IMAGIMIND
STUDIOS

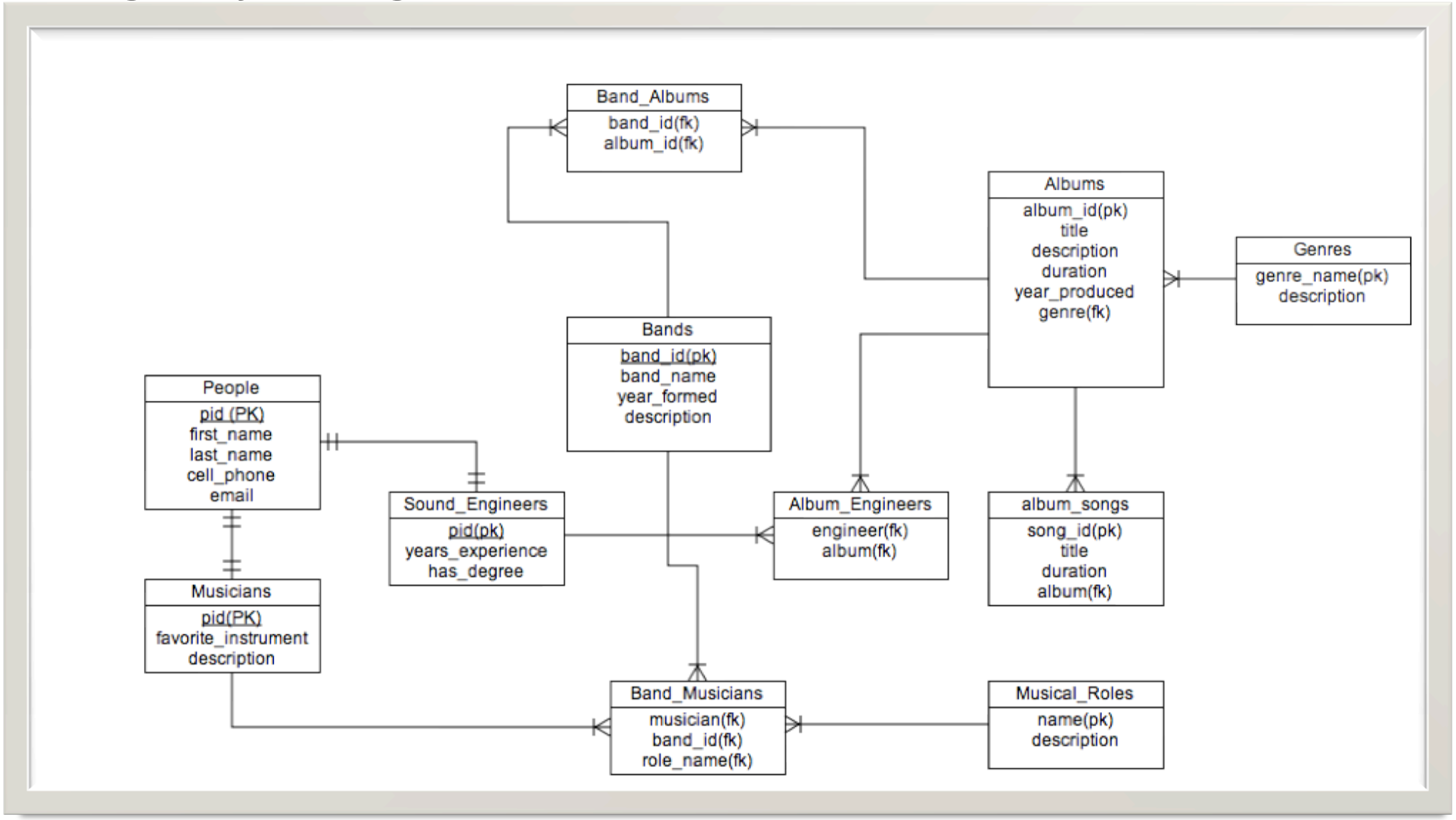
Andrew Langford, November 31 2013.

## Table of Contents

## ***Executive Summary***

Imagimind Studios is a limited liability corporation and recording studio stationed in Danbury, Connecticut. Their business revolves around recording and mixing music to produce albums. The purpose of this document is to conceptualize the database design for Imagimind Studios as well as describe its implementation in full. The Imagimind Studios database will store all data that is related to their central recording studio, including bands who produce music, engineers who record and mix their music, and the albums they collaboratively produce.  Starting with an ER diagram that represents the database on a high level, the document will describe each table in this diagram, functional dependencies in each table, and all the data collected to date. The procedures and reports are designed to allow for these individuals to obtain information regarding music, musicians, and engineers. Security has been implemented to make sure only administrators can create and update data in their respective authorization, while allowing anyone else to read data.

## ***ER Diagram for Imagimind Studios***



## ***Table Descriptions***

## ***People Table***

This table contains a list of all people working with Imagimind Studios, as well as their basic descriptive and contact information. The two subtypes of this table are musicians and sound engineers, therefore a musician can be a sound engineer and vice versa. The primary key is pid (people id).

**Functional Dependencies:**

pid => first_name, last_name, gender, cell_phone, email

**Create Table Statement:**

```
create table People(
pid char(8) not null,
first_name char(25) not null,
last_name char(25) not null,
gender char(1) check(gender = 'M' or gender = 'F'),
cell_phone char(12) not null,
email char(50) not null,
primary key(pid)
);
```

**Sample Data:**

| pid | first_name | last_name | gender | cell_phone | email |
|---|---|---|---|---|---|
| pid00001 | Jimi | Hendrix | M | 203-876-4435 | Experience@gmail.com |
| pid00002 | Robert | Shaw | M | 866-789-0989 | ofthebruce@aol.com |
| pid00003 | Tara | Noto | F | 233-477-9888 | namnam@hotmail.com |
| pid00004 | Sam | Rivot | M | 788-908-7656 | dilby@yahoo.com |
| pid00005 | Paul | Michaels | M | 765-976-0987 | semperfi@gmail.com |
| pid00006 | Samantha | Wilbert | F | 566-123-8767 | Wilbies4lyfe@marist.edu |
| pid00007 | Kyle | Ledoux | M | 203-766-8989 | ledouche@gmail.com |
| pid00008 | Stephanie | Haviland | F | 866-766-9765 | beauty@purchase.edu |
| pid00009 | Rebecca | Timman | F | 983-986-2172 | perogi@jwu.edu |
| pid00010 | John | Travolta | M | 184-826-9275 | ferret@yahoo.com |
| pid00011 | Bromhilde | Schnitzel | F | 178-928-9279 | weiseen@gmail.com |
| pid00012 | Candice | Sheronda | F | 878-964-9876 | datfooty@hotmail.com |
| pid00013 | Raqueem | Finagles | M | 826-927-7564 | rackemup@aol.com |
| pid00014 | Yolo | Yoyo | M | 827-277-8978 | Yoyofroyo@gmail.com |

## ***Musicians Table***

This table contains a list of all Musicians who have worked on albums at Imagimind studios. Each row describes musical preferences and a short description of a musician. Note that this is an entity subtype of People, therefore inherits all attributes of that table. The primary key is pid (people id).

**Functional Dependencies:**

pid => favorite_instrument, description

**Create Table Statement:**

```
create table Musicians(
pid char(8) not null,
favorite_instrument char(20) default('vocals'),
description text not null,
primary key(pid),
foreign key(pid) references people(pid)
)
```

**Sample Data:**

| pid | favorite_instrument | description |
|---|---|---|
| pid00014 | Violin | Classically trained concert violinist. Enjoys crumb cakes |
| pid00009 | Guitar | Expert Slide Guitarist |
| pid00013 | Percussion | Best Drummer in the Business |
| pid00008 | Vocals | Enjoys the sound of her natural instrument |
| pid00005 | Bass Guitar | Awful |
| pid00003 | Keyboard | All around talented musician |
| pid00001 | Guitar | Excellent lead and rhythm guitarist |

## ***Sound_Engineers Table***

The Sound_Engineer table contains a list of all engineers who record and mix music to produce the albums at Imagimind studios. It contains information about each engineer's years of experience in the field as well as whether or not they have obtained a degree in the field. Note that this is also an entity subtype of People, therefore inherits all attributes of that table. The primary key is pid (people identification)

**Functional Dependencies:**

pid => years_experience, has_degree

**Create Table Statement:**

```
create table sound_engineers(
pid char(8) not null,
years_experience int default(0),
has_degree boolean default('F'),
primary key(pid),
foreign key(pid) references people(pid)
);
```

**Sample Data:**

| pid | years_experience | has_degree |
|---|---:|---|
| pid00014 | 12 | t |
| pid00005 | 4 | f |
| pid00006 | 9 | t |
| pid00004 | 3 | f |
| pid00012 | 20 | t |

## ***Bands Table***

The Bands table contains a list of all the bands who have ever recorded an album here at Imagimind studios. Each row contains basic descriptive information about the band such as their name, year formed, and a musical description of the  band. The primary key is band_id.

**Functional Dependencies:**

band_id =>  band_name, year_formed, description

**Create Table Statement:**

```
create table Bands(
band_id char(8) not null,
band_name char(50) not null,
year_formed int not null,
description text not null,
primary key(band_id)
);
```

**Sample data:**

| band_id | band_name | year_formed | description |
|---|---|---|---|
| bid00001 | Flounder | 1992 | classically inspired band with bluegrass roots |
| bid00002 | Further | 1998 | Oldschool rockin good for the soul |
| bid00003 | Born to Be | 1997 | Native tongued naeive scoundrels |

## ***Musical Roles Table***

The Musical Roles Table contains a list of specific roles and descriptions a musician would play in a band such as Guitarist, Pianist, etc.  This table allows for each Musician to take on multiple roles in a band, since it is not uncommon for one musician to play multiple instruments on a single track. The primary key is name.

**Functional dependencies:**

Name => description

**Create Table Statement:**

```
create table Musical_Roles(
name char(25) not null,
description text not null,
primary key(name)
);
```

**Sample data:**

| **Name** | **description** |
|---|---|
| **Guitarist** | **Electric/Acoustic/Classical Guitarist. Anyone with a 6 string guitar** |
| **Pianist** | **One who specializes on the keys** |
| **vocalist** | **One skilled in vocal arts** |
| **Percussion** | **One who enjoys striking objects to create rhythmic pulse** |
| **Bassist** | **One who is....** |

## ***Band Musicians Table***

This table serves as the connection between bands and the musicians in those bands, as well as the role they play in said band

**Create Table Statement:**

```
create table Band_Musicians(
musician char(8) not null,
band_id char(8) not null,
role_name char(25) not null,
primary key (musician,band_id,role_name),
foreign key (musician) references Musicians(pid),
foreign key(band_id) references Bands(band_id),
foreign key (role_name) references Musical_Roles(name)
);
```

**Sample Data:**

| musician | band_id | role_name |
|---|---|---|
| pid00001 | bid00001 | Guitarist |
| pid00014 | bid00001 | vocalist |
| pid00013 | bid00001 | Percussion |
| pid00003 | bid00001 | Bassist |
| pid00014 | bid00002 | Pianist |
| pid00001 | bid00002 | vocalist |
| pid00001 | bid00002 | Guitarist |
| pid00008 | bid00002 | Bassist |
| pid00009 | bid00002 | Percussion |
| pid00005 | bid00003 | Guitarist |
| pid00005 | bid00003 | Bassist |
| pid00003 | bid00003 | Percussion |
| pid00013 | bid00003 | vocalist |

## ***Genres Table***

The Genres table contains a list of all musical genres which are currently being produced or have been produced at the studio, as well as a short description of that genre. The description is included since many new genres can be created with the fusion of one or more different genres. The possibilities are endless. If a new type of album is produced that falls under a different genre, a new record will be created. The primary key it the genre_name.

**Functional Dependencies:**

Genre_name => description

**Create Table Statement:**

```
create table Genres(
genre_name varchar(25) not null,
description text not null,
primary key(genre_name)
);
```

**Sample Data:**

| genre_name | description |
|---|---|
| Rock | Classic Rock through post hardcore rock |
| Reggae | Upbeat Island music |
| Jazz | Mixture of African and European classical styles |
| Country | Direct rip off of Blues |
| Blues | The only original American music |

## ***Albums Table***

The Albums table stores all information about albums produced at Imagimind Studios. The duration column is measured in hours:minutes:seconds. The primary key is album_id.

**Functional Dependencies:**

album_id => title, description, duration,  year_produced, genre

**Create Table Statement:**

```
create table Albums(
album_id char(8) not null,
title char(50) not null,
description text not null,
duration interval default('00:00:00'),
year_produced int not null,
genre varchar(25) not null,
primary key(album_id),
foreign key(genre) references Genres(genre_name)
);
```

**Sample data:**

| album_id | title | description | duration | year_produced | genre |
|---|---|---|---|---|---|
| alb00001 | Snow Blues | compilation of old timey blues tunes | 01:30:09 | 2003 | Blues |
| alb00002 | Hedge Knight | Intense, heavy, soulful | 02:30:14 | 2004 | Rock |
| alb00003 | Glade | Relaxing tunes for the bathtub | 04:24:18 | 1998 | Jazz |
| alb00004 | Cyrus | Terrible hillbilly blasphemy | 01:13:19 | 2013 | Country |
| alb00005 | The Experience | Good Workout Music | 03:12:07 | 1997 | Rock |
| alb00006 | Wonderous | Name says it all | 01:44:02 | 2001 | Jazz |
| alb00007 | Hey You | With the Face! | 01:40:17 | 1989 | Jazz |

## Band_Albums Table

This table serves as the connection between the albums produced and the bands that were recorded to make those albums.

**Create table statement:**

```
create table Band_Albums(
band_id char(8) not null,
album_id char(8) not null,
primary key(band_id,album_id),
foreign key(band_id) references Bands(band_id),
foreign key(album_id) references Albums(album_id)
);
```

**sample data:**

| band_id | album_id |
|---|---|
| bid00001 | alb00007 |
| bid00001 | alb00001 |
| bid00002 | alb00002 |
| bid00003 | alb00003 |
| bid00003 | alb00004 |
| bid00003 | alb00005 |
| bid00002 | alb00006 |

## ***Album Engineer Table***

This table serves as a connection between the albums produced and the engineers who produced them.

**Create Table Statement:**

```
create table Album_Engineers(
engineer char(8) not null,
album char(8) not null,
primary key(engineer,album),
foreign key(engineer) references sound_engineers(pid),
foreign key(album) references albums(album_id)
);
```

**Sample Data:**

| engineer | album |
|---|---|
| pid00014 | alb00001 |
| pid00014 | alb00002 |
| pid00005 | alb00001 |
| pid00006 | alb00001 |
| pid00004 | alb00002 |
| pid00006 | alb00003 |
| pid00014 | alb00003 |
| pid00005 | alb00004 |
| pid00012 | alb00005 |
| pid00006 | alb00006 |
| pid00012 | alb00006 |
| pid00006 | alb00007 |

## ***Album Songs Table***

This table stores all descriptive attributes about the songs that are contained in each album. The primary key is song_id.

**Create Table Statement**

```
create table album_songs(
song_id char(8) not null,
title varchar(25) not null,
track_num int not null,
duration interval default('00:00:00'),
album char(8) not null,
primary key(song_id),
foreign key(album) references Albums(album_id)
);
```

**Sample Data:**

| song_id | title | track_num | duration | album |
|---|---|---|---|---|
| sng00001 | Cray | 1 | 00:22:02 | alb00001 |
| sng00002 | Fragment | 2 | 00:44:06 | alb00001 |
| sng00003 | Snarls | 3 | 00:04:33 | alb00001 |
| sng00004 | Attitude | 1 | 00:44:22 | alb00002 |
| sng00005 | Too Late | 2 | 00:33:13 | alb00002 |
| sng00006 | Hopeless | 3 | 00:22:33 | alb00002 |
| sng00007 | Tiger Paws | 1 | 00:56:12 | alb00003 |
| sng00008 | Voltage | 1 | 01:09:22 | alb00004 |
| sng00009 | Harlot | 2 | 00:14:37 | alb00004 |
| sng00010 | Beastly | 3 | 00:55:51 | alb00004 |
| sng00011 | Limber | 1 | 00:11:19 | alb00005 |
| sng00012 | Alhoo | 2 | 00:12:14 | alb00005 |
| sng00013 | Grandpa Bones | 3 | 00:44:46 | alb00005 |
| sng00014 | Hoodinis Bra | 1 | 00:13:14 | alb00006 |
| sng00015 | Frolic | 2 | 00:44:56 | alb00006 |
| sng00016 | Endless | 1 | 03:44:46 | alb00007 |

## ***Views***

Here are some sample views.

**Jazz Guitarists**

```
create view Jazz_Guitarists as
select first_name as Jazz_Guitarist_First, last_name as Jazz_Guitarist_Last
from People
where pid in(
        select pid
        from musicians
        where pid in (select musician
                        from Band_Musicians
                        where role_name = 'Guitarist'
                         and band_id in (
                                select band_id
                                from Band_Albums
                                where album_id in(
                                        select album_id
                                        from Albums
                                        where genre = (
                                                select genre_name
                                                from genres
```

```
                    where genre_name = 'Jazz')))));
```

**view:**

| jazz_guitarist_first | jazz_guitarist_last |
|---|---|
| **Jimi** | **Hendrix** |
| **Paul** | **Michaels** |

## Jazz Pianists

```
create view Jazz_Pianists as
select first_name as Jazz_Pianist_First, last_name as Jazz_Pianist_Last
from People
where pid in(
        select pid
        from musicians
        where pid in (select musician
                        from Band_Musicians
                        where role_name = 'Pianist'
                         and band_id in (
                                select band_id
                                from Band_Albums
```

```
                    where album_id in(
                        select album_id
                        from Albums
                        where genre = (
                            select genre_name
                            from genres
                            where genre_name = 'Jazz')))));
```

**view:**

| jazz_pianist_first | jazz_pianist_last |
|---|---|
| **Yolo** | **Yoyo** |

## Blues_Songs

```
create view Blues_Songs as
select distinct
        bands.band_name as "Artist",
        album_songs.title as "song_name" ,
        album_songs.duration,
            albums.title as "album_title",
            albums.year_produced
from
      bands inner join band_albums on bands.band_id = band_albums.band_id
      inner join albums on band_albums.album_id = albums.album_id
      inner join album_songs on albums.album_id = album_songs.album
      inner join genres on albums.genre = genres.genre_name
      where genres.genre_name = 'Blues';
```

**view:**

| Artist | song_name | duration | album_title | year_produced |
|---|---|---|---|---|
| **Flounder** | **Cray** | **00:22:02** | **Snow Blues** | **2003** |
| **Flounder** | **Snarls** | **00:04:33** | **Snow Blues** | **2003** |
| **Flounder** | **Fragment** | **00:44:06** | **Snow Blues** | **2003** |

## **Rock Songs**

```
create view Rock_Songs as
select distinct bands.band_name as "Artist",
        album_songs.title as "song_name" ,
        album_songs.duration,
        albums.title as "album_title",
        albums.year_produced
from
        bands inner join band_albums on bands.band_id = band_albums.band_id
        inner join albums on band_albums.album_id = albums.album_id
        inner join album_songs on albums.album_id = album_songs.album
        inner join genres on albums.genre = genres.genre_name
        where genres.genre_name = 'Rock'
        group by bands.band_name, albums.title,  song_name, album_songs.duration,
albums.year_produced;
```

**view:**

| Artist | song_name | duration | album_title | year_produced |
|---|---|---|---|---|
| Further | Too Late | 00:33:13 | Hedge Knight | 2004 |
| Further | Hopeless | 00:22:33 | Hedge Knight | 2004 |
| Born to Be | Alhoo | 00:12:14 | The Experience | 1997 |
| Further | Attitude | 00:44:22 | Hedge Knight | 2004 |
| Born to Be | Limber | 00:11:19 | The Experience | 1997 |
| Born to Be | Grandpa Bones | 00:44:46 | The Experience | 1997 |

## Stored Procedures:

### Albums for a given band

**Function Definition:**

```
create function bandsAlbums(band text)
returns TABLE(title char(50), year_produced int, genre varchar(25)) as $$
begin
    return query select albums.title, albums.year_produced, albums.genre
    from bands
        inner join band_albums on bands.band_id = band_albums.band_id
        inner join albums on band_albums.album_id = albums.album_id
        where bands.band_name = 'Flounder';
end;
$$ language PLPGSQL
```

**Function Call:**

```
select bandsAlbums('Flounder')
```

**output:**

| bandsalbums |
|---|
| **("Hey You ",1989,Jazz)** |
| **("Snow Blues ",2003,Blues)** |

## Musicians for a given band

**Function definition:**

```
	create function bandsMusicians(band text)
	returns TABLE (first_name char(25), last_name char(25)) as $$
	begin
		return query select people.first_name, people.last_name
		from people
			inner join musicians on people.pid = musicians.pid
			inner join band_musicians b on b.musician = musicians.pid
			inner join bands on bands.band_id = b.band_id
			where bands.band_name = band;
	end;
	$$ language PLPGSQL
```

**Function Call:**

select bandsMusicians('Flounder');

**output:**

| bandsmusicians |
|---|
| **("Jimi ","Hendrix ")** |
| **("Yolo ","Yoyo ")** |
| **("Raqueem ","Finagles ")** |
| **("Tara ","Noto ")** |

## <u>Engineers for a Given Album</u>

**Function Definition:**
create function Albums_Engineers(albumname text)

```
returns TABLE (first_name char(25), last_name char(25)) as $$
begin
	return query select people.first_name, people.last_name
	from people
	inner join sound_engineers SE on SE.pid = People.pid
	inner join album_engineers AE on AE.engineer = SE.pid
	inner join albums on albums.album_id = AE.album
	where albums.title = albumname;
end;
$$ language PLPGSQL
```

**Function Call:**

```
select Albums_Engineers('Snow Blues')
```

**output:**

| albums_engineers |
|---|
| ("Yolo ","Yoyo ") |
| ("Paul ","Michaels ") |
| ("Samantha ","Wilbert ") |

## Albums an engineer has worked on

**Function Definition:**

```
create function engineersWorks(engineer_name text)
returns table (title char(50), year_produced int, genre varchar(25)) as $$
begin
        return query select albums.title, albums.year_produced, albums.genre
        from albums
        inner join album_engineers AE on albums.album_id = AE.album
        inner join sound_engineers SE on AE.engineer = SE.pid
        inner join people on people.pid = SE.pid
        where people.last_name = 'Yoyo';
end;
$$ language PLPGSQL
```

**Function Call:**

```
select engineersWorks('Yoyo')
```

**output:**

| engineersworks |
|---|
| **("Snow Blues ",2003,Blues)** |
| **("Hedge Knight ",2004,Rock)** |
| **("Glade ",1998,Jazz)** |

## Albums a musician has worked on

**Function Definition:**

```
create function musiciansWork(musician_name text)
returns table (title char(50), year_produced int, genre varchar(25)) as $$
begin
        return query select distinct albums.title, albums.year_produced, albums.genre
        from albums
        inner join band_albums BA on BA.album_id = albums.album_id
        inner join bands on bands.band_id = BA.band_id
        inner join band_musicians BM on BM.band_id = bands.band_id
        inner join Musicians on musicians.pid = BM.musician
        inner join People on people.pid = Musicians.pid
        where People.last_name = musician_name;
end;
$$ language PLPGSQL
```

**Function Call**

```
select musiciansWork('Hendrix')
```

**output:**

| musicianswork |
|---|
| **("Hedge Knight ",2004,Rock)** |
| **("Hey You ",1989,Jazz)** |
| **("Snow Blues ",2003,Blues)** |
| **("Wonderous ",2001,Jazz)** |

## Reports

Here are some sample reports that present useful information for a manager.

**Genre count for all albums:**

```
select genre, COUNT(*) as "total"
from albums
group by genre;
```

**report:**

| genre | total |
|---|---:|
| **Country** | **1** |
| **Blues** | **1** |
| **Rock** | **2** |
| **Jazz** | **3** |

**All engineers with degrees:**

```
select *
from people, sound_engineers
where people.pid = sound_engineers.pid
and has_degree = 'T';
```

**report:**

| pid | first_name | last_name | gender | cell_phone | email | pid | years_experience | has_degree |
|---|---|---|---|---|---|---|---|---|
| pid00014 | Yolo | Yoyo | M | 827-277-8978 | Yoyofroyo@gmail.com | pid00014 | 12 | t |
| pid00006 | Samantha | Wilbert | F | 566-123-8767 | Wilbies4lyfe@marist.edu | pid00006 | 9 | t |
| pid00012 | Candice | Sheronda | F | 878-964-9876 | datfooty@hotmail.com | pid00012 | 20 | t |

**Quantity of musician roles on roster:**

```
select role_name as "Role_In_Band", count(*) as "Total"
from Band_Musicians
```

```
group by role_name;
```

**report:**

| Role_In_Band | Total |
|---|---|
| **Pianist** | **1** |
| **Bassist** | **3** |
| **vocalist** | **3** |
| **Guitarist** | **3** |
| **Percussion** | **3** |

**Every Band's Roster:**

```
select distinct bands.band_name, band_musicians.role_name,
people.first_name,people.last_name
from bands inner join band_musicians on bands.band_id = band_musicians.band_id
inner join people on people.pid = band_musicians.musician
group by bands.band_name, band_musicians.role_name, people.first_name, people.last_name
order by bands.band_name
```

**report:**

| band_name | role_name | first_name | last_name |
|---|---|---|---|
| Born to Be | Bassist | Paul | Michaels |
| Born to Be | Guitarist | Paul | Michaels |
| Born to Be | Percussion | Tara | Noto |
| Born to Be | vocalist | Raqueem | Finagles |
| Flounder | Bassist | Tara | Noto |
| Flounder | Guitarist | Jimi | Hendrix |
| Flounder | Percussion | Raqueem | Finagles |
| Flounder | vocalist | Yolo | Yoyo |
| Further | Bassist | Stephanie | Haviland |
| Further | Guitarist | Jimi | Hendrix |
| Further | Percussion | Rebecca | Timman |
| Further | Pianist | Yolo | Yoyo |
| Further | vocalist | Jimi | Hendrix |

**Longest Album:**

```
select title, description, duration, year_produced, genre
from albums
where duration = (select max(duration) from albums)
```

**report:**

| title | description | duration | year_produced | genre |
|---|---|---|---|---|
| Glade | Relaxing tunes for the bathtub | 04:24:18 | 1998 | Jazz |

**Longest Song:**

```
select title, track_num, duration
from album_songs
where duration = (select max(duration) from album_songs);
```

**report:**

| title | track_num | duration |
|---|---|---|
| Endless | 1 | 03:44:46 |

**Number of songs engineers have worked on who have more than five years experience and have a degree:**

```
select people.first_name, people.last_name, count(album_songs.title)
from people
inner join sound_engineers SE on people.pid = SE.pid
inner join album_engineers AE on SE.pid = AE.engineer
inner join albums on albums.album_id = AE.album
inner join album_songs on albums.album_id = album_songs.album
where SE.years_experience > 5
and SE.has_degree = 'T'
group by people.first_name, people.last_name
```

**report:**

| first_name | last_name | count |
|---|---|---|
| Yolo | Yoyo | 7 |
| Samantha | Wilbert | 7 |
| Candice | Sheronda | 5 |

## **Security**

In order to maintain security and keep unscrupulous users from destroying data or falsely updating it, I’ve implemented an admins group with one admin to start off with. This admin has the ability to create other users and is allowed to insert,select, and update data within the database. This account is password protected.

```
create group admins
create user super_admin password 'imagimind';
alter group admins add user super_admin;
grant select, insert, update
on all tables in SCHEMA PUBLIC
to super_admin
create role super superuser in group admins
```

This second group will contain those who are not authorized to change or insert new data into the database, but still be able to query the database for information.

```
Create group  users
create role unauthorized_user
grant select on all tables in SCHEMA PUBLIC
to users
```

## Flaws and Possible Enhancements

This section depicts some weak points and possible ideas for enhancement of the database

-The current design only allows for an album to be considered part of a single genre. Often, musical groups will produce albums that would be considered more than one genre. A future enhancement in regards to this issue is to categorize songs by genre instead.

-The current design does not take into account the financials included in this business. Future enhancements should have branches which include accounts receivable and accounts payable

-The current design does not take into account the equipment necessary to run such a business.

-The current design does not take into account the actual booking of the studio

-A user should be able Query the database for all musicians by genre and by song.