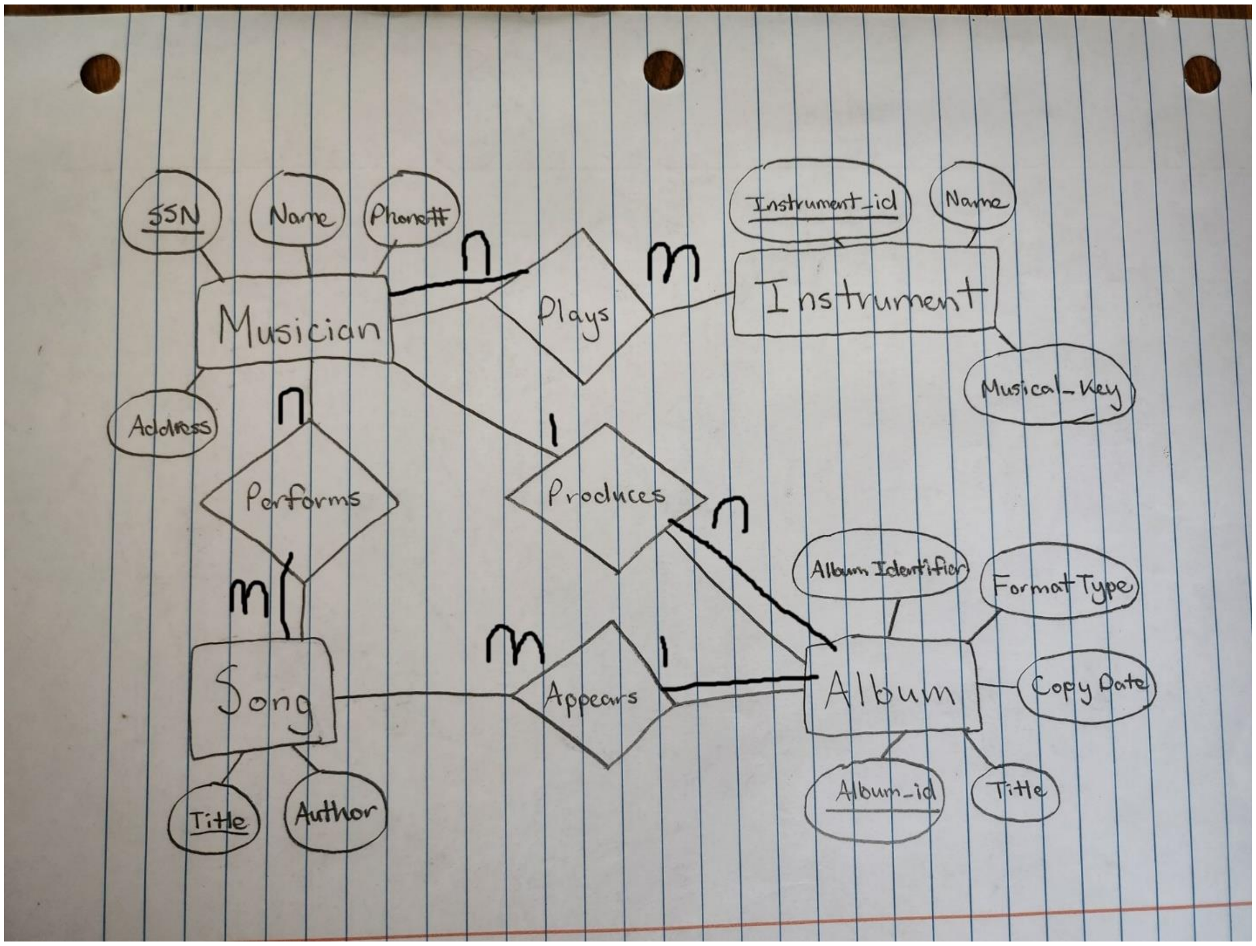
Group 10 members and percentage contributions

Cameron Cheng (33%)

Nathan Campos (33%)

Uchenna Onuigbo (33%)

ER Diagram



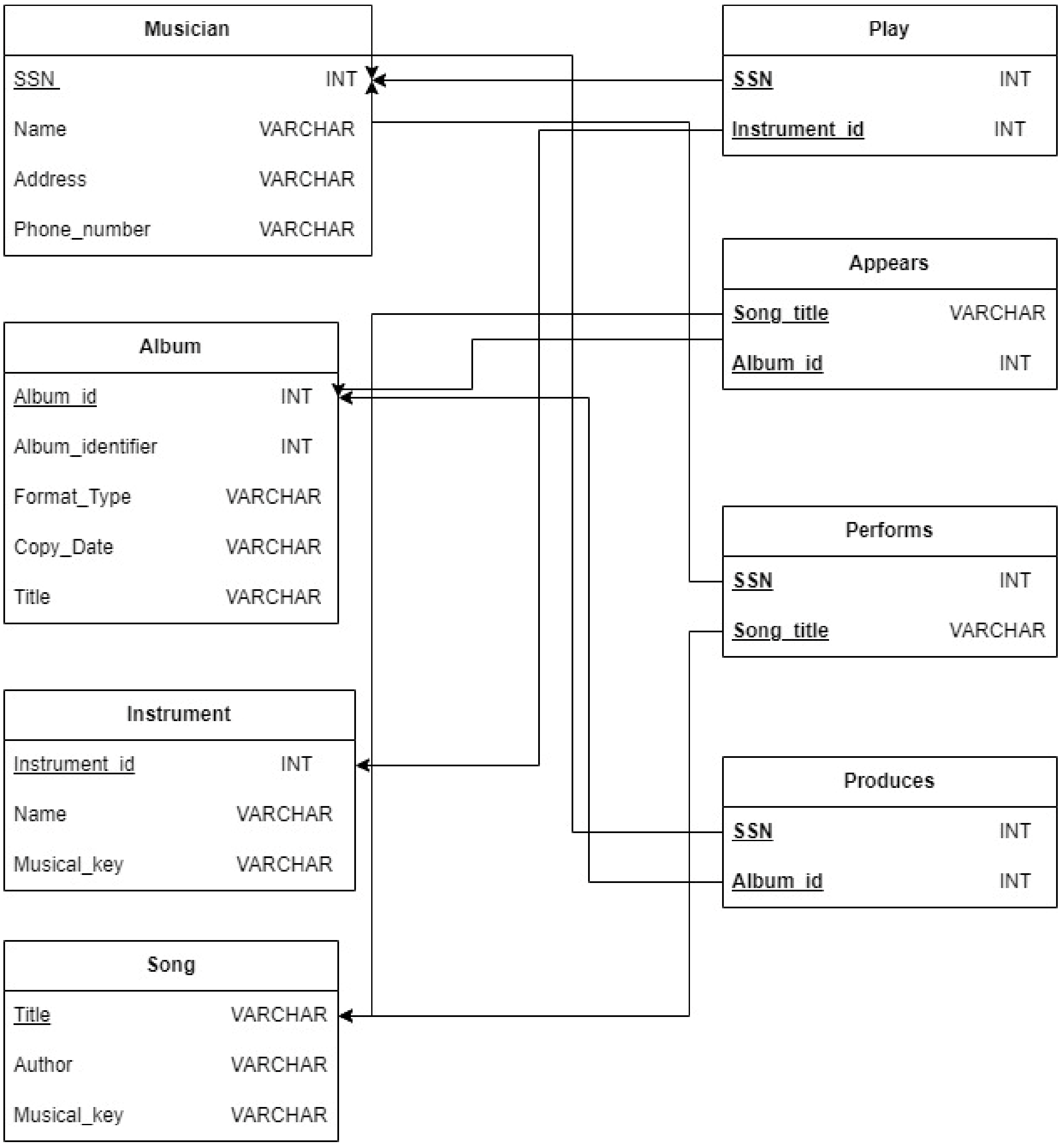
Constraints

- Songs require instruments to be played
- Each musician can play multiple instruments and vice versa
- Musicians perform many songs and vice versa
- Each album has many songs but no one song can appear in the album
- Musician act as the producer of one single album, but they can produce multiple albums

Database Schema Diagram

Underline = primary key,

Bold = foreign key



SQL Table Statements

```
CREATE TABLE Musician
(
  SSN INT PRIMARY KEY,
  Name VARCHAR(30),
  Address VARCHAR(30),
  Phone_number VARCHAR(30)
);

CREATE TABLE Instrument
(
  Instrument_id INT PRIMARY KEY,
  Name VARCHAR(30),
  Musical_key VARCHAR(10),
  SSN INT,
  FOREIGN KEY(SSN) REFERENCES Musician(SSN)
);

CREATE TABLE Album
(
  Album_id INT PRIMARY KEY,
  Album_title VARCHAR(50),
  Format VARCHAR(5),
  Copyright_date VARCHAR(50),
  Album_identifier INT
);

CREATE TABLE Songs
(
   Song_title VARCHAR(30) PRIMARY KEY,
   Author INT,
   FOREIGN KEY(Author) REFERENCES Album(Album_id)
);

CREATE TABLE Play
(
   SSN INT,
   Instrument_id INT,
   PRIMARY KEY(SSN, Instrument_id),
   FOREIGN KEY(SSN) REFERENCES Musician(SSN),
   FOREIGN KEY(Instrument_id) REFERENCES Instrument(Instrument_id)
);


CREATE TABLE Performs
(
   SSN INT,
   Song_title VARCHAR(30),
   PRIMARY KEY(SSN, Song_title),
   FOREIGN KEY(SSN) REFERENCES Musician(SSN),
   FOREIGN KEY(Song_title) REFERENCES Songs(Song_title)
);

CREATE TABLE Appears
(
   Song_title VARCHAR(30),
   Album_id INT,
   FOREIGN KEY(Song_title) REFERENCES Songs(Song_title),
    FOREIGN KEY(Album_id) REFERENCES Album(Album_id)
);

CREATE TABLE Produces
(
    SSN INT,
    Album_id INT,
    PRIMARY KEY(SSN, Album_id),
    FOREIGN KEY(SSN) REFERENCES Musician(SSN),
    FOREIGN KEY(Album_id) REFERENCES Album(Album_id)
);

INSERT INTO Musician VALUES(1, 'Alex', '2222 Mountain Ave', '909-999-7777');
INSERT INTO Musician VALUES(2, 'Jonathan', '3333 Speedhill Ave', '909-111-7777');
INSERT INTO Musician VALUES(3, 'Chris', '4444 Virus Blvd', '909-222-7777');

INSERT INTO Instrument VALUES(1, 'Trumbone', 'C', 1);
INSERT INTO Instrument VALUES(2, 'Ocarina', 'C-flat', 1);
INSERT INTO Instrument VALUES(3, 'Piano', 'C5', 3);

INSERT INTO Album VALUES(1, 'The End So Far', 'CD', '9/30/2022', 7);
INSERT INTO Album VALUES(2, 'The Nothing', 'CD', '9/13/2019', 13);
INSERT INTO Album VALUES(3, 'Disguise', 'CD', '6/7/2019', 5);
INSERT INTO Album VALUES(4, 'Toxicity', 'MC', '9/4/2001', 2);

INSERT INTO Songs VALUES('Adderall', 1);
INSERT INTO Songs VALUES('Cold', 2);
INSERT INTO Songs VALUES('Thoughts & Prayers', 3);
```

Triggers and Stored Procedures

cd_Album(): a function that returns the percentage of the CD album.

```
CREATE OR REPLACE FUNCTION cd_Album() RETURNS REAL AS '
    DECLARE
        total_formats INTEGER := 0;
        cd_formats INTEGER := 0;
        percentage REAL := 0.0;
        row_data ALBUM%ROWTYPE;
    BEGIN
        FOR row_data IN SELECT * FROM ALBUM
        LOOP
            total_formats := total_formats + 1;
            IF row_data.Format=''CD''
                THEN
                    cd_formats := cd_formats + 1;
            END IF;
        END LOOP;
    percentage := cd_formats / (total_formats * 1.0);
    RETURN percentage;
    END;
' LANGUAGE 'plpgsql';
```

total_songs(): a function that returns the total number of songs of all albums.

```
CREATE OR REPLACE FUNCTION total_songs() RETURNS INTEGER AS '
    DECLARE
        song_total INTEGER := 0;
        row_data SONGS%ROWTYPE;
    BEGIN
        FOR row_data IN SELECT * FROM SONGS
        LOOP
            song_total := song_total + 1;
        END LOOP;
    RETURN song_total;
    END;
' LANGUAGE 'plpgsql';
```

remove_album: a trigger to delete an album if all songs in the album are deleted.

```
CREATE OR REPLACE FUNCTION remove_album() RETURNS INTEGER AS '
```

```
BEFORE DELETE ON Album
FOR EACH ROW
DECLARE
    x INTEGER := 0;
BEGIN
    x := (SELECT * FROM Album);

END;
' LANGUAGE 'plpgsql';
```

song_restrict: a trigger to ensure that an album contains no more than 15 songs.

```
CREATE OR REPLACE FUNCTION song_limit() RETURNS TRIGGER AS '
    DECLARE
        song_total INTEGER := 0;
        row_data SONGS%ROWTYPE;
    BEGIN
        FOR row_data IN SELECT * FROM SONGS S WHERE S.Author = NEW.Author
        LOOP
            song_total := song_total + 1;
        END LOOP;
        IF song_total >= 15
                THEN
                    RAISE EXCEPTION ''Limit Reached. Cannot insert more than 15
songs'';
        END IF;
    RETURN NEW;
    END;
' LANGUAGE 'plpgsql';
```