

Simpsons Character Database

## Iglika Hadjiyska Database Management Spring 2016

Table of Contents

### Executive Summary 3

### Entity Relationship Diagram 4

### Tables 5-12

characters

episodes

voiceActors

occupations

animals

owners

jobs

voiceCharacter

characterEpisode

animalOwn

### Views 13-14

whichVoice

whichEpisode

### Reports 15-16

### Stored Procedures 17-19

frequencyVoice

jobschar

addOwnerAnimal

### Triggers 19

addOwnerAnimal()

### Security 20

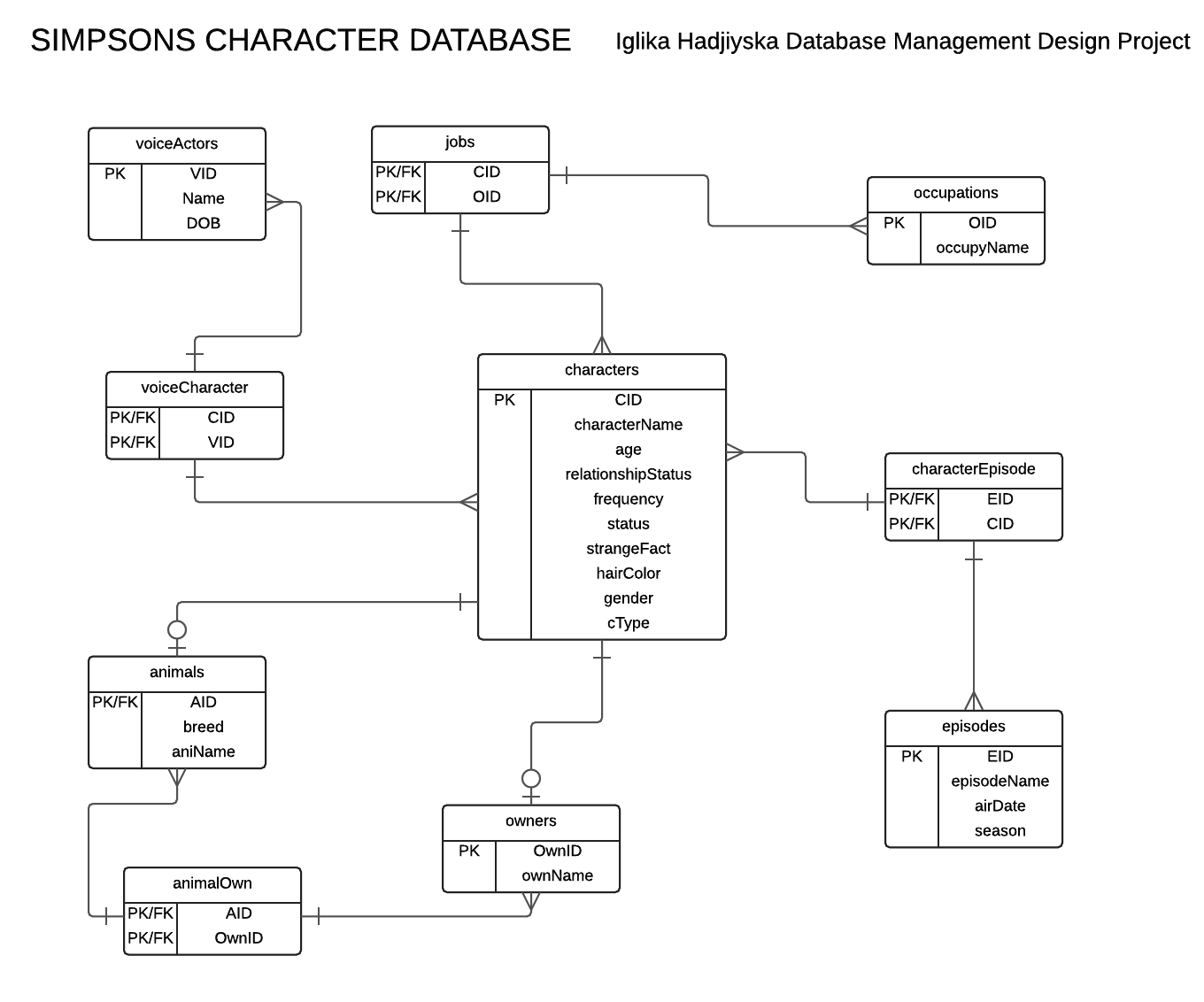
### Notes/Problems/Future Enhancements 21

Executive Summary

## This document is meant to showcase the structure and functions of a database designed for The Simpsons television show. The show has close to 2,000 characters and counting, as well as an enormous numbers of episodes and voice actors. As a result, this database has been compiled to track these characters and information about them. Since there is so much information, the scope of this database has been reduced to the first nine seasons, with only canon episodes and characters. This is meant to keep this instance of the database short and manageable although the implemented database would be larger in scale. The expected users can be the viewing public but more specifically, it would be the show runners, as an attempt to provide consistency in characters, which has been lost in the recent seasons of the show.

## The objective of this database is to provide quick access to characters in case of discontinuity within an episode, or to provide statistics about the characters themselves. As such, both the public and the writers would only be allowed certain security functions, as will be discussed later on.

Entity Relationship Diagram



Tables

## characters

This table lists the characters and their attributes

CREATE TABLE characters (

cid integer NOT NULL,

characterName varchar(100) NOT NULL DEFAULT 'not specified',

age integer NOT NULL,

relationshipStatus varchar(100) NOT NULL DEFAULT 'single',

frequency varchar(100) DEFAULT 'not specified',

status varchar(100) DEFAULT 'alive',

strangeFact text DEFAULT 'lives in the Simpsons universe',

hairColor varchar(100) DEFAULT 'not specified',

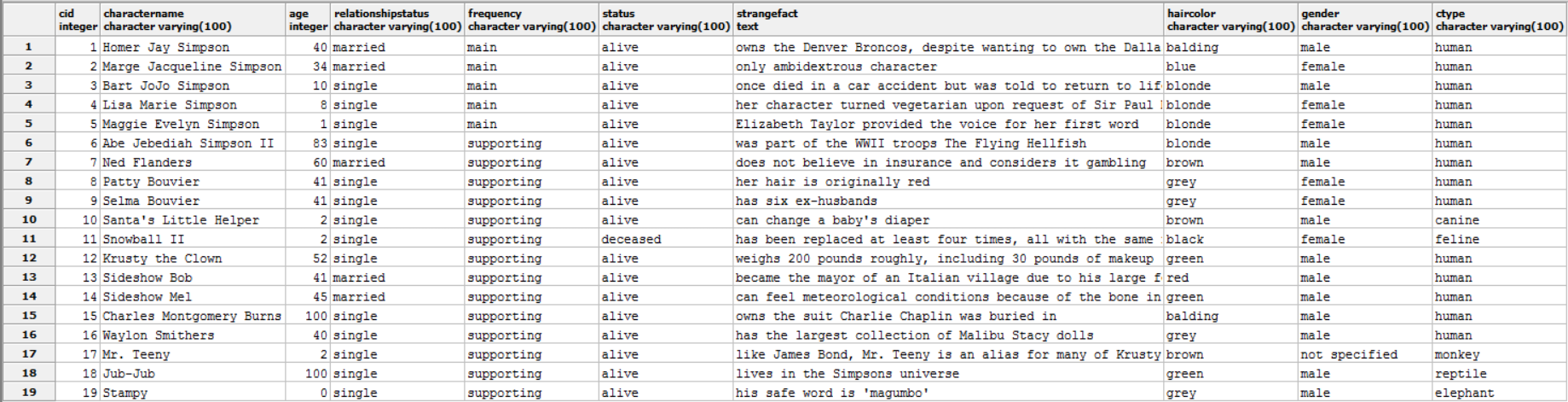
gender varchar(100) DEFAULT 'not specified',

cType varchar(100) DEFAULT 'not specified',

primary key (cid)

);

(cid) 🡪 characterName, age, relationshipStatus, frequency, status, strangeFact, hairColor, gender, cType



## episodes

This tables lists the first episode a character appeared in

CREATE TABLE episodes (

eid integer NOT NULL,

episodeName varchar(100) NOT NULL DEFAULT 'not specified',

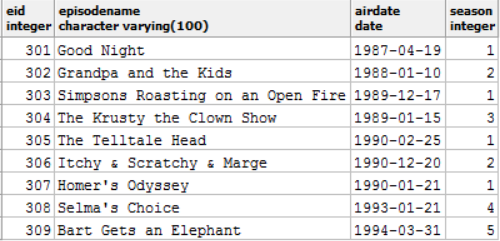
airDate date,

season int,

primary key (eid)

);

(eid) 🡪 episodeName, airdate, season



## voiceActors

This table showcases the voice actors of all the characters

CREATE TABLE voiceActors (

vid integer NOT NULL,

actorName varchar(100) NOT NULL DEFAULT 'not specified',

dob date,

primary key (vid)

);

(vid) 🡪 actorName, dob



## occupations

This table lists all of the jobs that characters have ever had

CREATE TABLE occupations (

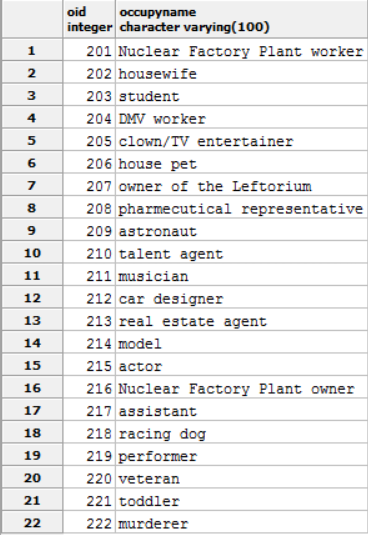
oid integer NOT NULL,

occupyName varchar(100) NOT NULL DEFAULT 'not specified',

primary key (oid)

);

(oid) 🡪 occupyName



## animals

This is an entity subtype of the table characters. Some characters are animals and have to be separated accounted for. The attribute ‘breed’ accounts for the specific type of animal, elaborating on the ‘cType’ attribute of the characters table.

CREATE TABLE animals (

aid integer NOT NULL references characters(cid),

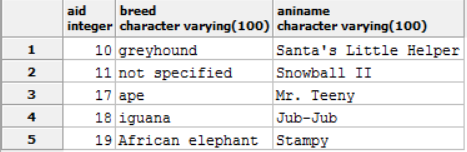
breed varchar(100) DEFAULT 'not specified',

aniName varchar(100) DEFAuLT 'not specified',

primary key (aid)

);

(aid) 🡪 breed, aniName



## owners

This is also an entity subtype of characters. Some of the animals are pets and as a result, must have owners.

CREATE TABLE owners (

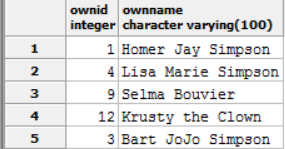
ownid integer NOT NULL references characters(cid),

ownName varchar(100) DEFAULT 'not specified',

primary key (ownid)

);

(ownid) 🡪 ownName



## jobs

This is a linking table between characters and occupations, meant to link together the character names with their job names.

CREATE TABLE jobs (

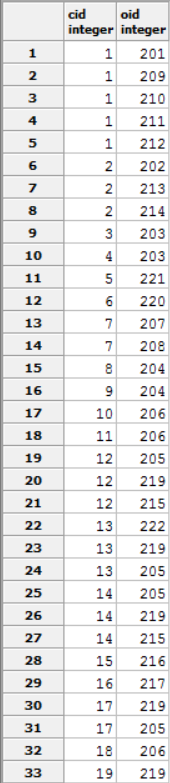
cid integer NOT NULL references characters(cid),

oid integer NOT NULL references occupations(oid),

primary key (cid, oid)

);

(cid,oid) 🡪



## voiceCharacter

This table is also a linking table between voiceActors and characters. It meant to link the two together, by their IDs.

CREATE TABLE voiceCharacter (

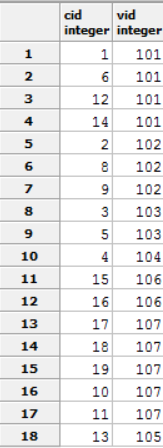
cid integer NOT NULL references characters(cid),

vid integer NOT NULL references voiceActors(vid),

primary key (cid, vid)

);

(cid, vid) 🡪



## characterEpisode

This table links characters and episodes, to show which characters were in which episdoes.

CREATE TABLE characterEpisode (

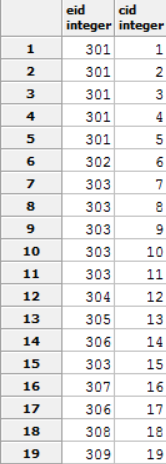
eid integer NOT NULL references episodes(eid),

cid integer NOT NULL references characters(cid),

primary key (eid, cid)

);

(eid, cid) 🡪



## animalOwn

This is a linking table between animals and owners, meant to link the two with their IDS.

CREATE TABLE animalOwn (

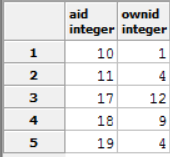
aid integer NOT NULL references animals(aid),

ownid integer NOT NULL references owners(ownid),

primary key(aid, ownid)

);

(aid, ownid) 🡪



Views

## whichVoice

This is a view meant to pair characters with their respective voice actors.

CREATE OR REPLACE VIEW whichVoice AS

SELECT c.characterName, va.actorName

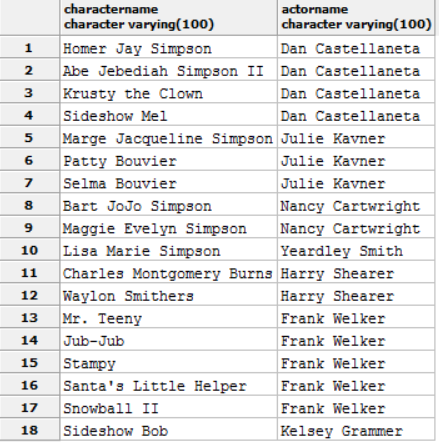
FROM characters c

INNER JOIN voiceCharacter vc

ON vc.cid=c.cid

INNER JOIN voiceActors va

ON vc.vid = va.vid;



## whichEpisode

This serves mostly the same purpose as the other view, but shows the characters with the episode that they first appeared in.

CREATE OR REPLACE VIEW whichEpisode AS

SELECT e.episodeName, e.airDate, c.characterName

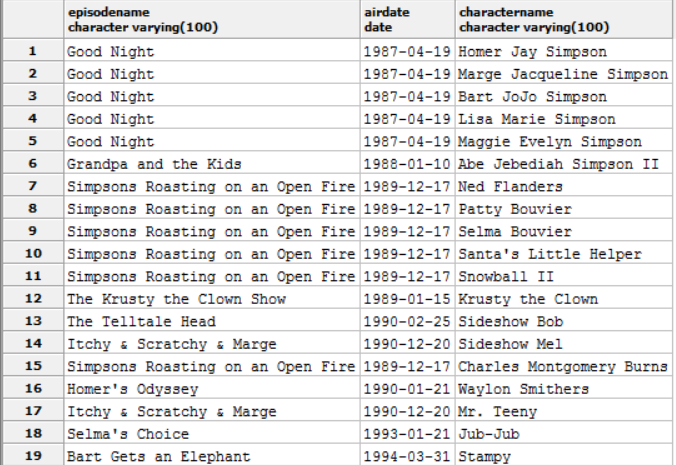
FROM episodes e

INNER JOIN characterEpisode ce

ON e.eid = ce.eid

INNER JOIN characters c

ON ce.cid = c.cid;



Reports

This report shows the jobs that each character has.

SELECT o.occupyName, c.characterName

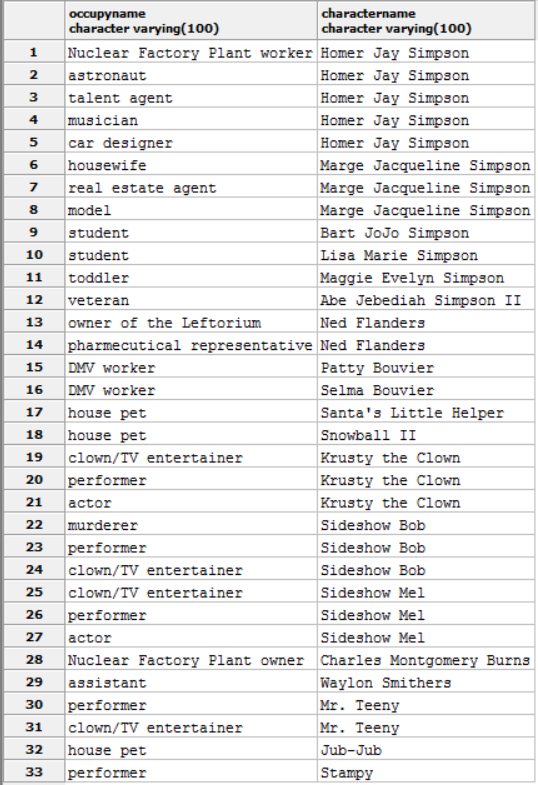
FROM occupations o

INNER JOIN jobs j

ON o.oid = j.oid

INNER JOIN characters c

ON j.cid = c.cid;



This report displays the character names of the owners with their respective pets.

SELECT o.ownName, a.aniName

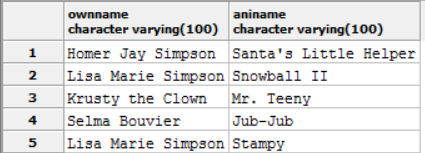
FROM owners o

INNER JOIN animalOwn ao

ON o.ownid = ao.ownid

INNER JOIN animals a

ON ao.aid = a.aid;



Stored Procedures

## frequencyVoice

This stored procedure takes in the name of a voice actor, then outputs the number of characters they voice for the show.

CREATE OR REPLACE FUNCTION frequencyVoice(varchar(100), REFCURSOR)

RETURNS refcursor as $$

DECLARE

actorNameInput varchar(100) := $1;

resultset REFCURSOR := $2;

BEGIN

open resultset for

SELECT count(actorNameInput)

FROM whichVoice

WHERE actorNameInput = actorName;

return resultset;

end;

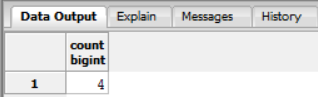
$$

language plpgsql;

SELECT frequentVoice('Dan Castellaneta', 'results');

FETCH ALL FROM results;

The return value for this is four characters, based on the sample data inputted.



## jobschar

This stored procedure displays all of the jobs that an inputted character has had.

CREATE OR REPLACE FUNCTION jobschar(varchar(100), REFCURSOR)

RETURNS refcursor as $$

DECLARE

characterInput varchar(100) := $1;

resultset REFCURSOR := $2;

BEGIN

open resultset for

SELECT o.occupyName, c.characterName

FROM occupations o

INNER JOIN jobs j

ON o.oid = j.oid

INNER JOIN characters c

ON j.cid = c.cid

where characterInput = c.characterName;

return resultset;

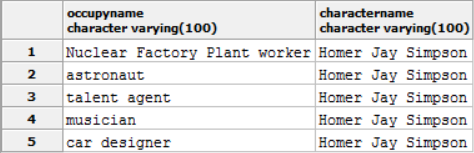
end;

$$

language plpgsql;

SELECT jobschar('Homer Jay Simpson', 'results');

FETCH ALL FROM results;



## addjobs()

This stored procedure updates the jobs linking table if a new character or occupation is added.

CREATE OR REPLACE FUNCTION addjobs()

RETURNS TRIGGER AS $body$

BEGIN

INSERT INTO jobs (cid, oid)

values(new.cid, new.oid);

END

$body$

language plpgsql;

Triggers

## addOccupation

This trigger updates on the occupation table, by adding any data inserted into the table into the jobs table.

CREATE TRIGGER addOccupation

AFTER INSERT ON occupations

EXECUTE PROCEDURE addjobs();

## addCharacter

This trigger updates on the character table alternatively. There must be two different triggers because there are two separate updates on two separate tables.

CREATE TRIGGER addCharacter

AFTER INSERT ON characters

EXECUTE PROCEDURE addjobs();

Security

The only users of this database would be the administrators and the public (user). The user role includes the writers of the show and other viewers who wish to access information. The admin naturally are granted all features because they must upkeep the database. The users are only given a SELECT permission so they can view but not alter any information within the database.

CREATE ROLE admin;

GRANT ALL ON ALL TABLES

IN SCHEMA PUBLIC

TO admin;

CREATE ROLE user

GRANT SELECT

ON ALL TABLES IN SCHEMA PUBLIC

TO user;

Notes, Problems and Enhancements

The implementation of the database was fairly easy, considering it was tracking on characters and a few other things associated with them, like voice actors and episodes. I chose to include less data to make the database more manageable, otherwise the database would be too big to display here, considering how many characters are in the first nine seasons. The problems I ran into were mostly trying to display certain kinds of information through views and queries, such as the table that displays owner names and animal names. I also encountered problems with the stored procedures and triggers, mostly with monitoring inserts on certain tables. If I could enhance the database more, I would attempt to add all of the data needed for the characters. I would hesitate from including seasons beyond season 10, for continuity reasons; most characters began to be altered beyond what would make sense for their character. There might also be redundant and/or contradicting data. Furthermore, it would be interesting to create a database of the entire town of Springfield, including stores, streets, and important town members. However, that database would be far beyond the scope allowed for this document.