Data Description:

Our application's database is based on APIs from DBPedia
Our data is retrieved using the SPARQL protocol.
Our data is divided to:

1. Data tables:
    - We generated our Data tables from DBPedia via SPARQL.
    - We sent a SPARQL query for each entity and its desired attributes, and constructed, from the returned results, a data table.
    - Every tuple in the entity is unique, meaning tuples represent one unique ontology entity. (Each artists, genre, song, album, composition, band and album has only one tuple representing them).
      We ensured this using the attribute 'wikipageID' which is the distinct id assigned to each ontology entity in DBPedia.
    - The data tables:
        a. MusicGenre - all the musical genres.
        b. MusicalArtist - musical artists (not bands).
        c. Band – musical bands (no solo artists).
        d. Songs – music singles and songs.
        e. Album – musical albums.
        f. ClassicalMusicComposition – classical music work.
2. Relation tables:
    - The attributes in the data tables, that were needed for connecting them to other data tables, usually consisted of many matches (For example: many genres for one artist).
    - The relation tables were created in order to reduce data duplication.
      They were also extracted from SPARQL and consisted of two attributes, one was an identifier for the tuple in the data table, and the other was all the matches in the desired attribute (flat table).
    - For example: Every artist was connected to many genres, in order to spare duplication of the artist's tuple for each genre, we decomposed the Artists table, taking only the Attributes: Artist's Identifier and each matching genre and connecting them together in a new flat table.

- After completing the tables creation, we used them to connect between our data tables. (The full process is explained in the 'scripts' section).
- The relation tables (full specification in the DB description section):
    a. Music Genres to its Origins, Subgenres, Derivations and Fusions.
    b. Songs, Albums, Artists & Bands to Music Genres.
    c. Songs and Albums to Artists and Bands.
    d. Artists to Bands.

Data processing:

We have written **Python** scripts to create our DB in three stages:

1. Connect to DBPedia (via requests to server) and send SPARQL queries to retrieve the desired data (the APIs).
   The data returns in JSON format.
2. Clean the data and organize results in CSV format.
   The results are divided into Data and Relation tables (CSV format).
3. Generate MySQL schema and queries for insertion and index declaration.

- Our python code is here: https://github.com/shircohen36/Database-Systems/tree/master/Code/BuildDB
  And enclosed in our submission under the 'BuildDB/Code' directory.
- The entire process (from start to finish) can be run by the script: **'runCodeDataToDB.py'.**
  (The APIs extraction process can take a while…).

Full Code Flow - Scripts (Python scripts, by order of execution):

1. Install Python package: SPARQLWrapper-1.7.6.
    a. We've enclosed this in our submission files.
        i. Installation instruction in the 'READEME' file.
        ii. Can Install from command with the line: **python setup.py install**
2. Extract the APIs:

a. Scripts:
    i. **extractAPISparqlData.py**
    ii. **extractAPISparqlRelation.py**
    iii. **(extractClassicalmusichelper.py).**
b. Description:
    i. Connects to DBPedia server and extracts APIs by sending SPARQL queries.
    ii. Our code contains:
        1. Queries from which we built our Data tables.
        2. Queries from which we built our Relation tables.
    iii. The data returns in Json format.
    iv. Then process the data retrieved for the query from DBPedia, organize results, clean text and encode in "utf-8".
    v. Combine all results for each entity and construct its data table.
    vi. Save tables in CSV format in the "DataTables" and "RelationTables" directories.
    vii. 'extractClassicalmusichelper.py' – extracting classical music information was slightly different from the rest of the data tables and therefore we created an additional script.

3. Cleaning the tables:
    a. Scripts:
        i. **cleanDataAddID.py**
    b. Description:
        i. Cleans all the data tables and adds a unique ID to each tuple (incrementing counter starting with '1').
        ii. Results are saved in the directory: "DataTablesClean".
4. Find top genres:
    a. Scripts:
        i. **findTopGenres.py**
    b. Description:
        i. Our app gives results to users by genres and sub genres search.

ii. We needed some key top genres such as "Rock music" and "Pop music" in order to begin descending down our genres tree. (information on genres and their sub genres is available in the extracted data, but we wanted to present only a few very general roots for the user to choose from).
iii. This script creates a data table called "MusicGenreTop", and a relation table connecting between the Music Genres to the Top Genres (their parents).
iv. Each genre in the Musical Genres table is connected to at least one Top Genre. (So all genres are accessible from at least one root).

5. Creating the "Songs" table:
    a. Scripts:
        i. **combineTwoTablesToOne.py**
    b. Description:
        i. The information in DBPedia on Songs was divided into 2 categories: Song and Single (the information was distinct).
        ii. After the extraction, we combined the two tables into one table, cleaned it and added IDs to all the tuples.
6. Alter and fix the relation tables.
    a. Scripts:
        i. **createRelationTablesByID.py**
    b. Description:
        i. As mentioned in the previews section, we created separate tables for attributes connection between our data tables.
        The data in these tables was consist of identifiers and values extracted from DBPedia.
        ii. This script takes these tables and turns them into proper relation tables, connecting between two data tables while ensuring information integrity.
        iii. The script goes over each row in the relation table and changes the identifier to the ID that matches it in the data table. (i.e the artist's identifier is changed to the

ID we assigned this artist in the Artists data table.)
And the same is done for the other attribute.

iv. This script preforms 2 tasks:
    1. Change the relation tables to connect tuples via the ID values we assigned to them (instead of DBPedia identifiers).
    2. Keep the integrity of the connection information by discarding any information which is not found in the data tables.
       i.e. if an Artist is connected to a song that is not found in the Songs table then this row (this connection) is discarded.

v. Results are saved in the directory: "RelationTablesByID"

7. Create the MySql schema and insertion queries.
    a. Scripts:
        i. **createMySQLTableFromCSV.py**
    b. Description:
        i. This script takes the tables we created and processed from the directories: **"**DataTablesClean" and "RelationTablesByID" and creates 2 SQL files that are stored in the directory "SQL_DB":
            1. **music-schema.sql**
            2. **music-data.sql**
        ii. music-schema contains the "Create Table" queries for all the tables in the above directories.
        iii. It also sets indexes on all the IDs.
        (ID for each tuple in the Data tables, and both the connecting IDs in the Relation tables).
        iv. Music-data contains the "Insert to <table> values ()" queries inserting the tuples in the CSV tables into their tables.
        v. After all the data is inserted into the Data tables, the ID is set to 'Auto-Increment' starting with the last id inserted (to prevent duplicate id). This is done for future data updates.
8. Run full code from start to finish.

a. Scripts:
    i. **runCodeDataToDB.py**
b. Description:
    i. This is the main script. This script runs all the scripts described in 1-7 by this order.
    ii. The output is the SQL files construction our database.

DB description

1. **Entities = Data tables:**
    a. MusicGenre (ID, wikiPageID, url, name, comment).
        - All the musical genres .
        - PRIMARY KEY (ID).
    b. MusicGenreTop (ID, name).
        - Most general genres.
        - PRIMARY KEY (ID)
    c. MusicalArtist (ID, wikiPageID, url, name, comment, activeYearsStartYear, activeYearsEndYear, background, imageLink).
        - Musical artists, musicians (not groups or bands).
        - PRIMARY KEY (ID)
    d. Band (ID, wikiPageID, url, name, comment, activeYearsStartYear, activeYearsEndYear, background, imageLink).
        - All bands (not solo musicians).
        - PRIMARY KEY (ID)
    e. Songs (ID, wikiPageID, url, name, comment, year, Album, subsequentWork, previousWork, imageLink)
        - All the songs from single and song categories in DBPedia.
        - PRIMARY KEY (ID)
    f. Album (ID, wikiPageID, url, name, comment, type, released, lastAlbum, nextAlbum).
        - All the music albums.

- PRIMARY KEY (ID)

g. ClassicalMusicComposition (ID, wikiPageID, url, comment, name, composer).
   - Classical music compositions.
   - Table mostly contains data in the comment attribute.
   - PRIMARY KEY (ID)

- Attributes explained:
  - name: name of tuple (Artists name, Genre name…).
  - url: the url address to the tuple page in DBPedia.
  - comment, description, background, type: a short info about the tuple.
  - ID: a distinct id we assigned to the tuples, the IDs are in a increasing order beginning with '1'.
  - wikipageID: distinct id assigned by dbpedia for the url of each tuple. (we used them to group results that returned in the API extraction part, and for the connections in our relation tables).
  - Imagelinke: link to a picture of the tuple (Artist, Band), extracted also from the APIs.

2. Relations = Relation tables:

a. Connect to genre:
   - MusicalArtist_MusicGenre (MusicalArtist, MusicGenre).
     - REFERENCES MusicalArtist(ID), REFERENCES MusicGenre(ID).
   - Band_MusicGenre (Band, MusicGenre).
     - REFERENCES Band(ID), REFERENCES MusicGenre(ID).
   - Songs_MusicGenre (Song, MusicGenre).
     - REFERENCES Songs(ID), REFERENCES MusicGenre(ID).
   - Album_MusicGenre (Album, MusicGenre).
     - REFERENCES Album(ID), REFERENCES MusicGenre(ID).

b. Connect artists/bands to musical work.

- Album_Band (Album, Artists).
    - REFERENCES Album(ID), REFERENCES Band(ID).
- Album_MusicalArtist (Album, Artists).
    - REFERENCES Album(ID), REFERENCES MusicalArtist(ID).
- Songs_Band (Song, Artists).
    - REFERENCES Songs(ID), REFERENCES Band(ID).
- Songs_MusicalArtist (Song, Artists).
    - REFERENCES Songs(ID), REFERENCES MusicalArtist(ID).

c. Connect genre to other genres:

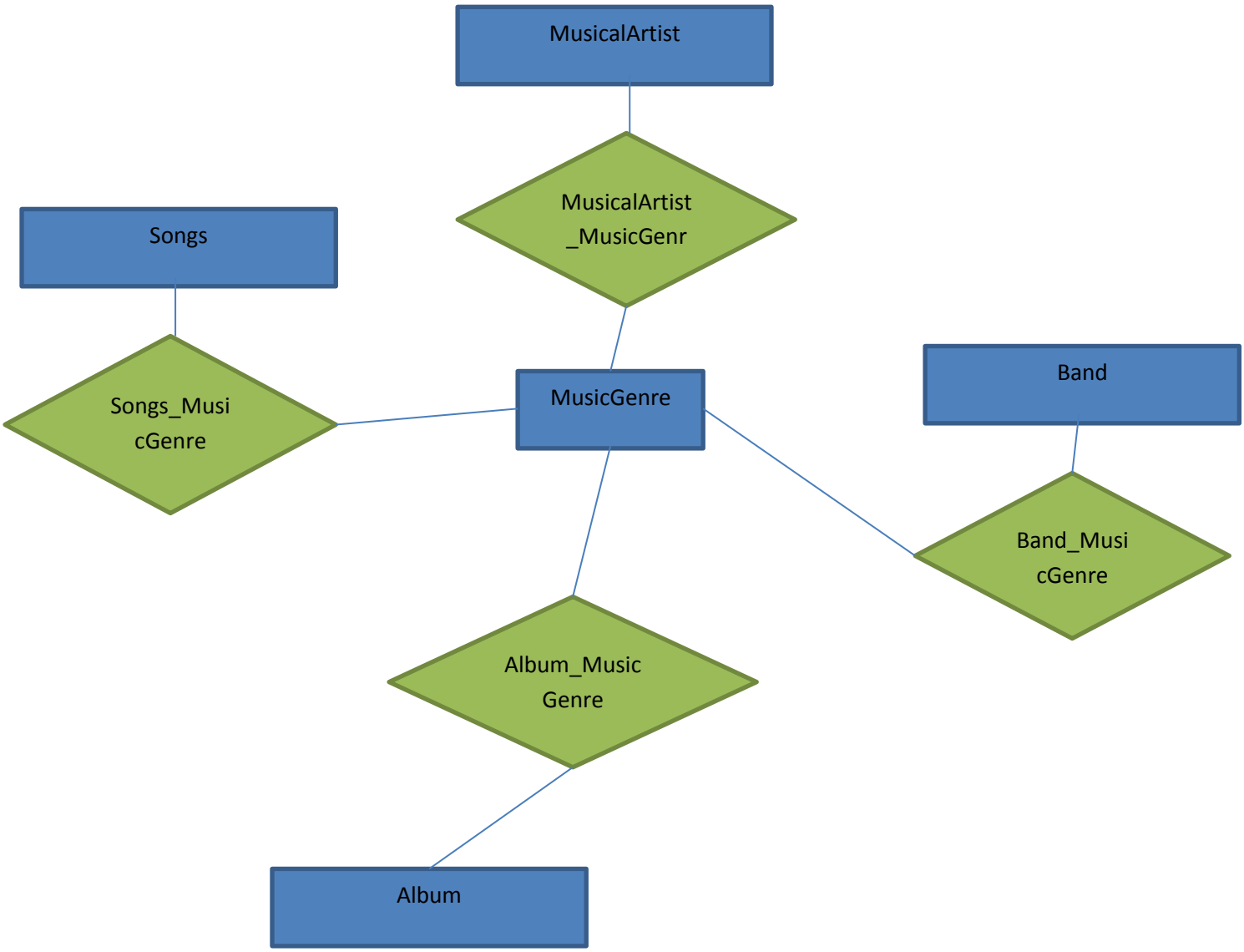
- MusicGenre_MusicGenreTop (MusicGenre, TopGenre).
    - REFERENCES MusicGenre(ID), REFERENCES MusicGenreTop(ID).
- MusicGenre_MusicStylisticOriginGenre (MusicGenre, MusicStylisticOriginGenre).
    - REFERENCES MusicGenre(ID), REFERENCES MusicGenre(ID).
- MusicGenre_MusicSubGenre (MusicGenre, MusicSubGenre).
    - REFERENCES MusicGenre(ID), REFERENCES MusicGenre(ID).
- MusicGenre_MusicDerivativeGenre (MusicGenre, MusicDerivativeGenre).
    - REFERENCES MusicGenre(ID), REFERENCES MusicGenre(ID).
- MusicGenre_MusicFusionGenre (MusicGenre, MusicFusionGenre).
    - REFERENCES MusicGenre(ID), REFERENCES MusicGenre(ID).

d. Connect between artists:

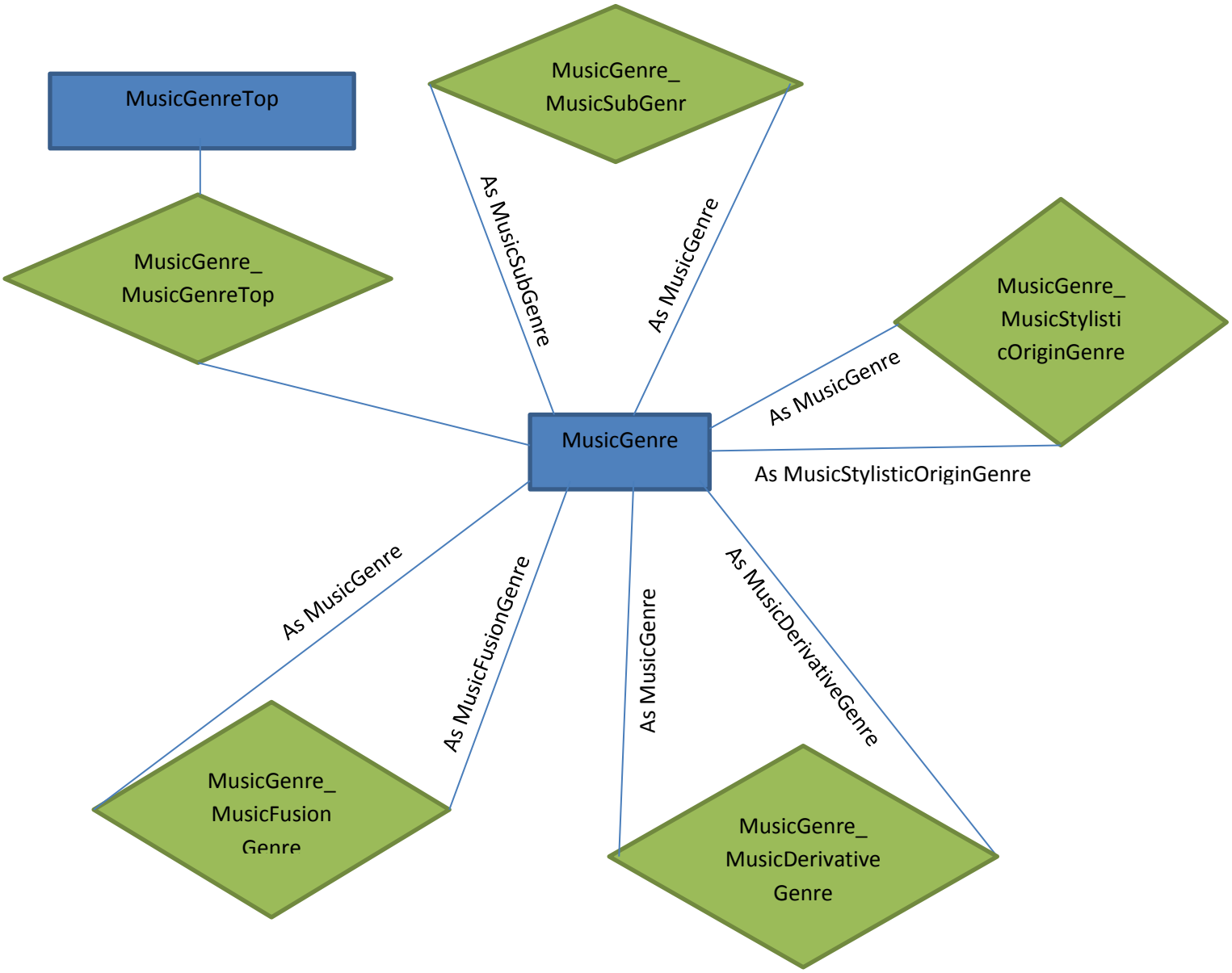
- Band_MusicalArtist (Band, BandMembers).
    - REFERENCES Band(ID), REFERENCES MusicalArtist(ID).

3. Our full database structure is described in the next page.

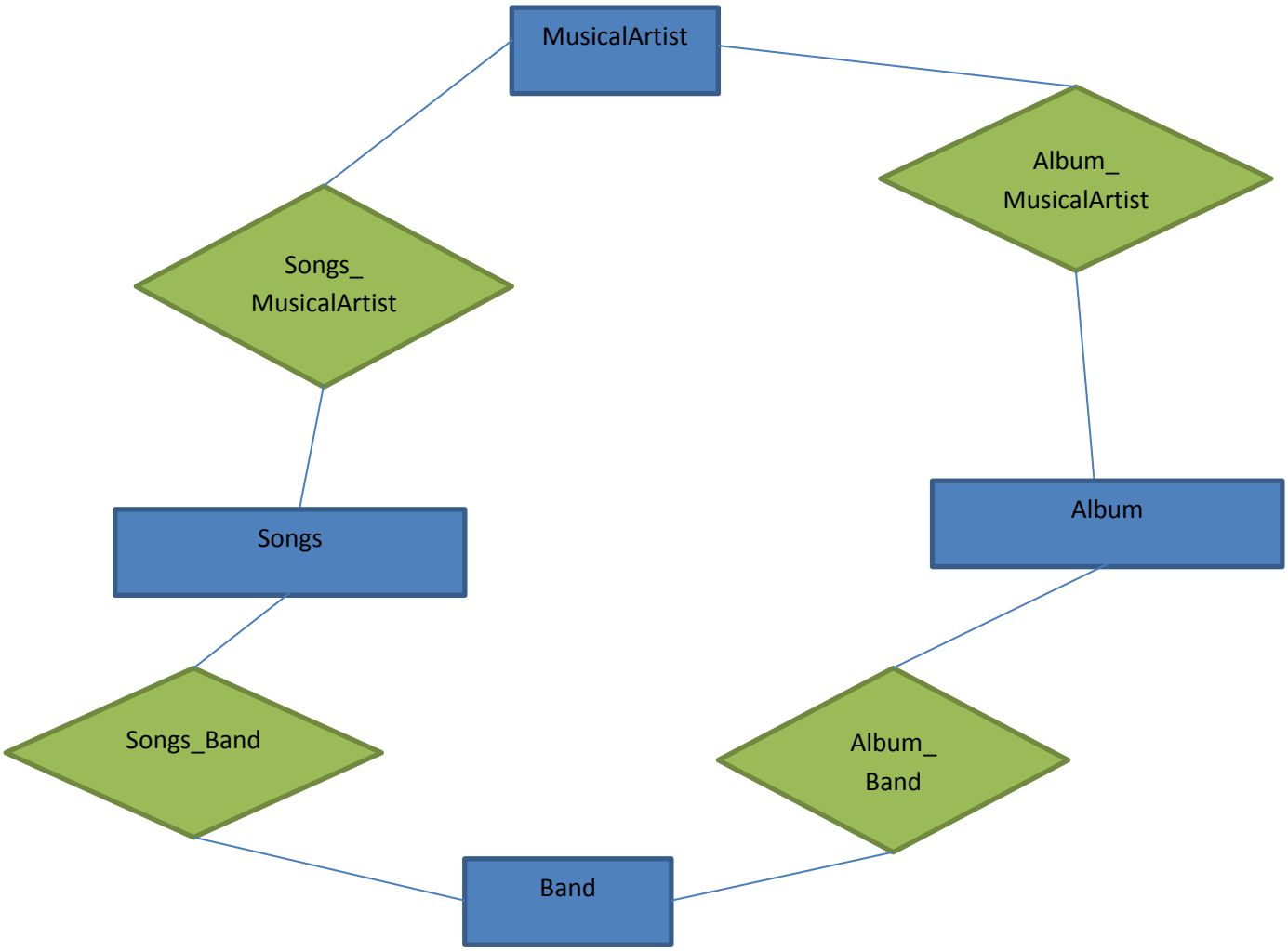
Connect to genre:



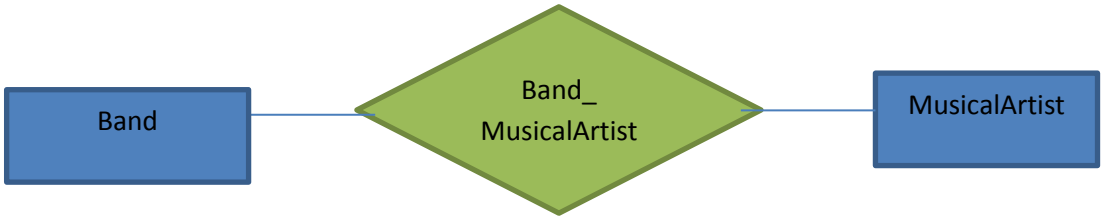
Connect genre to other genres:



Connect artists/bands to songs and albums:



Connect between artists:



Not connected entity (called for by code**):**

Code Structure:

Server side is written in php.
Client side in written in Html, CSS, JavaScript and JQuery.

Connection to the db:

Config.php:

Contains the connection to our database, we use the mysqli function that is a built in function in php for connection to mysql database.

Run query is a general function we use it to run our query.

SQL Functions:

Sql_funcs.php:

This page contains our SQL functions , and queries to the database. You can see our queries with documentation in this file.

General_funcs.php:

Includes all the php functions that are used in the pages to display the information to the user.

The general _funcs page is a connection between the pages that we display to the client and the SQL queries from the database.

Displaying pages:

MainPage.php:

This page is the starting point page in our website that we display to the client.

It includes a genre that we get from the MusicGenreTop table in the database and a sub genre we get from musicGenre table. You need first to choose a genre and then by using jquery we re-render the sub genre top down.

You need to choose a genre and a sub genre ,otherwise the submit will be disabled.

When you press on "submit" after choose genre and subgenre you go to OptionsPage.php.

We send the information about the id of the chosen sub genre in a cookie to the options page.

Loadsubcat.php:

We use this file to get the sub genre to display it in the main page.

OptionsPage.php:

Displays the sub the user has chosen.

You can see in this page information about the genre including: information about the genre, most popular year (by artists and by bands), start year of this genre, most active band and artist (the band and artist that have the largest number of active years in this genre) and etc.

We call the function that in general _function.php in order to get the information this page displays.

We receive the id of the genre with a cookie or with "get" depends on the way the user arrives to this page: mainPage sends to optionsPage a cookie with the information and the other pages send "get" methods to this page.

By using the id of this sub genre we get the information about this genre in this page.

By pressing on the artist name we go to ArtistPage.php, using the "get" method to get the artist id we want to display

ArtistPage.php:

We get the id of the artist by using "get" method.

This page displays artist information and also band information.

We check if the user chose an artist or a band.

By using the id (of the band or artist), we take information from the data base that relevant to this artist\band and display it, by using functions from the general_function.php

We also display some video related to this artist\band, using the API of you tube to get one song of this band\artist

Song.php:

We get the id of the song by using "get" method .

If the song comes from pressing on song in the ArtistPage.php we also get the artist id using get method (it help us to get the related song in youtube video)

We also display the video related to this song, using the API of you tube

Album.php:

We get the id of the album by using get method.

By using functions from general_function.php we get the information TO display the client.

Search.php:

You can search for Album or artist\band or for song insert the name and get result of name that includes the letters you write.

If you don’t find the song or artist\band or album you want , you can insert it to our database.

ClassicalComp.php:

If the client chose classical for genre and Classical music composition for sub genre then in the OptionsPage we get only song which are classical composition . by pressing one of this song we get to this page (ClassicalComp.php)

By using the id (using get method) of the composition we have chosen and by calling php function of the general_funcs page we display some information about the composition and get the composition video from youtube API.

update_song.php:

In the update_song you can update the song with missing information. We use the "post" method in order to get the information from the user.

Data_insert_func.php:

This function using post method to get the information and insert the info the user types to the db.

ShowVideo.php:
We use YouTube API to get videos of songs or artists by their name.

Update_artist_band:
In the update_ artist_band you can update the artist/band with missing information. We use the "post" method in order to get the information from the user.

MainCss.css:

The css file for the all website, includes all the style for our website

searchBox.css:

Style for the search box page

About.html:

The about page explains briefly about our site.

Js_funcs.js:

The java script fie of this project , mainly used to send the get method to other page.