Comedians DB

Rebecca Murphy
12/1/ 2014

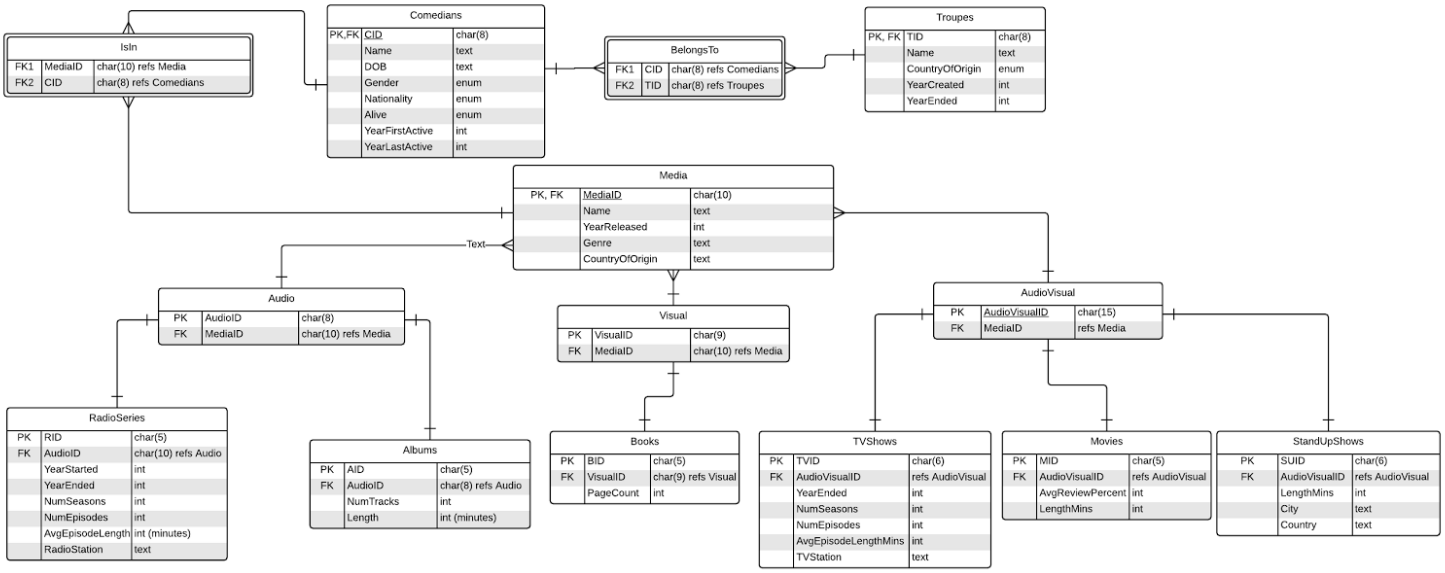
Table of Contents

Executive Summary……3
Entity Relationship Diagram……..4
Tables……...5
Comedians…… 5
Troupes…….. 6
BelongsTo……..7
Media……..8
IsIn……..9
Audio……..10
RadioSeries……...11
Album……12
Visual……13
Books……14
AudioVisual……...15
TVShows……...16
Movies……....17
StandUpShows……18
Views……....19
TroupesMembers View……19
ComediansMedia View……...20
TroupesMedia View……21
Reports and their Queries……22
British Comedians with TV Shows……....22
Media Featuring Female Comedians……23
Comedians that have released a form of Media in the past 20 years……24
Stored Procedures……25
GetTroupes('Comedian Name').....25
GetTroupeMembers('Troupe Name').....26
GetComedianMedia('Comedian Name').....27
GetTroupeMedia('Troupe Name').....28
Triggers……...29
Security……...30
Implementation Notes / Known Problems / Future Enhancements……..30

Executive Summary

The goal of this database what to provide users with a simple way to find new comedians and all of their material. It allows users to search by comedian, or troupe to find all media associate with said comedian or troupe, which the possible to qualify the media by type. The idea was that a user could qualify the type of media because they may only want a specific type for a specific purpose, e.g. wanting something to listen to for a car trip, the user may only want to return media of an audio type.

Entity Relationship Diagram



Tables

Comedians

The Comedians table holds the basic info for all the comedians in the database. This basic info is what would be of interest for someone looking for a new comedian to watch.

Create Statement

```
CREATE TYPE GENDER AS ENUM ('Female', 'Male', 'Other');
CREATE TABLE IF NOT EXISTS Comedians (
        CID             CHAR(8)         NOT NULL,
        Name            TEXT            NOT NULL,
        DOB             DATE            NOT NULL,
        Gender          GENDER          NOT NULL,
        Nationality     TEXT            NOT NULL,
        Alive           BOOL            NOT NULL,
        YearFirstActive INT             NOT NULL,
        YearLastActive  INT                     ,
        PRIMARY KEY (CID)
);
```

Functional Dependencies

CID → Name, DOB, Gender, Nationality, Alive, YearFirstActive, YearLastActive

Sample Data

	cid character(8)	name text	dob date	gender gender	nationality text	alive boolean	yearfirstactive integer	yearlastactive integer
1	C0000001	Eddie Izzard	1962	Male	British	t	1982	
2	C0000002	Graham Chapman	1941	Male	British	f	1960	1989
3	C0000003	Terry Gilliam	1940	Male	British	t	1967	
4	C0000004	Terry Jones	1942	Male	British	t	1969	
5	C0000005	John Cleese	1939	Male	British	t	1961	
6	C0000006	Michael Palin	1943	Male	British	t	1966	
7	C0000007	Kate Micucci	1980	Female	American	t	2001	
8	C0000008	Riki Lindhome	1979	Female	American	t	2001	
9	C0000009	Dara Ó Briain	1972	Male	Irish	t	1998	
10	C0000010	Dylan Moran	1971	Male	Irish	t	1992	
11	C0000011	Julian Barratt	1968	Male	British	t	1995	
12	C0000012	Noel Fielding	1973	Male	British	t	1996	

Troupes

Create Statement

```
CREATE TABLE IF NOT EXISTS Troupes (
        TID                 CHAR(8)             NOT NULL,
        Name                TEXT                NOT NULL,
        CountryOfOrigin     TEXT                NOT NULL,
        YearCreated         INT                 NOT NULL,
        YearEnded           INT                         ,
        PRIMARY KEY (TID)
);
```

Functional Dependencies

TID → Name, CountryOfOrigin, YearCreated, YearEnded

Sample Data

	tid character(8)	name text	countryoforigin text	yearcreated integer	yearended integer
1	T0000001	Monty Python	England	1969	1983
2	T0000002	Garfunkel and Oates	United States	2007	
3	T0000003	The Mighty Boosh	England	1998	2009

BelongsTo

Create Statement

```
CREATE TABLE IF NOT EXISTS BelongsTo (
        CID                     CHAR(8)         NOT NULL REFERENCES Comedians(CID),
        TID                     CHAR(8)         NOT NULL REFERENCES Troupes(TID)
);
```

Functional Dependencies

CID, TID →

Sample Data

	cid character(8)	tid character(8)
1	C0000002	T0000001
2	C0000003	T0000001
3	C0000004	T0000001
4	C0000005	T0000001
5	C0000006	T0000001
6	C0000007	T0000002
7	C0000008	T0000002
8	C0000011	T0000003
9	C0000012	T0000003

Media

Create Statement

```
CREATE TABLE IF NOT EXISTS Media (
        MediaID             CHAR(10)      NOT NULL,
        Name                TEXT          NOT NULL,
        YearReleased  INT             NOT NULL,
        Genre               TEXT          NOT NULL,
        CountryOfOrigin     TEXT          NOT NULL,
       PRIMARY KEY (MediaID)
);
```

Functional Dependencies

MediaID → Name, YearReleased, Genre, CountryOfOrigin

Sample Data

	mediaid character(10)	name text	yearreleased integer	genre text	countryoforigin text
1	Media00001	Monty Python and the Holy Grail	1975	Comedy	United Kingdom
2	Media00002	Monty Python's Flying Circus	1969	Sketch Comedy	United Kingdom
3	Media00003	All Over Your Face	2011	Comedy	United States
4	Media00004	Black Books	2000	Sitcom	United Kingdom
5	Media00005	This Is the Show	2010	Stand Up Comedy	United Kingdom
6	Media00006	Monty Pythons Big Red Book	1971	Comedy	United Kingdom
7	Media00007	The Boosh	2001	Surreal Comedy	United Kingdom

IsIn

Create Statement

```
CREATE TABLE IF NOT EXISTS IsIn (
        MediaID         CHAR(10)    NOT NULL REFERENCES Media(MediaID),
        CID             CHAR(8)     NOT NULL REFERENCES Comedians(TID)
);
```

Functional Dependencies

MediaID, CID→

Sample Data

	mediaid character(10)	cid character(8)
1	Media00001	C0000002
2	Media00001	C0000003
3	Media00001	C0000004
4	Media00001	C0000005
5	Media00001	C0000006
6	Media00002	C0000002
7	Media00002	C0000003
8	Media00002	C0000004
9	Media00002	C0000005
10	Media00002	C0000006
11	Media00003	C0000007
12	Media00003	C0000008
13	Media00004	C0000010
14	Media00005	C0000009
15	Media00006	C0000002
16	Media00006	C0000003
17	Media00006	C0000004
18	Media00006	C0000005
19	Media00006	C0000006
20	Media00007	C0000011
21	Media00007	C0000012

Audio

Create Statement

```
CREATE TABLE IF NOT EXISTS Audio (
        AudioID             CHAR(8)         NOT NULL,
        MediaID             CHAR(10)        NOT NULL REFERENCES Media(MediaID),
        PRIMARY KEY (AudioID)
);
```

Functional Dependencies

AudioID → MediaID

Sample Data

	audioid character(8)	mediaid character(10)
1	Audio001	Media00003
2	Audio002	Media00007

RadioSeries

Create Statement

```
CREATE TABLE IF NOT EXISTS RadioSeries (
        RID                             CHAR(5)         NOT NULL,
        AudioID                         CHAR(8)         NOT NULL REFERENCES
                                                        Audio(AudioID),
        YearEnded                       INT             NOT NULL,
        NumSeasons                      INT             NOT NULL,
        NumEpisodes             INT             NOT NULL,
        AvgEpisodeLengthMinINT          NOT NULL,
        RadioStation                    TEXT            NOT NULL,
        PRIMARY KEY (RID)
);
```

Functional Dependencies

RID → AudioID, YearEnded, NumSeasons, NumEpisodes, AvgEpisodeLengthMin, RadioStation

Sample Data

	rid character(5)	audioid character(8)	yearended integer	numseasons integer	numepisodes integer	avgepisodelengthmin integer	radiostation text
1	R0001	Audio002	2001	1	6	100	BBC

Album

Create Statement

```
CREATE TABLE IF NOT EXISTS Album (
        AID             CHAR(5)     NOT NULL,
        AudioID         CHAR(8)     NOT NULL REFERENCES Audio(AudioID),
        NumTracks       INT         NOT NULL,
        LengthMins      INT         NOT NULL,
        PRIMARY KEY (AID)
);
```

Functional Dependencies

AID → AudioID, NumTracks, LengthMins

Sample Data

	aid character(5)	audioid character(8)	numtracks integer	lengthmins integer
1	A0001	Audio001	10	25

Visual

Create Statement

```
CREATE TABLE IF NOT EXISTS Visual (
        VisualID            CHAR(9)     NOT NULL,
        MediaID             CHAR(10)    NOT NULL REFERENCES Media(MediaID),
        PRIMARY KEY (VisualID)
);
```

Functional Dependencies

VisualID → MediaID

Sample Data

	visualid character(9)	mediaid character(10)
1	Visual001	Media00006

Books

Create Statement

```
CREATE TABLE IF NOT EXISTS Books (
        BID             CHAR(5)         NOT NULL,
        VisualID        CHAR(9)         NOT NULL REFERENCES Visual(VisualID),
        PageCount       INT             NOT NULL,
        PRIMARY KEY (BID)
);
```

Functional Dependencies

BID → VisualID, PageCount

Sample Data

	bid character(5)	visualid character(9)	pagecount integer
1	B0001	Visual001	64

AudioVisual

Create Statement

```
CREATE TABLE IF NOT EXISTS AudioVisual (
        AudioVisualID       CHAR(15)        NOT NULL,
        MediaID             CHAR(10)        NOT NULL REFERENCES Media(MediaID),
        PRIMARY KEY (AudioVisualID)
);
```

Functional Dependencies

AudioVisualID → MediaID

Sample Data

	audiovisualid character(15)	mediaid character(10)
1	AudioVisual0001	Media00001
2	AudioVisual0002	Media00002
3	AudioVisual0003	Media00004
4	AudioVisual0004	Media00005

TVShows

Create Statement

```
CREATE TABLE IF NOT EXISTS TVShows (
        TVID                    CHAR(6)         NOT NULL,
        AudioVisualID           CHAR(15)        NOT NULL REFERENCES
                                                AudioVisual(AudioVisualID),
        YearEnded               INT             NOT NULL,
        NumSeasons              INT             NOT NULL,
        NumEpisodes             INT             NOT NULL,
        EpisodeLengthMins       INT             NOT NULL,
        TVStation               TEXT            NOT NULL,
        PRIMARY KEY (TVID)
);
```

Functional Dependencies

TVID → AudioVisualID, YearEnded, NumSeasons, NumEpisodes, EpisodeLengthMins, TVStation

Sample Data

	tvid character(6)	audiovisualid character(15)	yearended integer	numseasons integer	numepisodes integer	episodelengthmins integer	tvstation text
1	TV0001	AudioVisual0001	1974	4	45	30	BBC1
2	TV0002	AudioVisual0003	2004	3	18	25	Channel 4

Movies

Create Statement

```
CREATE TABLE IF NOT EXISTS Movies (
        MID                     CHAR(5)         NOT NULL,
        AudioVisualID           CHAR(15)        NOT NULL REFERENCES
                                                AudioVisual(AudioVisualID),
        AvgReviewPercent        INT             NOT NULL,
        LengthMins              INT             NOT NULL,
        PRIMARY KEY (MID)
);
```

Functional Dependencies

MID → AudioVisualID, AvgReviewPercent, LengthMins

Sample Data

	mid character(6)	audiovisualid character(15)	avgreviewpercent integer	lengthmins integer
1	M0001	AudioVisual0001	97	91

StandUpShows

Create Statement

```
CREATE TABLE IF NOT EXISTS StandUpShows (
        SUID                                    CHAR(6)         NOT NULL,
        AudioVisualID                   CHAR(15)        NOT NULL REFERENCES
                                                AudioVisual(AudioVisualID),
        LengthMins                              INT             NOT NULL,
        City                                    TEXT            NOT NULL,
        Country                                 TEXT            NOT NULL,
        PRIMARY KEY (SUID)
);
```

Functional Dependencies

SUID → AudioVisualID, LengthMins, City, Country

Sample Data

	suid character(6)	audiovisualid character(15)	lengthmins integer	city text	country text
1	SU0001	AudioVisual0004	102	London	England

Views

TroupesMembers View

Create Statement

```
DROP VIEW IF EXISTS TroupesMembers;
CREATE VIEW TroupesMembers
AS
SELECT DISTINCT t.name as "Troupe", c.name as "Comedian"
FROM Troupes as t, Comedians as c, BelongsTo as b
WHERE b.TID = t.TID AND b.CID = c.CID
ORDER BY t.name;
```

Sample Data

	Troupe text	Comedian text
1	Garfunkel and Oates	Kate Micucci
2	Garfunkel and Oates	Riki Lindhome
3	Monty Python	Graham Chapman
4	Monty Python	John Cleese
5	Monty Python	Michael Palin
6	Monty Python	Terry Gilliam
7	Monty Python	Terry Jones
8	The Mighty Boosh	Julian Barratt
9	The Mighty Boosh	Noel Fielding

ComediansMedia View

Create Statement

```
DROP VIEW IF EXISTS ComediansMedia;
CREATE VIEW ComediansMedia
AS
SELECT DISTINCT c.name as "Comedian", m.name as "Media"
FROM Media as m, Comedians as c, IsIn as relate
WHERE relate.MediaID = m.MediaID AND relate.CID = c.CID
ORDER BY c.name;

select * from ComediansMedia;
```

Sample Data

	Comedian text	Media text
1	Dara Ó Briain	This Is the Show
2	Dylan Moran	Black Books
3	Graham Chapman	Monty Python and the Holy Grail
4	Graham Chapman	Monty Pythons Big Red Book
5	Graham Chapman	Monty Python’s Flying Circus
6	John Cleese	Monty Python and the Holy Grail
7	John Cleese	Monty Pythons Big Red Book
8	John Cleese	Monty Python’s Flying Circus
9	Julian Barratt	The Boosh
10	Kate Micucci	All Over Your Face
11	Michael Palin	Monty Python and the Holy Grail
12	Michael Palin	Monty Pythons Big Red Book
13	Michael Palin	Monty Python’s Flying Circus
14	Noel Fielding	The Boosh
15	Riki Lindhome	All Over Your Face
16	Terry Gilliam	Monty Python and the Holy Grail
17	Terry Gilliam	Monty Pythons Big Red Book
18	Terry Gilliam	Monty Python’s Flying Circus
19	Terry Jones	Monty Python and the Holy Grail
20	Terry Jones	Monty Pythons Big Red Book
21	Terry Jones	Monty Python’s Flying Circus

TroupesMedia View

Create Statement

```
DROP VIEW IF EXISTS TroupesMedia;
CREATE VIEW TroupesMedia AS
SELECT DISTINCT t.name, mediaid
FROM BelongsTo AS b
join IsIn AS relate
ON
b.CID = relate.CID
join Troupes AS t
ON
b.TID = t.TID
ORDER BY t.name;
```

Sample Data

	name text	mediaid character(1
1	Garfunkel and Oates	Media00003
2	Monty Python	Media00001
3	Monty Python	Media00002
4	Monty Python	Media00006
5	The Mighty Boosh	Media00007

Reports and their Queries

British Comedians with TV Shows

Displays all British Comedians with TV Shows

Query

```
select distinct c.name as "Comedian", m.name as "TV Show"
from TVShows as TV, AudioVisual as AV,
IsIn as relate
join Media as m
on m.MediaID = relate.MediaID
join Comedians c
on relate.CID = c.CID
where
AV.MediaID = m.MediaID AND TV.AudioVisualID = AV.AudioVisualID
```

Sample Data

	Comedian text	TV Show text
1	Terry Jones	Monty Python's Flying Circus
2	Dylan Moran	Black Books
3	Graham Chapman	Monty Python's Flying Circus
4	Terry Gilliam	Monty Python's Flying Circus
5	John Cleese	Monty Python's Flying Circus
6	Michael Palin	Monty Python's Flying Circus

Media Featuring Female Comedians

Displays all media associated with female comedians

Query

```
select c.name, m.name
from IsIn as relate
join Comedians as c
on c.gender = 'Female' and relate.CID = c.CID
join Media as m
on m.MediaID = relate.MediaID;
```

Sample Data

	name text	name text
1	Kate Micucci	All Over Your Face
2	Riki Lindhome	All Over Your Face

Comedians that have released a form of Media in the past 20 years

Displays comedians that have semi-recently released something.

Query

```
select distinct c.name
from Media as m
join
Isin as relate
on
relate.MediaID = m.MediaID
join Comedians as c
on relate.CID = C.CID
where m.yearreleased > (2014-20);
```

Sample Data

	name text
1	Dylan Moran
2	Dara Ó Briain
3	Riki Lindhome
4	Noel Fielding
5	Kate Micucci
6	Julian Barratt

Stored Procedures

GetTroupes('Comedian Name')

Gets all troupes the specified comedian name has ever been in

Create Statement

```
DROP FUNCTION IF EXISTS GetTroupes(cname text);
CREATE OR REPLACE FUNCTION GetTroupes(cname text)
RETURNS SETOF TEXT AS $$
BEGIN
RETURN QUERY
SELECT t.name
from Troupes as t
where t.tid in
        (select TID
        from BelongsTo as relate
        join Comedians as c
        on relate.CID = c.CID and c.name = cname
        )
order by t.name;
END;
$$ LANGUAGE plpgsql;
```

Sample Data

```
select GetTroupes('Graham Chapman');
```

	gettroupes text
1	Monty Python

GetTroupeMembers(‘Troupe Name’)

Gets all members of specified troupe name

Create Statement

```
DROP FUNCTION IF EXISTS GetTroupeMembers(tname text);
CREATE OR REPLACE FUNCTION GetTroupeMembers(tname text)
RETURNS SETOF TEXT AS $$
BEGIN
RETURN QUERY
SELECT c.name
from Comedians as c
where c.cid in
        (select CID
        from BelongsTo as relate
        join Troupes as t
        on relate.TID = t.TID and t.name = tname
        )
order by c.name;
END;
$$ LANGUAGE plpgsql;
```

Sample Data

select GetTroupeMembers('Monty Python');

	gettroupemembers text
1	Graham Chapman
2	John Cleese
3	Michael Palin
4	Terry Gilliam
5	Terry Jones

GetComedianMedia('Comedian Name')

Gets all media associated with the comedian

Create Statement

```
DROP FUNCTION IF EXISTS GetComedianMedia(cname text);
CREATE OR REPLACE FUNCTION GetComedianMedia(cname text)
RETURNS SETOF TEXT AS $$
BEGIN
RETURN QUERY
SELECT m.name
from Media as m
where m.MediaID in
        (select MediaID
        from IsIn as relate
        join Comedians as c
        on relate.CID = c.CID and c.name = cname
        )
order by m.name;
END;
$$ LANGUAGE plpgsql;
```

Sample Data

```
select GetComedianMedia('Dylan Moran');
```

	getcomedianmedia text
1	Black Books

GetTroupeMedia(‘Troupe Name’)

Gets all media of specified troupe name

Create Statement

```
DROP FUNCTION IF EXISTS GetTroupeMedia(tname text);
CREATE OR REPLACE FUNCTION GetTroupeMedia(tname text)
RETURNS SETOF TEXT AS $$
BEGIN
RETURN QUERY
SELECT m.name
from Media as m
where m.MediaID in
        (select MediaID
        from IsIn as relate
        join Comedians as c
        on relate.CID = c.CID
        join BelongsTo as b
        on b.CID = c.CID
        join Troupes as t
        on b.TID = t.TID and tname = t.name
        )
order by m.name;
END;
$$ LANGUAGE plpgsql;
```

Sample Data

```
select GetTroupeMedia('Monty Python');
```

	gettroupemedia text
1	Monty Python and the Holy Grail
2	Monty Pythons Big Red Book
3	Monty Python’s Flying Circus

Triggers

This trigger listens for a delte on the BelongsTo table, and inserts the deleted record into a table called quit_troupe, which keeps a record of comedians that formerly belonged to a troupe.

Create statement

```
drop trigger if exists delete_relation_listener on BelongsTo;
drop function if exists delete_listener();
drop table if exists quit_troupe;

create table quit_troupe (
        CID                     CHAR(8)         NOT NULL,
        TID                     CHAR(8)         NOT NULL
);

create function delete_listener() returns trigger as $$
begin
   insert into quit_troupe values (old.CID, old.TID);
 return null;
end;
$$ language plpgsql;
create trigger delete_relation_listener after delete on BelongsTo
 for each row execute procedure delete_listener();
```

Test Statement

```
Delete from BelongsTo where 'C0000002' = CID;
Select * from quit_troupe;
```

Test Output

	cid character(8)	tid character(8)
1	C0000002	T0000001

Security

For this database there would be 2 basic roles, or types of users.

Admin: Has permission to change, update, and maintain the database

```
CREATE ROLE Admin
GRANT SELECT, INSERT, UPDATE, ALTER
ON ALL TABLES IN SCHEMA PUBLIC
TO Admin;
```

User: A normal user who would only be able to see the database and perform queries.

```
CREATE ROLE User
GRANT SELECT
ON ALL TABLES IN SCHEMA PUBLIC
TO User
```

Implementation Notes / Known Problems / Future Enhancements

The goal of the implementation was to be as general as possible with the media types, so new kinds of media could be added easily. However, this lead to a lot of weak entities being used if the tables were to remain normalized. It also leads to a lot of foreign key use, which may be tedious and confusing for user at first. There was also plans to be a separate attribute for each of the tables for the type of media, for mediums available, e.g. you could buy an albumn on Itunes, CD, etc. However, I couldn't think of a good way to implement this without creating more weak entities which would have created an even messier ER diagram. Also currently a media entry can only have one genre, which really limits the categorization of media, but it was not implemented for a similar reason as mediums available.

Future enhancements would be to create a way to let media have more than one genre. There is also not a way to show breaks in years a comedian or a troupe was active, currently it is just the original range.

I tried to make my trigger check that numbers in all the tables were never negative. However, SQL would not let me. I left the code I did in the sql script just to show what I was trying to do, but I ended up doing a simplified version.