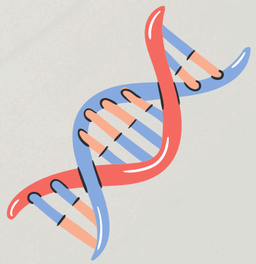
Music DNA



A Music Lineage Database Design

Colin Martin

Table Of Contents

Executive Summary

The music industry is extremely vast. There are many artists that create incredible songs that can inspire a lot of other musicians in their work. Whether it be through the usage of sampling, or a simple cover, a lot of songs are tied together due to the inspirations of musical masterpieces from the past. Some songs do fail to give proper credit though, and legal trouble can spark.

This database encapsulates this entire idea, demonstrating how songs and artists are connected to each other through various samples, techniques, inspirations, and so on. As you continue through my work, you will see an ER diagram visualizing this concept, followed by a breakdown of all of the tables and its contents. After that, you will see the views, queries, stored procedures, triggers, and roles that I implemented into the SQL.

The purpose of this database is to demonstrate a cohesive design, with the inclusion of a fun and interesting topic. The way music travels is a lot more extensive than you may initially realize, and this database will help paint that picture.

ER Diagram

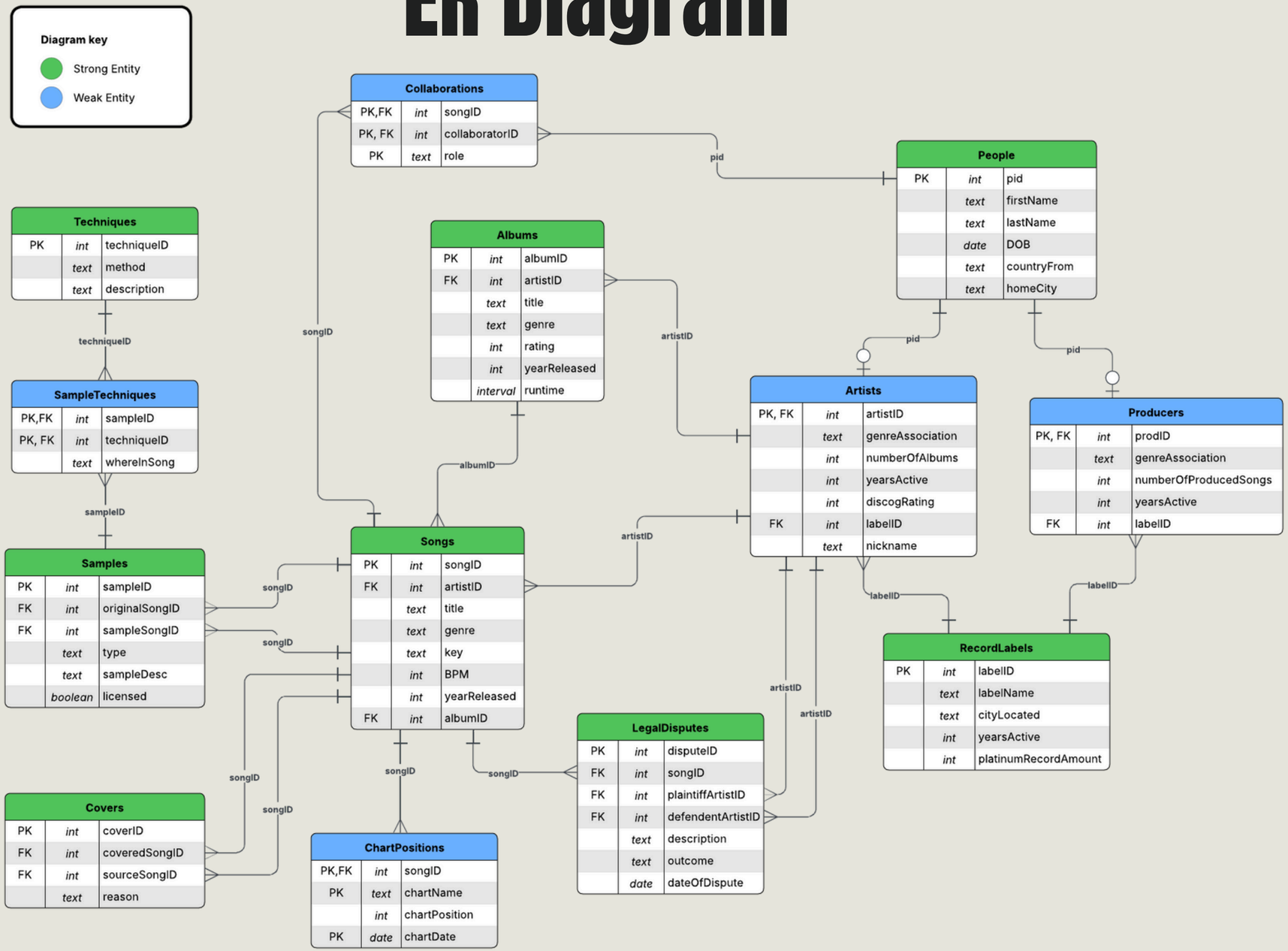


Diagram key
Strong Entity
Weak Entity
Collaborations
PK,FK int songID
PK, FK int collaboratorID
PK text role
People
PK int pid
text firstName
text lastName
date DOB
text countryFrom
text homeCity
Techniques
PK int techniqueID
text method
text description
Albums
PK int albumID
FK int artistID
text title
text genre
int rating
int yearReleased
interval runtime
SampleTechniques
PK,FK int sampleID
PK, FK int techniqueID
text whereInSong
Artists
PK, FK int artistID
text genreAssociation
int numberOfAlbums
int yearsActive
int discogRating
FK int labelID
text nickname
Producers
PK, FK int prodID
text genreAssociation
int numberOfProducedSongs
int yearsActive
FK int labelID
Samples
PK int sampleID
FK int originalSongID
FK int sampleSongID
text type
text sampleDesc
boolean licensed
Songs
PK int songID
FK int artistID
text title
text genre
text key
int BPM
int yearReleased
FK int albumID
RecordLabels
PK int labelID
text labelName
text cityLocated
int yearsActive
int platinumRecordAmount
LegalDisputes
PK int disputeID
FK int songID
FK int plaintiffArtistID
FK int defendentArtistID
text description
text outcome
date dateOfDispute
Covers
PK int coverID
FK int coveredSongID
FK int sourceSongID
text reason
ChartPositions
PK,FK int songID
PK text chartName
int chartPosition
PK date chartDate
pid
techniqueID
sampleID
songID
albumID
artistID
labelID

Tables

People

```
CREATE TABLE People (
    pid             int not null,
    firstName       text not null,
    lastName        text, -- some artists are a singular name
    DOB             date not null,
    countryFrom     text not null,
    homeCity        text not null,
 primary key(pid)
); -- end People
```

The People table stores every person that is in this included in this database, including their origin and date of birth. Note that some last names are null as some artists only go by a one word name.

Functional Dependencies:
pid → [firstName, lastName, DOB, countryFrom, homeCity]

	pid [PK] integer	firstname text	lastname text	dob date	countryfrom text	homecity text
1	1	Yung	Gravy	1996-03-19	United States	Rochester
2	2	Kanye	West	1977-06-08	United States	Atlanta
3	3	Lucky	Chops	2006-12-19	United States	New York City
4	4	Robin	Thicke	1977-03-10	United States	Los Angeles
5	5	Marvin	Gaye	1939-04-02	United States	Washington D.C.
6	6	Player		1977-05-06	United States	Los Angeles
7	7	The	Temptations	1961-03-21	United States	Detroit
8	8	Stevie	Wonder	1950-05-13	United States	Saginaw
9	9	Coolio		1963-08-01	United States	Monessen
10	10	Vanilla	Ice	1967-10-31	United States	Dallas
11	11	Queen		1970-06-27	England	London
12	12	David	Bowie	1947-01-08	England	Brixton
13	13	Lenny	Kravitz	1964-05-26	United States	New York City
14	14	Childish	Gambino	1983-09-25	United States	Edwards Air Force Base
15	15	Steely	Dan	1971-02-06	United States	Annandale-on-Hudson
16	16	Kendrick	Lamar	1987-06-17	United States	Compton
17	17	Jay	Rock	1985-03-31	United States	Watts
18	18	Pharell	Williams	1973-04-05	United States	Virginia Beach
19	19	Beach	House	2004-06-24	United States	Baltimore
20	20	Journey		1973-12-31	United States	San Francisco
21	21	Time	Check	1994-12-02	United States	Poughkeepsie
22	22	Alan	Labouseur	1968-01-23	United States	Albany

RecordLabels

```sql
-- RecordLabel --
CREATE TABLE RecordLabels (
    labelID                 int not null,
    labelName               text not null,
    cityLocated             text not null,
    yearsActive             int,
    platinumRecordAmount    int check(platinumRecordAmount >= 0),
 primary key(labelID)
); -- end RecordLabel
```

	labelid [PK] integer	labelname text	citylocated text	yearsactive integer	platinumrecordamount integer
1	100	Republic Records	New York City	30	600
2	101	GOOD Music	Chicago	21	21
3	102	Interscope Records	Santa Monica	36	1000
4	103	Motown	Detroit	67	53
5	104	RSO Records	London	10	22
6	105	$BK Records	New York City	9	10
7	106	Parlophone	London	129	45
8	107	Virgin Records	Los Angeles	53	77
9	108	RCA Records	New York City	124	400
10	109	pgLang	Los Angeles	5	3
11	110	Top Dog Entertainment	Carson	21	51
12	111	SubPop Records	Seattle	39	5
13	112	Columbia Records	Washington D.C.	38	331
14	113	Warner Records	Los Angeles	66	389

The RecordLabel table stores every record label that a person (artist/producer) may be signed to, including where they are located, how long they have been around, and how many platinum records they have tied to their name.

Functional Dependencies:
labelID → [labelName, cityLocated, yearsActive, platinumRecordAmount]

Artists

```sql
-- Artists --
CREATE TABLE Artists (
    artistID            int not null references People(pid),
    genreAssociation    text,
    numberOfAlbums      int,
    yearsActive         int,
    discogRating        int check(discogRating between 1 and 10),
    labelID             int references RecordLabels(labelID),
    nickname            text,
 primary key(artistID)
); -- end Artists
```

The Artists table connects to People and stores every person that is an artist, including their main-associated genre, how many albums they have made, how long they have been an active artist, their discography rating, their record label (if any), and any associated nickname they may have.

Functional Dependencies:
artistID → [genreAssociation, numberOfAlbums, yearsActive, discogRating, labelID, nickname]

	artistid [PK] integer	genreassociation text	numberofalbums integer	yearsactive integer	discograting integer	labelid integer	nickname text
1	1	Hip-Hop	7	9	7	100	Mr. Buttersworth
2	2	Hip-Hop	15	29	9	101	Yeezy
3	3	Brass Pop	7	19	9	[null]	The Chops
4	4	R&B	8	24	6	102	Thicke
5	5	Soul	25	27	9	103	The Prince of Soul
6	6	Rock	5	35	8	104	The Guys that Made Baby Come Back
7	7	Soul	43	65	8	103	The Emperors of Soul
8	8	R&B	23	64	10	103	
9	9	Hip-Hop	8	35	7	112	
10	10	Hip-Hop	6	39	5	105	
11	11	Rock	15	54	9	106	The Kings of Arena Rock
12	12	Rock	26	54	8	106	Ziggy Stardust
13	13	Funk	11	35	8	107	
14	14	Hip-Hop	4	16	10	108	Bino
15	15	Rock	9	37	9	113	Dan
16	16	Hip-Hop	7	20	10	109	K-Dot
17	17	Hip-Hop	3	8	8	110	
18	18	Hip-Hop	2	32	7	107	Skateboard P
19	19	Indie	8	20	8	111	
20	20	Rock	15	51	9	112	
21	21	A Cappella	3	30	10	[null]	Marists BEST A Cappella Group

Producers

```sql
-- Producers --
CREATE TABLE Producers (
    prodID                  int not null references People(pid),
    genreAssociation        text,
    numberOfProducedSongs   int not null,
    yearsActive             int,
    labelID                 int references RecordLabels(labelID),
 primary key(prodID)
); -- end Producers
```

The Producers table connects to People and stores every person that is a producers, including their main-associated genre, how many songs they produced, how long they have been producing (if also an artist, this yearsActive value **can** be different), and their potential record label

Functional Dependencies:
prodID → [genreAssociation, numberOfProducedSongs, yearsActive, labelID]

	prodid [PK] integer	genreassociation text	numberofproducedsongs integer	yearsactive integer	labelid integer
1	2	Hip-Hop	600	29	101
2	4	R&B	125	31	102
3	5	Soul	50	27	103
4	8	R&B	200	64	103
5	13	Funk	193	36	107
6	14	Hip-Hop	50	17	108
7	16	Hip-Hop	30	21	109
8	18	Hip-Hop	661	33	107
9	22	Alpaca-Rock	19	20	102

Albums

```sql
-- Albums --
CREATE TABLE Albums(
    albumID         int not null,
    artistID        int not null references Artists(artistID),
    title           text not null,
    genre           text not null,
    rating          int check(rating between 1 and 10),
    yearReleased    int not null,
    runtime         interval not null,
 primary key(albumID)
); -- end Albums
```

The Albums table stores the album that a song belongs to, including the artist that made it, the main genre of it, its rating (1 to 10), the year it came out, and how long it runs.

Functional Dependencies:
albumID → [artistID, title, genre, rating, yearReleased, runtime]

	albumid [PK] integer	artistid integer	title text	genre text	rating integer	yearreleased integer	runtime interval
1	200	1	Cheryl (Single)	Hip-Hop	9	2017	00:02:49
2	201	2	Graduation	Hip-Hop	9	2007	00:54:29
3	202	3	NYC	Brass Pop	8	2015	00:35:37
4	203	4	Blurred Lines	R&B	7	2013	01:02:00
5	204	5	Live at the London Palladium	Soul	8	1977	01:18:00
6	205	6	Player (Self Titled)	Soft Rock	8	1977	00:39:56
7	206	7	The Temptations Sing Smok...	Soul	8	1964	00:33:49
8	207	8	Songs in the Key of Life	R&B	10	1976	01:45:00
9	208	9	Gangstas Paradise	Hip-Hop	8	1964	01:04:00
10	209	10	To The Extreme	Hip-Hop	6	1990	00:57:53
11	210	11	Hot Space	Rock	9	1982	00:48:18
12	211	13	5	Funk Rock	8	1998	01:15:00
13	212	14	Kauai	R&B	8	2014	00:28:07
14	213	15	The Royal Scam	Jazz Fusion	9	1976	00:41:17
15	214	16	good kid, m.A.A.d city	Hip-Hop	10	2012	01:32:00
16	215	19	Teen Dream	Indie Rock	9	2010	00:48:46
17	216	20	Frontiers	Rock	9	1983	00:43:47
18	217	21	Offce Hours	A Cappella	10	2024	00:10:19

Songs

```sql
-- Songs --
CREATE TABLE Songs (
    songID          int not null,
    artistID        int not null references Artists(artistID),
    title           text not null,
    genre           text not null,
    key             text not null,
    BPM             int not null check(BPM > 0),
    yearReleased    int not null,
    albumID         int references Albums(albumID),
 primary key(songID)
); -- end Songs
```

The Songs table stores every song that an artist made, including its main genre, the key it is in, its tempo (BPM), the year it came out, and the album that it belongs to.

Functional Dependencies:
songID → [artistID, title, genre, key, BPM, yearReleased, albumID]

	songid [PK] integer	artistid integer	title text	genre text	key text	bpm integer	yearreleased integer	albumid integer
1	300	1	Cheryl	Hip-Hop	C Minor	76	2017	200
2	301	2	Champion	Hip-Hop	F Sharp Major	102	2007	201
3	302	3	My Girl	Brass Funk	C Major	105	2015	202
4	303	4	Blurred Lines	R&B	G Major	120	2013	203
5	304	5	Got to Give it Up	Soul	D Major	123	1977	204
6	305	6	Baby Come Back	Soft Rock	F Minor	156	1977	205
7	306	7	My Girl	Soul	C Major	105	1964	206
8	307	8	Pastime Paradise	Soul	C Minor	79	1976	207
9	308	9	Gangstas Paradise	Hip-Hop	A Flat Major	80	1995	208
10	309	10	Ice Ice Baby	Hip-Hop	D Minor	116	1990	209
11	310	11	Under Pressure	Rock	D Major	114	1981	210
12	311	13	Thinking of You	Funk Rock	A Major	167	1998	211
13	312	14	Sober	R&B	C Major	98	2014	212
14	313	15	Kid Charlemagne	Jazz Fusion	C Major	97	1976	213
15	314	16	Money Trees	Hip-Hop	G Major	72	2012	214
16	315	19	Silver Soul	Indie Rock	D Major	135	2010	215
17	316	20	Seperate Ways (Worlds Apart)	Rock	C Major	131	1983	216
18	317	21	Seperate Ways (Worlds Apart)	A Cappella	B flat Major	131	2024	217

Samples

```sql
-- Samples --
CREATE TABLE Samples (
    sampleID        int not null,
    originalSongID  int not null references Songs(songID),
    sampledSongID   int not null references Songs(songID),
    type            text not null check (type in ('Sample', 'Interpolation', 'Remix', 'Mashup', 'Sound-Alike')),
    sampleDesc      text,
    licensed        boolean not null,
 primary key(sampleID),
 check (originalSongID != sampledSongID)
); -- end Samples
```

The Samples table stores every song that samples another song, including its type of sample, the description of the sample, and whether or not the sample has proper approval.

Functional Dependencies:
sampleID → [originalSongID, sampledSongID, type, sampleDesc, licensed]

	sampleid [PK] integer	originalsongid integer	sampledsongid integer	type text	sampledesc text	licensed boolean
1	400	305	300	Sample	Melodic sample from chorus	true
2	401	313	301	Interpolation	Interpolates instrumental groove	true
3	402	307	308	Sample	Sample of main instrumental and chorus	true
4	403	310	309	Sample	Sample of bassline and piano	false
5	404	311	312	Sample	Sample of guitar riff and chord progression	true
6	405	315	314	Sample	Vocal and instrumental flipped into beat	true
7	406	304	303	Sound-Alike	Song sounds a little to similar to another one	false

Techniques

```sql
-- Techniques --
CREATE TABLE Techniques (
    techniqueID     int not null,
    method          text not null,
    description     text,
 primary key(techniqueID)
); -- end Techniques
```

	techniqueid [PK] integer	method text	description text
1	500	Chopping	Slicing and rearranging segments of the original audio
2	501	Looping	Repeating a section of audio as a rhythmic or melodic loop
3	502	Pitch Shifting	Changing the pitch of the original audio sample
4	503	Time Stretching	Altering the speed of the sample without affecting pitch
5	504	EQ Filtering	Isolating or enhancing frequencies in the original sample
6	505	Interpolation	Replaying or recreating a sample rather than directly samplin...

The Techniques table stores different way that a sample can be used, along with the description of those methods

Functional Dependencies:
techniqueID → [method, description]

SampleTechniques

```sql
-- SampleTechniques
CREATE TABLE SampleTechniques (
    sampleID        int not null references Samples(sampleID),
    techniqueID     int not null references Techniques(techniqueID),
    whereInSong     text,
 primary key(sampleID, techniqueID)
); -- end SampleTechniques
```

The SampleTechniques table connects Samples and Techniques and stores the different sampling techniques used in each sampled song, including where in the song those techniques are used.

Functional Dependencies:
sampleID, techniqueID → [whereInSong]

	sampleid [PK] integer	techniqueid [PK] integer	whereinsong text
1	400	500	Used in the hook
2	400	501	Looped during the intro and chorus
3	401	505	Replayed groove in main beat
4	402	501	Looped as the songs base melody
5	402	504	EQ filtered to emphasize synth line
6	403	501	Looped bassline in entire instrument...
7	403	502	Pitch shifted slightly for tempo match
8	404	500	Chopped guitar riff in bridge
9	404	504	Filtered mid frequencies of original
10	405	503	Time-stretched intro vocals
11	405	500	Chopped instrumental to create hook

LegalDisputes

```sql
-- LegalDisputes --
CREATE TABLE LegalDisputes (
    disputeID           int not null,
    songID              int not null references Songs(songID),
    plaintiffArtistID   int not null references Artists(artistID),
    defendantArtistID   int not null references Artists(artistID),
    description         text,
    outcome             text check(outcome in ('settled', 'won', 'lost', 'dismissed', 'ongoing', 'undisclosed')),
    dateOfDispute       date not null,
 primary key(disputeID)
); -- end LegalDisputes
```

	disputeid [PK] integer	songid integer	plaintiffartistid integer	defendantartistid integer	description text	outcome text	dateofdispute date
1	600	303	5	4	Feel and sound of Blurred Lines was deemed too similar to Marvins song	settled	2015-03-10
2	601	309	11	10	Ice Ice Baby copied bassline from Under Pressure by Queen and David Bo...	settled	1991-01-01

The LegalDisputes table stores each song that had been involved in legal trouble, including the plaintiff, the defendant, the description of the dispute, its outcome, and the date it occurred.

Functional Dependencies:
disputeID → [songID, plaintiffArtistID, defendantArtistID, description, outcome, dateOfDispute]

ChartPositions

```sql
-- ChartPositions --
CREATE TABLE ChartPositions (
    songID          int not null references Songs(songID),
    chartName       text not null,
    peakPosition    int not null, -- note that a position of 1 means the best
    chartDate       date not null,
 primary key(songID, chartName, chartDate)
); -- end ChartPositions
```

The ChartPositions table connects to the Songs table and stores each song that made it onto a well known chart, including the highest rank it achieved, and the date of that achievement.

Functional Dependencies:
songID, chartName, chartDate → [peakPosition]

	songid [PK] integer	chartname [PK] text	peakposition integer	chartdate [PK] date
1	300	Billboard Hot 100	78	2017-06-01
2	301	Billboard Hot 100	18	2007-09-10
3	302	Jazz Digital Songs	13	2015-03-18
4	303	Billboard Hot 100	1	2013-07-15
5	304	Billboard R&B	1	1977-06-10
6	305	Billboard Hot 100	1	1978-01-02
7	306	Billboard Hot 100	1	1965-01-08
8	307	Billboard Soul	13	1976-12-01
9	308	Billboard Hot 100	1	1995-10-15
10	309	Billboard Hot 100	1	1990-11-03
11	310	UK Singles Chart	29	1981-12-15
12	311	Adult Alternative Songs	31	1998-07-05
13	312	R&B/Hip-Hop Digital Songs	41	2014-11-12
14	313	Billboard 200	66	1976-09-20
15	314	Billboard Hot 100	30	2012-11-05
16	315	Billboard Rock Songs	22	2010-04-02
17	316	Billboard Hot 100	8	1983-04-10
18	317	A Cappella Weekly	1	2024-03-01

Covers

```sql
-- Covers --
CREATE TABLE Covers (
    coverID             int not null,
    coveredSongID       int not null references Songs(songID),
    sourceSongID        int not null references Songs(songID),
    reason              text,
 primary key(coverID),
 check(coveredSongID != sourceSongID)
); -- end Covers
```

	coverid [PK] integer	coveredsongid integer	sourcesongid integer	reason text
1	700	302	306	Live brass reinterpretation
2	701	317	316	The arrangement was too damn good

The Covers table stores each song that covered another one, including the reason for that cover.

Functional Dependencies:
coverID→ [coveredSongID, sourceSongID, reason]

Collaborations

```sql
-- Collaborations --
CREATE TABLE Collaborations (
    songID          int not null references Songs(songID),
    collaboratorID  int not null references People(pid),
    role            text check(role in ('feature', 'producer', 'co-writer')),
 primary key(songID, collaboratorID, role)
); -- end Collaborations
```

	songid [PK] integer	collaboratorid [PK] integer	role [PK] text
1	303	18	producer
2	310	12	feature
3	313	22	producer
4	314	17	feature

The Collaborations table connects People and Songs and stores each song that had a collaboration between a main artist and another person, including the role of the collaborator.

Functional Dependencies:
songID, collaboratorID, role→ [empty set]

Views

SongLineage

```
CREATE VIEW SongLineage AS
    SELECT s.songID, s.title AS songTitle, 'Sample' AS relationType, src.title AS relatedSongTitle
    FROM Samples samp INNER JOIN Songs s ON samp.sampledSongID = s.songID
                      INNER JOIN Songs src ON samp.originalSongID = src.songID
    UNION
    SELECT c.coveredSongID, s.title AS songTitle, 'Cover' AS relationType, src.title AS relatedSongTitle
    FROM Covers c INNER JOIN Songs s ON c.coveredSongID = s.songID
                  INNER JOIN Songs src ON c.sourceSongID = src.songID;
```

Displays the relationship between songs and represents their overall lineage in terms of where samples / covers came from. It gives us the songID for the sample/cover, the titles for both songs, and a signifier of whether or not the relation is a sample or cover.

	songid integer	songtitle text	relationtype text	relatedsongtitle text
1	300	Cheryl	Sample	Baby Come Back
2	303	Blurred Lines	Sample	Got to Give it Up
3	312	Sober	Sample	Thinking of You
4	301	Champion	Sample	Kid Charlemagne
5	308	Gangstas Paradise	Sample	Pastime Paradise
6	302	My Girl	Cover	My Girl
7	317	Seperate Ways (Worlds Apart)	Cover	Seperate Ways (Worlds Apart)
8	309	Ice Ice Baby	Sample	Under Pressure
9	314	Money Trees	Sample	Silver Soul

TopRatedAlbums

```sql
CREATE VIEW TopRatedAlbums AS
    SELECT a.title AS albumTitle, p.firstName, p.lastName, ar.nickname, a.rating, a.runtime
    FROM Albums a INNER JOIN Artists ar ON a.artistID = ar.artistID
                  INNER JOIN People p ON p.pid = ar.artistID
    WHERE a.rating >= 9;
```

Displays all of the albums that are considered “top rated,” specifically a 9 or higher. This also gives us each artist responsible for the great album, their potential nickname, the specific album rating and the runtime.

	albumtitle text	firstname text	lastname text	nickname text	rating integer	runtime interval
1	Cheryl (Single)	Yung	Gravy	Mr. Buttersworth	9	00:02:49
2	Graduation	Kanye	West	Yeezy	9	00:54:29
3	Songs in the Key of Life	Stevie	Wonder		10	01:45:00
4	Hot Space	Queen		The Kings of Arena Rock	9	00:48:18
5	The Royal Scam	Steely	Dan	Dan	9	00:41:17
6	good kid, m.A.A.d city	Kendrick	Lamar	K-Dot	10	01:32:00
7	Teen Dream	Beach	House		9	00:48:46
8	Frontiers	Journey			9	00:43:47
9	Offce Hours	Time	Check	Marists BEST A Cappella Group	10	00:10:19

NumberOneHits

```sql
CREATE VIEW NumberOneHits AS
    SELECT s.title AS songTitle, p.firstName, p.lastName, c.chartName, c.chartDate
    FROM ChartPositions c INNER JOIN Songs s ON c.songID = s.songID
                          INNER JOIN Artists a ON s.artistID = a.artistID
                          INNER JOIN People p on a.artistID = p.pid
    WHERE c.peakPosition = 1;
```

Displays all of the songs that achieved a beloved number one. It also shows the artists, the respective chart and date of achievement.

	songtitle text	firstname text	lastname text	chartname text	chartdate date
1	Blurred Lines	Robin	Thicke	Billboard Hot 100	2013-07-15
2	Got to Give it Up	Marvin	Gaye	Billboard R&B	1977-06-10
3	Baby Come Back	Player		Billboard Hot 100	1978-01-02
4	My Girl	The	Temptations	Billboard Hot 100	1965-01-08
5	Gangstas Paradise	Coolio		Billboard Hot 100	1995-10-15
6	Ice Ice Baby	Vanilla	Ice	Billboard Hot 100	1990-11-03
7	Seperate Ways (Worlds Apart)	Time	Check	A Cappella Weekly	2024-03-01

Queries / Reports

All Sampled Songs In a Different Key

```
SELECT s.sampleID, samp.title AS sampledSongTitle, samp.key AS sampledKey,
       orig.title AS originalSongTitle, orig.key AS originalKey
FROM Samples s INNER JOIN Songs samp on s.sampledSongID = samp.songID
               INNER JOIN Songs orig on s.originalSongID = orig.songID
WHERE samp.key != orig.key;
```

This query returns every sampled song that were made in a different key than the song it comes from. Each song is listed with its respective key so that the difference is apparent.

	sampleid integer	sampledsongtitle text	sampledkey text	originalsongtitle text	originalkey text
1	400	Cheryl	C Minor	Baby Come Back	F Minor
2	401	Champion	F Sharp Major	Kid Charlemagne	C Major
3	402	Gangstas Paradise	A Flat Major	Pastime Paradise	C Minor
4	403	Ice Ice Baby	D Minor	Under Pressure	D Major
5	404	Sober	C Major	Thinking of You	A Major
6	405	Money Trees	G Major	Silver Soul	D Major
7	406	Blurred Lines	G Major	Got to Give it Up	D Major

All Songs Produced By Alan

```sql
SELECT s.title AS songTitle, p.firstName, p.lastName, s.genre, s.yearReleased
FROM Collaborations c INNER JOIN Songs s on c.songID = s.songID
                      INNER JOIN Artists a on s.artistID = a.artistID
                      INNER JOIN People p on a.artistID = p.pid
WHERE c.collaboratorID = 22 AND c.role = 'producer';
```

	songtitle text	firstname text	lastname text	genre text	yearreleased integer
1	Kid Charlemagne	Steely	Dan	Jazz Fusion	1976

This query returns every song that the one and only Alan Labouseur produced. It also returns the name of the artist for that song, the genre of the song, and the year the song came out.

All Top Charters by Discography

```
SELECT s.title as song, a.nickname, a.discogRating, c.chartName, c.peakPosition
FROM Songs s INNER JOIN Artists a on s.artistID = a.artistID
             INNER JOIN chartPositions c on s.songID = c.songID
WHERE c.peakPosition = 1
ORDER BY a.discogRating DESC;
```

This query gets every song that reached number one on a respective chart, in order from discography rating from highest to lowest. It also returns the artist nickname and the chart they peaked.

	song text	nickname text	discograting integer	chartname text	peakposition integer
1	Seperate Ways (Worlds Apart)	Marists BEST A Cappella Group	10	A Cappella Week...	1
2	Got to Give it Up	The Prince of Soul	9	Billboard R&B	1
3	My Girl	The Emperors of Soul	8	Billboard Hot 100	1
4	Baby Come Back	The Guys that Made Baby Come Back	8	Billboard Hot 100	1
5	Gangstas Paradise		7	Billboard Hot 100	1
6	Blurred Lines	Thicke	6	Billboard Hot 100	1
7	Ice Ice Baby		5	Billboard Hot 100	1

Most Common Sample Techniques

```sql
SELECT t.method, COUNT(st.sampleID) as techniqueCount
FROM SampleTechniques st INNER JOIN Techniques t on st.techniqueID = t.techniqueID
GROUP BY t.method
ORDER BY techniqueCount ASC;
```

This query counts us the most common sampling techniques that were used on the sampled songs, ordered from least to most common.

	method text	techniquecount bigint
1	Interpolation	1
2	Time Stretching	1
3	Pitch Shifting	1
4	EQ Filtering	2
5	Looping	3
6	Chopping	3

Stored Procedures

getProducerSongCount

```
CREATE OR REPLACE FUNCTION getProducerSongCount(pid INT)
RETURNS INT AS
$$
DECLARE
    count INT;
BEGIN
    SELECT COUNT(*) INTO count
    FROM Collaborations
    WHERE collaboratorID = pid AND role = 'producer';
    RETURN count;
END;
$$
LANGUAGE plpgsql;
```

```
SELECT getProducerSongCount(18);
```

	getproducersongcount integer
1	1

This stored procedure represents a function that will allow us to input the id of a person in the database and return the number of songs they produced.

getSampledByArtist

```
CREATE OR REPLACE FUNCTION getSampledByArtist(artistPID INT)
RETURNS TABLE (
    sampledSongTitle TEXT,
    originalSongTitle TEXT,
    samplingArtist TEXT
)
AS
$$
BEGIN
    RETURN QUERY
    SELECT
        s2.title AS sampledSongTitle,
        s1.title AS originalSongTitle,
        p2.firstName || ' ' || COALESCE(p2.lastName, '') AS samplingArtist
    FROM Songs s1 INNER JOIN Samples sp ON s1.songID = sp.originalSongID
                  INNER JOIN Songs s2 ON s2.songID = sp.sampledSongID
                  INNER JOIN Artists a2 ON a2.artistID = s2.artistID
                  INNER JOIN People p2 ON p2.pid = a2.artistID
    WHERE s1.artistID = artistPID;
END;
$$
LANGUAGE plpgsql;
```

```
SELECT *
FROM getSampledByArtist(13);
```

	sampledsongtitle text	originalsongtitle text	samplingartist text
1	Sober	Thinking of You	Childish Gambino

This stored procedure represents a function where you can input an artistID and have a table returned that will show you the details of the newly sampled song (the title, artist, and the original song of the artist you choose to input).

getSongCollaborations

```
CREATE OR REPLACE FUNCTION getSongCollaborations(song_id INT)
RETURNS TABLE (
    collaborator TEXT,
    role TEXT
)
AS
$$
BEGIN
    RETURN QUERY
    SELECT
        p.firstName || ' ' || COALESCE(p.lastName, '') AS collaborator,
        c.role
    FROM Collaborations c
    JOIN People p ON p.pid = c.collaboratorID
    WHERE c.songID = song_id;
END;
$$
LANGUAGE plpgsql;
```

```
SELECT *
FROM getSongCollaborations(314);
```

	collaborator text	role text
1	Jay Rock	feature

This stored procedure represents a function where you can input a songID and it will return the potential collaborator on it and their role on the song.

Triggers

toManyTimesSampled

```
CREATE OR REPLACE FUNCTION tooManyTimesSampled()
RETURNS TRIGGER AS
$$
DECLARE
    sampleCount INT;
BEGIN
    SELECT COUNT(*) INTO sampleCount
    FROM Samples
    WHERE originalSongID = NEW.originalSongID;

    IF sampleCount >= 3 THEN
        RAISE NOTICE 'Song ID % has now been sampled % times.', NEW.originalSongID, sampleCount + 1;
    END IF;

    RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER trg_tooManyTimesSampled
BEFORE INSERT ON Samples
FOR EACH ROW
EXECUTE FUNCTION tooManyTimesSampled();
```

This trigger will automatically give you a warning if a song is starting to be entered as a sample too often.

This can be important mitigating legal trouble potential, as the more a song gets sampled, the less original it is. So the warning may be necessary to make that clear.

```
INSERT INTO Samples (sampleID, originalSongID, sampledSongID, type, sampleDesc, licensed)
VALUES              (319, 305, 600, 'Sample', 'Use in verse melody', TRUE),
                    (320, 305, 601, 'Sample', 'Instrumental bridge', TRUE),
                    (321, 305, 602, 'Sample', 'Bassline sample', TRUE),
                    (322, 305, 603, 'Sample', 'Subtle drum loop reuse', TRUE);
```

Data Output | Messages | Notifications

```
NOTICE:  Song ID 305 has now been sampled 4 times.
NOTICE:  Song ID 305 has now been sampled 5 times.
```

warnLowRatingArtist

```
CREATE OR REPLACE FUNCTION warnLowRatingArtist()
RETURNS TRIGGER AS
$$
DECLARE
    rating INT;
BEGIN
    SELECT discogRating INTO rating
    FROM Artists
    WHERE artistID = NEW.artistID;

    IF rating < 5 THEN
        RAISE NOTICE 'Artist ID % has a discography rating below 5. Track: "%"', NEW.artistID, NEW.title;
    END IF;

    RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER trg_warnLowRatingArtist
BEFORE INSERT ON Songs
FOR EACH ROW
EXECUTE FUNCTION warnLowRatingArtist();
```

This trigger will automatically raise a notice when adding a song made by an artists with a bad discography rating. Its purpose is to essentially just warns you when adding a song by an artist that is not considered to be very good.

Unfortunately, according to this database, Alan does not seem to have a successful music career...

```
INSERT INTO Artists (artistID, genreAssociation, numberOfAlbums, yearsActive, discogRating, labelID, nickname)
VALUES              (22,       'Alpaca-Rock',    7,              12,          4,            100,     '')

INSERT INTO Songs (songID, artistID,  title,            genre,        key,        BPM,  yearReleased,  albumID)
VALUES            (999,    22,        'Noise Symphony', 'Avant-Garde', 'F# Minor', 60,   2025,          NULL);
```

Data Output | Messages | Notifications

```
NOTICE:  Artist ID 22 has a discography rating below 5. Track: "Noise Symphony"
INSERT 0 1

Query returned successfully in 128 msec.
```

Security

Security

```sql
CREATE ROLE Agent;
GRANT SELECT
ON Artists, Songs, ChartPositions, Albums
TO Agent

CREATE ROLE Admin;
GRANT ALL
ON ALL TABLES IN SCHEMA PUBLIC
TO Admin;

CREATE ROLE MusicReviewer;
GRANT INSERT, UPDATE
ON Albums, Artists
TO MusicReviewer
```

Agent: An artist's agent is going to need access to their client's information, including their music (songs and albums), and their success (chart positions).

Admin: The administrator of this database is going to need access to **everything**, along with the ability to make any changes deemed necessary.

MusicReviewer: A music reviewer is going to need access to inserting and updating values such as discogRating (artist) and rating (album), so that they can properly insert the values of their review scores.

Implementation Notes, Known Problems, and Future Enhancements

Implemenation Notes, Known Problems, Future Enhancements

- This database has the potential to hold **a lot** of information. There are many songs in this world, with a lot of them serving as inspirations for other people. With that being said, since this is a project, I had to limit what I implemented, as this only includes a handful of music.
- Plenty of more tables can be added to this design, one of which being “bands.” I thought about implementing that here, however I would have to added each individual member, and three bands would take up the majority of the people table, so I decided not to do that here and is why I just included one word groups as first names. This is a great thing to take care of in the future. This same exact logic is why I did not implement an instrument (and thus a song instruments) table.
- I did have a difficult time trying to fit artists on the SongLineage view, as the query got extremely complex with many joins. This is certainly a great future enhancement to consider.
- Overall, this implementation took a good amount of time, but served as incredible database design practice. I put in a lot of effort to make this exactly how I wanted it, and I feel that it payed off, as I really like the way this came out!