Project A: Extending Microblog

In this first project, you will spend an intensive three weeks understanding the implementation of a small web log ("blog") application, Microblog, and extending it to add additional functionality.

The project must be done individually, although you are strongly encouraged to ask for help from the TAs, classmates, and friends. The goal here is for you to understand, top-to-bottom how a database-backed web application works. In project B, you will design and implement your own database-backed web application from scratch.

Before you start

Read the handout "Using Oracle in the TLAB". This will explain how to log in to the tlab server, how to configure your environment, and how to access oracle using SQL*Plus. Make sure that your environment is working correctly.

Getting and installing Microblog

To install Microblog, log into your account on tlab-server and do the following:

```
cd ~/public_html
tar xvfz ~pdinda/HANDOUT/microblog.tgz
cd microblog
more README
```

The README file will give you detailed instructions on how to configure Microblog and verify that it is working. You should be able to visit your Microblog via http://tlab-login/~you/microblog/blog.html. At this point, you should be able to log in as the root user and do the following: add and delete users, add and revoke user permissions, write messages, see a summary of all messages, and see all the messages.

Note that we do expect that your blog system will be running and visible at http://tlab-login/~you/microblog/blog.html. We will have a master list of the blogs available on the course web site so that you'll be able to check out the progress of other student's blogs. We will provide you with a mechanism that will not permit other students to see your code, only its execution.

How does Microblog work?

The state of Microblog (users, their permissions, and the messages) lives entirely in the database. That state consists of a sequence and three tables (see blog.sql). The blog_users table contains the username, password, and email address of each Microblog user. The password is stored in cleartext, which is insecure. The blog_actions table contains the names of all the possible actions that a user may take in Microblog. The blog_permissions table maps from Microblog usernames to the actions that they have permission to take. An action should only be taken if the action exists in the blog_actions table, the user exists in the blog_users table, and the user has permission for the action in the blog_permissions table. By default, the following actions are made available: manage-users, query-messages, delete-any-messages, delete-own-messages, and write-messages. Two users are added by default: root, with password rootroot, who can do anything, and none, with password nonenone, who can do nothing.

The blog_message_id sequence provides a way of getting a unique number for tagging each message (the message id). The blog_messages table contains the actual messages. In addition to a message id, each message has an author (who must exist in the blog_messages blog_users table), a subject, a timestamp, and the text of the message (which can be HTML). A message also has a field called "respid", which is the id of the message to which it is responding. This referential behavior means that the messages form a tree. By default, there is a message with id zero, written by user none, and referring to itself that is installed into the database. This message, which is never displayed, is the root of the message tree.

Microblog has a notion of user sessions. The state of a session is kept in the web browser, in a cookie named MicroblogSession. When users log into Microblog they are given a cookie that contains their login name and password in cleartext (this is not secure) and is set to expire in about one hour. Without a cookie, the user can only log in. When presented with a cookie, Microblog uses its contents to authenticate the user and check to see if he has the necessary permissions (authorization) to do what he wants.

The following is a more detailed description of a Microblog session:

- ⇨ The user visits http://tlab-server/~you/microblog/blog.html
- ⇨ blog.html is loaded
  - o It creates an HTML frameset, consisting of a left frame and a right frame
  - o The left frame is filled with actions.html, whose different links are targeted at the right frame.
  - o The right frame is filled by executing blog.pl with the "act" parameter set to "query"
  - o blog.pl notices that there is no cookie and so forces the "login" action.
  - o The login form is displayed.
- ⇨ The user fills out the login form and hits submit.
- ⇨ blog.pl is invoked with the "act" parameter set to "login", and the "loginrun" parameter set to one.
  - o blog.pl extracts the "user" and "password" parameters provided by the form, and does a SQL query to see if the combination exists in the database.
  - o If the combination exists, it creates a cookie with the combination and passes it back to the user's browser. Next, it displays the query form (provided the user is permitted query-messages).
  - o If the combination does not exist, blog.pl complains, does not return a cookie, and displays the login form again.
- ⇨ At this point, the user's browser has the cookie, which is good for one hour.

- ⇨ The default "query" action, if there is a cookie, and the user/password combination in the cookie is valid, and the user has query-messages permission, displays the query form and then does a SQL query to generate a summary of all the messages in the database which it then displays as an HTML table.
- ⇨ The user fills out the query form and hits submit.
- ⇨ Seeing that the "queryrun" parameter exists, blog.pl, if there is a cookie, and the user/password combination in the cookie is valid, and the user has query-messages permission, executes a SQL query that fetches all the matching messages. It then prints them formatted in HTML.
- ⇨ On every interaction, the cookie is refreshed, so the one hour time limit on the cookie is basically the maximum idle time before the user is logged out.
- ⇨ The other actions are very similar to the query action.

Project Steps

In this project, you will extend Microblog to provide the following functionality. Each extension is marked with a percentage that is intended to reflect its perceived difficulty level. You may do the extensions in any order, although we believe the order presented will be the easiest. We believe the best approach to building each extension is the following:

1. Read and understand blog.sql and blog.pl
2. Design the SQL statements that are needed
3. Test the SQL statements using SQL*Plus.
4. Embed the SQL statements into Perl functions (see UserAdd(), for example)
5. Write the Perl logic to call the functions at the appropriate times.

You should avoid becoming bogged down in the Perl and, especially, the HTML, parts of the code. As this is a databases course, you will be graded on correctness, not the appearance of the final product. Just ask us for help in person or on the newsgroup if you can't figure something out.

Implement query (30%) : As handed out, message queries completely ignore all the arguments and simply return all messages. Extend MessageQuery so that it is possible to query messages by any combination of author, title, date range, and content pattern match. You will find that it is easiest to add query-by-author and title first. Query by date will require that you become familiar with Time::ParseDate. Query by pattern match will require that you become familiar with the Oracle regular expressions system (REGEXP_LIKE()). It should be possible to query using regular expressions for author, title, and content.

Add reply and delete (20%) : In the message summary display and the message query display, it should be possible to reply to or delete a message in a single click. To do so, modify MessageSummary and MessageQuery so that when they print a message, they also print an anchor tag that leads to deletion or update. For example:

```
<a href="blog.pl?act=delete&deleterun=1&id=533">delete</a>
```

```
<a href="blog.pl?act=reply&respid=533">reply</a>
```

These urls will then point to your delete and reply functionality. A user is allowed to delete any of his own messages if he has delete-own-messages permission, and any messages if he has delete-any-messages permission. In order to reply, he must have write-messages permission. A message can only be deleted if no other messages refer to it.

Add tree display (20%) : Modify message summary and message query so that they display the messages as a tree. You may find the Oracle-specific "CONNECT BY" feature to be useful here, although it is not necessary.

Add guest user, registration, and register-and-post (30%) : Introduce the notion of guest user who only has permission to read and query messages. The guest user does not need to log in. If the guest user attempts to reply to a message or to post a new message, he should be shown a screen that lets him *simultaneously* register (or log in if he's already registered) and post. That is, the register-and-post page has a single form and only one submit button. The pseudocode for this page will look something like this:

1. Get username, password, and message subject and text.
2. Check to see if user already exists. If so, log him in and post the message.
3. If the user does not exist, do the following as a *single transaction*:
    a. Create user with password (notice that this can fail due to the user name already existing or due to the password being incorrect
    b. Give the user permission to post (and other appropriate permissions)
    c. Log in the user
    d. Post the message (notice that this can fail too)

Notice that step 3 requires that you group at least three SQL inserts (user, permissions, and message) and a sequence operation (message sequence number) together as a single transaction that either all succeeds or all fails.

Where to go for help

The goal of this project is to learn how a database-backed web application works. Don't fall into the trap of spending lots of time generating pretty HTML or particularly elegant Perl. Get the SQL right first and make the Perl return what you need. We don't want you to get stuck on the details of Perl, CGI, or HTML, so please use the following resources:

- Newsgroups and instant messaging, as described on the course web page. Don't forget to help others with problems that you've already overcome.
- Office hours. Make sure to use the office hours made available by the instructor and the TAs.
- Handouts: we have prepared handouts on using Oracle in the TLAB and a high-level introduction to the Perl programming language.
- Videos. There are videos available on the course web site that will bring you up to speed on Linux.
- Web content. You will find many examples of Perl CGI and DBI programming on the web.

Hand-in

To hand in the project, you will do the following:

- ⇨ Get a copy of your extended blog up and running in your account on tlab-login.
- ⇨ Email the instructor and the TAs with a copy of all the files that you have modified or added. At minimum, this will be blog.pl and blog.sql. In the email, also supply a URL for your running blog and the root account name and password.

Extra Credit (30% Maximum)

You can gain extra credit by trying the following extensions:

Add image upload capability for replies and posts : Currently, all posts consist of text only. You will add the ability to attach one image to each post and you will update your message display (MessageQuery) to show the images. You will want to look at ~pdinda/HANDOUTS/blob* and ~pdinda/HANDOUTS/upload* for examples of how to handle binary data in Oracle and upload files in a CGI script. While you need only associate a single image with a message, you are required to store your images in a separate table, blog_images, that can associate multiple images with each message.

Security: Microblog is horrendously insecure because passwords are stored in cleartext in the database and cookie file, and are sent in cleartext over the network. Read about more secure approaches to session management and implement one of them.

Multipage display: As currently specified, every query or message summary displays all matches at once. This won't scale as the number of messages grows. Add support that splits the display across multiple pages.

Graphical tree: Use a tool like GraphViz (installed in ~pdinda/HANDOUTS) to generate graphical pictures of your message tree, ideally letting a user click on a node to view it in a separate window. See https://www.ecotonoha.com for an example of this taken to an extreme.