Homework 5: Web Application

bearTunes

Due @ 10:00 PM on Tuesday, May 1st

Overview

For this assignment, you'll be implementing portions of a database-backed web application using Ruby on Rails. If you've been ignoring the buzz, Ruby on Rails is the hot new web application framework that makes development (relatively) quick and (usually) painless. After spending the first two programming assignments hacking away at the dark underbelly of a DBMS, you now get to take a step back and put all of that functionality to use! That something is bearTunes, a web application used to catalog music.

Why Ruby on Rails?

The guiding principle of Ruby on Rails is "convention over configuration." Based on some simple assumptions about naming conventions, it does a lot of the gross administrative junk for you, making application development much faster and more bug-free. It lets you focus on the application logic without having to get bogged down in configuration issues. To further simplify things, the Rails architecture draws clean lines between the different functional pieces of your application architecture: Model, Controller, and View. (You'll find all of the files in the beartunes/app/* directories, labeled pretty much as expected).

> For a quick tutorial on how to use Ruby on Rails, check out this site:
> http://www.onlamp.com/pub/a/onlamp/2005/01/20/rails.html

Model

The Model consists of the classes representing your database tables. For each table in your database, there will be a corresponding Model class. You'll never have to directly communicate with your database: all actions are carried out through the Model classes. Think of them as providing an API that your Ruby application uses to access the database backend (in this case, Postgresql). You won't have to make any modifications to any of the Models for this project, but you should take a look at the files just to get familiar with them.

Controller

The Controller is where you specify what data should be retrieved from the database. The code in the Controller retrieves and manipulates data from the Model, possibly based on user input, and then gives it to the View so that it can be displayed. Here is where the

necessary SQL commands will be generated; you have the option of writing the queries yourself (using the command find_by_sql), or just using find to automatically generate the queries, based on the given parameters. Each action defined in the Controller classes will compute values (or sets of tuples) and store them in variables, and the corresponding View class can reference those variables and display them to the user. For example, to retrieve all album tuples and their associated artists, you would include the following code in your Controller:

```
@albums = Album.find(:all, :include => "artist")
```

Or, to achieve the same results:

```
@albums = find_by_sql("select * from albums A1, artists A2
where A1.artist_id = A2.id)
```

View

The View controls the display of your results in the web browser. In this case, that comes in the form of HTML sprinkled with embedded Ruby code. The design and layout of your web pages will all be specified here. There will be one .rhtml file for each action in the Controller classes. The format of these files is standard HTML, with all Ruby code enclosed with percent-signs: <% something like this %>. Within the Ruby code snippets, you can access the values passed in from the Controller by sticking a @ at the start of the variable name. So, for example, if the following variable assignment appeared in your Controller:

```
@artists = Artist.find(:all)
```

To print out the names of all of the artists, you would include the following code in your View:

```
<% @artists.each do |artist| %>
          <%= artist.name %><br>
<% end %>
```

Setup

Before you get to work, you have to set up your environment. You will be able to run this project on the following inst machines: pentagon, rhombus, or cube. For this project you'll be running your own copy of both the database and the webserver. First, make a new directory and `cd` into it, then call the following command:

```
setuphw5
```

When that finishes, you should see two new directories: `beartunes/` and `pgdata/`. To start up the application, move into the `beartunes/` directory and call the following command:

```
./RUN_SERVER
```

This script will start up Postgresql and the WEBrick, the webserver. **Don't run this script in the background!** If you need access to the terminal, open another one. When you're finished, simply press CTRL-C to close down both the database and the webserver.

Accessing bearTunes

When you start the webserver, you will notice the following block of text telling you which port the webserver is running on:

```
---------------------------------
+ RUNNING SERVER ON PORT: ##### +
---------------------------------
```

To access your application, simply point your favorite browser to `http://MACHINE:PORT`, where `PORT` is the one listed in the output to `RUN_SERVER` and `MACHINE` is one of the following three, depending on which one you're logged into:

```
pentagon.cs.berkeley.edu
rhombus.cs.berkeley.edu
cube.cs.berkeley.edu
```

Note that if you're developing this at home, you'll probably end up replacing `MACHINE` with `localhost`. You'll have to figure out the `PORT` yourself: if you can get it running at home, you will know this ☺.

Database schema

The following is the schema of the underlying database. In all relations, `id` is the primary key. No foreign key constraints are specified by the database management system (though the intended relationships are obvious by the naming convention). Instead, the constraints are specified in the Model (look at the files in `beartunes/app/model` to see how that's done).

```
artists(id, name, bio)
albums (id, artist_id, name, year, genre_id, rating)
tracks (id, album_id, name, number, length)
genres (id, name)
```

Filling the database

First things first: put some data in your database! One way to do this is to start up the application, open it in a web browser, and use the provided forms. If you use this approach, it will probably make your life easier to first insert at least one genre, then an artist, then an album, and then some tracks. You can also automatically fill the database with some pre-compiled data. Just `cd` into the `beartunes/` directory and call the following command:

```
./LOAD_DATA
```

Alternatively, you can provide your own data in the form of a SQL file, with the following command:

```
./LOAD_DATA SQLFILE
```

If you use this approach, you **must** make sure the following set of lines appears at the top of the file:

```
DELETE FROM tracks;
DELETE FROM albums;
DELETE FROM artists;
DELETE FROM genres;
```

And the following lines appear at the end of the file:

```
ALTER SEQUENCE genres_id_seq RESTART WITH #MAX+1#;
ALTER SEQUENCE artists_id_seq RESTART WITH #MAX+1#;
ALTER SEQUENCE albums_id_seq RESTART WITH #MAX+1#;
ALTER SEQUENCE tracks_id_seq RESTART WITH #MAX+1#;
```

Where `#MAX+1#` is the number after the highest `id` value in the relevant table. **If you don't do this, trying to insert new tuples through the web interface will not work!**

If you're a fan of iTunes, you can also import your iTunes library into the bearTunes database. Open iTunes and select `File > Export Library...` to export your iTunes library to an XML file. Then `cd` into the `beartunes/script/hw5/` directory and call the following command:

```
java iTunesLibraryLoader XMLFILE SQLFILE
```

Where `XMLFILE` is the file that was generated by iTunes and `SQLFILE` is the output file that will be created. **(Note: this took 2 days to convert my library, and it's a mere 16 GB, so be prepared to wait a while for it to complete!)** When this is finished, `cd` to the `beartunes/` directory and call the following command:

```
./LOAD_DATA SQLFILE
```

Your assignment

After taking some time to play around with bearTunes, you'll now get to add some functionality to it. The following are the classes you'll need to either create or extend, **which potentially includes changes to both the Controller and View**. You'll find all of the code you need in the `beartunes/app/` directory (use the existing files for reference). You should only need to make changes to the files in the `controllers/` and `views/` directories. For example, to modify the artist/list action, you would have to make changes to the files `controllers/artist` and `views/artist/`. You should be using the Controller classes to fetch all of the specified data from the database – the View classes are just responsible for displaying the data, not doing any computation or manipulation! **Unless otherwise specified, all orderings should be in ascending order**. Also, any pages we ask you to provide a link to already exist – unless otherwise specified, you should not need to create any additional files. Be sure to pay close attention to the **order in which the data is displayed**, and the **pages to which the data is linked!**

1. track/list

For each Track in the database, list the relevant Artist name and Track name. They should be ordered by (Track name, Artist name). When clicked on, the Artist name should lead to the artist/show page for that particular Artist. Clicking on the Track name should lead to **album/show page for the relevant Album**. Finally, somewhere on the page, put a link to the page to track/new.

2. album/show

For the specified Album, display the relevant Artist name and Album name. Clicking on the Artist name should lead to the artist/show page for that particular Artist. Also list the Genre name, year it was released, and rating. Clicking on the Genre name should lead to the genre/show page for that particular Genre. Finally, for all Tracks on the specified Album, list the name, number, and length. They should be ordered by number.

3. artist/show

For the specified Artist, display the relevant Artist name. For all Albums **by that artist**, list the Album name, year, rating, and Genre. Clicking on the Album name should lead to the album/show page for that particular Album, and clicking on the Genre name should lead to the genre/show page for that particular Genre. These albums should be ordered by (Album year, Album name). Along with the listed information, for each Album give the **total length**. Finally, somewhere on the page, display the average rating over all Albums by the given Artist.

4. genre/show

For all Albums **in the specified Genre**, list the relevant Artist name, Album name, and year. Clicking on the Album name should lead to the album/show page for that particular Album, and clicking on the Artist name should lead to the artist/show page for that particular Artist. These albums should be ordered by (Artist name, Album name, year).

5. album/best_albums << NEW ACTION >>

For all Albums with the **highest rating in the database**, list the relevant Artist name, Album name, and year. Clicking on the Album name should lead to the album/show page for that particular Album, and clicking on the Artist name should lead to the artist/show page for that particular Artist. These Albums should be ordered by (Artist name, Album name, year). **Note: you will need to create a new View file for this action, as one does not already exist.**

6. artist/top_ten_artists << NEW ACTION >>

For the Artist with the **top ten average Album ratings in the database**, list the relevant Artist name and average rating. Clicking on the Artist name should lead to the artist/show page for that particular Artist. These Artists should be ordered by (average Album rating, Artist name). There should only be ten Artists displayed; in the case of a tie that causes you to have to choose, it's pretty much arbitrary. **Note: you will need to create a new View file for this action, as one does not already exist.**

Submission

Submit the following files from within the `beartunes/app/` directory:

```
controllers/album_controller.rb
controllers/artist_controller.rb
controllers/genre_controller.rb
controllers/track_controller.rb
views/album/best_albums.rhtml
views/album/show.rhtml
views/artist/show.rhtml
views/artist/top_ten_artists.rhtml
views/genre/show.html
views/track/list.rhtml
```

As usual, submit a `README` file that contains both partner's names and logins, and a description of any bugs you're aware of but weren't able to fix. Good luck!