distilled - A brief guide to the standard object modeling language. Addison Wesley Longman, July 2001.

2. Larne Pekowsky. JavaServer Pages. Addison Wesley, April 2000.

3. Eric Freeman. JavaSpaces Principles, Patterns, and Practice. Addison Wesley, Novermber 1999.

4. Ken Arnold and James Gosling. The Java Programming Language Second Edition. Addison Wesley, February 2000.

5. H.M.Deitel and P.J.Deitel. JAVA 2 Platform - How to program. Prentice Hall, New Jersey, 2002.

6. Falkner, Galbraith, et al. Beginning JSP Web Development. Wrox Press, 2001.

7. Elmasri and navathe. Fundamentals of Database Systems, third edition. Addison Wesley, June 2000.

8. David M. Geary. Advanced JavaServer Pages. Prentice Hall PTR, 2001.

9. William B. Sanders. JavaScript DESIGN. New Riders, 2002.

10. PostgreSQL Reference Manual for version 7.3. <http://www.postgresql.org/docs/ >