MUSIC MATCH DATABASE

By Robert Mitola

02 December 2013

This document describes a database for understanding the relationships between musicians and all aspects of their participation in the music scene.

Table of Contents

Executive Summary

This document describes a database for understanding the relationships between musicians and all aspects of their participation in the music scene. These aspects include instruments they play or have played and bands they have been in or are in currently. Bands have relationships with managers, genres, and a current record label if the band is under a label.

The primary purpose of such a database is to serve a website which allows its users to query information about particular artists or bands. The functionalities intended for use by visitors of the site can be understood by looking at the Views section of this document.

There are a number of requirements and restrictions that were taken into account while designing this database. They are as follows:

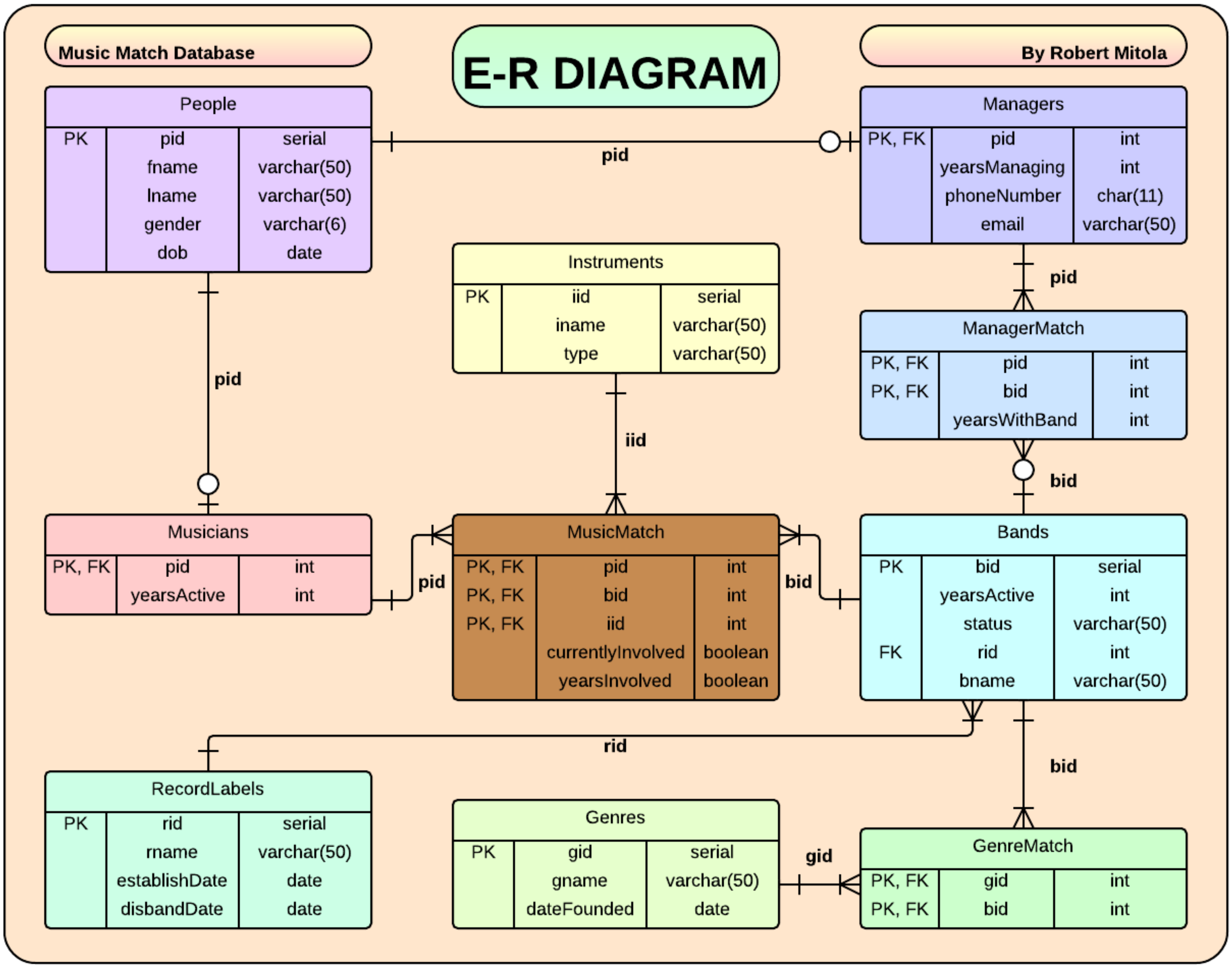
People can be managers, musicians, or both. A person does not have to be a manager or musician, but all managers and musicians are people. Basic information about each person should be provided.

Musicians can perform for one or more bands. They also can play one or more instruments, including "vocals," under the same band or many bands. It should be taken into account which instrument is played by the musician in each band he or she is a part of.

Managers can work for one or more bands, and basic contact information can be provided for each manager.

Bands can have one or more members and zero or more managers. A band can also fall under one or more genres, but must have one. If a band does not fit a defined genre, it falls under "independent" or "alternative." A band can have, but is not required to have, one record label; the one it is currently signed under.

Entity Relationship Diagram



People

PK	pid	serial
	fname	varchar(50)
	lname	varchar(50)
	gender	varchar(6)
	dob	date

Managers

PK, FK	pid	int
	yearsManaging	int
	phoneNumber	char(11)
	email	varchar(50)

Instruments

PK	iid	serial
	iname	varchar(50)
	type	varchar(50)

ManagerMatch

PK, FK	pid	int
PK, FK	bid	int
	yearsWithBand	int

Musicians

PK, FK	pid	int
	yearsActive	int

MusicMatch

PK, FK	pid	int
PK, FK	bid	int
PK, FK	iid	int
	currentlyInvolved	boolean
	yearsInvolved	boolean

Bands

PK	bid	serial
	yearsActive	int
	status	varchar(50)
FK	rid	int
	bname	varchar(50)

RecordLabels

PK	rid	serial
	rname	varchar(50)
	establishDate	date
	disbandDate	date

Genres

PK	gid	serial
	gname	varchar(50)
	dateFounded	date

GenreMatch

PK, FK	gid	int
PK, FK	bid	int

Tables: Create Statements, Functional Dependencies, Sample Data

People Table

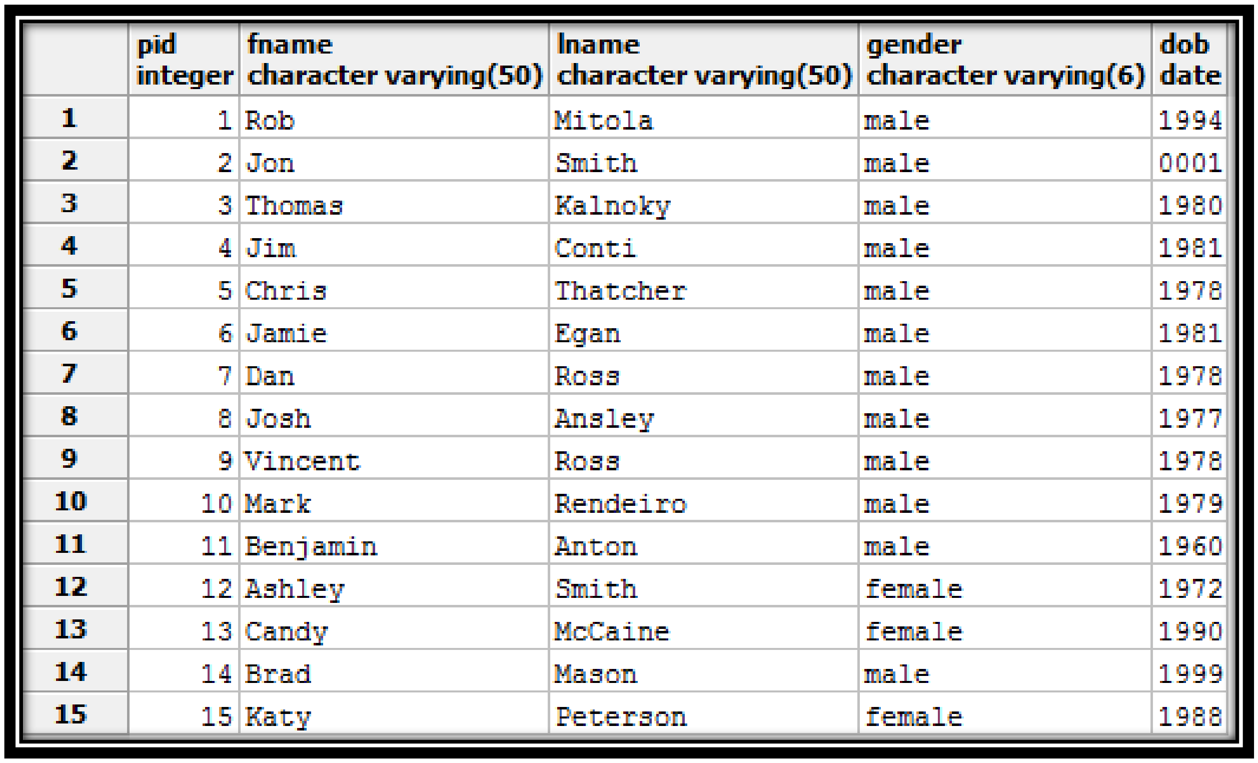
Create Statement

```sql
CREATE TABLE people(
	pid SERIAL PRIMARY KEY,
	fname VARCHAR(50) NOT NULL,
	lname VARCHAR(50) NOT NULL,
	gender VARCHAR(6) NOT NULL, /*Can be male, female, or other.*/
	dob DATE NOT NULL
);
```

Functional Dependencies

pid → fname, lname, gender, dob

Sample Data



	pid integer	fname character varying(50)	lname character varying(50)	gender character varying(6)	dob date
1	1	Rob	Mitola	male	1994
2	2	Jon	Smith	male	0001
3	3	Thomas	Kalnoky	male	1980
4	4	Jim	Conti	male	1981
5	5	Chris	Thatcher	male	1978
6	6	Jamie	Egan	male	1981
7	7	Dan	Ross	male	1978
8	8	Josh	Ansley	male	1977
9	9	Vincent	Ross	male	1978
10	10	Mark	Rendeiro	male	1979
11	11	Benjamin	Anton	male	1960
12	12	Ashley	Smith	female	1972
13	13	Candy	McCaine	female	1990
14	14	Brad	Mason	male	1999
15	15	Katy	Peterson	female	1988

Managers Table

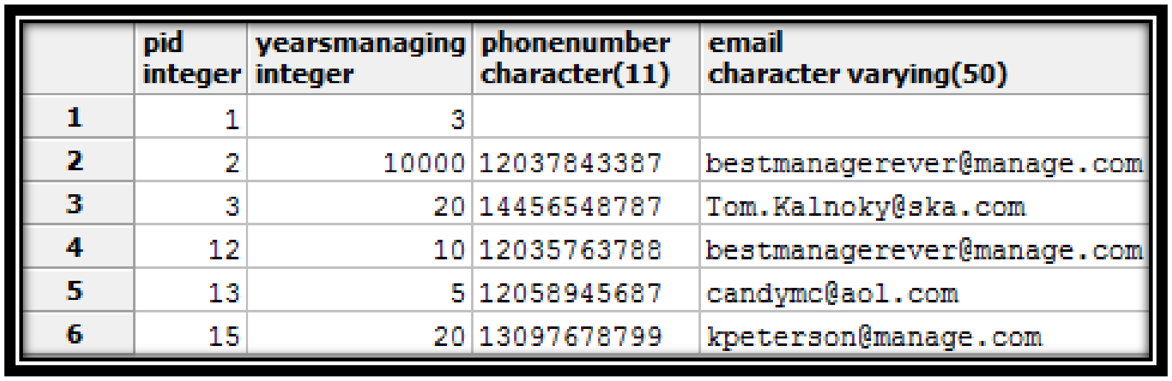
Create Statement

```
CREATE TABLE managers(
        pid INT NOT NULL PRIMARY KEY,
        yearsManaging INT NOT NULL,
        phoneNumber CHAR(11),
        email VARCHAR(50),
FOREIGN KEY(pid) REFERENCES people(pid)
);
```

Functional Dependencies

pid → yearsManaging, phoneNumber, email

Sample Data



	pid integer	yearsmanaging integer	phonenumber character(11)	email character varying(50)
1	1	3		
2	2	10000	12037843387	bestmanagerever@manage.com
3	3	20	14456548787	Tom.Kalnoky@ska.com
4	12	10	12035763788	bestmanagerever@manage.com
5	13	5	12058945687	candymc@aol.com
6	15	20	13097678799	kpeterson@manage.com

Musicians Table

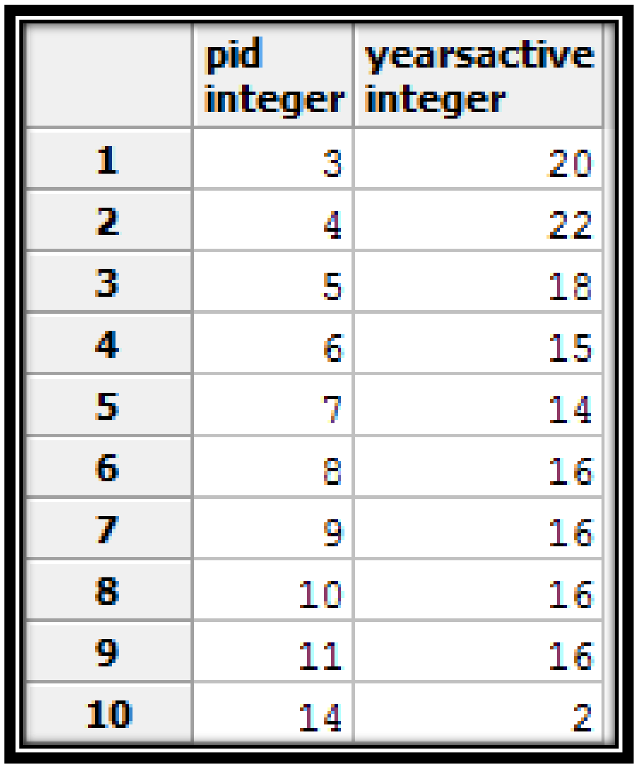
Create Statement

```
CREATE TABLE musicians(
        pid INT NOT NULL PRIMARY KEY,
        yearsActive INT NOT NULL,
FOREIGN KEY(pid) REFERENCES people(pid)
);
```

Functional Dependencies

pid → yearsActive

Sample Data



	pid integer	yearsactive integer
1	3	20
2	4	22
3	5	18
4	6	15
5	7	14
6	8	16
7	9	16
8	10	16
9	11	16
10	14	2

Instruments Table

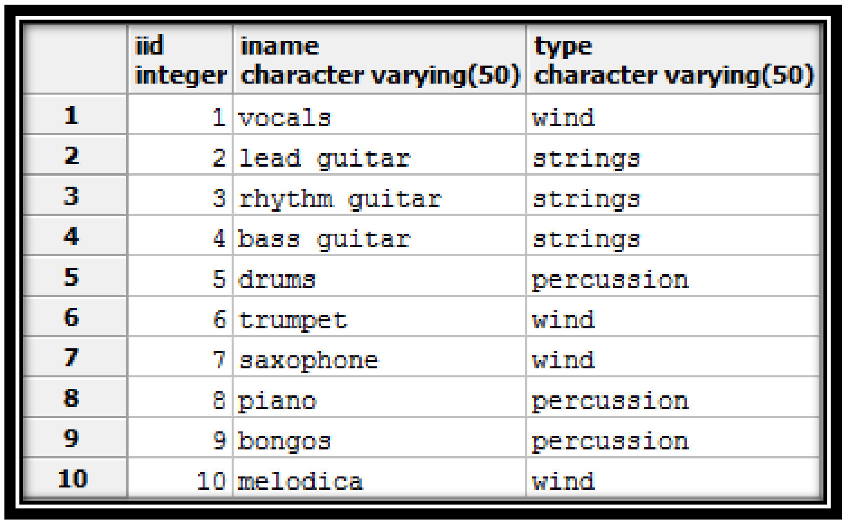
Create Statement

```
CREATE TABLE instruments(
        iid SERIAL PRIMARY KEY,
        iname VARCHAR(50) NOT NULL,
        type VARCHAR(50) NOT NULL
);
```

Functional Dependencies

iid → iname, type

Sample Data



	iid integer	iname character varying(50)	type character varying(50)
1	1	vocals	wind
2	2	lead guitar	strings
3	3	rhythm guitar	strings
4	4	bass guitar	strings
5	5	drums	percussion
6	6	trumpet	wind
7	7	saxophone	wind
8	8	piano	percussion
9	9	bongos	percussion
10	10	melodica	wind

Genres Table

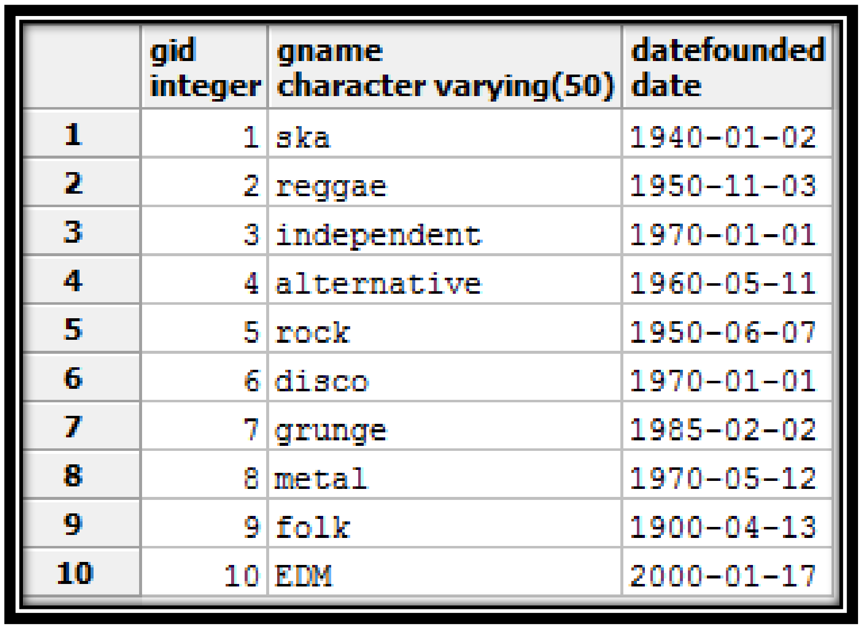
Create Statement

```
CREATE TABLE genres(
	gid SERIAL PRIMARY KEY,
	gname VARCHAR(50) NOT NULL,
	dateFounded DATE
);
```

Functional Dependencies

gid → gname, dateFounded

Sample Data



	gid integer	gname character varying(50)	datefounded date
1	1	ska	1940-01-02
2	2	reggae	1950-11-03
3	3	independent	1970-01-01
4	4	alternative	1960-05-11
5	5	rock	1950-06-07
6	6	disco	1970-01-01
7	7	grunge	1985-02-02
8	8	metal	1970-05-12
9	9	folk	1900-04-13
10	10	EDM	2000-01-17

Record Labels Table

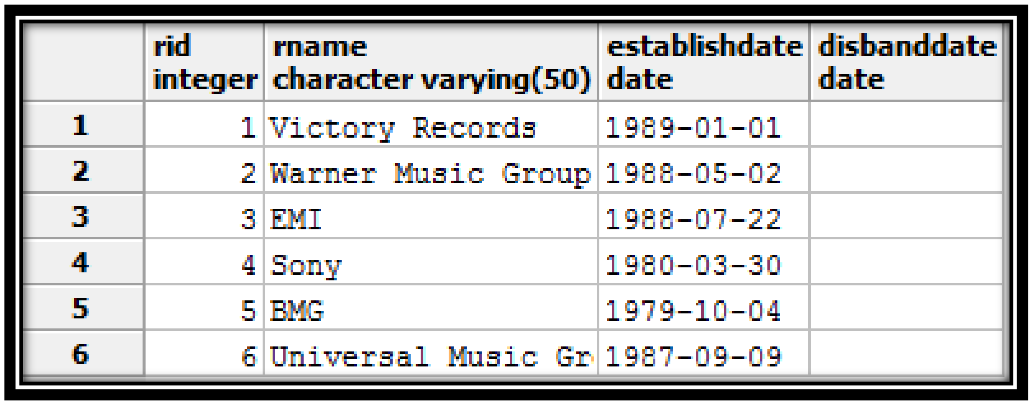
Create Statement

```
CREATE TABLE recordLabels(
	rid SERIAL PRIMARY KEY,
	rname VARCHAR(50) NOT NULL,
	establishDate DATE NOT NULL,
	disbandDate DATE
);
```

Functional Dependencies

rid → rname, establishDate, disbandDate

Sample Data



	rid integer	rname character varying(50)	establishdate date	disbanddate date
1	1	Victory Records	1989-01-01	
2	2	Warner Music Group	1988-05-02	
3	3	EMI	1988-07-22	
4	4	Sony	1980-03-30	
5	5	BMG	1979-10-04	
6	6	Universal Music Gr	1987-09-09	

Bands Table

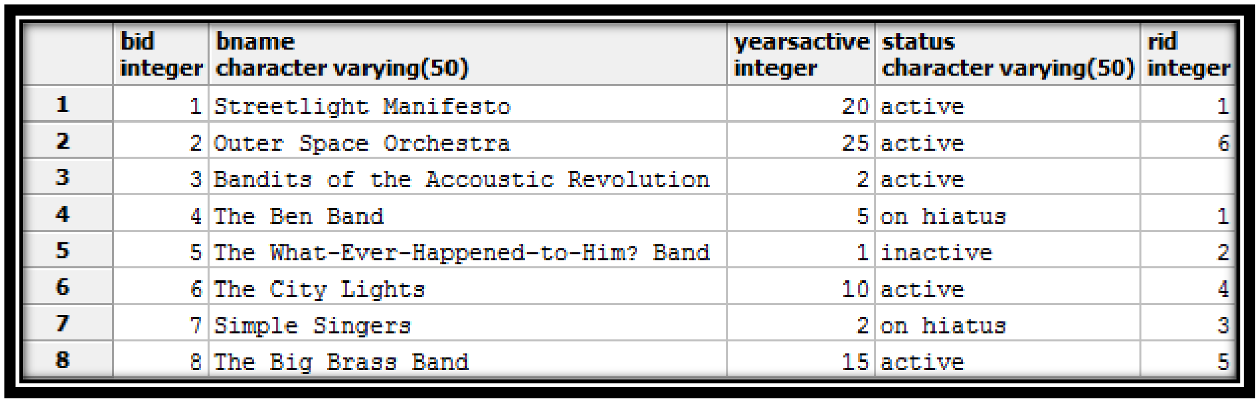
Create Statement

```
CREATE TABLE bands(
        bid SERIAL PRIMARY KEY,
        bname VARCHAR(50) NOT NULL,
        yearsActive INT NOT NULL,
        status varchar(50) NOT NULL,
        rid INT,
FOREIGN KEY(rid) REFERENCES recordLabels(rid)
);
```

Functional Dependencies

bid $\rightarrow$ bname, yearsActive, status, rid

Sample Data



	bid integer	bname character varying(50)	yearsactive integer	status character varying(50)	rid integer
1	1	Streetlight Manifesto	20	active	1
2	2	Outer Space Orchestra	25	active	6
3	3	Bandits of the Accoustic Revolution	2	active	
4	4	The Ben Band	5	on hiatus	1
5	5	The What-Ever-Happened-to-Him? Band	1	inactive	2
6	6	The City Lights	10	active	4
7	7	Simple Singers	2	on hiatus	3
8	8	The Big Brass Band	15	active	5

Manager Match Table

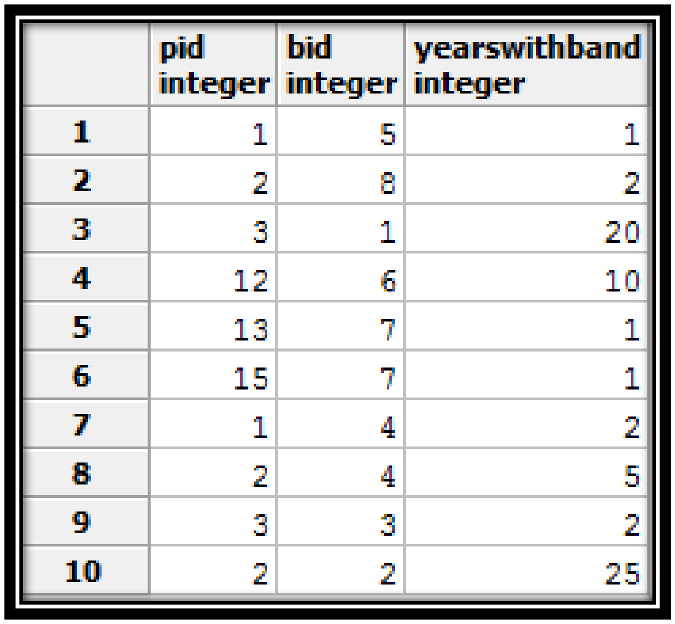
Create Statement

```
CREATE TABLE managerMatch(
	pid INT NOT NULL,
	bid INT NOT NULL,
	yearsWithBand INT NOT NULL,
PRIMARY KEY(pid, bid),
FOREIGN KEY(pid) REFERENCES managers(pid),
FOREIGN KEY(bid) REFERENCES bands(bid)
);
```

Functional Dependencies

pid, bid → yearsWithBand

Sample Data



	pid integer	bid integer	yearswithband integer
1	1	5	1
2	2	8	2
3	3	1	20
4	12	6	10
5	13	7	1
6	15	7	1
7	1	4	2
8	2	4	5
9	3	3	2
10	2	2	25

Genre Match Table

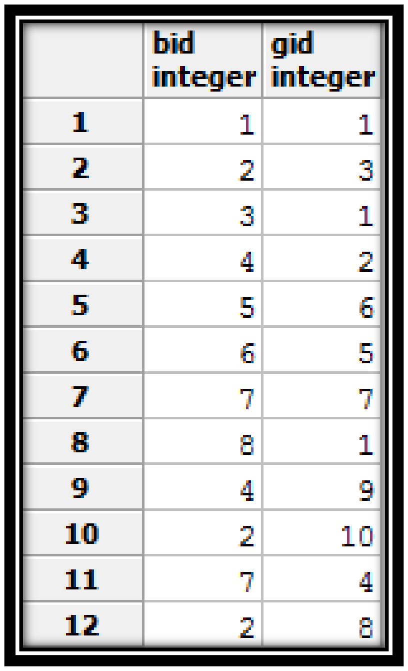
Create Statement

```
CREATE TABLE genreMatch(
	bid INT NOT NULL,
	gid INT NOT NULL,
PRIMARY KEY(bid, gid),
FOREIGN KEY(bid) REFERENCES bands(bid),
FOREIGN KEY(gid) REFERENCES genres(gid)
);
```

Functional Dependencies

bid, gid

Sample Data



	bid integer	gid integer
1	1	1
2	2	3
3	3	1
4	4	2
5	5	6
6	6	5
7	7	7
8	8	1
9	4	9
10	2	10
11	7	4
12	2	8

Music Match Table

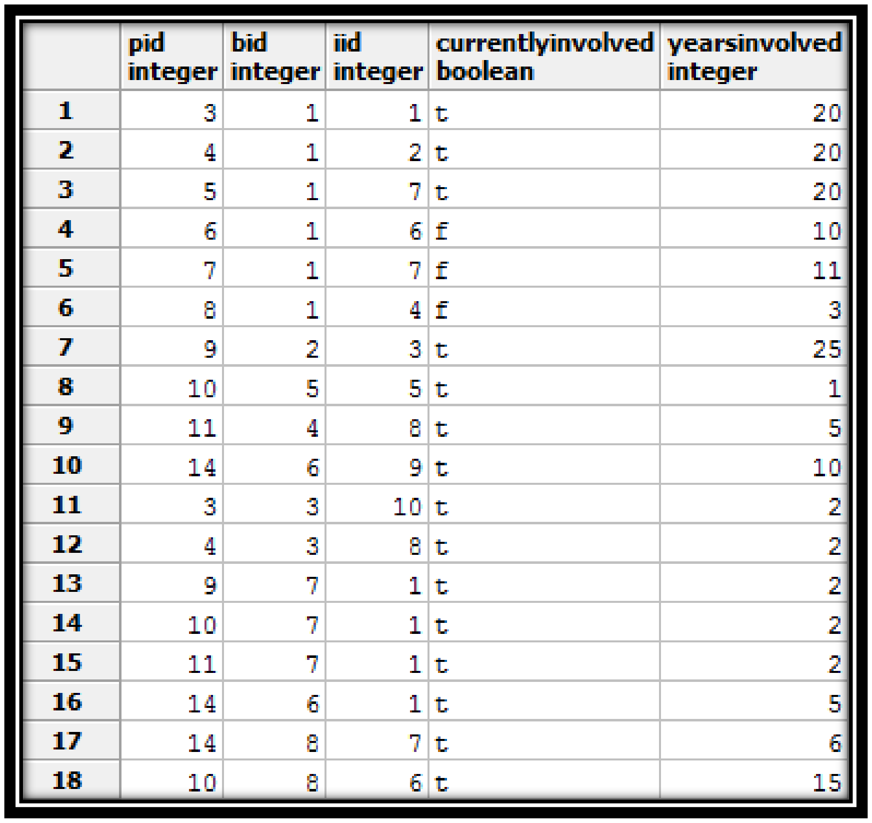
Create Statement

```
CREATE TABLE musicMatch(
        pid INT NOT NULL,
        bid INT NOT NULL,
        iid INT NOT NULL,
        currentlyInvolved BOOLEAN NOT NULL,
        yearsInvolved INT NOT NULL,
PRIMARY KEY(pid, bid, iid),
FOREIGN KEY(pid) REFERENCES musicians(pid),
FOREIGN KEY(bid) REFERENCES bands(bid),
FOREIGN KEY(iid) REFERENCES instruments(iid)
);
```

Functional Dependencies

pid, bid, iid → currentlyInvolved, yearsInvolved

Sample Data



	pid integer	bid integer	iid integer	currentlyinvolved boolean	yearsinvolved integer
1	3	1	1	t	20
2	4	1	2	t	20
3	5	1	7	t	20
4	6	1	6	f	10
5	7	1	7	f	11
6	8	1	4	f	3
7	9	2	3	t	25
8	10	5	5	t	1
9	11	4	8	t	5
10	14	6	9	t	10
11	3	3	10	t	2
12	4	3	8	t	2
13	9	7	1	t	2
14	10	7	1	t	2
15	11	7	1	t	2
16	14	6	1	t	5
17	14	8	7	t	6
18	10	8	6	t	15

Views

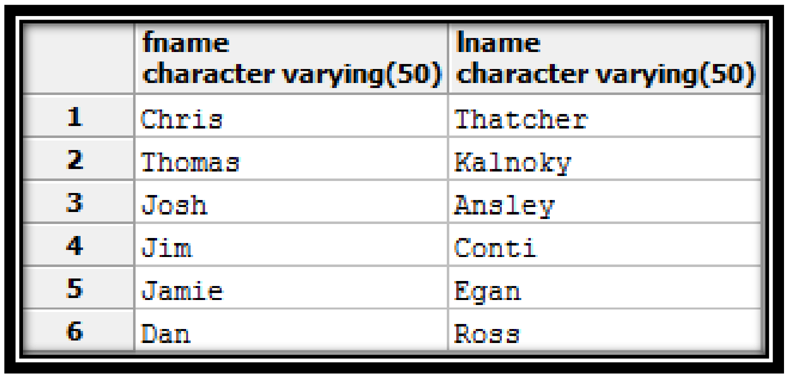
Streetlight Members View

Someone may be interested in finding out who all the members of a particular band are. Let us say, for instance, this band is Streetlight Manifesto. This view will return all the members of the band. To find out the members of another band, all that needs to be changed is the band name under the WHERE clause. This could be defined in a website by inserting user input into the SQL view via PHP.

View Statement

```
CREATE VIEW streetlight_members AS
SELECT DISTINCT p.fname, p.lname
FROM people p, musicians m, musicMatch mm, bands b, instruments i
WHERE b.bname = 'Streetlight Manifesto'
AND b.bid = mm.bid
AND p.pid = m.pid
AND m.pid = mm.pid
GROUP BY p.lname, p.fname;
```

Results



	fname character varying(50)	lname character varying(50)
1	Chris	Thatcher
2	Thomas	Kalnoky
3	Josh	Ansley
4	Jim	Conti
5	Jamie	Egan
6	Dan	Ross

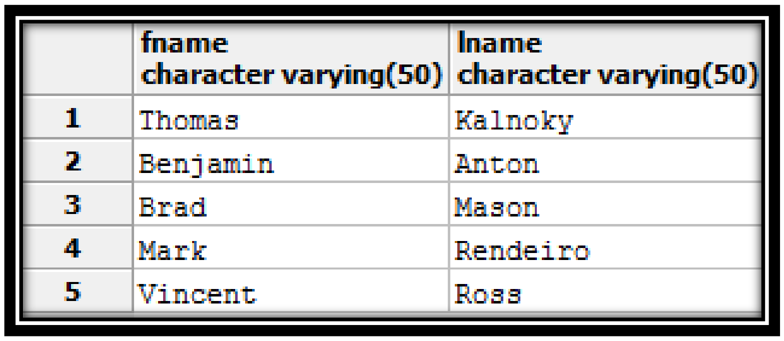
Vocalists View

An agent or hobbyist may be interested in finding out who plays what instrument within the music scene. The following creates a view which holds all the vocalists within the database. Like mentioned before, the instrument can be changed with user input from PHP on a website.

View Statement

```
CREATE VIEW vocalists AS
SELECT DISTINCT p.fname, p.lname
FROM people p, musicians m, musicMatch mm, instruments i
WHERE i.iname = 'vocals'
AND i.iid = mm.iid
AND p.pid = m.pid
AND m.pid = mm.pid
GROUP BY p.fname, p.lname;
```

Results



	fname character varying(50)	lname character varying(50)
1	Thomas	Kalnoky
2	Benjamin	Anton
3	Brad	Mason
4	Mark	Rendeiro
5	Vincent	Ross

Reports and Their Queries

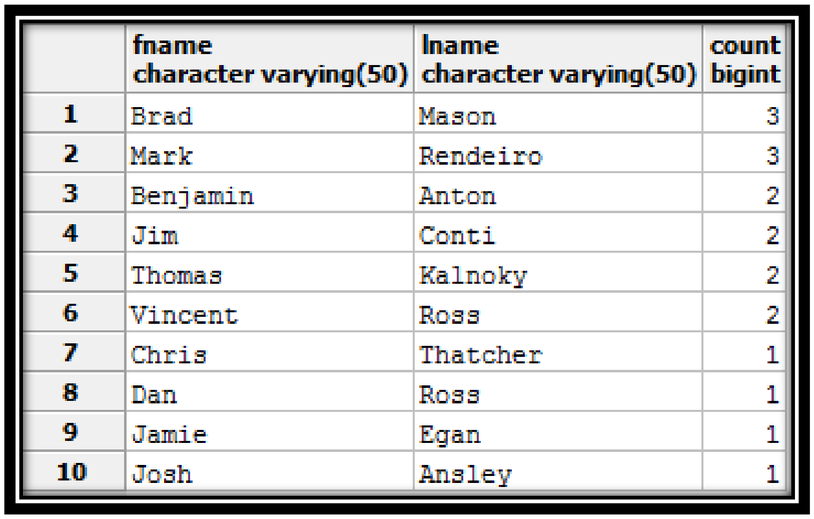
Number of Bands Played In Report

This report shows how many different bands within the database every musician has played in, regardless of their instrument or amount of time spent with the band. It orders the results from most bands played in to least to try and show the most prominent musicians.

Report Query

```
SELECT DISTINCT p.fname, p.lname, COUNT(b.bid)
FROM people p, musicians m, musicMatch mm, bands b
WHERE p.pid = m.pid
AND m.pid = mm.pid
AND mm.bid = b.bid
GROUP BY p.fname, p.lname
ORDER BY COUNT(b.bid) DESC;
```

Results



	fname character varying(50)	lname character varying(50)	count bigint
1	Brad	Mason	3
2	Mark	Rendeiro	3
3	Benjamin	Anton	2
4	Jim	Conti	2
5	Thomas	Kalnoky	2
6	Vincent	Ross	2
7	Chris	Thatcher	1
8	Dan	Ross	1
9	Jamie	Egan	1
10	Josh	Ansley	1

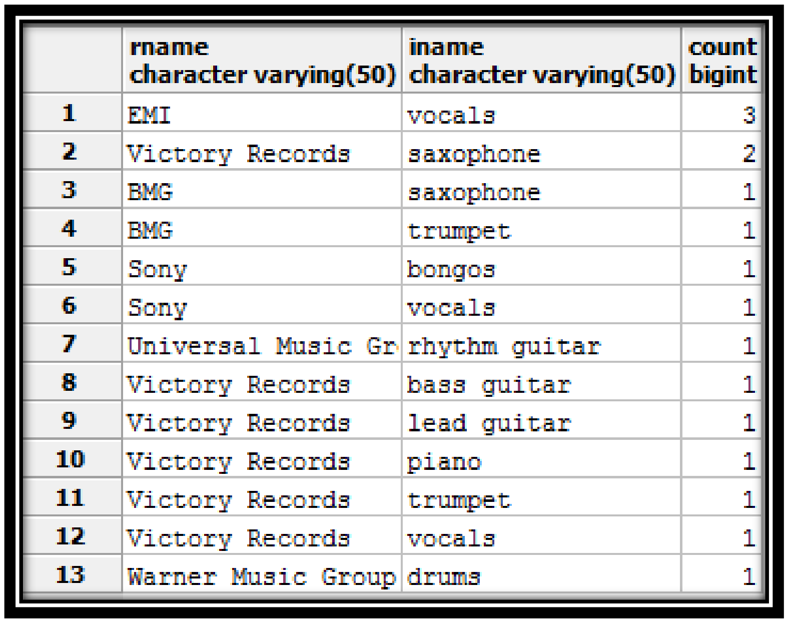
Most Popular Instrument Report

This report determines which record company favors a single instrument the most, and what that instrument is, by analyzing the instances of an instrument being used in any scenario and looking at which record company that scenario falls under. This is a good way to view trends and what certain commercial sounds the companies are looking for in new artists. The record company which favors a single instrument the most is listed first, the one which does so second to most next, and so on.

Report Query

```
SELECT DISTINCT r.rname, i.iname, COUNT(mm.iid)
FROM instruments i, musicMatch mm, bands b, recordLabels r
WHERE r.rid = b.rid
AND b.bid = mm.bid
AND i.iid = mm.iid
GROUP BY r.rname, i.iname
ORDER BY COUNT(mm.iid) DESC;
```

Results



	rname character varying(50)	iname character varying(50)	count bigint
1	EMI	vocals	3
2	Victory Records	saxophone	2
3	BMG	saxophone	1
4	BMG	trumpet	1
5	Sony	bongos	1
6	Sony	vocals	1
7	Universal Music Gr	rhythm guitar	1
8	Victory Records	bass guitar	1
9	Victory Records	lead guitar	1
10	Victory Records	piano	1
11	Victory Records	trumpet	1
12	Victory Records	vocals	1
13	Warner Music Group	drums	1

Stored Procedures

Bands Under Label Stored Procedure

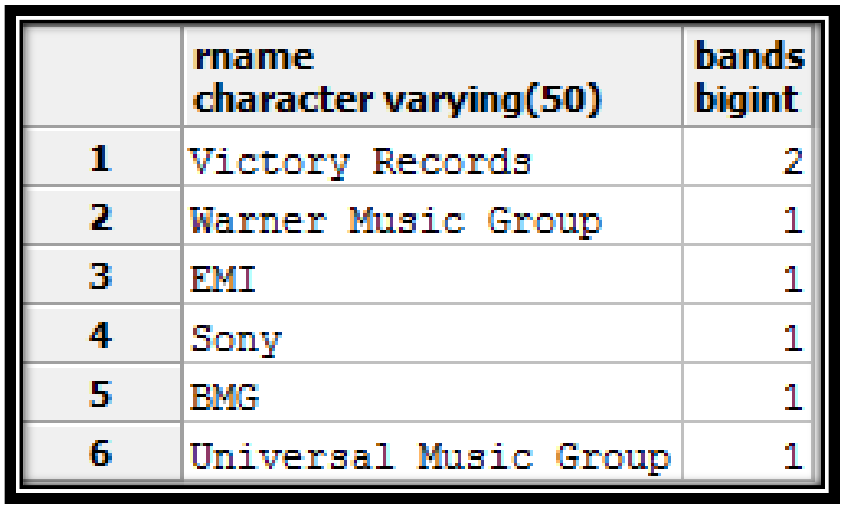
This stored procedure counts and returns the number of bands under the given label in the parameter field. This is useful for seeing the success of a record label in acquiring or maintaining contracts, or as seen in the triggers section of this report, making sure that an administrator does not alter a label with more than zero bands under it.

Stored Procedure Statement

```
CREATE FUNCTION bands_under_label(VARCHAR) RETURNS BIGINT AS $$
        SELECT COUNT(b.*)
        FROM bands b, recordLabels r
        WHERE r.rname = $1
        AND r.rid = b.rid;
$$ LANGUAGE sql;
```

Results of Querying

```
SELECT rname, bands_under_label(rname) AS bands
FROM recordlabels;
```



	rname character varying(50)	bands bigint
1	Victory Records	2
2	Warner Music Group	1
3	EMI	1
4	Sony	1
5	BMG	1
6	Universal Music Group	1

Years Managing Stored Procedure

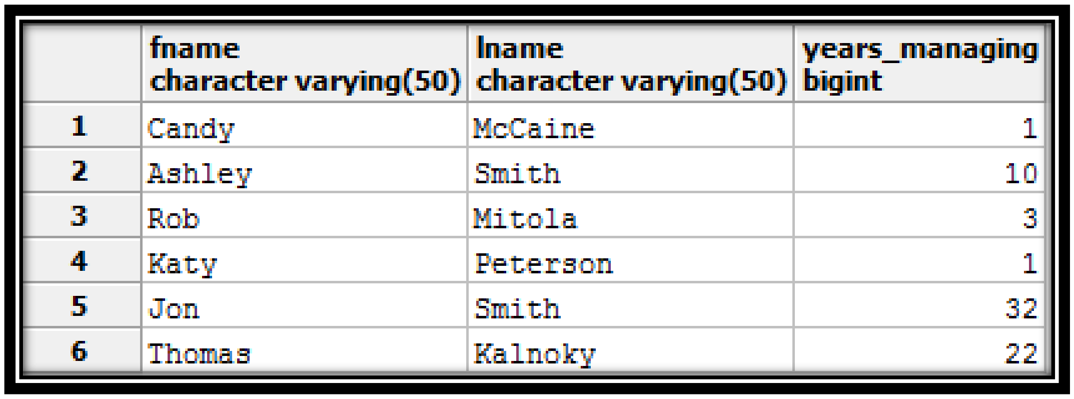
This stored procedure calculates and returns the sum of the number of years a manager has dedicated to all bands collectively. It should be noted that while a manager may work with more than one band at a time, this stored procedure will add the years dedicated to each band separately, even if they are the same years. For this reason this stored procedure should only be used for assuring a manager's YearsManaging field is not more than the collective amount of years managing each band.

Stored Procedure Statement

```
CREATE FUNCTION years_managing(VARCHAR, VARCHAR) RETURNS BIGINT AS
$$
	SELECT DISTINCT SUM(mm.yearsWithBand)
	FROM managers m, managerMatch mm, people p
	WHERE p.fname = $1
	AND p.lname = $2
	AND p.pid = m.pid
	AND m.pid = mm.pid;
$$ LANGUAGE sql;
```

Results of Querying

```
SELECT DISTINCT p.fname, p.lname, years_managing(p.fname, p.lname)
FROM people p, managers m
WHERE p.pid IN(
	SELECT pid
	FROM managers
	);
```



	fname character varying(50)	lname character varying(50)	years_managing bigint
1	Candy	McCaine	1
2	Ashley	Smith	10
3	Rob	Mitola	3
4	Katy	Peterson	1
5	Jon	Smith	32
6	Thomas	Kalnoky	22

Triggers

Check Bands Under Label Trigger

This trigger procedure keeps somebody from altering a record label's name when there are bands signed under it. This is useful for preventing people from altering a record label's name to make it appear a band is signed to a label it is actually not, or to keep administrators from accidentally changing a record label in use. On the chance that a record label were to change its name, this trigger would have to be removed.

Trigger Function

```
CREATE FUNCTION check_bands_under_label() RETURNS TRIGGER AS
$check_bands_under_label$
        DECLARE
        numberBands INTEGER :=
        (SELECT COUNT(b.*)
        FROM bands b, recordLabels r
        WHERE r.rid = NEW.rid
        AND r.rid = b.rid);
        BEGIN
                IF
                        numberBands > 0
                THEN
                        RAISE EXCEPTION 'Cannot alter a record label with bands signed
                        under it.';
                END IF;
                RETURN NEW;
        END;
$check_bands_under_label$ LANGUAGE plpgsql;
```

Trigger

```
CREATE TRIGGER check_bands_under_label BEFORE UPDATE ON recordLabels
        FOR EACH ROW EXECUTE PROCEDURE check_bands_under_label();
```

Check Years Managing Trigger

This trigger makes sure that the yearsManaging value of the managers table is never updated to be greater than the collective sum of yearsWithBand values of the managerMatch table. This is important because it would be bad to say a manager has spent a certain amount of years managing, when his or her practice shows differently. This trigger makes sure all years in the field are updated within the database before the total is.

Trigger Function

```
CREATE FUNCTION check_years_managing() RETURNS TRIGGER AS
$check_years_managing$
        DECLARE
        yearsInvolved INTEGER :=
        (SELECT DISTINCT SUM(mm.yearsWithBand)
        FROM managers m, managerMatch mm
        WHERE m.pid = NEW.pid
        AND m.pid = mm.pid);
        yearsManaging INTEGER :=
        (SELECT DISTINCT yearsManaging
        FROM managers
        WHERE pid = NEW.pid);
        BEGIN
                IF
                        yearsInvolved < yearsManaging
                THEN
                        RAISE EXCEPTION 'A manager cannot have a yearsManaging
                        value greater
                        than the sum of the yearsWithBand values in managerMatch.';
                END IF;
                RETURN NEW;
        END;
$check_years_managing$ LANGUAGE plpgsql;
```

Trigger

```
CREATE TRIGGER check_years_managing AFTER UPDATE ON managers
        FOR EACH ROW EXECUTE PROCEDURE check_years_managing();
```

Security

Considering that the implementation of this database is intended for a website for a large audience of music lovers to utilize, there will have to be a few different access levels. At the base level there are the fans, who should not be able to alter anything within the database; only view it. At the highest level is the head administrator, who in contrast can edit everything. More details and grant statements are as follows:

System Administrator

- Have the ability to control everything about the database.
- Owns the database and website.

```
GRANT SELECT, INSERT, UPDATE, DELETE, INDEX, ALTER, CREATE, DROP
ON musicMatch.*
TO sys_admin@localhost IDENTIFIED BY 'admin_password';
```

Band Managers & Record Label Officials

- Can update information about the band.
- Can change the band's record label.
- Can view members' status within the band.

```
GRANT SELECT, UPDATE,
ON bands
TO sys_admin@localhost IDENTIFIED BY 'admin';
GRANT SELECT
ON musicMatch.*
TO manager@localhost IDENTIFIED BY ‘manager_password’;
```

Site Visitors

- This is the largest group of users of the database.
- Should not be able to edit anything.
- Should be allowed to view everything.

```
GRANT SELECT
ON musicMatch.*
TO basic_user@localhost IDENTIFIED BY ‘password’;
```

Implementation Notes

In order to implement this database, a few changes will have to be made. For the most part, the addition of certain web based database tables is essential. These include, but are not limited to, tables which store user information and login credentials, tables which hold information about popular searches, and tables that handle forum posts should the website have a forum.

Aside from the addition of web based tables and functionalities, implementing the database on any machine requires running first the create statements, followed by the insert statements for initial information which is predetermined to be added to the database to start it off and test-run it, followed by creating users via the grant statements found in the security section of this document.

When adding information to the database, the order in which to do so based upon the entity relationships should be noted. The best course of action when adding information is to first add the necessary people; followed by categorizing them into the managers and musicians table; followed by adding a record label, instruments, and genre if necessary (although in most instances it probably is not); followed by adding band information. Finally, the information should be tied together through the designated match tables.

Known Problems

It was discovered when creating triggers that there is no real way to differentiate between managers that have worked with multiple bands at the *same time* versus managers that have worked with multiple bands at *different times*. This creates a problem for verifying the actual amount of years a manager has spent managing. A possible fix for this problem is, instead of storing the years a manager has worked with a band as an integer value, include start and end dates stored as date values. This same problem, and same possible solution, applies to musicians and their correlating time spent playing instruments for bands. There is no real way to differentiate between a musician playing two or more different instruments for a band at the same time or at different times. Likewise there is no way to differentiate a musician's involvement in multiple bands at the same time or different times.

The bands table lacks information in that it does not describe when the band was founded or when, if it did, disband. A future version of this database would update that.

The information in the database is somewhat "loose." The fact that information about how long managers have been managing and how long musicians have been playing is stored in multiple areas creates some problems for verification and efficiency when adding new information. These problems were attempted to be solved by the example triggers and stored procedures, which can be applied with minor changes to various areas of the database.

Future Enhancements

A future enhancement of this database would first seek to correct all afore mentioned problems. This could be done, as was explained before, by editing the fields in the tables of the database, and by providing more constraints via triggers and stored procedures.

The database could also be upgraded to provide a much larger scope of information about artists and their work. For example, past record deals could be taken into account instead of just the current one. Band pictures and histories could be stored in the database to essentially provide a social Wikipedia of musicians and their interconnected lives.

An upgraded version of this database could also integrate user input into it. Through administrative verification users might be allowed to add musicians, bands, genres, and record labels that were not already a part of the database. This would help it grow very quickly.