Name of Application: Battle of the Bands
Author: Scott Pullen

Access Information:

- URL: http://cscie253.dce.harvard.edu/~spullen/final_project
- User: user/253user
- Administrator: admin/253admin
- Recommended Browser: Chrome (Firefox should work though, don't use IE)

Problem Statement:

The goal of this application is to provide a nomination process for bands to be given a slot for a "Battle of the Bands" show. Shows can happen multiple times in a calendar year, but usually are only scheduled twice a year. Each show has a designated voting period where users are allowed to vote for bands that have entered the show. Show's voting period should not overlap. Shows are performed at venues.

Users are able to make a profile for their band. A user that creates a band profile is called a "maintainer." Band profiles include the basic information about the band (biography, members and website). The band profile also contains a list of members and what instrument they play. Maintainers are able to upload sample tracks that users can download and listen to. Users can maintain many bands.

For each show there is a nomination process, where users get to vote for bands that have entered. Bands have to enter into the nomination process in order to get user votes. Bands can only enter once per nomination process for a given show. Users can vote for bands that have entered the nomination process. Users can only vote during specified voting periods. After the voting period has ended, the top five bands with the highest average votes will be given a slot to play the Battle of the Bands show.

Database Implementation:

Note: Included with this report is a master SQL file spullenfp.sql which contains everything required to set up the database including table definitions, procedures and triggers along with the associated tests. As a convenience, I've created two additional files containing the procedures and triggers along with their tests, as the master file is pretty big. They are spullenfp_triggers.sql and spullenfp_procs.sql.
I should also note that all table constraints have been tested. Triggers and Stored Procedures have also been tested. These can be seen in the master SQL file spullenfp.sql.

Stored Procedures:

I have two stored procedures, add_band_member and add_music_file. The goal of both procedures is to find the first available id number for new tbMembers or tbMusicFiles entries then insert the new entry. The input parameters for add_band_member are the bandId, name and instrument, and an output parameter for the new member id. The parameters for add_music_file are bandId, title and file_path, and an output parameter that returns the new music file id. If either method finds that the max number of ids has been assigned for a band -1 is returned. One interesting issue I ran into while developing these procedures was that the query needed to find the next available does not work if multiple rows are returned. In order to solve this problem I needed to limit the number of rows returned. To achieve this I did a subquery on the query to that finds the available ids, where the query to find the available ids returned the row number. I then had to place a restriction on the row number such that it was greater than 0 but less than or equal to 1. Doing this only returns one row (if available) so I could assign into a variable I declared in my procedure.

Note: I wasn't able to get this to work through coldfusion code, see Known Issues section.

Triggers:

I have implemented seven triggers. I have one trigger than prevents a show from being inserted when the voting start date or voting end date overlaps with another show. The overlap check trigger is named TR_tbShows_votingOverlap. The other six provide auto-number functionality. They are named TR_tbUsers_in, TR_tbVenues_in, TR_tbShows_in, TR_tbBands_in, TR_tbEntries_in and TR_tbVotes_in. When an insert happens on each respective table the next value from the sequence is retrieved and the value is set.

Transactions:

I also implemented a couple transactions in the coldfusion code. I have a transaction when a new band member or music file for a band. The transaction is wrapped around a request to get the next member id or music file id for a band then insert the new entry with the requested value. Although this currently would not be a problem since only one person is allowed to create new band members and music files, in the future this could change and any one in the band could have access to manage members and music files. In the future case if the transaction wasn't there, there could be problems with requesting the new id.

Miscellaneous PL/SQL:

I also wrote a small PL/SQL piece to initialize tbMemberIds and tbMusicFileIds. All it does it loop through the numbers 1 to 90 and insert them into the respective table.

Relevant Queries:

The list contains some of interesting queries used for this application. This is not a complete list (I did not include any INSERT/UPDATE/DELETE queries as they are boilerplate).

getUpcomingShows:

This query gets the first four upcoming shows.

```
SELECT show_id, show_date, voting_start_date, voting_end_date, venue_id, name,
street, city, state, zip
  FROM (
    SELECT show_id, show_date, voting_start_date, voting_end_date, venue_id, name,
street, city, state, zip, rownum r
    FROM (
      SELECT tbShows.show_id,
             tbShows.show_date,
             tbShows.voting_start_date,
             tbShows.voting_end_date,
             tbVenues.venue_id,
             tbVenues.name,
             tbVenues.street,
             tbVenues.city,
             tbVenues.state,
             tbVenues.zip
      FROM tbShows
      INNER JOIN tbVenues ON tbVenues.venue_id = tbShows.venue_id
      WHERE tbShows.show_date >= sysdate
      ORDER BY show_date ASC
    )
    WHERE rownum <= 4
  )
  WHERE r >= 1
```

getVenues:

```
SELECT *
FROM tbVenues
ORDER BY name
```

getShows:

```
SELECT tbShows.show_id,
       tbShows.show_date,
       tbShows.voting_start_date,
       tbShows.voting_end_date,
       tbVenues.venue_id,
       tbVenues.name,
       tbVenues.street,
       tbVenues.city,
       tbVenues.state,
       tbVenues.zip
FROM tbShows
INNER JOIN tbVenues ON tbVenues.venue_id = tbShows.venue_id
ORDER BY show_date DESC
```

getShowEntries:

This query has two different versions. The first version is if the user is logged in. In this case I get their rating. If the user is not logged in it just gets the current state of the entry and omits retrieving the user rating portion (as the user is not logged in).

If user is logged in:

```
SELECT tbEntries.entry_id,
       tbEntries.show_id,
       tbBands.band_id,
       tbBands.name,
       averageEntryRatingView.average_rating,
       tbVotes.vote_id,
       tbVotes.rating,
       tbVotes.user_id
FROM tbEntries
INNER JOIN tbBands ON tbBands.band_id = tbEntries.band_id
LEFT OUTER JOIN averageEntryRatingView ON averageEntryRatingView.entry_id =
      tbEntries.entry_id
LEFT OUTER JOIN tbVotes ON tbVotes.entry_id = tbEntries.entry_id AND
      tbVotes.user_id = #currentUser.user_id#
WHERE tbEntries.show_id =
            <cfqueryparam cfsqltype="CF_SQL_NUMERIC" value="#URL.show_id#">
ORDER BY case when average_rating is null then 1 else 0 end, average_rating DESC
```

If user is logged out:

```
SELECT tbEntries.entry_id,
       tbEntries.show_id,
       tbBands.band_id,
       tbBands.name,
       averageEntryRatingView.average_rating
FROM tbEntries
INNER JOIN tbBands ON tbBands.band_id = tbEntries.band_id
LEFT OUTER JOIN averageEntryRatingView ON averageEntryRatingView.entry_id =
      tbEntries.entry_id
WHERE show_id = <cfqueryparam cfsqltype="CF_SQL_NUMERIC" value="#URL.show_id#">
ORDER BY case when average_rating is null then 1 else 0 end, average_rating DESC
```

getBands:

This query has two forms. It conditionally restricts the result set by the band name. The band name can be entered into the band search form which is in the navigation.

```
SELECT *
FROM tbBands
<cfif URL.name NEQ -1>
WHERE upper(name) LIKE
  upper(trim(<cfqueryparam cfsqltype="CF_SQL_VARCHAR" value="%#URL.name#%">))
</cfif>
ORDER BY name
```

getBandCurrentEntries:

This is query is similar to the getShowEntries query. If the user is logged in it retrieves the rating that

the user gave otherwise it just retrieves the list.

User logged in:

```
SELECT tbEntries.entry_id,
       tbEntries.show_id,
       tbShows.voting_start_date,
       tbShows.voting_end_date,
       tbShows.show_date,
       tbVenues.name,
       tbVenues.street,
       tbVenues.city,
       tbVenues.state,
       tbVenues.zip,
       averageEntryRatingView.average_rating,
       tbVotes.vote_id,
       tbVotes.rating,
       tbVotes.user_id
FROM tbEntries
INNER JOIN tbShows ON tbShows.show_id = tbEntries.show_id
INNER JOIN tbVenues ON tbVenues.venue_id = tbShows.venue_id
LEFT OUTER JOIN averageEntryRatingView ON averageEntryRatingView.entry_id =
      tbEntries.entry_id
LEFT OUTER JOIN tbVotes ON tbVotes.entry_id = tbEntries.entry_id AND
      tbVotes.user_id = #currentUser.user_id#
WHERE tbEntries.band_id = <cfqueryparam cfsqltype="CF_SQL_NUMERIC"
      value="#URL.band_id#"> AND
            tbShows.voting_start_date <= sysdate AND
            tbShows.voting_end_date >= sysdate
ORDER BY tbShows.show_date DESC
```

If no user is logged in:

```
SELECT tbEntries.entry_id,
       tbEntries.show_id,
       tbShows.voting_start_date,
       tbShows.voting_end_date,
       tbShows.show_date,
       tbVenues.name,
       tbVenues.name,
       tbVenues.street,
       tbVenues.city,
       tbVenues.state,
       tbVenues.zip,
       averageEntryRatingView.average_rating
FROM tbEntries
INNER JOIN tbShows ON tbShows.show_id = tbEntries.show_id
INNER JOIN tbVenues ON tbVenues.venue_id = tbShows.venue_id
LEFT OUTER JOIN averageEntryRatingView ON averageEntryRatingView.entry_id =
      tbEntries.entry_id
WHERE tbEntries.band_id = <cfqueryparam cfsqltype="CF_SQL_NUMERIC"
            value="#URL.band_id#"> AND
      tbShows.voting_start_date <= sysdate AND
```

```
      tbShows.voting_end_date >= sysdate
ORDER BY tbShows.show_date DESC
```

getBandPastEntries:

Again this query is different based on whether the user is logged in or not. If the user is logged in then the query retrieves the user's rating. If the user is not logged in then it just retrieves the list.

```
SELECT tbEntries.entry_id,
       tbEntries.show_id,
       tbShows.voting_start_date,
       tbShows.voting_end_date,
       tbShows.show_date,
       tbVenues.name,
       tbVenues.name,
       tbVenues.street,
       tbVenues.city,
       tbVenues.state,
       tbVenues.zip,
       averageEntryRatingView.average_rating
FROM tbEntries
INNER JOIN tbShows ON tbShows.show_id = tbEntries.show_id
INNER JOIN tbVenues ON tbVenues.venue_id = tbShows.venue_id
LEFT OUTER JOIN averageEntryRatingView ON averageEntryRatingView.entry_id =
      tbEntries.entry_id
WHERE tbEntries.band_id = <cfqueryparam cfsqltype="CF_SQL_NUMERIC"
            value="#URL.band_id#"> AND
      tbShows.voting_end_date < sysdate
ORDER BY tbShows.show_date DESC
```

currentUser:

```
SELECT *
FROM tbUsers
WHERE user_id = <cfqueryparam cfsqltype="CF_SQL_VARCHAR" value="#Session.userId#">
```

Forms/Reports:

Forms:

- User Registration
- User Login
- User Swap Role (change user to admin or take away admin privileges)
- Venue Create/Update
- Show Create/Update/Delete
- Band Create/Update
- Band Member Create/Update/Delete
- Band Music File Create/Update/Delete
- Band Search

Reports:

- Show Entry Standings
- Band Current Show Entry Standings
- Band Past Show Standings

Special Features:

User sessions:

Users are allowed to create accounts. Once they create an account they are able to login or log out of the site. A user can either be a regular user or an administrator. If a user is logged in they are able to create and manage bands, enter bands into shows and vote for bands entrants for shows. If a user is logged in and also an administrator they are allowed to create venues, shows and are also allowed to grants other users administrator privileges. Administrators can also do everything that a regular user can do.

Users that are not logged in get a “read-only” view of the site. They can see bands, shows and venues, but they can't create bands or vote.

Band Search:

A user can search for a band. On the navigation of the site there is a input form. A user can enter the name or partial name of the band they are looking for and it will find them. For this I piggy backed on the bands list page. What happens is the query that finds the bands will conditionally add a restriction to the query if the form value is present.

File Upload:

I implemented a file uploader. This is used to upload music files for bands. The music file assets are stored in folders that are designated by the band id. If the directory for the band does not exist it is created. Each file that is uploaded is given a unique filename. The file name is what is stored in the

database and the path to the file is reconstructed when presented to the user for download.

Testing:

Everything done on the database has been tested. For tables this includes the testing of Entity Integrity rules, Referential Integrity rules and column constraints. For stored procedures and triggers tests were written to verify that they worked properly. The tests can be seen in the master SQL file (*spullenfp.sql*).

Form Validation:

I felt that the form validation methods that coldfusion provided were really weak, none of the examples presented seemed user friendly. So I used jQuery and a validation plugin for jQuery to handle validation for all forms. This works similar to the built in coldfusion javascript validation but it gives the developer much more control over how things are validated.

Exception Handling:

I used coldfusion exception handling to prevent page crashes to the user. They are implemented on pages that take values from the user.

Flash Messages:

I implemented a flash (Session) message scheme. If an error does occur that I want to provide feedback I simply set the Session alert variable and the page will display it and then deleted it. Same goes for success messages (like when a entry is successfully created a notice will be displayed to the user).

Known Issues:

- Band member update doesn't validate, it has something to do with having the create form on the same page and the validation code I used can't distinguish between the different forms. The

database does catch the issues, and as a work around I present this as an alert after the form is submitted.

- The music file download sometimes doesn't work. This might be due to permissions issues.
- Music file delete doesn't physically delete the file. This might be due to permissions issues.
- I tried to execute the store procedures via coldfusion and was running into errors. I was getting a "ORA-01461: can bind a LONG value only for insert into a LONG column" error. I was able to successfully execute the procedure in sqlplus without any errors.

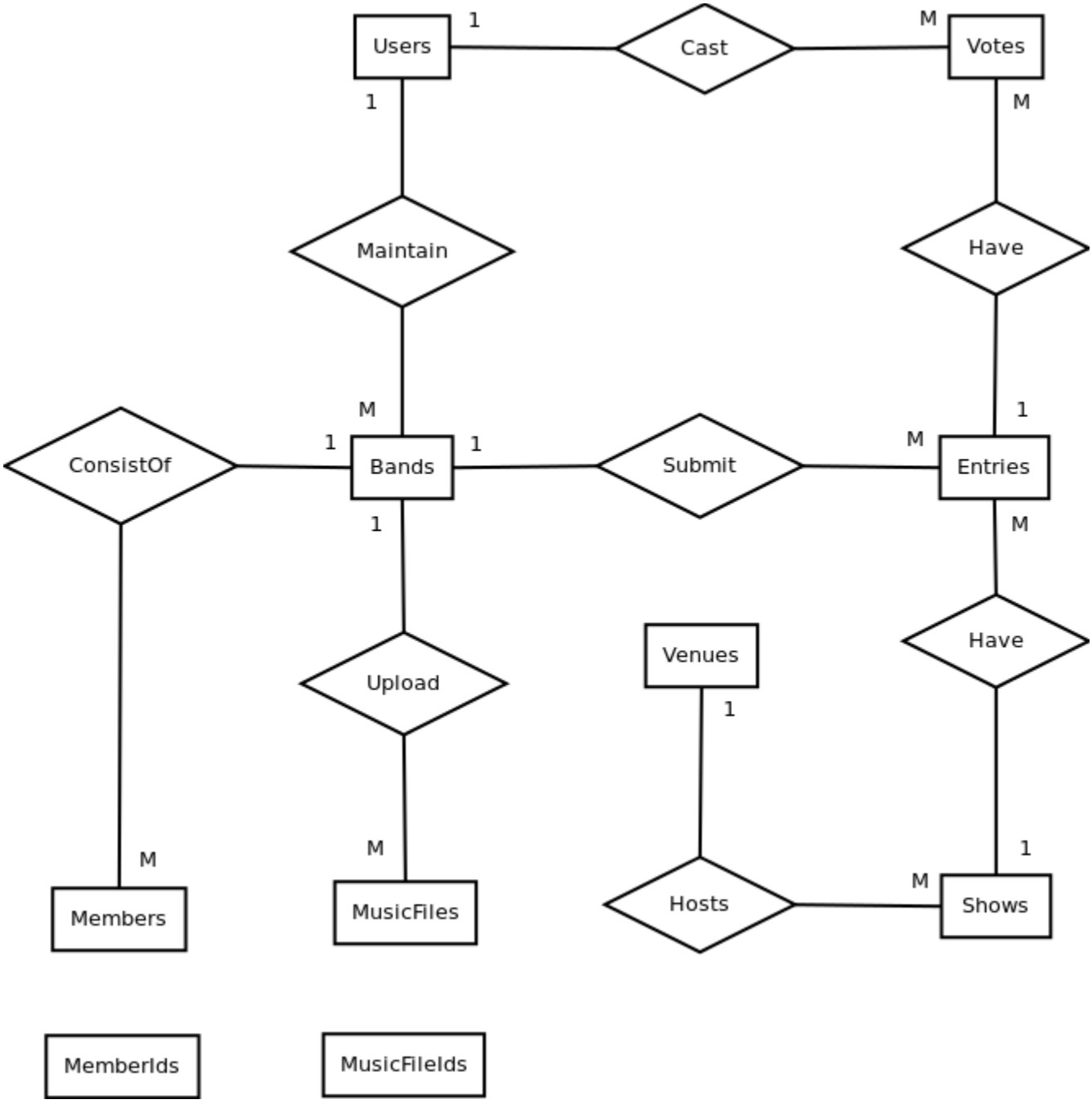
Sources:

- jQuery 1.8.3, http://jquery.com/
  - Dependency for jQuery Validation Plugin
- jQuery Validation 1.10.0, http://bassistance.de/jquery-plugins/jquery-plugin-validation/
  - Used for form validations.
- Twitter Bootstrap 2.2.1, http://twitter.github.com/bootstrap/
  - Used for the style of the site

Original Work:

I hereby certify that this project was prepared especially for this course, and that this or a similar version has not been submitted to any other course.

ERD Diagrams:



Record Diagrams:

USERS:

user_id	username	email	pwd	first_name	last_name	is_admin
PK						
00001	User1	test@test.com	abd37189109caef	John	Smith	1
00029	User2	test2@test.com	bdbf128971666a3	Jane	Doe	0
00010	User3	test3@test.com	765236723273723	Steve	Jackson	0

VENUES:

venue_id	name	street	city	state	zip	website	phone
PK							
01	House of Blues	15 Lansdowne St.	Boston	MA	02215	http://www.houseofblues.com	888-693-2345

SHOWS:

show_id	venue_id	show_date	created_at	voting_start_date	voting_end_date
PK					
	FK				
0021	01	10/20/2011	05/01/2011	07/01/2011	09/20/2011
0022	01	06/05/2012	11/01/2011	12/01/2011	05/04/2012

BANDS:

band_id	maintainer_id	name	biography	website
PK				
	FK			
0001	00001	RatTrap	We got started...	www.rat-trap.com
0002	00002	Goldrush	Heavy-alt rock group with...	www.goldrush.com

MEMBERS:

band_id	member_id	name	instrument
PK			
FK			
0001	01	John Smith	Guitar
0001	02	Joe Stone	Drums
0002	01	Jane Doe	Vocals
0002	02	Tom D.	Guitar

MUSICFILES:

band_id	music_file_id	title	file_path	upload_date
PK				
FK				
0001	01	A song about nothin'	./assets/0001/01.mp3	10/01/2011
0001	02	Coo.	./assets/0001/02.mp3	12/21/2011
0002	01	Shinin' Gold	./assets/0002/01.mp3	08/12/2009

ENTRIES:

entry_id	band_id	show_id	entry_date
PK			
	FK	**FK**	
00000001	0001	0021	05/20/2011
00000002	0002	0021	06/05/2011

VOTES:

vote_id	entry_id	user_id	rating	date_voted
PK				
	FK	**FK**		
00000001	00000001	00010	4	07/21/2011
00000002	00000002	00029	1	08/02/2011

I have a view called *averageEntryRatingView* used to calculate the average rating for an entry.

I have two helper tables to find the first available member_id and music_file_id for tbMembers and tbMusicFiles. Both tables just have values from 1 to 90.