# Music Industry Query System

## Team Members

- Ankush Israney
- Jaidev Ramakrishna

## Application URL

http://resin.cci.drexel.edu/~avi26/MusicIndustry

## Project Description

This application provides a high level view of the music industry from the business side. It does contain metadata about the musical aspects, but the entities and relationships, and the queries that the application is capable of running, have been formulated so that one may examine the business side of the music industry. One can use this type of system to perform analytics, mine information and query facts pertaining to the functioning of the industry.

The front end is JSP files that dynamically generate HTML data to display. The back end is written in Java. It consists of -

- An Analyst class and its associated servlet for generating the interface.
- A Query class and its associated servlet for building and running SQL queries.
- One class for each table in our database, each of which contains entity-specific methods.

## Entities & Schema Translation

The schema contains several entities, all of which have their own table. Due to the nature and combination of the constraints on the relationship sets, only Collaborates requires its own table. The other relationships - Performs, Manages, Contract, Composes, Releases, Contains, and Records - are encoded in entity sets. Hence, we have the following list of tables –

- Artist
- Song
- Composer
- Album
- Producer
- Record_Label
- Manager
- Collaborates

Where there is a key participation, we were able to include the fields of the relationship in one of the entities. The collaborates relationship has neither a key nor a participation constraint. Hence, it warrants its own table.

It should be noted that certain modifications to the original schema were necessary since we would need assertions to implement them, which have not been covered in class. We have several relationships in our ER model where one of the entities has a key + participation constraint and the other entity has only a participation constraint. We have enforced this rule by importing the primary key of the entity with only a participation constraint into the relation definition of the other entity as a foreign key. The participation constraint on the side without a key constraint is implicitly enforced during data acquisition without any additional Modifications/Insertions/Deletions in the database, and without any triggers or assertions for these rules.

## Business Rules

- An artist performs on at least one song; each song can have exactly one performing artist.
- Songs can be collaborated upon by any number of artists and artists can collaborate on any number of songs.
- Each song may have at most one composer.
- A song is contained on exactly one album; an album has at least one song.
- A producer records at least one album; an album has exactly one producer.
- An album is released by exactly one Record Label; one label releases at least one album.
- A manager manages at least one artist; an artist is managed by exactly one manager.
- An artist has exactly one contract with a Record Label.

## Data Acquisition

We acquired our data by manually scraping Wikipedia and other sources. We have attempted to make it as realistic as possible. It can be seen that our music metadata (concerning Artist, Song, and Album) is factually accurate. We have taken a few liberties with business data (concerning Record Label, Manager, and Producer) in order to present a complete and coherent data set. The nature of this project necessitates a densely connected and balanced set of data in order to be able to return meaningful answers.

We initially loaded the data into MS Excel sheets, exported them to CSV, and then wrote methods in Java to convert them to SQL queries. This was relatively trivial, since the CSV format allowed us to use commas as delimiters for separating fields while parsing the file line by line. The schema.sql file contains all add, drop, and create statements. It can be run in order to fully load data into Postgres.

## User Interface

The UI is a highly minimalistic web interface – using very basic HTML and JSP. All CSS embellishments are inline. The workflow presented to the user is simple and streamlined. The user first selects the entity that they wish to examine, is able to search or list tuples, and then select from among entity-specific queries.

- The user interface uses a landing page called index.jsp, from where the analyst interface can be launched.

- The analyst interface is housed in initial.jsp, and it offers a list of entites for the user to select.
- Once the user has selected an entity, the entity_action.jsp page offers the option of searching for a specific entity using primary key (Collaborates uses Artist ID), or of listing all tuples in the entity set.
- From here, each entity has its own jsp page for dispaying content and selecting facts to query.

We do not provide an admin interface to add and remove data, since the highly connected nature of our data, with its complex constraint structure, would make such operations either too unwieldy to easily perform, or would result in significant degradation in the quality of the data due to cascading modifications. Since our data is more or less static, we are not using on-delete rules in the table definitions.

## Other Details

The code requires at least Java 7 to compile and run. It will fail to compile on previous versions.

The zip archive contains our source code. We have provided java and jsp source code files in two folders. We have also provided a WEB-INF folder that may be placed directly on resin, if it is to be deployed and executed.

The schema.sql contains all instructions needed to recreate our database. It must be run manually before running the application.

We have tested our application on multiple web browsers on various operating systems. The content and layout seem to be reproduced without any glitches even on phone browsers, but it is best viewed on Chrome in 1920x1080 resolution.

The ER diagram is contained in a separate PDF.