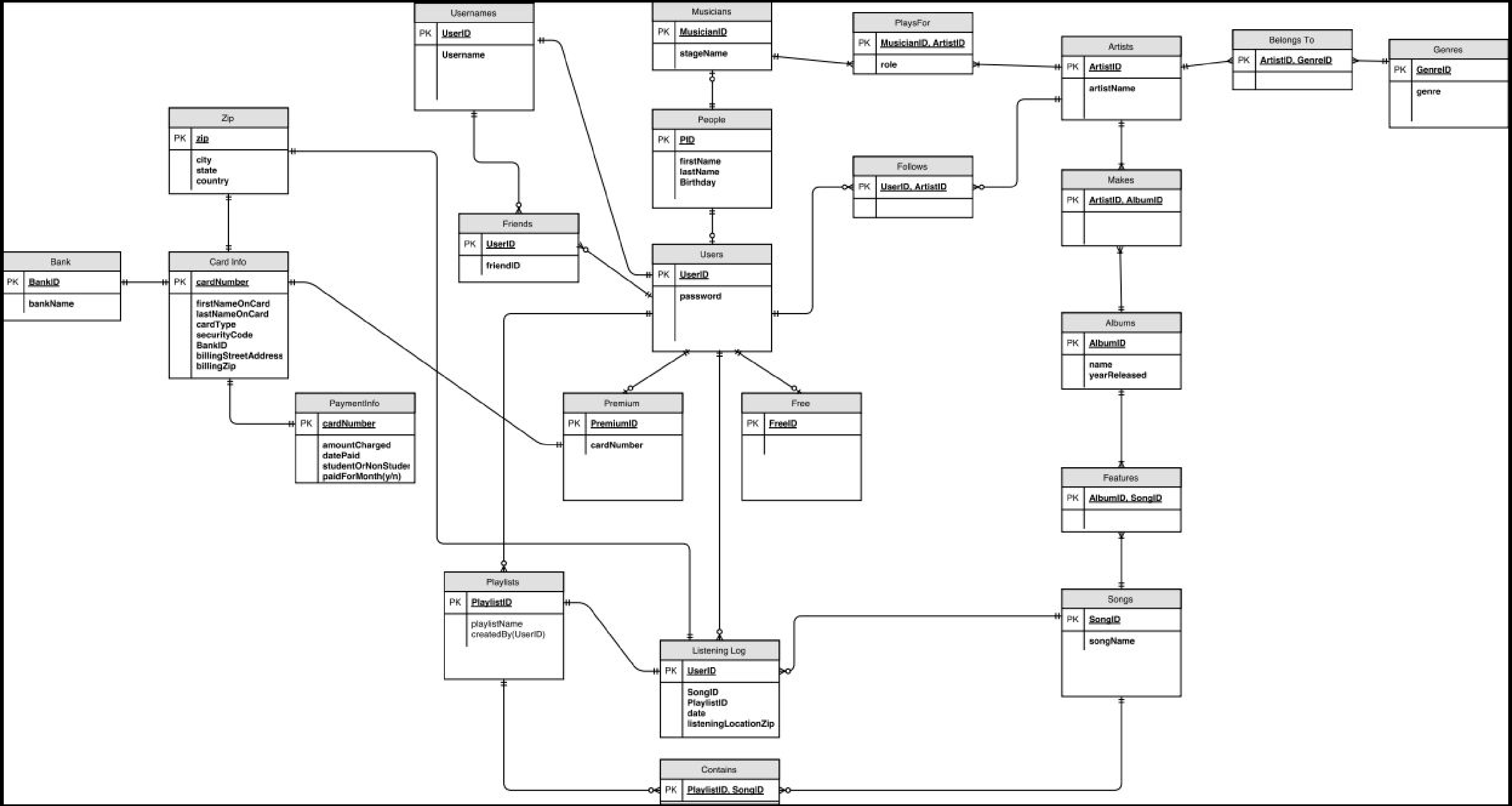
Spotify®

## Table of Contents

With millions of users worldwide, Spotify needs a precise and meaningful database to store and manage its data. This data combined with powerful queries will convey the information necessary for Spotify statistics, analytics, and storage. The purpose of this database is to show the relations between users, artists, songs and more to better Spotify's analysis in the inner workings of its data, to arrange the data more meaningfully, and to provide precise service to its users. This database helps in cataloging payment information for premium users, holding playlist information, keeping logs of what the user is listening to, and more. In the following slides, one will find an outline of the Entity-Relationship Diagram, its functional dependencies, the code used to make it, and some sample data to show how it runs. The slides also delve into stored procedures, triggers, views, security, problems, and future enhancements of the Spotify database.

## Executive Summary

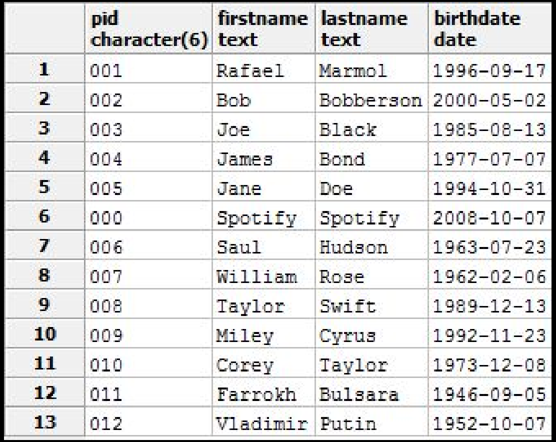


## Entity Relationship Diagram

**<u>PEOPLE:</u>** lists all people and basic attributes

```
CREATE TABLE People (
      PID char(6) NOT NULL UNIQUE,
      firstName text NOT NULL,
      lastName text NOT NULL,
      birthDate DATE NOT NULL,
      PRIMARY KEY (PID)
);
```

<u>Functional Dependencies</u>
PID -> firstName, lastName, birthDate

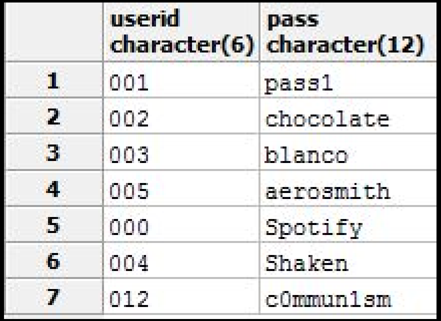


| | pid character(6) | firstname text | lastname text | birthdate date |
|---|---|---|---|---|
| 1 | 001 | Rafael | Marmol | 1996-09-17 |
| 2 | 002 | Bob | Bobberson | 2000-05-02 |
| 3 | 003 | Joe | Black | 1985-08-13 |
| 4 | 004 | James | Bond | 1977-07-07 |
| 5 | 005 | Jane | Doe | 1994-10-31 |
| 6 | 000 | Spotify | Spotify | 2008-10-07 |
| 7 | 006 | Saul | Hudson | 1963-07-23 |
| 8 | 007 | William | Rose | 1962-02-06 |
| 9 | 008 | Taylor | Swift | 1989-12-13 |
| 10 | 009 | Miley | Cyrus | 1992-11-23 |
| 11 | 010 | Corey | Taylor | 1973-12-08 |
| 12 | 011 | Farrokh | Bulsara | 1946-09-05 |
| 13 | 012 | Vladimir | Putin | 1952-10-07 |

**<u>USERS:</u>** lists all users and basic attributes

```
CREATE TABLE Users (
      UserID char(6) NOT NULL UNIQUE references people(PID),
      pass char(12) NOT NULL,
      PRIMARY KEY (UserID)
);
```

<u>Functional Dependencies</u>
UserID -> pass



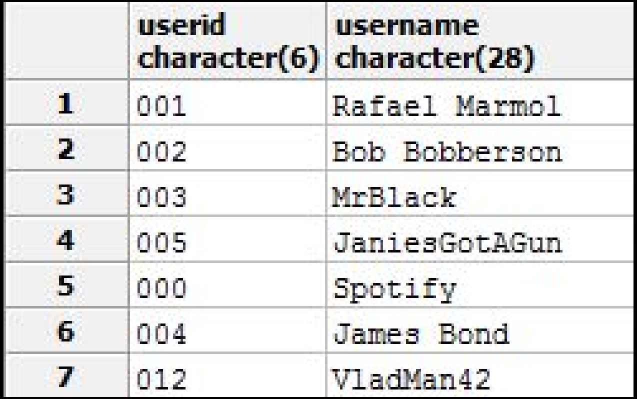
| | userid character(6) | pass character(12) |
|---|---|---|
| 1 | 001 | pass1 |
| 2 | 002 | chocolate |
| 3 | 003 | blanco |
| 4 | 005 | aerosmith |
| 5 | 000 | Spotify |
| 6 | 004 | Shaken |
| 7 | 012 | c0mmun1sm |

## Tables

**USERNAMES:** lists all usernames for user id

```
CREATE TABLE Usernames (
      UserID char(6) NOT NULL UNIQUE references Users(UserID),
      username char(28) NOT NULL UNIQUE,
      PRIMARY KEY (UserID)
);
```

Functional Dependencies
UserID -> username

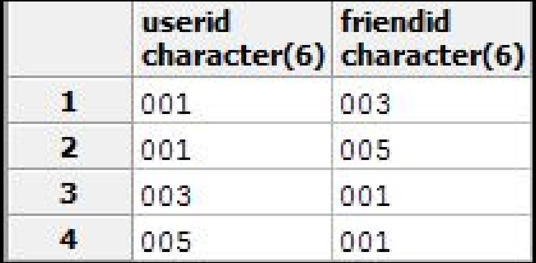


| | userid character(6) | username character(28) |
|---|---|---|
| 1 | 001 | Rafael Marmol |
| 2 | 002 | Bob Bobberson |
| 3 | 003 | MrBlack |
| 4 | 005 | JaniesGotAGun |
| 5 | 000 | Spotify |
| 6 | 004 | James Bond |
| 7 | 012 | VladMan42 |

**FRIENDS:** lists userid's and friendid's

```
CREATE TABLE Friends (
      UserID char(6) NOT NULL references Users(UserID),
      friendID char(6) NOT NULL references Usernames(UserID),
      PRIMARY KEY(UserID, friendID)
);
```

Functional Dependencies
UserID -> friendID

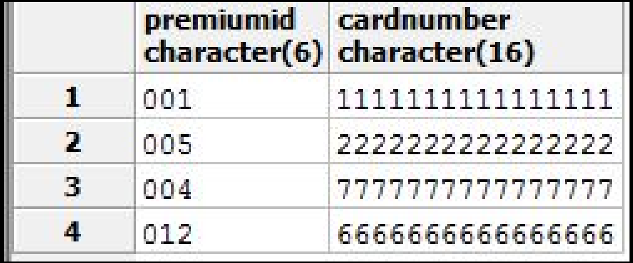


| | userid character(6) | friendid character(6) |
|---|---|---|
| 1 | 001 | 003 |
| 2 | 001 | 005 |
| 3 | 003 | 001 |
| 4 | 005 | 001 |

## Tables

**PREMIUM:** lists all premium users and attributes

```
CREATE TABLE Premium (
      PremiumID char(6) NOT NULL UNIQUE references Users(UserID),
      cardNumber char(16) NOT NULL UNIQUE,
      PRIMARY KEY (PremiumID)
);
```

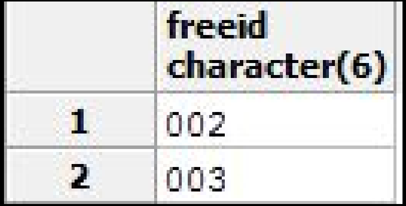


| | premiumid character(6) | cardnumber character(16) |
|---|---|---|
| 1 | 001 | 1111111111111111 |
| 2 | 005 | 2222222222222222 |
| 3 | 004 | 7777777777777777 |
| 4 | 012 | 6666666666666666 |

Functional Dependencies
PremiumID -> cardNumber

**FREE:** lists all people and basic attributes

```
CREATE TABLE Free (
      FreeID char(6) NOT NULL UNIQUE references Users(UserID),
      PRIMARY KEY(FreeID)
);
```



| | freeid character(6) |
|---|---|
| 1 | 002 |
| 2 | 003 |

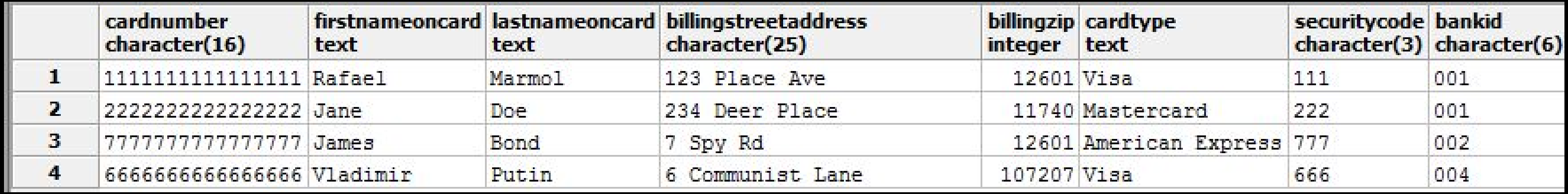
Functional Dependencies
FreeiD ->

# Tables

**CARDINFO:** lists all card information and attributes

```
CREATE TABLE CardInfo (
      cardNumber char(16) NOT NULL UNIQUE references Premium(cardNumber),
      firstNameOnCard text NOT NULL,
      lastNameOnCard text NOT NULL,
      billingStreetAddress char(25) NOT NULL,
      billingZip int NOT NULL,
      cardType text NOT NULL,
      securityCode char(3) NOT NULL,
      bankID char(6) NOT NULL,
      PRIMARY KEY (cardNumber)
);
```

Functional Dependencies
cardNumber -> firstNameOnCard, lastNameOnCard, billingStreetAddress, billingZip, cardType, securityCode, bankID



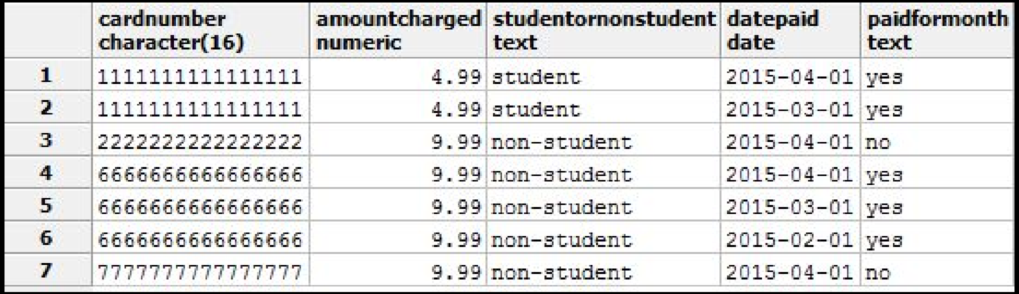
| | cardnumber character(16) | firstnameoncard text | lastnameoncard text | billingstreetaddress character(25) | billingzip integer | cardtype text | securitycode character(3) | bankid character(6) |
|---|---|---|---|---|---|---|---|---|
| 1 | 1111111111111111 | Rafael | Marmol | 123 Place Ave | 12601 | Visa | 111 | 001 |
| 2 | 2222222222222222 | Jane | Doe | 234 Deer Place | 11740 | Mastercard | 222 | 001 |
| 3 | 7777777777777777 | James | Bond | 7 Spy Rd | 12601 | American Express | 777 | 002 |
| 4 | 6666666666666666 | Vladimir | Putin | 6 Communist Lane | 107207 | Visa | 666 | 004 |

## Tables

**PAYMENTINFO:** lists card numbers and payment history

```
CREATE TABLE PaymentInfo (
      cardNumber char(16) NOT NULL references CardInfo(cardNumber),
      amountCharged decimal NOT NULL,
      studentOrNonStudent text NOT NULL,
      datePaid date NOT NULL,
      paidForMonth text NOT NULL,
      CONSTRAINT paidForMonth CHECK (paidForMonth = 'yes' OR paidForMonth = 'no'),
      CONSTRAINT studentOrNonStudent CHECK (studentOrNonStudent = 'student' OR studentOrNonStudent = 'non-
student'),
      PRIMARY KEY (cardNumber, datePaid)
);
```

Functional Dependencies
cardNumber-> amountCharged, studentOrNonStudent, datePaid, paidForMonth



| | cardnumber character(16) | amountcharged numeric | studentornonstudent text | datepaid date | paidformonth text |
|---|---|---|---|---|---|
| 1 | 1111111111111111 | 4.99 | student | 2015-04-01 | yes |
| 2 | 1111111111111111 | 4.99 | student | 2015-03-01 | yes |
| 3 | 2222222222222222 | 9.99 | non-student | 2015-04-01 | no |
| 4 | 6666666666666666 | 9.99 | non-student | 2015-04-01 | yes |
| 5 | 6666666666666666 | 9.99 | non-student | 2015-03-01 | yes |
| 6 | 6666666666666666 | 9.99 | non-student | 2015-02-01 | yes |
| 7 | 7777777777777777 | 9.99 | non-student | 2015-04-01 | no |

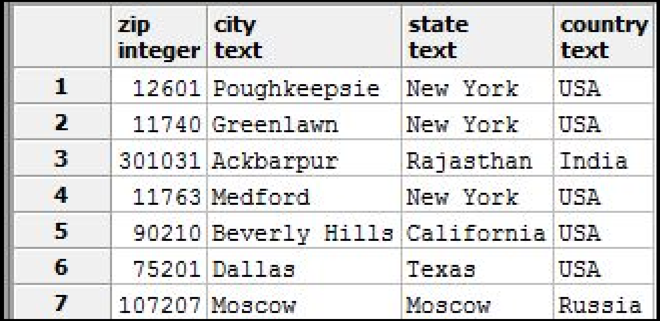
## Tables

**ZIP:** lists all zip codes and attributes

```
CREATE TABLE Zip (
      zip int NOT NULL,
      city text NOT NULL,
      state text NOT NULL,
      country text NOT NULL,
      PRIMARY KEY (zip)
);
```

Functional Dependencies

Zip -> city, state, country



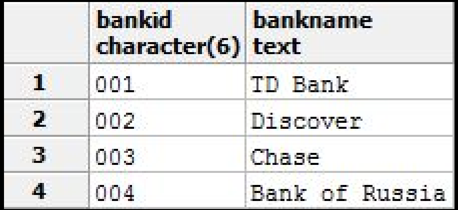
| | zip integer | city text | state text | country text |
|---|---|---|---|---|
| 1 | 12601 | Poughkeepsie | New York | USA |
| 2 | 11740 | Greenlawn | New York | USA |
| 3 | 301031 | Ackbarpur | Rajasthan | India |
| 4 | 11763 | Medford | New York | USA |
| 5 | 90210 | Beverly Hills | California | USA |
| 6 | 75201 | Dallas | Texas | USA |
| 7 | 107207 | Moscow | Moscow | Russia |

**BANK:** lists all banks

```
CREATE TABLE Bank (
      BankID char(6) NOT NULL UNIQUE,
      bankName text NOT NULL,
      PRIMARY KEY (BankID)
);
```

Functional Dependencies

BankID-> bankName



| | bankid character(6) | bankname text |
|---|---|---|
| 1 | 001 | TD Bank |
| 2 | 002 | Discover |
| 3 | 003 | Chase |
| 4 | 004 | Bank of Russia |

# Tables

**PLAYLISTS:** lists all playlists and basic attributes

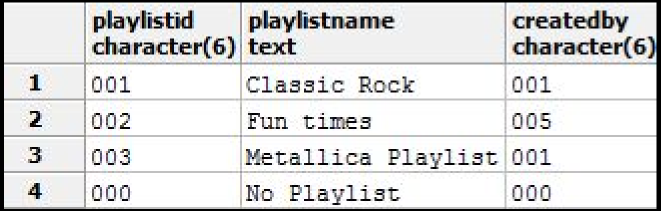
```
CREATE TABLE Playlists (
      PlaylistID char(6) NOT NULL UNIQUE,
      playlistName text NOT NULL,
      createdBy char(6) NOT NULL references Users(UserID),
      PRIMARY KEY(PlaylistID)
);
```

Functional Dependencies
PlaylistID -> playlistName, createdBy

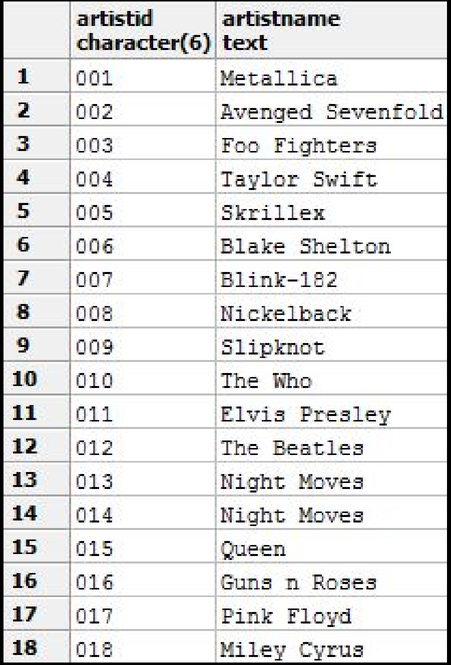
**ARTISTS:** lists all artists

```
CREATE TABLE Artists (
      ArtistID char(6) NOT NULL UNIQUE,
      artistName text NOT NULL,
      PRIMARY KEY (ArtistID)
);
```

Functional Dependencies
ArtistID -> artistName



| | playlistid<br>character(6) | playlistname<br>text | createdby<br>character(6) |
|---|---|---|---|
| 1 | 001 | Classic Rock | 001 |
| 2 | 002 | Fun times | 005 |
| 3 | 003 | Metallica Playlist | 001 |
| 4 | 000 | No Playlist | 000 |



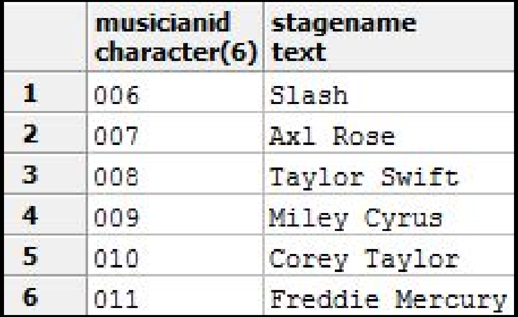
| | artistid<br>character(6) | artistname<br>text |
|---|---|---|
| 1 | 001 | Metallica |
| 2 | 002 | Avenged Sevenfold |
| 3 | 003 | Foo Fighters |
| 4 | 004 | Taylor Swift |
| 5 | 005 | Skrillex |
| 6 | 006 | Blake Shelton |
| 7 | 007 | Blink-182 |
| 8 | 008 | Nickelback |
| 9 | 009 | Slipknot |
| 10 | 010 | The Who |
| 11 | 011 | Elvis Presley |
| 12 | 012 | The Beatles |
| 13 | 013 | Night Moves |
| 14 | 014 | Night Moves |
| 15 | 015 | Queen |
| 16 | 016 | Guns n Roses |
| 17 | 017 | Pink Floyd |
| 18 | 018 | Miley Cyrus |

# Tables

**MUSICIANS:** lists musicians

```
CREATE TABLE Musicians (
      MusicianID char(6) NOT NULL UNIQUE references People(PID),
      stageName text NOT NULL,
      PRIMARY KEY (MusicianID)
);
```

Functional Dependencies
MusicianID -> stageName

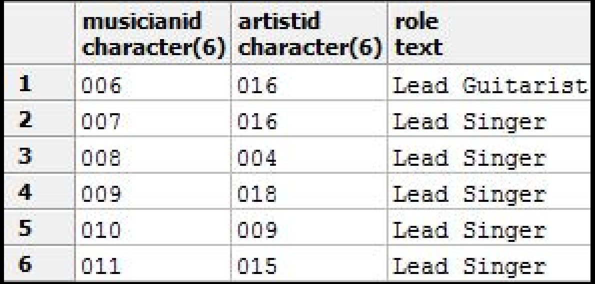


| | musicianid character(6) | stagename text |
|---|---|---|
| 1 | 006 | Slash |
| 2 | 007 | Axl Rose |
| 3 | 008 | Taylor Swift |
| 4 | 009 | Miley Cyrus |
| 5 | 010 | Corey Taylor |
| 6 | 011 | Freddie Mercury |

**PLAYSFOR:** lists who’s in a band and what their role is

```
CREATE TABLE PlaysFor (
      MusicianID char(6) NOT NULL references Musicians(MusicianID),
      ArtistID char(6) NOT NULL references Artists(ArtistID),
      role text NOT NULL,
      PRIMARY KEY (MusicianID, ArtistID)
);
```

Functional Dependencies
(MusicianID, ArtistID) -> role



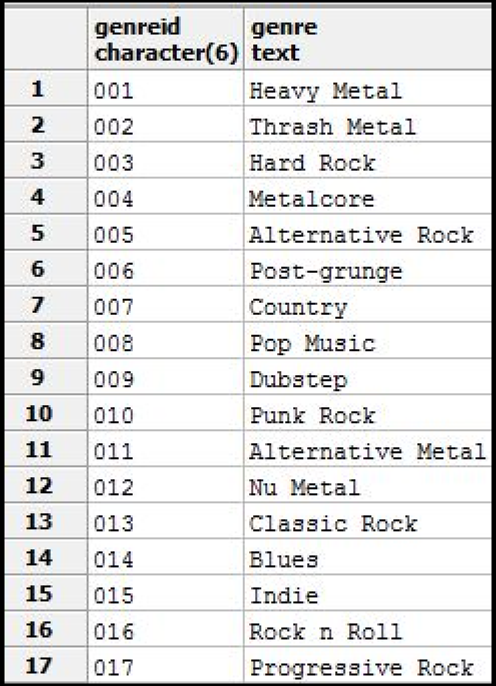
| | musicianid character(6) | artistid character(6) | role text |
|---|---|---|---|
| 1 | 006 | 016 | Lead Guitarist |
| 2 | 007 | 016 | Lead Singer |
| 3 | 008 | 004 | Lead Singer |
| 4 | 009 | 018 | Lead Singer |
| 5 | 010 | 009 | Lead Singer |
| 6 | 011 | 015 | Lead Singer |

## Tables

**GENRES:** lists all genres

```
CREATE TABLE Genres (
      GenreID char(6) UNIQUE NOT NULL,
      genre text NOT NULL,
      PRIMARY KEY(GenreID)
);
```

Functional Dependencies
GenreID -> genre



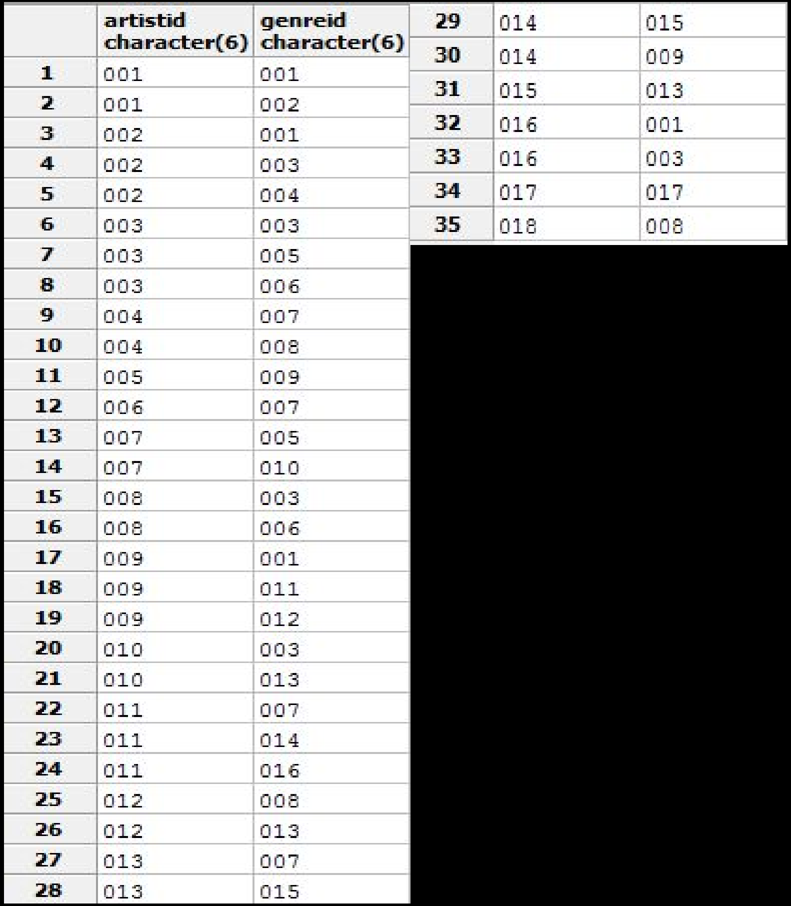
| | genreid character(6) | genre text |
|---|---|---|
| 1 | 001 | Heavy Metal |
| 2 | 002 | Thrash Metal |
| 3 | 003 | Hard Rock |
| 4 | 004 | Metalcore |
| 5 | 005 | Alternative Rock |
| 6 | 006 | Post-grunge |
| 7 | 007 | Country |
| 8 | 008 | Pop Music |
| 9 | 009 | Dubstep |
| 10 | 010 | Punk Rock |
| 11 | 011 | Alternative Metal |
| 12 | 012 | Nu Metal |
| 13 | 013 | Classic Rock |
| 14 | 014 | Blues |
| 15 | 015 | Indie |
| 16 | 016 | Rock n Roll |
| 17 | 017 | Progressive Rock |

## Tables

**BELONGSTO:** lists artistid’s and genreid’s

```
CREATE TABLE BelongsTo (
      ArtistID char(6) NOT NULL references Artists(ArtistID),
      GenreID char(6) NOT NULL references Genres(GenreID),
      PRIMARY KEY (ArtistID, GenreID)
);
```

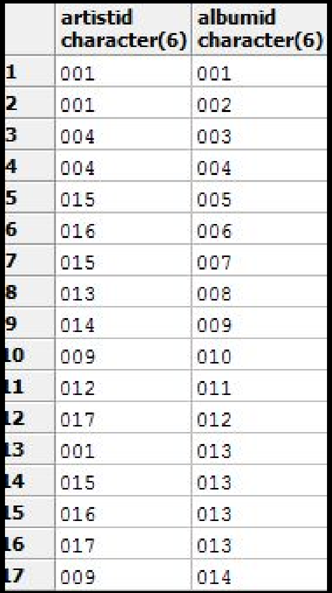
Functional Dependencies
(ArtistID, GenreID) ->



| | artistid character(6) | genreid character(6) |
|---|---|---|
| 1 | 001 | 001 |
| 2 | 001 | 002 |
| 3 | 002 | 001 |
| 4 | 002 | 003 |
| 5 | 002 | 004 |
| 6 | 003 | 003 |
| 7 | 003 | 005 |
| 8 | 003 | 006 |
| 9 | 004 | 007 |
| 10 | 004 | 008 |
| 11 | 005 | 009 |
| 12 | 006 | 007 |
| 13 | 007 | 005 |
| 14 | 007 | 010 |
| 15 | 008 | 003 |
| 16 | 008 | 006 |
| 17 | 009 | 001 |
| 18 | 009 | 011 |
| 19 | 009 | 012 |
| 20 | 010 | 003 |
| 21 | 010 | 013 |
| 22 | 011 | 007 |
| 23 | 011 | 014 |
| 24 | 011 | 016 |
| 25 | 012 | 008 |
| 26 | 012 | 013 |
| 27 | 013 | 007 |
| 28 | 013 | 015 |
| 29 | 014 | 015 |
| 30 | 014 | 009 |
| 31 | 015 | 013 |
| 32 | 016 | 001 |
| 33 | 016 | 003 |
| 34 | 017 | 017 |
| 35 | 018 | 008 |

**MAKES:** lists artistid's and albumid's

```
CREATE TABLE Makes (
      ArtistID char(6) NOT NULL references Artists(ArtistID),
      AlbumID char(6) NOT NULL references Albums(AlbumID),
      PRIMARY KEY (ArtistID, AlbumID)
);
```

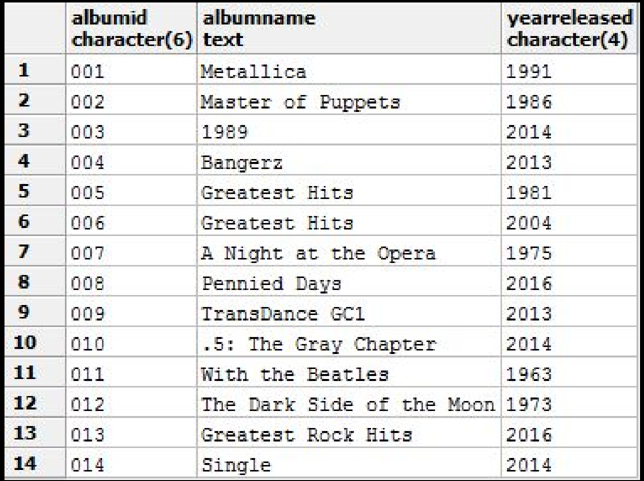


| | artistid character(6) | albumid character(6) |
|---|---|---|
| 1 | 001 | 001 |
| 2 | 001 | 002 |
| 3 | 004 | 003 |
| 4 | 004 | 004 |
| 5 | 015 | 005 |
| 6 | 016 | 006 |
| 7 | 015 | 007 |
| 8 | 013 | 008 |
| 9 | 014 | 009 |
| 10 | 009 | 010 |
| 11 | 012 | 011 |
| 12 | 017 | 012 |
| 13 | 001 | 013 |
| 14 | 015 | 013 |
| 15 | 016 | 013 |
| 16 | 017 | 013 |
| 17 | 009 | 014 |

Functional Dependencies
(ArtistID, AlbumID) ->

**ALBUMS:** lists all albums and basic attributes

```
CREATE TABLE Albums (
      AlbumID char(6) UNIQUE NOT NULL,
      albumName text NOT NULL,
      yearReleased char(4) NOT NULL
);
```



| | albumid character(6) | albumname text | yearreleased character(4) |
|---|---|---|---|
| 1 | 001 | Metallica | 1991 |
| 2 | 002 | Master of Puppets | 1986 |
| 3 | 003 | 1989 | 2014 |
| 4 | 004 | Bangerz | 2013 |
| 5 | 005 | Greatest Hits | 1981 |
| 6 | 006 | Greatest Hits | 2004 |
| 7 | 007 | A Night at the Opera | 1975 |
| 8 | 008 | Pennied Days | 2016 |
| 9 | 009 | TransDance GC1 | 2013 |
| 10 | 010 | .5: The Gray Chapter | 2014 |
| 11 | 011 | With the Beatles | 1963 |
| 12 | 012 | The Dark Side of the Moon | 1973 |
| 13 | 013 | Greatest Rock Hits | 2016 |
| 14 | 014 | Single | 2014 |

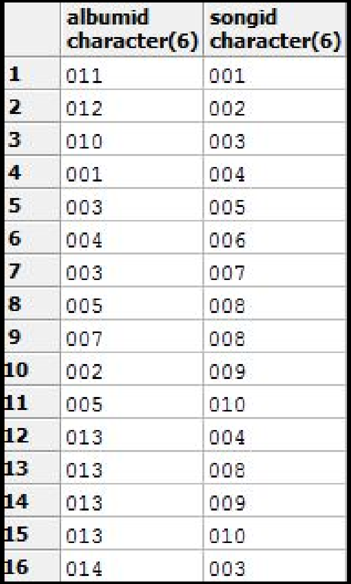
Functional Dependencies
AlbumID -> albumName, yearReleased

## Tables

**FEATURES:** lists albumid's and songid's

```
CREATE TABLE Features (
      AlbumID char(6) NOT NULL references Albums(AlbumID),
      SongID char(6) NOT NULL references Songs(SongID),
      PRIMARY KEY(AlbumID, SongID)
);
```

Functional Dependencies
(AlbumID, SongID) ->

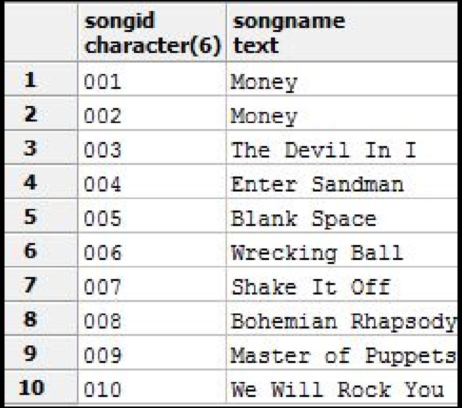


| | albumid character(6) | songid character(6) |
|---|---|---|
| 1 | 011 | 001 |
| 2 | 012 | 002 |
| 3 | 010 | 003 |
| 4 | 001 | 004 |
| 5 | 003 | 005 |
| 6 | 004 | 006 |
| 7 | 003 | 007 |
| 8 | 005 | 008 |
| 9 | 007 | 008 |
| 10 | 002 | 009 |
| 11 | 005 | 010 |
| 12 | 013 | 004 |
| 13 | 013 | 008 |
| 14 | 013 | 009 |
| 15 | 013 | 010 |
| 16 | 014 | 003 |

**SONGS:** lists songid and name

```
CREATE TABLE Songs(
      SongID char(6) UNIQUE NOT NULL,
      songName text NOT NULL,
      PRIMARY KEY(SongID)
);
```

Functional Dependencies
SongID -> songName



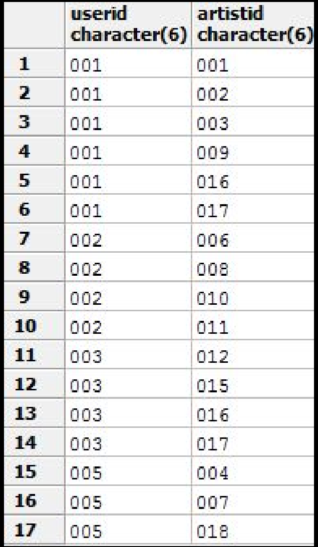
| | songid character(6) | songname text |
|---|---|---|
| 1 | 001 | Money |
| 2 | 002 | Money |
| 3 | 003 | The Devil In I |
| 4 | 004 | Enter Sandman |
| 5 | 005 | Blank Space |
| 6 | 006 | Wrecking Ball |
| 7 | 007 | Shake It Off |
| 8 | 008 | Bohemian Rhapsody |
| 9 | 009 | Master of Puppets |
| 10 | 010 | We Will Rock You |

## Tables

**FOLLOWS:** lists userid's and artistid's

```
CREATE TABLE Follows (
      UserID char(6) NOT NULL references Users(UserID),
      ArtistID char(6) NOT NULL references Artists(ArtistID),
      PRIMARY KEY(UserId, ArtistID)
);
```

Functional Dependencies
(UserID, ArtistID) ->

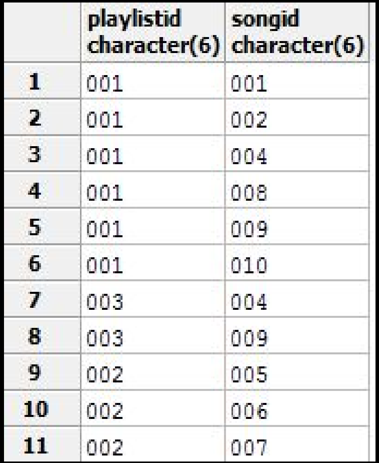


| | userid character(6) | artistid character(6) |
|---|---|---|
| 1 | 001 | 001 |
| 2 | 001 | 002 |
| 3 | 001 | 003 |
| 4 | 001 | 009 |
| 5 | 001 | 016 |
| 6 | 001 | 017 |
| 7 | 002 | 006 |
| 8 | 002 | 008 |
| 9 | 002 | 010 |
| 10 | 002 | 011 |
| 11 | 003 | 012 |
| 12 | 003 | 015 |
| 13 | 003 | 016 |
| 14 | 003 | 017 |
| 15 | 005 | 004 |
| 16 | 005 | 007 |
| 17 | 005 | 018 |

**CONTAINS:** lists playlistid's and songid's

```
CREATE TABLE Contains (
      PlaylistID char(6) NOT NULL references Playlists(PlaylistID),
      SongID char(6) NOT NULL references Songs(SongID),
      PRIMARY KEY(PlaylistID,SongID)
);
```

Functional Dependencies
(PlaylistID, SongID) ->



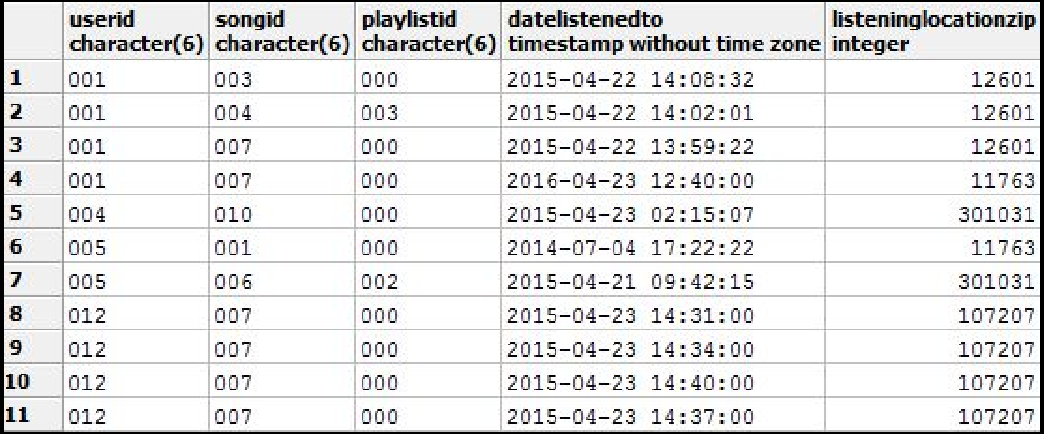
| | playlistid character(6) | songid character(6) |
|---|---|---|
| 1 | 001 | 001 |
| 2 | 001 | 002 |
| 3 | 001 | 004 |
| 4 | 001 | 008 |
| 5 | 001 | 009 |
| 6 | 001 | 010 |
| 7 | 003 | 004 |
| 8 | 003 | 009 |
| 9 | 002 | 005 |
| 10 | 002 | 006 |
| 11 | 002 | 007 |

## Tables

**LISTENINGLOG:** lists all userid's and listening attributes

```
CREATE TABLE ListeningLog (
      UserID char(6) NOT NULL references Users(UserID),
      SongID char(6) NOT NULL references Songs(SongID),
      PlaylistID char(6) NOT NULL references Playlists(PlaylistID),
      dateListenedTo timestamp NOT NULL,
      listeningLocationZip int NOT NULL references Zip(zip),
      PRIMARY KEY (UserID, dateListenedTo)
);
```

Functional Dependencies
(UserID, dateListenedTo) -> SongID, PlaylistID, listeningLocationZip

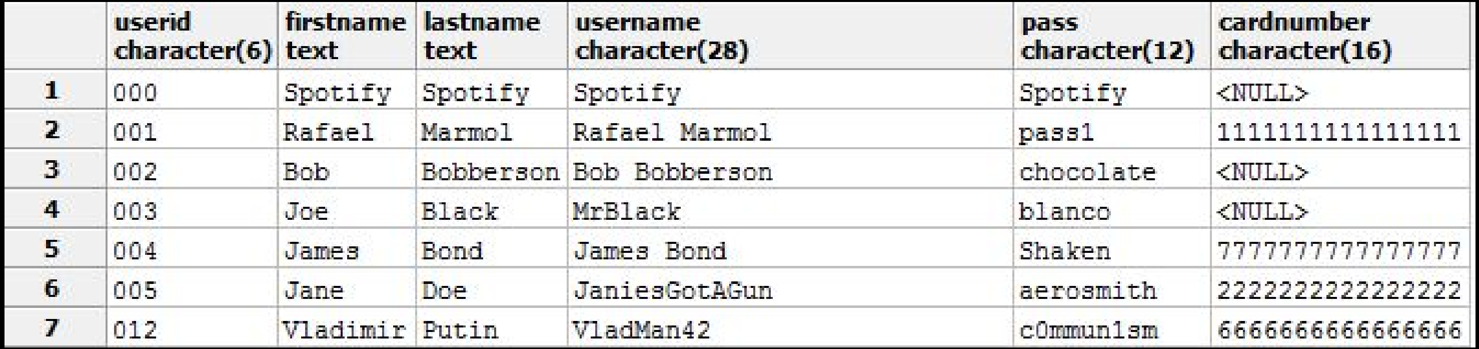


| | userid character(6) | songid character(6) | playlistid character(6) | datelistenedto timestamp without time zone | listeninglocationzip integer |
|---|---|---|---|---|---|
| 1 | 001 | 003 | 000 | 2015-04-22 14:08:32 | 12601 |
| 2 | 001 | 004 | 003 | 2015-04-22 14:02:01 | 12601 |
| 3 | 001 | 007 | 000 | 2015-04-22 13:59:22 | 12601 |
| 4 | 001 | 007 | 000 | 2016-04-23 12:40:00 | 11763 |
| 5 | 004 | 010 | 000 | 2015-04-23 02:15:07 | 301031 |
| 6 | 005 | 001 | 000 | 2014-07-04 17:22:22 | 11763 |
| 7 | 005 | 006 | 002 | 2015-04-21 09:42:15 | 301031 |
| 8 | 012 | 007 | 000 | 2015-04-23 14:31:00 | 107207 |
| 9 | 012 | 007 | 000 | 2015-04-23 14:34:00 | 107207 |
| 10 | 012 | 007 | 000 | 2015-04-23 14:40:00 | 107207 |
| 11 | 012 | 007 | 000 | 2015-04-23 14:37:00 | 107207 |

**<u>VIEWS:</u>** a query that is set to a specific name so you can call it by the name instead of typing out the whole query each time

**<u>VIEW:</u>** `UserInfo` shows list of users' names, usernames, passwords, and credit card numbers

```
CREATE OR REPLACE VIEW UserInfo as
select u.userid, firstName, lastName, username, pass, cardnumber
from users u INNER JOIN people p ON u.userid = p.pid
          LEFT OUTER JOIN premium pr ON u.userid = pr.premiumid
          LEFT OUTER JOIN free f ON u.userid = f.freeid
          INNER JOIN usernames n ON u.userid = n.userid;
```

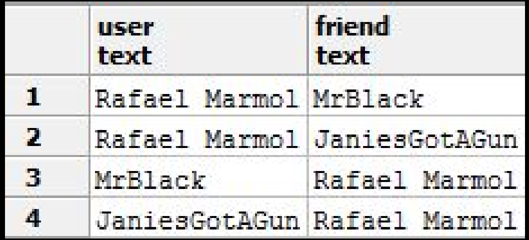


| | userid character(6) | firstname text | lastname text | username character(28) | pass character(12) | cardnumber character(16) |
|---|---|---|---|---|---|---|
| 1 | 000 | Spotify | Spotify | Spotify | Spotify | <NULL> |
| 2 | 001 | Rafael | Marmol | Rafael Marmol | pass1 | 1111111111111111 |
| 3 | 002 | Bob | Bobberson | Bob Bobberson | chocolate | <NULL> |
| 4 | 003 | Joe | Black | MrBlack | blanco | <NULL> |
| 5 | 004 | James | Bond | James Bond | Shaken | 7777777777777777 |
| 6 | 005 | Jane | Doe | JaniesGotAGun | aerosmith | 2222222222222222 |
| 7 | 012 | Vladimir | Putin | VladMan42 | c0mmun1sm | 6666666666666666 |

## Views

**VIEW:** `FriendsList` shows list of usernames and friends' usernames

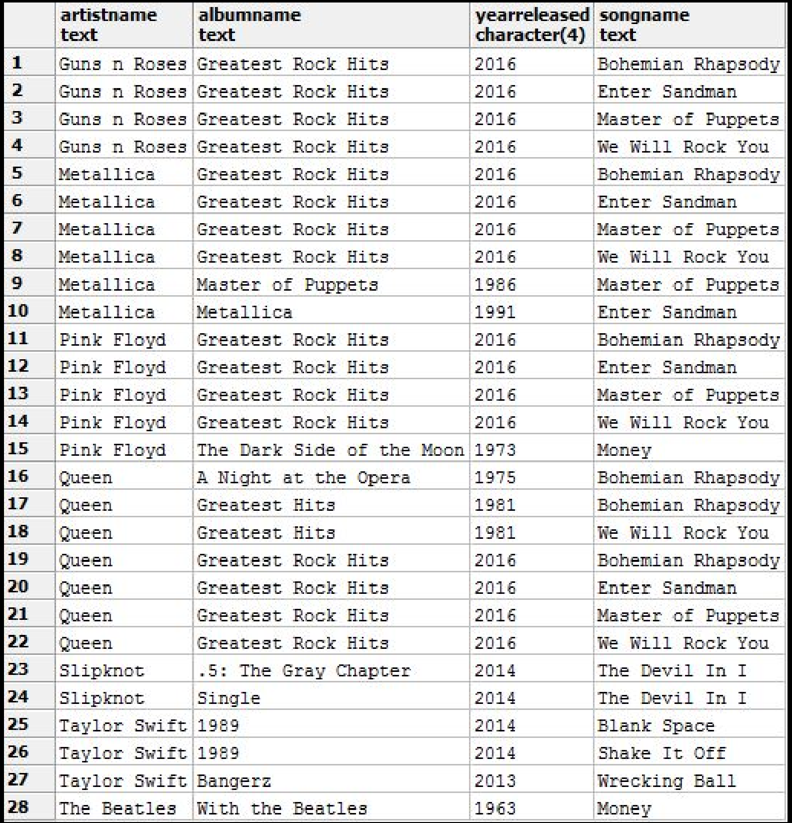
```
CREATE OR REPLACE VIEW FriendsList as
select showusernamefor(u.userid) as User, showusernamefor(friendid) as Friend
from friends f INNER JOIN users u ON f.userid = u.userid
          INNER JOIN usernames n ON n.userid = u.userid
ORDER BY User ASC;
```



| | user<br>text | friend<br>text |
|---|---|---|
| 1 | Rafael Marmol | MrBlack |
| 2 | Rafael Marmol | JaniesGotAGun |
| 3 | MrBlack | Rafael Marmol |
| 4 | JaniesGotAGun | Rafael Marmol |

**VIEW:** `MusicLibrary` shows list of artists, albums, years, and songs in database

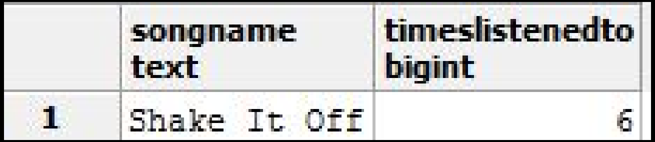
```
CREATE OR REPLACE VIEW MusicLibrary as
select distinct artistName, albumName, yearReleased, songName
from artists a INNER JOIN belongsto b ON b.artistid = a.artistid
            INNER JOIN genres g ON b.genreid = g.genreid
            INNER JOIN makes m ON m.artistid = a.artistid
            INNER JOIN albums al ON al.albumid = m.albumid
            INNER JOIN features f ON f.albumid = al.albumid
            INNER JOIN songs s ON s.songid = f.songid
ORDER BY artistName ASC;
```



| | artistname<br>text | albumname<br>text | yearreleased<br>character(4) | songname<br>text |
|---|---|---|---|---|
| 1 | Guns n Roses | Greatest Rock Hits | 2016 | Bohemian Rhapsody |
| 2 | Guns n Roses | Greatest Rock Hits | 2016 | Enter Sandman |
| 3 | Guns n Roses | Greatest Rock Hits | 2016 | Master of Puppets |
| 4 | Guns n Roses | Greatest Rock Hits | 2016 | We Will Rock You |
| 5 | Metallica | Greatest Rock Hits | 2016 | Bohemian Rhapsody |
| 6 | Metallica | Greatest Rock Hits | 2016 | Enter Sandman |
| 7 | Metallica | Greatest Rock Hits | 2016 | Master of Puppets |
| 8 | Metallica | Greatest Rock Hits | 2016 | We Will Rock You |
| 9 | Metallica | Master of Puppets | 1986 | Master of Puppets |
| 10 | Metallica | Metallica | 1991 | Enter Sandman |
| 11 | Pink Floyd | Greatest Rock Hits | 2016 | Bohemian Rhapsody |
| 12 | Pink Floyd | Greatest Rock Hits | 2016 | Enter Sandman |
| 13 | Pink Floyd | Greatest Rock Hits | 2016 | Master of Puppets |
| 14 | Pink Floyd | Greatest Rock Hits | 2016 | We Will Rock You |
| 15 | Pink Floyd | The Dark Side of the Moon | 1973 | Money |
| 16 | Queen | A Night at the Opera | 1975 | Bohemian Rhapsody |
| 17 | Queen | Greatest Hits | 1981 | Bohemian Rhapsody |
| 18 | Queen | Greatest Hits | 1981 | We Will Rock You |
| 19 | Queen | Greatest Rock Hits | 2016 | Bohemian Rhapsody |
| 20 | Queen | Greatest Rock Hits | 2016 | Enter Sandman |
| 21 | Queen | Greatest Rock Hits | 2016 | Master of Puppets |
| 22 | Queen | Greatest Rock Hits | 2016 | We Will Rock You |
| 23 | Slipknot | .5: The Gray Chapter | 2014 | The Devil In I |
| 24 | Slipknot | Single | 2014 | The Devil In I |
| 25 | Taylor Swift | 1989 | 2014 | Blank Space |
| 26 | Taylor Swift | 1989 | 2014 | Shake It Off |
| 27 | Taylor Swift | Bangerz | 2013 | Wrecking Ball |
| 28 | The Beatles | With the Beatles | 1963 | Money |

**VIEW:** `MostPopularSong` shows most popular song to date

```
CREATE OR REPLACE VIEW MostPopularSong as
select songName, count(songName) as timeslistenedto
from listeninglog l INNER JOIN songs s ON l.songid = s.songid
group by songName
order by count(songName) desc
limit 1;
```



|  | songname text | timeslistenedto bigint |
|---|---|---|
| 1 | Shake It Off | 6 |

**STORED PROCEDURES:** These are functions that can be utilized to create statements or make calculations instead of going through the hassle of writing/rewriting queries

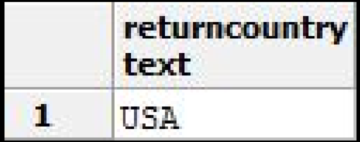
**1. STORED PROCEDURE:** `ReturnCountry` this automatically shows the country for a given zipcode (helps later with `PossibleCardTheft`)

```
create or replace function ReturnCountry(int) returns text as
$$
declare
   zip_input int       := $1;

begin
   return (
      select country
      from   zip
       where  zip = zip_input);

end;
$$
language plpgsql;
```

```
Select returnCountry(11763);
```



|   | returncountry text |
|---|---|
| 1 | USA |

**2. STORED PROCEDURE:** `lastDateListened` this automatically returns the most recent date a user has listened to something (helps later with `possibledeadusers` )

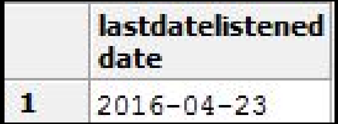
```
create or replace function lastDateListened(text) returns date as
$$
declare
   user_input text       := $1;

begin
   return (
      select datelistenedto
      from   listeninglog
      where userid = user_input
      order by datelistenedto DESC
      limit 1

       );

end;
$$
language plpgsql;
```

```
Select lastDateListened('001');
```



| | lastdatelistened date |
|---|---|
| 1 | 2016-04-23 |

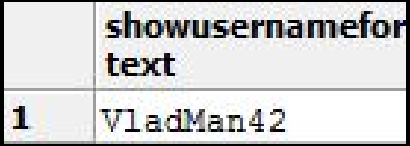
**<u>3. STORED PROCEDURE:</u>** `showUserNameFor(userid)` this automatically returns username for the given userid (helps later with `FriendsList` view)

```
create or replace function showUserNameFor(text) returns text as
$$
declare
   user_input text       := $1;

begin
   return (
      select username
      from   usernames
      where userid = user_input
       );

end;
$$
language plpgsql;
```

```
Select showUserNameFor('012');
```



| | showusernamefor<br>text |
|---|---|
| 1 | VladMan42 |

# Stored Procedures

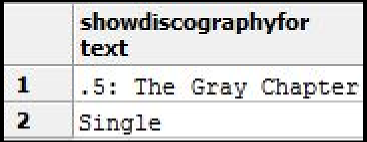
**4. STORED PROCEDURE:** `showDiscographyFor(artistName)` this automatically returns the given artist’s albums

```
create or replace function showDiscographyFor(text) returns setof text as
$$
declare
   artist_input text       := $1;

begin
   return query(
      select albumName
      from   makes m INNER JOIN artists a ON m.artistid = a.artistid
                INNER JOIN albums al ON m.albumid = al.albumid
      where artistName = artist_input
       );

end;
$$
language plpgsql;
```

```
Select showDiscographyFor(‘Slipknot’);
```



| | showdiscographyfor text |
|---|---|
| 1 | .5: The Gray Chapter |
| 2 | Single |

# Stored Procedures

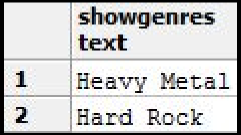
**5. STORED PROCEDURE:** `showGenres` this automatically returns a table of genres that the input artist falls under (helps later with `showRelatedArtistsFor` function)

```
create or replace function showGenres(text) returns setof text as
$$
declare
   artist_input text       := $1;

begin
   return query(
      select genre
      from   belongsTo b INNER JOIN artists a ON b.artistID = a.artistID
               INNER JOIN genres g ON b.genreID = g.genreID
     where artistName = artist_input
       );

end;
$$
language plpgsql;
```

```
Select showGenres('Guns n Roses');
```



| | showgenres text |
|---|---|
| 1 | Heavy Metal |
| 2 | Hard Rock |

# Stored Procedures

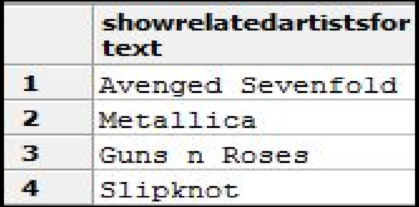
**6. STORED PROCEDURE:** `showRelatedArtistsFor` this automatically returns a table of artists that fall under the same genres as the input artist

```
create or replace function showRelatedArtistsFor(text) returns setof text as
$$
declare
   artist_input text       := $1;

begin
   return query(
      select distinct artistName
      from   belongsTo b INNER JOIN artists a ON b.artistID = a.artistID
                INNER JOIN genres g ON b.genreID = g.genreID
       where  genre IN (select showgenres(artist_input))
       );

end;
$$
language plpgsql;
```

```
Select showRelatedArtistsFor('Metallica');
```



| | showrelatedartistsfor text |
|---|---|
| 1 | Avenged Sevenfold |
| 2 | Metallica |
| 3 | Guns n Roses |
| 4 | Slipknot |

## Stored Procedures

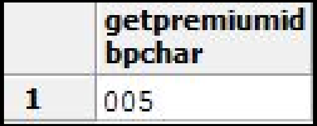
**7. STORED PROCEDURE:** `getPremiumID(char)` this automatically returns the premiumid that matches the cardnumber in paymentinfo (used later in `checkPayment()` )

```
create or replace function getpremiumid(character) returns character as
$$
declare
   char_input character(16)       := $1;

begin
   return (select distinct premiumid
          from premium p INNER JOIN cardinfo c ON p.cardnumber = c.cardnumber
                    INNER JOIN paymentinfo pi ON c.cardnumber = pi.cardnumber
               WHERE pi.cardnumber = char_input);

end;
$$
language plpgsql;
```

```
Select getpremiumid('1111111111111111');
```



|   | getpremiumid bpchar |
|---|---|
| 1 | 005 |

**8. STORED PROCEDURE:** `checkPayment` this automatically deletes a premium user and information from premium and adds them as a free user if they have not paid for the month (sample on trigger)

```
CREATE OR REPLACE FUNCTION checkPayment() RETURNS trigger AS
$$
declare
yn text;
card char(16);
id char(6);
    BEGIN

        IF TG_OP = 'INSERT' then
        yn = NEW.paidformonth;
        card = NEW.cardnumber;
        id = (Select getpremiumid(card));

            IF (yn = 'no') THEN
                DELETE FROM paymentinfo
                WHERE cardnumber = card;
                DELETE FROM cardinfo
                WHERE cardnumber = card;
                DELETE FROM premium
                WHERE premiumid = id;
                INSERT INTO free (freeid)
                VALUES (id);
                return new;
                END IF;
            END IF;
        return null;

    END;
$$
    LANGUAGE plpgsql;
```

**9. STORED PROCEDURE:** `replaceBandMember` this automatically deletes a member in the band that has the same role as the member being added (sample on trigger)

```
CREATE OR REPLACE FUNCTION replaceBandMember() RETURNS trigger AS
$$
    BEGIN
        IF TG_OP = 'INSERT' then

            IF ( NEW.role IN (select role from playsfor where artistid = NEW.artistID) ) then
            DELETE FROM playsfor
            WHERE artistid = NEW.artistid
            AND role = NEW.role
            AND musicianid != NEW.musicianid;
            return new;
            END IF;


        END IF;
        return null;

    END;
$$
    LANGUAGE plpgsql;
```

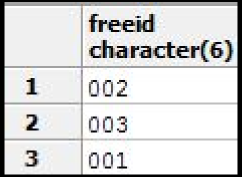
# Stored Procedures

**TRIGGERS:** activated at insert, update, or delete and runs a specific function

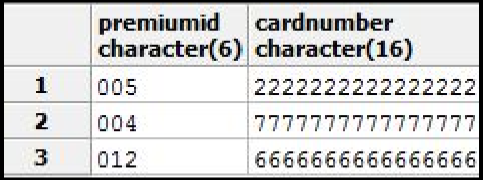
TRIGGER: `checkPayment` whenever a premium user has not paid for the month, they are deleted from premium and added to free

```
CREATE TRIGGER checkPayment AFTER INSERT OR UPDATE ON paymentinfo
    FOR EACH ROW EXECUTE PROCEDURE checkPayment();
```

```
INSERT INTO PaymentInfo (cardNumber, amountCharged, studentOrNonStudent, datepaid, paidformonth)
     VALUES('1111111111111111', '9.99', 'non-student', '2016-04-01', 'no');
```

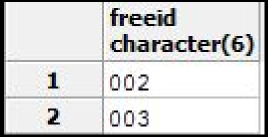


| | freeid character(6) |
|---|---|
| 1 | 002 |
| 2 | 003 |
| 3 | 001 |

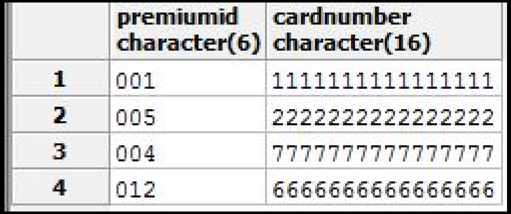


| | premiumid character(6) | cardnumber character(16) |
|---|---|---|
| 1 | 005 | 2222222222222222 |
| 2 | 004 | 7777777777777777 |
| 3 | 012 | 6666666666666666 |

**BEFORE**



| | freeid character(6) |
|---|---|
| 1 | 002 |
| 2 | 003 |



| | premiumid character(6) | cardnumber character(16) |
|---|---|---|
| 1 | 001 | 1111111111111111 |
| 2 | 005 | 2222222222222222 |
| 3 | 004 | 7777777777777777 |
| 4 | 012 | 6666666666666666 |

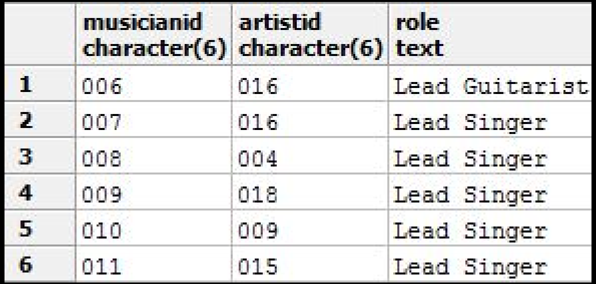
**AFTER**

## Triggers

TRIGGER: `replaceBandMember` whenever a member is added to the band that has the same role as an older member, they are replaced by the new one

```
CREATE TRIGGER replacebandmember AFTER INSERT ON playsfor
    FOR EACH ROW EXECUTE PROCEDURE replacebandmember();
```

```
INSERT INTO playsfor (musicianid, artistid, role)
VALUES ('010', '016' ,'Lead Singer');
```

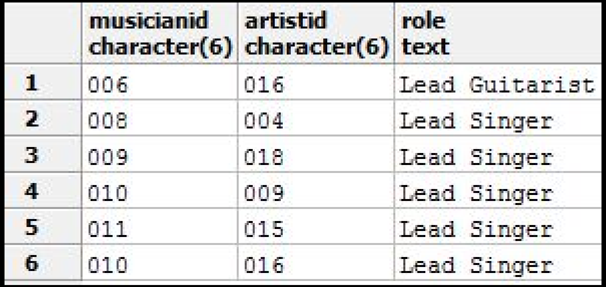


| | musicianid character(6) | artistid character(6) | role text |
|---|---|---|---|
| 1 | 006 | 016 | Lead Guitarist |
| 2 | 007 | 016 | Lead Singer |
| 3 | 008 | 004 | Lead Singer |
| 4 | 009 | 018 | Lead Singer |
| 5 | 010 | 009 | Lead Singer |
| 6 | 011 | 015 | Lead Singer |

<- is replaced

**BEFORE**

With ->



| | musicianid character(6) | artistid character(6) | role text |
|---|---|---|---|
| 1 | 006 | 016 | Lead Guitarist |
| 2 | 008 | 004 | Lead Singer |
| 3 | 009 | 018 | Lead Singer |
| 4 | 010 | 009 | Lead Singer |
| 5 | 011 | 015 | Lead Singer |
| 6 | 010 | 016 | Lead Singer |

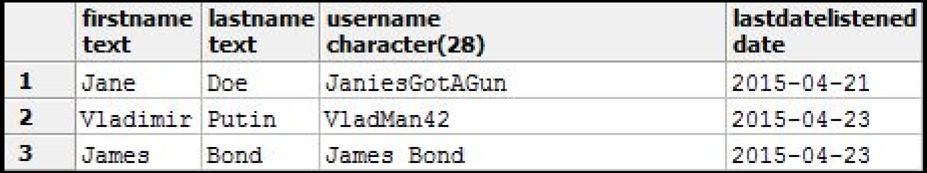
**AFTER**

## Triggers

**REPORTS:** Interesting Queries - shown below are queries that exemplify the true analytical power of the database

1. Query to show list of users who most likely aren't using Spotify anymore (checks if user's last listening date is >= 1 year and if so, they are listed)

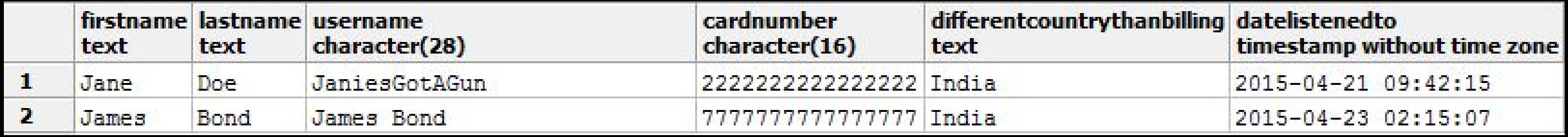
```
SELECT distinct firstName, lastName, username, lastDateListened(u.userid)
FROM listeninglog l INNER JOIN Users u ON l.userid = u.userid
          INNER JOIN people p ON p.pid = u.userid
          INNER JOIN usernames n ON u.userid = n.userid
WHERE (DATE_PART('year', current_date::date) - DATE_PART('year', lastdatelistened(l.
userid)::date)) >= 1
AND (DATE_PART('month', current_date::date) - DATE_PART('month', lastdatelistened(l.
userid)::date)) >= 0
AND (DATE_PART('month', current_date::date) - DATE_PART('month', lastdatelistened(l.
userid)::date)) <= 11;
```



| | firstname text | lastname text | username character(28) | lastdatelistened date |
|---|---|---|---|---|
| 1 | Jane | Doe | JaniesGotAGun | 2015-04-21 |
| 2 | Vladimir | Putin | VladMan42 | 2015-04-23 |
| 3 | James | Bond | James Bond | 2015-04-23 |

2. Query that shows list of users and credit cards that might be stolen. Compares country of billing zip code to that of the listening location zip code and if they're different, than it comes up as possily stolen)

```
CREATE OR REPLACE VIEW PossibleCardTheft AS
SELECT firstname, lastname, username, p.cardNumber, ReturnCountry(listeningLocationZip)
as DifferentCountryThanBilling, datelistenedTo
FROM listeninglog l  INNER JOIN users u ON l.userid = u.userid
          INNER JOIN premium p ON u.userid = p.premiumid
          INNER JOIN cardinfo c ON p.cardNUmber = c.cardnumber
          INNER JOIN people pl ON pl.pid = u.userid
          INNER JOIN usernames n ON n.userid = u.userid
WHERE ReturnCountry(listeningLocationZip) != ReturnCountry(billingZip)
```



| | firstname text | lastname text | username character(28) | cardnumber character(16) | differentcountrythanbilling text | datelistenedto timestamp without time zone |
|---|---|---|---|---|---|---|
| 1 | Jane | Doe | JaniesGotAGun | 2222222222222222 | India | 2015-04-21 09:42:15 |
| 2 | James | Bond | James Bond | 7777777777777777 | India | 2015-04-23 02:15:07 |

**SECURITY:** grants specific select, insert, update, delete commands to different users

**DATABASE ADMIN** - can change, update, and maintain database

```
create role database_admin
grant select,insert,update on all tables in schema public
to database_admin
```

**PREMIUM USER** - can view artists, albums, and songs; can change their card information; can add/delete friends, playlists, following

```
create role premium_user
grant select paymentinfo, friends, usernames, playlists, artists, albums, songs
to premium_user
grant update cardinfo
to premium_user
grant insert, update, delete friends, follows, playlists, contains
to premium_user
```

**FREE USER** - can view artists, albums, and songs; can add/delete friends, playlists, following

```
create role free_user
grant select friends, usernames, playlists, artists, albums, songs
to free_user
grant insert, update, delete friends, follows, playlists, contains
to free_user
```

## Security

The implementation went well with only few minor issues. The sample data used was generic, but held up extremely well with playing around with some interesting queries. One issue would probably be not accounting for free users having to listen to advertisements and premium users not having to. Also, when it comes to showing which artist made the song on an album with multiple artists, each song shows every collaborator as its artist instead of its particular one. In addition, these problems could be fixed with future enhancements by perhaps making another ads entity and directly connecting songs to artists as well without having to many artistID’s throughout the other tables. One would also want to populate the database with much more data to implement it to its full potential in future use. Overall, I am satisfied with the database I created and I believe it holds up in real world application.

## Implementation Notes/Known Problems/Future Enhancements