One Million Songs SQL Project

Walt Panfil and Trinity Merrell

CSSE 333, Hewner

Front-end

AJAX

Middle

Ruby

Back-end

MySQL

Data source:

* Million song dataset
* <http://labrosa.ee.columbia.edu/millionsong/>
* <http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/>

Table sketch

* LISTENER (Last.fm + EchoNest data, hopefully combined)
* SONG
* LISTENS\_TO (listener, song, play count)
* ARTIST
* GENRE
* TAGS (foreign key to song, tag word, maybe foreign key to tagging listener)
* FAVORITE\_SONGS (Associated with user, defined by user input only)

Potential Features (From most to least ambitious)

* Song recommendations based on input
* Plays all songs with same key & similar tempo simultaneously
* Songs by year (Loudness, Genre\*, Tracks, etc.)
  + \*Perhaps also Yahoo Ratings to analyze genre ratings by year
* Song/Genre/Artist popularity by demographics
* Percentages of genre by year

**1.1 A Database**

* Must be SQL based
* Must have at least 7 tables, and these tables should be materially interrelated
* Must be designed by you, and you must submit at ER diagram to me within the first 3 weeks of the project. I will be checking to make sure your design is good.
* Your database must correctly use all the main DB features we talk about in class - triggers, stored procedure etc.

**1.2 A web-backend**

* I recommend a language like ruby, python, etc. but it's up to you
* Your can **not** use an object-relational mapping layer (i.e. ruby on rails, etc.) - you must write your queries directly
* For ease of setup, I personally recommend you setup a unix virtual machine

**1.3 An import script/mechanism**

* Could be importing a datafile or querying an API (that last one can get interesting though)
* Must get your data from some real external source.
* Must be able to setup the database from scratch
* Must be able to be run more than once
* For B work, there can be some manual steps in the process
* For A work, should work automatically from the raw source

**1.4 A front end**

* Must be interactive, allow access
* For B work, does not have to be pretty (think Banner)
* For B work, must fulfill a variety of use cases
* For A work, must be pretty and nice
* For A work, must be "complete"
* For A work, must use AJAX and Javascript