SQL Statements

CREATE TABLES

Creating the database tables, this can be done automatically in the python initialize.py script in the git repository.

```
CREATE TABLE SONG
    (SongID INT NOT NULL AUTO_INCREMENT,
    TotalPlays INT DEFAULT 0,
    MonthlyPlays INT DEFAULT 0,
    Title VARCHAR(32),
    Duration VARCHAR(32),
    MusicFile VARCHAR(255),
    PRIMARY KEY (SongID));

CREATE TABLE ARTIST
    (ArtistID INT NOT NULL AUTO_INCREMENT,
    Name VARCHAR(32),
    About VARCHAR(1500),
    ProfilePicture VARCHAR(255),
    BannerPicture VARCHAR(255),
    TotalPlays INT DEFAULT 0,
    MonthlyPlays INT DEFAULT 0,
    PRIMARY KEY (ArtistID));

CREATE TABLE ALBUM
    (AlbumID INT NOT NULL AUTO_INCREMENT,
    IsSingle BOOLEAN NOT NULL,
    CoverArt VARCHAR(255),
    ReleaseDate DATE,
    Genre VARCHAR(255),
    Title VARCHAR(32),
    PRIMARY KEY (AlbumID));

CREATE TABLE USER
    (UserID INT NOT NULL AUTO_INCREMENT,
    Username VARCHAR(32) NOT NULL,
    PasswordHash CHAR(255) NOT NULL,
    IsPremium BOOLEAN NOT NULL,
    SubRenewDate DATE,
    PRIMARY KEY (UserID),
    UNIQUE (Username));

CREATE TABLE ADVERTISEMENT
    (AdID INT NOT NULL AUTO_INCREMENT,
    Duration VARCHAR(32),
    Company VARCHAR(32),
    SoundFile VARCHAR(255),
    PRIMARY KEY (AdID));

CREATE TABLE ADMIN
    (AdminID INT NOT NULL,
    PRIMARY KEY (AdminID),
    FOREIGN KEY (AdminID) REFERENCES USER(UserID) ON DELETE CASCADE ON
    UPDATE CASCADE);
```

```
CREATE TABLE PLAYLIST
    (PlaylistID INT NOT NULL AUTO_INCREMENT,
    PlaylistName VARCHAR(32),
    CreatorID INT,
    PRIMARY KEY (PlaylistID),
    FOREIGN KEY (CreatorID) REFERENCES USER(UserID) ON DELETE SET NULL
    ON UPDATE CASCADE);

CREATE TABLE PLAYLIST_CONTAINS
    (PlaylistID INT NOT NULL,
    SongID INT NOT NULL,
    PRIMARY KEY (PlaylistID, SongID),
    FOREIGN KEY (PlaylistID) REFERENCES PLAYLIST(PlaylistID) ON DELETE
    CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (SongID) REFERENCES SONG(SongID)ON DELETE CASCADE ON
    UPDATE CASCADE);

CREATE TABLE ALBUM_CONTAINS
    (AlbumID INT NOT NULL,
    SongID INT NOT NULL,
    PRIMARY KEY (AlbumID, SongID),
    FOREIGN KEY (AlbumID) REFERENCES ALBUM(AlbumID) ON DELETE CASCADE
    ON UPDATE CASCADE,
    FOREIGN KEY (SongID) REFERENCES SONG(SongID) ON DELETE CASCADE ON
    UPDATE CASCADE);

CREATE TABLE HAS
    (AlbumID INT NOT NULL,
    ArtistID INT NOT NULL,
    PRIMARY KEY (AlbumID, ArtistID),
    FOREIGN KEY (AlbumID) REFERENCES ALBUM(AlbumID) ON DELETE CASCADE
    ON UPDATE CASCADE,
    FOREIGN KEY (ArtistID) REFERENCES ARTIST(ArtistID) ON DELETE CASCADE ON
    UPDATE CASCADE);

CREATE TABLE DISTRIBUTOR
    (DistributorID INT NOT NULL AUTO_INCREMENT,
    DistributorName VARCHAR(32),
    PRIMARY KEY (DistributorID));

CREATE TABLE REPRESENTS
    (ArtistID INT NOT NULL,
    DistributorID INT NOT NULL,
    PRIMARY KEY (ArtistID, DistributorID),
    FOREIGN KEY (ArtistID) REFERENCES ARTIST(ArtistID) ON DELETE CASCADE ON
    UPDATE CASCADE,
    FOREIGN KEY (DistributorID) REFERENCES DISTRIBUTOR(DistributorID)ON
    DELETE CASCADE ON UPDATE CASCADE);

CREATE TABLE WRITES
    (SongID INT NOT NULL,
    ArtistID INT NOT NULL,
    PRIMARY KEY (SongID, ArtistID),
    FOREIGN KEY (SongID) REFERENCES SONG(SongID) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (ArtistID) REFERENCES ARTIST(ArtistID) ON DELETE CASCADE ON UPDATE CASCADE);
```

```
CREATE TABLE STEM
    (SongID INT NOT NULL,
    StemNo INT NOT NULL,
    MusicFile VARCHAR(255),
    PRIMARY KEY (SongID, StemNo),
    FOREIGN KEY (SongID) REFERENCES SONG(SongID) ON DELETE CASCADE ON UPDATE CASCADE);
```

INSERT

Inserting data into the database. This happens at a few points:
- Registering a new user.
- Premium users creating playlists and adding songs to them.
- Administrator functions, such as adding new admins, advertisements, songs, albums, artists, and linking the three together.

```
INSERT INTO USER (Username, PasswordHash, IsPremium) VALUES (@Username, @PasswordHash, FALSE);

INSERT INTO PLAYLIST (PlaylistName, CreatorID) VALUES (@PlaylistID, @UserID);
INSERT INTO PLAYLIST_CONTAINS (PlaylistID, SongID) VALUES (@PlaylistID, @SongID);

INSERT INTO ADMIN (AdminID) VALUES (@UserID)
INSERT INTO ADVERTISEMENT (Duration, Company, SoundFile) VALUES (@Duration, @Company, @SoundFile);
INSERT INTO SONG (Title, Duration, MusicFile, TotalPlays, MonthlyPlays)
    VALUES (@Title, @Duration, @MusicFile, 0, 0);
INSERT INTO ALBUM (Title, IsSingle, CoverArt, ReleaseDate, Genre)
    VALUES (@Title, @IsSingle, @CoverArt, @ReleaseDate, @Genre);
INSERT INTO ARTIST (Name, About, ProfilePicture, BannerPicture, TotalPlays, MonthlyPlays)
    VALUES (@Name, @About, @ProfilePicture, @BannerPicture, 0, 0);

INSERT INTO WRITES VALUES(@SongID, @ArtistID);
INSERT INTO HAS VALUES(@AlbumID, @ArtistID);
INSERT INTO ALBUM_CONTAINS VALUES(@AlbumID, @SongID);
```

DELETE

Removing data from the database. There are several ways to do this:
- Premium users can delete their playlists, and remove songs from their playlists.
- Administrators have permissions to delete users along with anything that they can insert, listed in the INSERT subsection.

```
DELETE FROM PLAYLIST WHERE PlaylistID=@PlaylistID;
DELETE FROM PLAYLIST_CONTAINS WHERE PlaylistID=@PlaylistID AND SongID=@SongID;

DELETE FROM USER WHERE UserID=@UserID;
DELETE FROM ADMIN WHERE AdminID=@UserID;
DELETE FROM ADVERTISEMENT WHERE AdID=@AdID;
DELETE FROM SONG WHERE SongID=@SongID;
DELETE FROM ALBUM WHERE AlbumID=@AlbumID;
DELETE FROM ARTIST WHERE ArtistID=@ArtistID;
DELETE FROM WRITES WHERE SongID=@SongID AND ArtistID=@ArtistID;
DELETE FROM HAS WHERE ArtistID=@ArtistID AND AlbumID=@AlbumID;
DELETE FROM ALBUM_CONTAINS WHERE SongID=@SongID AND AlbumID=@AlbumID;
```

UPDATE

Updating data in the database. This does not happen as often, but is important to keep track of the following functionality:
- Users can update their IsPremium status by either subscribing or cancelling premium.
- Whenever a song is played it needs to increment the number of MonthlyPlays and TotalPlays associated with that song and all artists of that song.
- Administrators can update Albums and Artists.
- Administrators can also clear the MonthlyPlays associated with Songs and Artists.

```
UPDATE USER SET IsPremium=TRUE, SubRenewDate=@SubRenewDate WHERE UserID=@UserID;
UPDATE USER SET IsPremium=FALSE, SubRenewDate=NULL WHERE UserID=@UserID;

UPDATE SONG SET TotalPlays = TotalPlays + 1, MonthlyPlays = MonthlyPlays + 1 WHERE SongID=@SongID;
UPDATE ARTIST SET TotalPlays = TotalPlays + 1, MonthlyPlays = MonthlyPlays + 1
    WHERE ArtistID IN (
        SELECT ArtistID FROM WRITES WHERE SongID=@SongID
    );

UPDATE ARTIST
    SET Name=@Name, About=@About, ProfilePicture=@ProfilePicture, BannerPicture=@BannerPicture
    WHERE ArtistID=@ArtistID;
UPDATE ALBUM
    SET Title=@Title, IsSingle=@IsSingle, CoverArt=@CoverArt, ReleaseDate=@ReleaseDate, Genre=@Genre
    WHERE AlbumID=@AlbumID;

UPDATE SONG SET MonthlyPlays=0 WHERE SongID=@SongID;
UPDATE ARTIST SET MonthlyPlays=0 WHERE ArtistID=@ArtistID;
```

SELECT

This is our most prominent SQL query type. It happens on nearly every page to fetch information from the database to display to the user. These functionalities include:
- Selecting data belonging to a single ID, such as a Song or User.
- Selecting data belonging to a User by Username, as Username is a unique field in the database.
- Selecting information on a song, artist, or album and pairing it to another artist or album that may or may not be present in the database.
- Selecting all artists that wrote an album or song, and all albums that contain a song.
- Determining if data already exists in the database, by calling SELECT and ensuring the result is empty.
- Finding the cover art of an album that contains a song by SongID.

```
SELECT <* or a specific set of attributes> FROM <table> WHERE <ID>=@ID;
SELECT * FROM SONG WHERE SongID=@SongID;
SELECT CoverArt FROM ALBUM WHERE AlbumID=@AlbumID;

SELECT * FROM USER WHERE Username=@Username;

SELECT Name FROM ARTIST, WRITES WHERE ARTIST.ArtistID = WRITES.ArtistID AND WRITES.SongID=@SongID;
SELECT Name FROM ARTIST, HAS WHERE ARTIST.ArtistID = HAS.ArtistID AND HAS.AlbumID=@AlbumID;
SELECT Title FROM ALBUM, ALBUM_CONTAINS
    WHERE ALBUM.AlbumID = ALBUM_CONTAINS.AlbumID AND ALBUM_CONTAINS.SongID=@SongID;

SELECT * FROM WRITES, ARTIST WHERE SongID=@SongID AND WRITES.ArtistID=ARTIST.ArtistID;
SELECT * FROM ALBUM_CONTAINS, ALBUM WHERE SongID=@SongID AND ALBUM_CONTAINS.AlbumID=ALBUM.AlbumID;

SELECT * FROM PLAYLIST_CONTAINS WHERE PlaylistID=@PlaylistID AND SongID=@SongID;

SELECT CoverArt FROM ALBUM WHERE AlbumID IN (SELECT AlbumID FROM ALBUM_CONTAINS WHERE SongID=@SongID);
```