CS4420/5420 Database Project
Spring 2022
NOTE: ERD Due before Project

This project involves the construction of a music chart database using the flat-table spreadsheet data as handed out in class. The objective is to design and implement a Third-Normal-Form (3NF) database in SQL, using good relational modeling based on the given data and the business rules provided here. The designed tables are then to be populated with the imported data, and then used to produce the results for the queries listed here. Grading criteria will be posted on the website.

Deliverables:
For this project, turn in the following components, neatly arranged in a two-pocket folder (no 3-ring binders PLEASE!):

1) ERD- a **neat, machine produced,** labeled ERD showing graphically how your tables are related

2) TABLES- the SQL table descriptions for all tables in the 3NF database

3) INIT SQL- the SQL source used to populate the tables from the original table

4) QUERY SQL- the SQL query source AND results for each of the 10 required queries. These must be neatly compiled in order, including a) the query number (1..10), b) the original question being asked, c) the SQL query source, and d) the resulting table output. The queries have been designed, for the most part, to have their results fit on a single page (with a couple of exceptions). **Readability for the grader** is the goal.

5) An approximately 1.5 to 2 page write-up describing the design decisions made for each original column from the initial table, and why or why not separate tables were used. See Business Rule #10 below.

Arrange these in the above order, divided between the two folder pockets, CLEARLY IDENTIFYING EACH COMPONENT so that it is easy to find and grade.

NOTE: The ERD is not to be handwritten, and will be due 1 week after this assignment is handed out. To avoid redoing the work, the student should not complete the implementation until the ERD has been graded and approved.

The Original 'Database':
The data was apparently collected as a collaborative internet effort to capture the music chart data for all songs reaching the 'Hot-100' charts. The original file contained data for more years, and contained several additional columns of data of interest only to the most devoted music industry fan. Originally an XLS file, this data was exported to a Comma Separated Value (CSV) form, and then 'massaged' by a custom text-processing program to search for inconsistencies and try to fix them, while eliminating less-important columns.

It is evident that the original data was not well designed from a database standpoint... a spreadsheet is **not** a database... and this one in particular is not even in 1NF. The goal

is to rearrange this data into a well-designed 3NF database containing multiple tables, so that the queries included in this document can be easily constructed.

The Basic Approach:
Because raw MySQL tables are not easily shareable among different machines, the process for this project will start by using the provided MySQL code to import the CSV data into one big, ugly,non-3NF SQL table. This table represents a direct conversion of the spreadsheet, as is, into a MySQL table, with no improvements. Next, the *well-designed* tables necessary to implement the 3NF database design will be created using SQL code to establish the needed, empty tables corresponding to the ERD. Third, the contents of the big table, plus some SQL code, will be used to populate the new and improved database tables, establishing the correct foreign-key links among the tables, again according to the ERD. Finally, the completed 3NF database will be used to run several SQL queries to answer questions about the now easily-accessible data.

Business Rules:
Use the following rules when designing the 3NF database. **Read these carefully; failure to conform to these could make the resulting database more complicated and less accessible for the assigned queries.** Follow instructions.

1) It is possible to use the **PREFIX** attribute as a PK, as it is guaranteed to be unique and non-null. An autoincrement PK is probably a better, more efficient choice. This attribute, however, should be kept in the database, as it contains some readable info, which might be useful in debugging the db links later on.

2) The following attributes/fields must be omitted from the original data, as they are minimally important and will not be used for this project: **SOURCE, SYMBOL, VERIFIED, GENRE**

3) Do not attempt to atomicize (like splitting into first-name, last-name) the **ARTIST** or **ARTISTINVERTED** attributes, just use them as is, and associate them with an appropriate PK. There is no consistent algorithm that can do this without A.I. or human intervention.

4) Artists and Writers are both People, and, since they might be a single person taking part in two different relationships to a song, they should be contained in a single PEOPLE table, with appropriate artificial PK. Note that it is NOT necessary or desirable to flag their function as either Artist or Writer, as this can be found from their relationships to the songs. Think about the M:N relationships involved.

5) The **TRACKS** are the central table for the database, with a Track equivalent to a Song in normal terminology. It makes sense to start the design process with this table, sorting out what is part of the song entity, and what needs to be in its own table because it is a separate entity.

6) The **TRACKTITLE** is not guaranteed unique, as it is the title of the song, and there are multiple but different songs having the same title. An Artist can perform many Tracks. For the purposes of this project, and because no atomicity will be established, a single Track, by definition, can be performed only by one 'artist' entity. For this project, a duet by two artists will be stored as a single entry in the people table– there is no way to break up the artists without lots of manual labor. It is not possible to determine if two songs with the same title are the same song directly; however, this could be determined by their Writer information.

7) A Person can write many songs, and a song can be written by many people, some of whom may also be artists. It is not necessary to distinguish whether someone is an artist, writer, or both when storing People; one could query the db to discover that, however, by seeing if they performed and/or wrote any songs. Also, due to atomicity issues, a named group will be treated as a single Person, and no attempt will be made to identify the people within a group for this project. A group IS a 'People' entity.

8) WrittenBy1..8 is a repeating group that allows up to 8 people to be credited with writing a song. Week1..75 is a repeating group allowing a track to be on the chart for up to 75 weeks. These need to be removed and stored in a manner acceptable to the Relational DB Model.

9) Songs may enter the chart late in their entry-date year, and not peak to their highest position until the following year. This means that the song year may not be the same as the year the song entered, as it is really the 'year the song peaked'.

10) The handling of any other data columns, and identification of apparent transitive dependencies is left to each students judgment, with the explanation for the decisions made for each attribute to be included in the write-up described in part 5 of the Deliverables, above.

Required Queries:

A list of the queries which will be required to be turned in will be provided on the ERD due date. Graduate and undergraduate students will have slightly different queries to be processed. The results are to be submitted as described in the Deliverables section.