PUNK

Alexander Hochberg

4-25-2014

Database Systems: Design Project

TABLE OF CONTENTS

EXECUTIVE SUMMARY

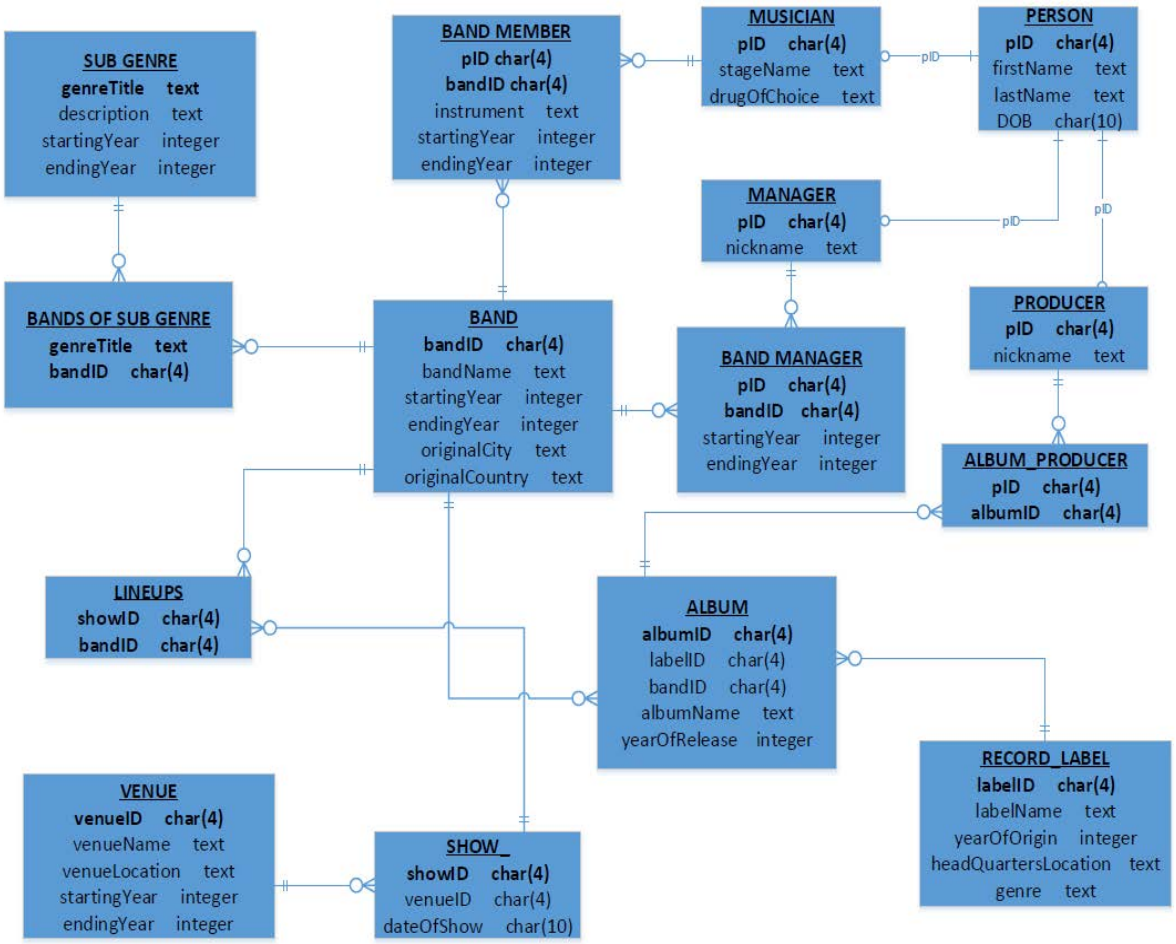
Overview_____

Between the years of 1974 and 1976 in the United Kingdom and the United States, a new genre of music emerged called **Punk**. This new form of music was not about talented musicianship, intricate song-writing or aesthetically- pleasing melodies, but rather just fast, loud and raw energy. Since its birth with bands like the Ramones, the Clash, and Television, a punk subculture emerged, expressing teenage rebellion and anti-establishment rage. Various subgenres such as garage punk, skate punk and hardcore punk have branch from its initial arrival and still continue to grow today.

Objectives_____

The main objective of this paper is to detail a complete database design to structure the noteworthy bands that contributed to punk music. This will allow useful information about details of punk bands to be accessible to music enthusiasts. The document will include details about people involved within a band including musicians, managers, producers, band's albums along with their corresponding record labels, band's shows at specific venues and punk subgenres. The document is sectioned into tables, views, reports with queries, triggers, security, implementation notes, known problems and future enhancement. The database was designed using Postgresql.

ENTITY RELATIONSHIP DIAGRAM



TABLES

Band_____

The Band table contains the basic important information about a band including their name, starting year, ending year (if applicable), city of origin and country of origin. The artificial key bandID has been created for the table's primary key.

Create Statement_____

```
-- BAND --
CREATE TABLE band (
  bandID                char(4)     not null,
  bandName              text        not null,
  startingYear          integer     not null,
  endingYear            integer,
  originalCity          text        not null,
  originalCountry       text        not null,
 primary key(bandID)
);
```

Functional Dependencies_____

bandID-> bandName, startingYear, endingYear, originalCity, originalCountry

Sample Data_____

	bandid character(4)	bandname text	startingy integer	endingy integer	originalcity text	originalcountry text
1	R451	The Ramones	1974	1996	Queens, New York	United States
2	C227	The Clash	1976	1986	London	England
3	S115	Sex Pistols	1975	1978	London	England
4	D998	Dead Kennedys	1978	1986	San Francisco, California	United States
5	B424	Black Flag	1976		Hermosa Beach, California	United States
6	M112	Misfits	1977		Lodi, New Jersy	United States
7	B882	Bikini Kill	1990	1997	Olympia, Washington	United States

Person_____

The person table contains the primary information that every type of person share; first name, last name and date of birth. The artificial key pID acts as the table's primary key.

Create Statement_____

```
-- PERSON --
CREATE TABLE person (
  pID               char(4)         not null,
  firstName         text            not null,
  lastName          text            not null,
  DOB               char(10)        not null,
 primary key(pID)
);
```

Functional Dependencies_____

pID->firstName, lastName, DOB

Sample Data_____

	pid character	firstname text	lastname text	dob character(10)					
1	DR32	Douglas	Colvin	09-18-1951	22	EB18	Raymond	Pepperell	10-17-1958
2	JR85	John	Cummings	10-8-1948	23	KF18	Geoffrey	Lyall	05-30-1949
3	JR31	Jeffrery	Hyman	11-24-1950	24	DH18	Darren	Henley	06-11-1952
4	TR84	Thomas	Erdelyi	06-17-1948	25	RG18	Ron	Greer	01-07-1952
5	JS33	John	Mellor	06-26-1955	26	KH76	Kathleen	Hanna	10-12-1968
6	MS45	Micheal	Jones	01-01-1953	27	KW76	Kathi	Wilcox	02-19-1967
7	PS45	Paul	Simmon	12-21-1955	28	TV76	Tobi	Vail	11-22-1965
8	NH44	Nicholas	Headon	07-30-1955	29	BK76	Billy	Karren	01-21-1969
9	SJ40	Stephan	Jones	08-10-1955	30	JO55	Gerrard	Caiafa	04-21-1959
10	PC17	Paul	Cook	04-19-1956	31	DC55	Dennis	Cadena	06-02-1961
11	GM66	Glen	Matlock	08-27-1956	32	EA55	Eric	Arce	11-21-1961
12	JR72	John	Lydon	01-31-1956	33	GD55	Glenn	Danzig	10-11-1966
13	DF11	Daniel	Feinberg	05-07-1941	34	GG79	Greg	Ginn	12-10-1954
14	BR82	Bernard	Rhodes	06-11-1947	35	HR79	Henry	Rollins	02-13-1954
15	MM86	Malcolm	McLaren	01-22-1946	36	CD79	Chuck	Dukowski	08-08-1955
16	CT53	Chris	Thomas	01-13-1947	37	RV79	Roberto	Valverde	12-17-1952
17	DG88	Dave	Goodman	02-29-1951	38	MM57	Monte	Melnick	10-17-1943
18	GS65	Guy	Stevens	04-13-1943	39	EM77	Elliot	Mazer	05-17-1947
19	SP90	Samuel	Pearlman	06-08-1943	40	GL80	Glen	Lockett	07-28-1951
20	MF70	Mickey	Foote	02-15-1940	41	DR64	Daniel	Rabinowitz	06-06-1949
21	CL55	Craig	Leon	01-07-1952	42	SN46	Stuart	Nallerman	09-16-1967

Musician_____

The Musician table contains information about a musician including his/her stage name and what drug they do (if applicable). The table is a subtype of the Person table, and thus contains the primary key pID.

Create Statement_____

```
-- MUSICIAN --
CREATE TABLE musician (
  pID           char(4) not null  references person(pID),
  stageName     text,
  drugOfChoice text,
 primary key(pID)
);
```

Functional Dependencies_____

pID-> stageName, drugOfChoice

Sample Data_____

	pid character(4)	stagename text	drugofchoice text
1	DR32	Dee Dee Ramone	heroin
2	JR85	Johnny Ramone	
3	JR31	Joey Ramone	
4	TR84	Tommy Ramone	
5	JS33	Joe Strummer	
6	MS45	Mick Jones	
7	PS45		
8	NH44	Topper Headon	heroin
9	SJ40	Steve Jones	
10	PC17		
11	GM66		
12	JR72	Johnny Rotten	
13	EB18	East Bay Ray	cocaine
14	KF18	Klaus Flouride	cocaine
15	DH18	D.H. Peligro	cocaine
16	RG18	Skip	heroin
17	KH76		
18	KW76		
19	TV76		
20	BK76		
21	JO55	Jerry Only	heroin
22	DC55	Dez Cadena	
23	EA55	Chupacabra	
24	GD55		
25	GG79		
26	HR79		LSD
27	CD79		LSD
28	RV79	Robo	cocaine

Manager_____

The Manager table contains information about a band manager including his/her nickname (if applicable). The table is a subtype of the Person table, and thus contains the primary key pID.

Create Statement_____

```
-- MANAGER --
CREATE TABLE manager (
  pID          char(4) not null  references person(pID),
  nickname     text,
 primary key(pID)
);
```

Functional Dependencies_____

pID-> nickname

Sample Data_____

	pid character(4)	nickname text
1	DF11	Danny Fields
2	BR82	
3	MM86	
4	MM57	Mont
5	CD79	

Producer_____

The Producer table contains information about an album producer including his/her nickname (if applicable). The table is a subtype of the Person table, and thus contains the primary key pID.

Create Statement_____

```
-- PRODUCER --
CREATE TABLE producer (
  pID          char(4) not null  references person(pID),
  nickName     text,
 primary key(pID)
);
```

Functional Dependencies_____

pID-> nickname

Sample Data_____

	pid character(4)	nickname text
1	CT53	
2	DG88	
3	GS65	
4	SP90	Sandy Pearlman
5	MF70	
6	TR84	Tommy Ramone
7	CL55	
8	EM77	
9	GL80	Spot
10	DR64	Daniel Rey
11	SN46	

Band Member_____

The Band Member table connects musicians with the corresponding bands they played with. It includes what instrument the band member plays, what year they joined the band and what year their left the band (if applicable). The table uses the foreign keys pID from the Musician table and bandID from the Band table as a composite primary key.

Create Statement_____

```
-- BAND_MEMBER --
CREATE TABLE band_member (
  pID              char(4)     not null  references musician(pID),
  bandID           char(4)     not null  references band(bandID),
  instrument       text        not null,
  startingYear     integer     not null,
  endingYear       integer,
 primary key(pID, bandID)
);
```

Functional Dependencies_____

pID, bandID-> instrument, startingYear, endingYear

Sample Data_____

	pid character(4)	bandid character(	instrument text	startingy integer	endingye integer
1	DR32	R451	Bass	1974	1996
2	JR85	R451	Guitar	1974	1996
3	JR31	R451	Vocals	1974	1996
4	TR84	R451	Drums	1974	1982
5	JS33	C227	Vocals	1976	1986
6	MS45	C227	Guitar	1976	1986
7	PS45	C227	Bass	1976	1986
8	NH44	C227	Drums	1976	1986
9	SJ40	S115	Guitar	1975	1978
10	PC17	S115	Drums	1975	1978
11	GM66	S115	Bass	1975	1978
12	JR72	S115	Vocals	1975	1978
13	EB18	D998	Guitar	1978	1986
14	KF18	D998	Bass	1978	1986
15	DH18	D998	Drums	1978	1986
16	RG18	D998	Vocals	1984	1986
17	KH76	B882	Vocals	1990	1997
18	KW76	B882	Bass	1990	1997
19	TV76	B882	Drums	1990	1997
20	BK76	B882	Guitar	1992	1997
21	JO55	M112	Bass	1977	
22	DC55	M112	Guitar	2001	
23	EA55	M112	Drums	2010	
24	GD55	M112	Vocals	1977	1983
25	GG79	B424	Guitar	1976	
26	HR79	B424	Vocals	1976	1981
27	CD79	B424	Bass	1978	1983
28	RV79	B424	Drums	1976	1979

Band Manager_____

The Band Manager table connects managers with the band they managed. The table includes what year they started managing the band and what year they stopped (if applicable). The table uses the foreign keys pID from the Manager table and bandID from the Band table as a composite primary key.

Create Statement_____

```
-- BAND_MANAGER --
CREATE TABLE band_manager (
  pID              char(4)      not null  references person(pID),
  bandID           char(4)      not null  references band(bandID),
  startingYear     integer      not null,
  endingYear       integer,
 primary key(pID, bandID)
);
```

Functional Dependencies_____

pID, bandID-> startingYear, endingYear

Sample Data_____

	pid character(4)	bandid character(	startingyear integer	endingyear integer
1	DF11	R451	1973	1976
2	BR82	C227	1976	1986
3	MM86	S115	1975	1978
4	MM57	R451	1976	1981
5	CD79	B424	1979	1984

Record Label_____

The Record Label table contains information about record labels that put out noteworthy punk rock albums. The table includes the record label's name, their year of origin, their headquarters' location and what type of genre of music they distribute. The artificial key labelID was created as the table's primary key.

Create Statement_____

```
-- RECORD_LABEL --
CREATE TABLE record_label(
  labelID               char(4) not null,
  labelName             text    not null,
  yearOfOrigin          text    not null,
  headQuartersLocation  text    not null,
  genre                 text    not null,
 primary key(labelID)
);
```

Functional Dependencies_____

labelID-> labelName, yearOfOrigin, headQuartersLocation, genre

Sample Data_____

	labelid character(4)	labelname text	yearoforigin text	headquarterslocation text	genre text
1	SR33	Sire Records	1966	Vermont, United States	Various
2	PR73	Philips Records	1950	Zwolle, Netherlands	Various
3	CR14	Columbia Records	1888	New York, United States	Various
4	VR64	Virgin Records	1972	Hollywood, California	Various
5	AT63	Alternative Tentacles	1979	San Francisco, California	Alternative
6	SS56	SST Records	1966	Taylor, Texas	Punk Rock
7	KR40	Kill Rock Stars	1991	Portland, Oregon	Various

Album_____

The Album table contains the essential information about band's albums, including the album's name, the album's year of release, the foreign key bandID from the table band, the foreign key labelID from the table label and the artificial key albumID as the primary key.

Create Statement_____

```
-- ALBUM --
CREATE TABLE album(
  albumID         char(4)      not null,
  labelID         char(4)                  references record_Label(labelID),
  bandID          char(4)      not null    references band(bandID),
  albumName       text         not null,
  yearOfRelease   integer      not null,
 primary key(albumID)
);
```

Functional Dependencies_____

albumID-> labelID, bandID, albumName, yearOfRelease

Sample Data_____

	albumid character(4)	labelid character(4)	bandid character(4)	albumname text	yearofrelease integer
1	RA01	SR33	R451	Ramones	1976
2	RA02	SR33	R451	Leave Home	1977
3	RA03	PR73	R451	Rocket to Russia	1977
4	CL01	CR14	C227	The Clash	1977
5	CL02	CR14	C227	Give Em Enough Rope	1978
6	CL03	CR14	C227	London Calling	1977
7	SP01	VR64	S115	Never Mind the Bollocks	1977
8	SP02	VR64	S115	The Great Rock n Roll Swindle	1979
9	DK01	AT63	D998	Give Me Convenience or Give Me Death	1987
10	BF01	SS56	B424	Damaged	1981
11	BF02	SS56	B424	Everything Went Black	1981
12	BF03	SS56	B424	The First Four Years	1983
13	MI01	AT63	M112	Famous Monsters	1999
14	MI02	AT63	M112	Walk Among Us	1982
15	BK01	KR40	B882	Pussy Whipped	1993

Album Producer_____

The Album Producer table connects a producer with the album they produced. It contains the foreign keys pID from the producer table and albumID from the album table as a composite primary key.

Create Statement_____

```
--ALBUM_PRODUCER--
CREATE TABLE album_producer (
  pID           char(4)    not null    references producer(pID),
  albumID       char(4)    not null    references album(albumID),
 primary key(pID ,albumID)
);
```

Functional Dependencies_____

pID, albumID->

Sample Data_____

	pid character(4)	albumid character(4)
1	CT53	SP01
2	DG88	SP02
3	GS65	CL03
4	SP90	CL02
5	MF70	CL01
6	CL55	RA03
7	MF70	RA02
8	MF70	RA01
9	EM77	DK01
10	GL80	BF01
11	GL80	BF02
12	GL80	BF03
13	DR64	MI01
14	DR64	MI02
15	SN46	BK01

Sub-Genre_____

The Sub-Genre table contains the sub-genre's title, a brief description explaining the sub-genre's characteristics, and its starting year and ending year (if applicable). The genreTitle acts as the table's primary key.

Create Statement_____

```
-- SUB_GENRE --
CREATE TABLE sub_genre(
  genreTitle          text        not null,
  description         text,
  startingYear        integer     not null,
  endingYear          integer,
 primary key(genreTitle)
);
```

Functional Dependencies_____

genreTitle->description, startingYear, endingYear

Sample Data_____

	genretitle text	description text	startingyear integer	endingyear integer
1	Garage Punk	Lo-Fi aesthetics over catchy melodies	1973	
2	Hardcore Punk	Fast, agressive and screams	1979	
3	Glam Punk	Otherwise known as Glitter punk, fuses punk and glam rock	1970	1976
4	Anarcho Punk	Grew out of UK. Promotes anarchism	1972	1983
5	Riot Grrl	Feminist encouraging punk/indie	1990	
6	Noise Rock	Incorporates atonality an dissonance	1981	
7	Christian Punk	Punk rock with some degree of Christian lyrical content	1987	
8	Horror Punk	Mixes gothic and punk rock sounds with morbid and violent imagery	1978	

Bands of Sub Genre_____

The Bands Of Sub Genre table connects bands with their specific sub-genre of punk. The table uses the foreign keys of genreTitle from the sub-genre table and bandID from the band table as its composite primary key.

Create Statement_____

```
-- BANDS_OF_SUB_GENRE --
CREATE TABLE bands_of_sub_genre(
  genreTitle    text       not null    references sub_genre(genreTitle),
  bandID        char(4)    not null    references band(bandID),
 primary key(genreTitle, bandID)
);
```

Functional Dependencies_____

genreTitle, bandID->

Sample Data_____

	genretitle text	bandid character(4)
1	Riot Grrl	B882
2	Garage Punk	R451
3	Hardcore Punk	B424
4	Hardcore Punk	D998
5	Horror Punk	M112

Venue_____

The Venue table contains information about the legendary concert venues of punk. This includes the venue's name, the venue's location, its year of creation and its year of demise, if the venue has shutdown. The atrfical key of a venueID has been created for the table's primary key.

Create Statement_____

```
-- VENUE --
CREATE TABLE venue(
  venueID           char(4)      not null,
  venueName         text         not null,
  venueLocation     text         not null,
  startingYear      integer      not null,
  endingYear        integer,
 primary key(venueID)
);
```

Functional Dependencies_____

venueID-> venueName, venueLocation, startingYear, endingYear

Sample Data_____

	venueid character(4)	venuename text	venuelocation text	startingyear integer	endingyear integer
1	CBG4	CBGBs	315 Bowery St. Manhattan, NY, United States	1973	2006
2	CLU8	82 Club	82 E.4th St. Manhattan, NY, United States	1958	1978
3	GG62	The Cafe au Go Go	152 Bleecker Manhattan, NY, United States	1961	
4	100C	100 Club	100 Oxford St. London, England	1942	
5	TMU3	The Mushroom	43 Board St. Seattle, Washington	1987	
6	OKH9	OK Hotel	100 Steve St. Olympia, Washington	1992	

Show_____

The Show_ table contains information about an actual punk concert at its specified concert venue. The table contains the date of the show, the foreign key venueID for the venue table and the artificial key showID, which acts as the table's primary key.

Create Statement_____

```
-- SHOW_ --
CREATE TABLE show_(
  showID          char(4)      not null,
  venueID         char(4)      not null,
  dateOfShow      char(10)     not null,
 primary key(showID)
);
```

Functional Dependencies_____

showID-> venueID, dateOfShow

Sample Data_____

	showid character(4)	venueid character(4)	dateofshow character(10)
1	SH01	CBG4	04-17-1972
2	SH02	OKH9	10-08-1995
3	SH03	100C	08-14-1975
4	SH04	CLU8	05-10-1980

Lineups_____

The Lineups table connects punk shows with the bands that played at that show. The table uses the foreign keys showID from the Show table and the bandID from the band table as a composite primary key.

Create Statement_____

```
-- LINEUPS --
CREATE TABLE lineups(
  showID   char(4)   not null   references show_(showID),
  bandID   char(4)   not null   references band(bandID),
 primary key(showID, bandID)
);
```

Functional Dependencies_____

showID, bandID->

Sample Data_____

	showid character(4)	bandid character(4)
1	SH01	R451
2	SH01	C227
3	SH02	B882
4	SH03	C227
5	SH03	S115
6	SH04	D998
7	SH04	B424
8	SH04	M112

QUERIES

Query 1_____

Query 1 finds all punk drummers that were actively playing in the year 1979 and displays their first name, last name, stage name, punk genre and a description of that genre.

```
SELECT p.firstName, p.lastName, m.stageName, sg.genreTitle, sg.description
FROM person p,
     musician m,
     band_member bm,
     band b,
     bands_of_sub_genre bosg,
     sub_genre sg
WHERE p.pID = m.pID
AND       m.pID = bm.pID
AND       p.pID = bm.pID
AND       bm.bandID = b.bandID
AND       bosg.genreTitle = sg.genreTitle
AND       bosg.bandID = bm.bandID
AND       bm.instrument = 'Drums'
AND       bm.startingYear <= 1979
AND       (bm.endingYear >=1979 OR bm.endingYear IS NULL);
```

Results_____

	firstname text	lastname text	stagename text	genretitle text	description text
1	Thomas	Erdelyi	Tommy Ramone	Garage Punk	Lo-Fi aesthetics over catchy melodies
2	Darren	Henley	D.H. Peligro	Hardcore Punk	Fast, agressive and screams
3	Roberto	Valverde	Robo	Hardcore Punk	Fast, agressive and screams

Query 2_____

Query 2 finds all the punk bands that lasted shorter than the average punk bands life span and displays their band name and how many years they were together.

```
SELECT b.bandName, (b.endingYear - b.startingYear) yearsTogether
FROM band b
WHERE (b.endingYear - b.startingYear)<
                (SELECT avg(b.endingYear - b.startingYear)
                 FROM band b
                 WHERE b.endingYear IS NOT NULL);
```

Results_____

	bandname text
1	Sex Pistols
2	Dead Kennedys
3	Bikini Kill

Query 3_____

Query 3 finds all the punk albums that were released in the year 1985 and later and displays their band name, album name, the year the album was released, the producer of the album's first name and last name, and the record label it was distribute through.

```
SELECT b.bandName,
       a.albumName,
       a.yearOfRelease,
       p.firstName,
       p.lastName,
       rl.labelName
FROM person p,
     producer pr,
     album_producer ap,
     album a,
     record_label rl,
     band b
WHERE p.pID=pr.pID
AND ap.pID=p.pID
AND ap.pID=pr.pID
AND a.albumID=ap.albumID
AND a.labelID=rl.labelID
AND b.bandID=a.bandID
AND a.yearOfRelease >=1985
ORDER BY a.yearOFRelease ASC;
```

Results_____

	bandname text	albumname text	yearofrelease integer	firstname text	lastname text	labelname text
1	Dead Kennedys	Give Me Convenience or Give Me	1987	Elliot	Mazer	Alternative Tentacles
2	Bikini Kill	Pussy Whipped	1993	Stuart	Nallerman	Kill Rock Stars
3	Misfits	Famous Monsters	1999	Daniel	Rabinowitz	Alternative Tentacles

VIEWS

View 1_____

View 1 creates a view of every punk show, the name of the venue it was held, the bands that played at that show, the date of the show and all the band members that played at that time.

```
CREATE VIEW ConcertInformation AS
SELECT v.venueName AS Venue,
       s.dateOfShow AS ConcertDate,
       b.bandName AS Band,
       p.firstName AS FirstName,
       p.lastName AS LastName,
       m.stageName AS StageName
FROM person p,
     musician m,
     band_member bm,
     band b,
     lineups l,
     show_ s,
     venue v
WHERE p.pID=m.pID
AND m.pID= bm.pID
AND p.pID=bm.pID
AND bm.bandID=b.bandID
AND b.bandID=l.bandID
AND l.bandID=bm.bandID
AND l.showID=s.showID
AND s.venueID= v.venueID;
```

Results_____

	venue text	concertdate character(10)	band text	firstname text	lastname text	stagename text
1	CBGBs	04-17-1972	The Ramones	Douglas	Colvin	Dee Dee Ramone
2	CBGBs	04-17-1972	The Ramones	John	Cummings	Johnny Ramone
3	CBGBs	04-17-1972	The Ramones	Jeffrery	Hyman	Joey Ramone
4	CBGBs	04-17-1972	The Ramones	Thomas	Erdelyi	Tommy Ramone
5	100 Club	08-14-1975	The Clash	John	Mellor	Joe Strummer
6	CBGBs	04-17-1972	The Clash	John	Mellor	Joe Strummer
7	100 Club	08-14-1975	The Clash	Micheal	Jones	Mick Jones
8	CBGBs	04-17-1972	The Clash	Micheal	Jones	Mick Jones
9	100 Club	08-14-1975	The Clash	Paul	Simmon	
10	CBGBs	04-17-1972	The Clash	Paul	Simmon	
11	100 Club	08-14-1975	The Clash	Nicholas	Headon	Topper Headon
12	CBGBs	04-17-1972	The Clash	Nicholas	Headon	Topper Headon
13	100 Club	08-14-1975	Sex Pistols	Stephan	Jones	Steve Jones
14	100 Club	08-14-1975	Sex Pistols	Paul	Cook	
15	100 Club	08-14-1975	Sex Pistols	Glen	Matlock	
16	100 Club	08-14-1975	Sex Pistols	John	Lydon	Johnny Rotten
17	82 Club	05-10-1980	Dead Kennedys	Raymond	Pepperell	East Bay Ray
18	82 Club	05-10-1980	Dead Kennedys	Geoffrey	Lyall	Klaus Flouride
19	82 Club	05-10-1980	Dead Kennedys	Darren	Henley	D.H. Peligro
20	82 Club	05-10-1980	Dead Kennedys	Ron	Greer	Skip
21	OK Hotel	10-08-1995	Bikini Kill	Kathleen	Hanna	
22	OK Hotel	10-08-1995	Bikini Kill	Kathi	Wilcox	
23	OK Hotel	10-08-1995	Bikini Kill	Tobi	Vail	
24	OK Hotel	10-08-1995	Bikini Kill	Billy	Karren	
25	82 Club	05-10-1980	Misfits	Gerrard	Caiafa	Jerry Only
26	82 Club	05-10-1980	Misfits	Dennis	Cadena	Dez Cadena
27	82 Club	05-10-1980	Misfits	Eric	Arce	Chupacabra
28	82 Club	05-10-1980	Misfits	Glenn	Danzig	
29	82 Club	05-10-1980	Black Flag	Greg	Ginn	
30	82 Club	05-10-1980	Black Flag	Henry	Rollins	
31	82 Club	05-10-1980	Black Flag	Chuck	Dukowski	
32	82 Club	05-10-1980	Black Flag	Roberto	Valverde	Robo

View 2_____

View 2 creates a view of all the musicians' first and last names that had a drug problem, along with their stage name, band name, instrument and type of drug

```
CREATE VIEW DrugProblems AS
SELECT p.firstName,
       p.lastName,
       mu.stageName,
       b.bandName,
       bm.instrument,
       mu.drugOfChoice
FROM person p,
     musician mu,
     band_member bm,
     band b
WHERE mu.drugOfChoice IS NOT NULL
AND b.bandID=bm.bandID
AND p.pID=mu.pID
AND p.pID=bm.pID
AND mu.pID=bm.pID;
```

Results_____

	firstname text	lastname text	stagename text	bandname text	instrument text	drugofchoice text
1	Douglas	Colvin	Dee Dee Ramone	The Ramones	Bass	heroin
2	Nicholas	Headon	Topper Headon	The Clash	Drums	heroin
3	Raymond	Pepperell	East Bay Ray	Dead Kennedys	Guitar	cocaine
4	Geoffrey	Lyall	Klaus Flouride	Dead Kennedys	Bass	cocaine
5	Darren	Henley	D.H. Peligro	Dead Kennedys	Drums	cocaine
6	Ron	Greer	Skip	Dead Kennedys	Vocals	heroin
7	Gerrard	Caiafa	Jerry Only	Misfits	Bass	heroin
8	Henry	Rollins		Black Flag	Vocals	LSD
9	Chuck	Dukowski		Black Flag	Bass	LSD
10	Roberto	Valverde	Robo	Black Flag	Drums	cocaine

STORED PROCDURES

Stored Procedure 1_____

Stored Procedure 1 checks to make sure that a punk show is not scheduled at the same venue and time as another punk show. If so, the stored procedure raises an exception to notify the database administrator.

```
CREATE FUNCTION check_show_() RETURNS TRIGGER AS $$
BEGIN
IF EXISTS (SELECT showID
           FROM show_
           WHERE venueID=NEW.venueID
           AND dateOfShow= NEW.dateOfShow)
           THEN
           RAISE EXCEPTION 'Already scheduled a show at that venue and time';
           END IF;
           RETURN NEW;
           END;
$$ LANGUAGE plpgsql;
```

Stored Procedure 2_____

Stored Procedure 2 checks to make sure that a band member's first year with a band is equal to or after the band's year of creation, or that a band member's last year with a band is equal to or before the band's year of demise. If this is not the case, an exception is raised to notify the database administrator of the error.

```
CREATE FUNCTION check_years() RETURNS TRIGGER AS $$
BEGIN
IF EXISTS (SELECT b.bandID
           FROM band_memeber bm,
                band b
           WHERE bm.bandID=b.bandID
           AND (bm.startingYear >= b.startingYear
               OR bm.endingYear <= b.endingYear))
           THEN
           RAISE EXCEPTION 'Incorrect band member spanning years';
           END IF;
           RETURN NEW;
           END;
$$ LANGUAGE plpgsql;
```

TRIGGERS

Trigger 1_____

Trigger 1 calls the stored procedure check_show_( ) when inserting a new row in the table show_ to make sure that a punk show is not scheduled at the same venue and time as the show that is being inserted.

```
CREATE TRIGGER check_show BEFORE INSERT ON show_
        FOR EACH ROW EXECUTE PROCEDURE check_show_();
```

Trigger 2_____

```
CREATE TRIGGER check_years BEFORE INSERT ON band_member
        FOR EACH ROW EXECUTE PROCEDURE check_years();
```

Trigger 2 calls the stored procedure check_years( ) when inserting a new row in the table band_member to make sure that a band members first and last years with a band are within the bands active time span.

SECURITY

There are two primary user roles for the database; music_admin and music_enthusiasts. The music_admin has been granted all privileges on all tables (select, insert, update), so he/her has the ability to add, make changes and make queries within the database. The music_enthusiasts role has been grant only the privilege to select on all tables, so the music_enthusiast is able to make queries within the database but is not able to add or make changes to the information with the schema.

Music_admin_____

```
CREATE ROLE music_admin;
GRANT SELECT, INSERT, UPDATE
ON
sub_genre,
bands_of_sub_genre,
band,
band_member,
musician,
manager,
person,
producer,
album_producer,
band_manager,
album,
record_label,
show_,
venue,
lineups
TO music_admin;
```

Music_enthusiasts_____

```
CREATE ROLE music_enthusiasts;
GRANT SELECT
ON
sub_genre,
bands_of_sub_genre,
band,
band_member,
musician,
manager,
person,
producer,
album_producer,
band_manager,
album,
record_label,
show_,
venue,
lineups
TO music_enthusiasts;
```

IMPLEMENATION NOTES

When implementing this database design, it is at the administrator digression to decide what punk bands have contributed enough to the genre of punk to be added into the database. Also, the term “punk” can be defined very loosely, thus that, it becomes the administrator’s subjective opinion what bands are to be considered “punk bands.

It is important to note that although this specific database has been created around the music genre of Punk, this database design could be used to map out any genre of music depending on the administrator’s data input. Also if expanded, this database structure could potentially be used to for all genres of music.

KNOWN PROBLEMS

- Within the Band_Member table, band members are not able to play multiple instruments. When inputting sample data, a band member's primary instrument was added. Although, this becomes a problem if a band member plays two instruments, such as Vocals and Guitar. Because the 'instrument' field takes a data type Text, multiple instruments could hypothetically be added; i.e. 'Vocals and Guitar', but this would compromise queries that group band members by their instrument. To fix this problem an extra table could have been added to record all the instrument a band member plays.
- In the case that a band reunites, which is very common, the startingYear and endingYear fields within the Band table would not be able to illustrate this. Instead, the database structure only allows the bands initial beginning year and break up year to be recorded. To fix this, extra fields for reuniting could be added.
- In the case that a compilation album is made between two bands, the Album table only allows one bandID to be recorded. Although this is not a very common issue, additional tables could fix this problem.

FUTURE ENHANCEMENTS

The Punk Database could be enhanced with the addition of restrictions on specific fields, so that inconsistencies could not occur during data entry. Helpful restrictions would include but are not limited to: denying bands from being entered into the Lineup table if they are not actively playing during its date (band.startingYear < show_.dateOfShow<band.endingYear), producers being restricted from producing albums before/after they are alive and band members making albums before/after they are alive. Another enhancement that may be useful for understanding the genre of punk would be addition tables that would map out what bands where influences on other bands. If more time, effort and possibly payment were available, these enhancements could be made.