CSE 412

Team Project, Phase II

Table of Contents

Overview

This project aimed to demonstrate various queries on a database filled with synthetic data. A schema was defined, by which the tables were created, and a script was used to generate the data. The database contains 970,531 rows over 11 tables. Queries were defined in a manner to be interesting, applicable, and nontrivial.

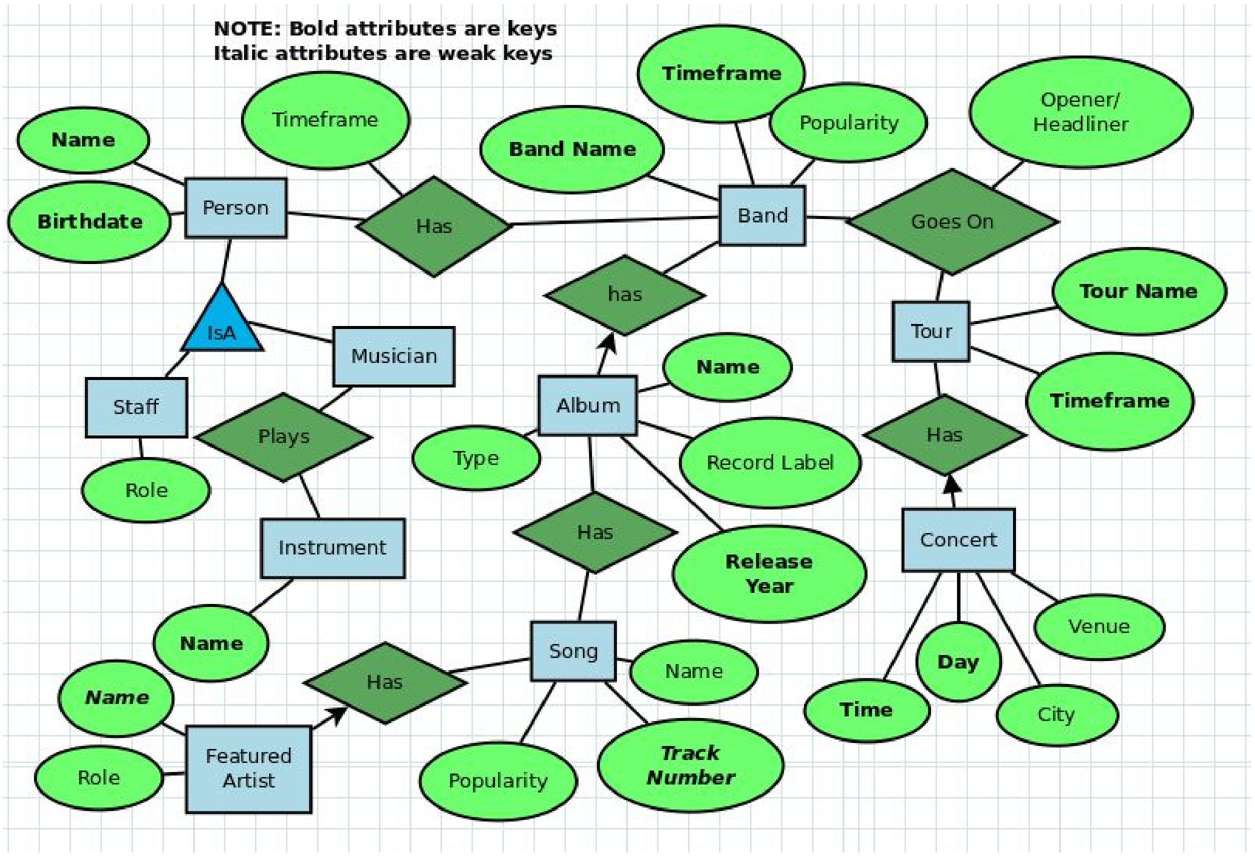
The first part of this project was to define a database. A schema was defined to do this, based on the ER diagram (shown later in this document). This schema was refined and filled out with constraints.

The second part of the project was to generate data for the tables. As previously noted, a script was used for this. A shell script called the other two scripts; one in SQL, and one in python. The SQL script creates the database and the tables. The aim of the python script was to provide semi-reasonable random data for all the tables, while making sure that this same data set could be generated again as needed. A random seed was used to accomplish this. The script was given lists of words (nouns, adjectives, first/last names, etc) to generate data from; for instance, a band name can be an adjective + noun pair. It then makes a few thousand bands, and for each band it makes band members, albums, etc; for each album, it makes songs; and so on. This ensures that the data that the script generates is consistent. All data is inserted directly into the database.

The third part of the project was to write the queries. These were created so as to yield interesting data which could not be obtained trivially. They explore multiple aspects of the database, from popularity to number of staff to release years.

A front-end was also built for this. A website hosts a page at which users can query the database, and a form is available for a user to easily add data to the database. See http://cse412.michaeljscott.net/ .

ER Diagram



This was the original diagram. The following points outline the changes.

- For *Band*, the ***Timeframe*** attribute was changed to a pair of DATEs, and *Popularity* was renamed to *band_billboard_rating*.
- *Person* uses foreign keys ***band_name*** and ***band_start_date***.
- The subclasses of *Person* were reworked; now the two entities are *staff* and *band_member*. Both have from/to dates, and foreign keys to person and band. *Staff* has a role.
- *Plays* is now a table; it links people to instruments.
- *Featured_artist* is now linked to both *song* and *band*.
- Other entities have various foreign keys.
- Indices were also added to some of the entities.

ER to SQL

Translation From Diagram to Schema

For the most part, the diagram was translated directly into SQL. A consistent naming scheme was used for attributes, and logical data types were used. Foreign keys were added in many of the tables, because many of them depend on the band. Actions to be taken for DELETE or UPDATE actions were also defined, as well as indices on some of the tables.

A good example of a table is as follows:

```
CREATE TABLE person
  (
     band_name VARCHAR (50),
     band_start_date DATE,
     person_name VARCHAR (50),
     person_birthdate DATE,
     INDEX (person_name, person_birthdate),
     PRIMARY KEY (band_name, band_start_date, person_name,
person_birthdate),
     FOREIGN KEY (band_name, band_start_date) REFERENCES
band (band_name, band_start_date)
     ON DELETE CASCADE
     ON UPDATE CASCADE
  );
```

Here we create a table for a *Person*. A Person has attributes of *band_name*, *band_start_date*, *person_name*, and *person_birthdate*. All four attributes in this example table are primary keys; this is not the case in other tables. *Band_name* and *band_start date* are foreign keys to band. The other two attributes are foreign keys for other tables.

We can also see that the update and delete actions are specified as cascading. This helps keep data synchronized across tables when information is changed. Cascading deletes and updates were down because the tables below were dependant. Setting null was also considered, but this was decided against for a couple reasons. First of all, this makes sure that the database only contains real and current data. If fields could be set null, then data could quickly become out of date, representing a previous state of the database. Second, this eliminates rows of the database that only have partial information and protects against hanging links. It would not be useful to have a row of data with only a couple fields filled.

The last element of this example of particular note is the index. The index is used to speed up queries on *person* by telling the database to sort by the specified fields.

Check Constraints and Assertions

- For the band, band member, and tour tables, the end date must be after the beginning date. Additionally, a concert must occur on or after the starting date of its tour.
- All relations on persons in the database, including featured artists and staff, cannot have numeric characters in their names.
- An instrument cannot be null.
- The release year of an album or song must during or after the first year of the band's activity.
- If an album has one song, it is called a single, and given the integer value of 0 in the album_type field. If an album has less than 6 songs, it is an EP, and given an integer value of 1 in the album_type field. Otherwise, the album is an LP, and given an integer value of 2 in the album_type field.
- If a featured artist is inserted who is a member of the releasing band at the time of the album's release, an error is thrown.

SQL Queries

Phase2Q1Join1.sql

All bassoon players on tour after the start of 2000

This query shows the functionality of being able to view the instrument of a particular musician as well as use their tour dates to provide a timeline for that musician and their use of said instrument. This functionality also allows for easy collection of statistics, for example if a user wanted to know in this case if there were more bassoon players on tour before or after 2000, given different parameters the database is functionally capable of giving that information.

Phase2Q2Join2.sql

All people in bands where the band had a top 100 hit

This query shows the functionality between the members of a band and their band's success tethered to their records' hits. This query shows that there is a relation between the musicians and their bands as well as a relation between the band and everything that the band produces, and the reception of that production, all of this can be tied back to the musician for data collection.

Phase2Q3Nested.sql

The name of the bands with the oldest band members

This query reinforces that there is a relationship between the band members and the bands, also that functionally extends to the information about the band members as well. In this case it is shown that the bands with the oldest band members is trackable, the same functionality would extend to other queries which take into account information about band members, for example this database could also track the name of the bands with the most guitarists.

Phase2Q4GroupBy1.sql

All bands that have more than 10 staff members

This query is more simplistic than some of the others; it establishes that there is a way to track the number of members in a band and that the information is easily retrievable. Because of functionality that was demonstrated in other queries, it is also possible to look for band members who have been in or are in bands with more than ten staff members.

Phase2Q5GroupBy2.sql

All bands that released an album after the average year in which albums were released

This query demonstrates the relationship between the album and the band, as well as the album and the statistics of the album. This functionally shows that a relationship exists between the band and the album as well as the album and its statistics, this allows for alternate queries for example this functionality would allow a user to query for all the bands that released an album with a higher than average rating, or bands that have released more albums than average. This combined with functionalities from other queries would allow even more in depth queries, for example "All members who play the guitar that were part of a band that released an album with a higher than average rating" would be possible, as the database is a complete relationship.

Phase2Q6.sql

All band members and bands that started a tour sometime during, or after, 2016.

This query demonstrates the relationship between bands, their tours and the the timeline of the tour. This relationship is expansive in that because member connects to bands it would also be possible to show members who were in a band that started a tour after 2016. This relationship allows users of the database to easily see what bands are touring and when.

Phase2Q7.sql

All featured artists that have ever played for a band that ever had a concert in Yuma, Arizona on 2011-02-23, and the instrument that the featured artist plays and/or the role they had

This is a more complicated query which shows the complete interaction between featured artist and songs, songs and bands, and bands and their tours. This query is an example of one that is only possible in a complete database where all the relations are connected while also demonstrating a more realistic query lookup-- in the sense that a concert attendee wishes to recall information from a concert they previously went to but can't remember any other information.

User Interaction Query

A website allowing a user to modify the database.

This query is flexible, allowing a user to add data to the database as they will. It allows to query bands, insert bands, and run custom queries. It also has a count functionality that totals the rows in each table.

Functional Dependencies

Featured Artist->Song

The Featured Artist has a functional dependency with the song that they are in. this is because the featured artist uniquely determines the song. This means that given a featured artist it is possible to map back to the song that they performed in.

Concert->Tour

The Tour is functionally dependent on the concert because the concert maps to the tour in a way that if one knows information on a concert it is unique to the tour. For example the performers of a band at a concert and the timeline that the concert takes place will also fall within the information of the tour.

Album->Band

The album is functionally dependent on the band because the album will contain all information that is uniquely determined by the band, for example performers in an album will be members of the band who that album belongs to, in addition the year the album came out will be the year the band released said album.