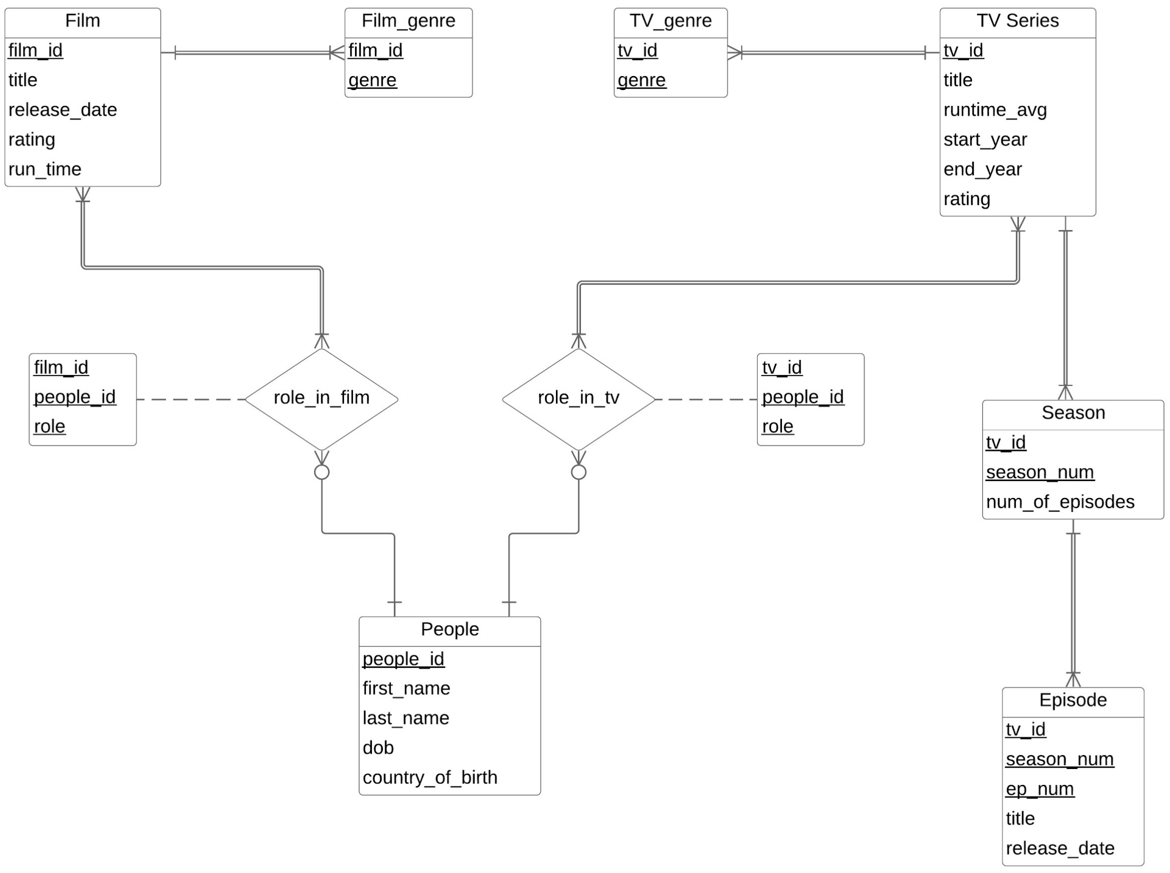
Introduction

The database that we chose to create for our project was IMDb (Internet Movie Database) inspired. IMDb is a robust collection of all many different pieces of media that include, but are not limited to: films, television programs, and home videos. With the amount of information and data that crosses over between these medians, a database model is the perfect type of approach to handling and sorting this data properly.

Our goal for this project is to create an accurate model that encapsulates the important aspects of the real IMDb and highlight some of the features that make it a continuously reliable source for handling intricacies in data. Another important aspect we realized early on was that the user will likely want to search and sort by specific aspects of the media they consume, such as number of episodes, watch time, and even release window. Other notable findings were that lots of the same people can be involved with these projects, this helped shape the approach we took to creating our database model.

ER Model



Our team's creation of the ER diagram was our first big step into deciding how exactly our system would run. We decided to break it up into three major sections, these being our people table, TV series table, and our film table. We chose these three because they most accurately represent the model we were trying to replicate, as home video could easily be grouped in with movies released in the theatre. TV series on the other hand had other distinctions that separated them from being grouped in with movies, such as runtimes for episodes and times or release.

One thing that we had to consider in our model was how different TV series were from movies. TV series are generally not released all at once, so we had to create more tables to correctly represent information on season and the information about those seasons, such as number of episodes, and release dates. We decided it is easier to include tv_id in Season table and tv_id and season_num in Episode table and create composite primary keys instead of having relationships like hasSeason and hasEpisode which created not only 2 additional tables but also required we introduced artificial primary keys to season and episode which resulted in a lot of duplicate data.

The ER diagram we decided to go with was not our first design, we initially had many different tables that would act as different positions that people held. This turned out to be counter intuitive because it would create many different tables that would hold lots of redundant information, as many of these people would cross over with each other or hold multiple positions. Instead we chose to create a general people table that gathers the general informational all people would have in common and handle their roles through relations with film and Tv series tables. This change not only cut down on redundancy, but also made the relation of people with the media easier to represent.

Relational Schema:

Film(film_id, title, release_year, rating, run_time)
TVSeries(tv_id, title, runtime_avg, start_year, end_year, rating)
Season(tv_id, season_num, num_of_episodes)
Episode(tv_id, season_num, ep_num, title, release_date)
People(people_id, first_name, last_name, dob)

Role_in_film(film_id, people_id, role)
Role_in_tv(tv_id, people_id, role)
film_gen(film_id, genre)
tv_gen(tv_id, genre)

Our schema was created so that we could easily integrate people with their roles into our data. Having separate relations for roles in TV series and films made it more intuitive to support a large cast, or multiple people heading different roles. This concept can also be seen with how genre is presented as a relation of TV series and film tables respectively. Each of the main tables contains the primary key that is referenced for the relations as needed. Finally, because of how different TV series is than movies, we needed to create a table for

seasons and for episodes as TV series are often broken up using this convention and can have varying degrees of length and episode number.
Some of our functional dependencies include film_id to everything else in the table, as the primary key is the main dependency. This can also be seen in all of our other tables, where the same sort of relationship exists with all of our other primary keys and attributes

SQL Schema:

```
CREATE TABLE Film(
    film_id INT,
    title VARCHAR(255) NOT NULL,
    release_year INT NOT NULL,
    rating FLOAT NOT NULL,
    run_time_min INT NOT NULL,
    PRIMARY KEY (film_id)
);
CREATE TABLE TvSeries(
    tv_id INT,
    title VARCHAR(255) NOT NULL,
    runtime_avg INT NOT NULL,
    start_year INT NOT NULL,
    end_year VARCHAR(255) NOT NULL,
    rating FLOAT NOT NULL,
    PRIMARY KEY (tv_id)
);
CREATE TABLE Season(
    tv_id INT,
    season_num INT NOT NULL,
    num_of_episodes INT NOT NULL,
    PRIMARY KEY (tv_id, season_num),
    FOREIGN KEY (tv_id) REFERENCES TvSeries
    ON DELETE CASCADE
);
CREATE TABLE Episode(
    tv_id INT,
    season_num INT NOT NULL,
    ep_num INT NOT NULL,
    title VARCHAR(255),
    release_date DATE,
    PRIMARY KEY (tv_id, season_num, ep_num),
    FOREIGN KEY (tv_id, season_num) REFERENCES Season
    ON DELETE CASCADE
);
CREATE TABLE People(
    people_id INT NOT NULL,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    dob DATE NOT NULL,
    country_of_birth VARCHAR(255) NOT NULL,
    PRIMARY KEY (people_id)
);
CREATE TABLE Role_in_film(
    film_id INT,
    people_id INT NOT NULL,
    roleIn VARCHAR(255) NOT NULL,
    PRIMARY KEY (film_id, people_id, roleIn),
    FOREIGN KEY (film_id) REFERENCES Film
    ON DELETE CASCADE,
    FOREIGN KEY (people_id) REFERENCES People
    ON DELETE CASCADE
);
CREATE TABLE Role_in_tv(
    tv_id INT,
    people_id INT NOT NULL,
    roleIn VARCHAR(255) NOT NULL,
    PRIMARY KEY (tv_id, people_id, roleIn),
    FOREIGN KEY (tv_id) REFERENCES TvSeries
    ON DELETE CASCADE,
    FOREIGN KEY (people_id) REFERENCES People
    ON DELETE CASCADE
);
CREATE TABLE Film_gen(
    film_id INT,
    genre VARCHAR(255) NOT NULL,
    PRIMARY KEY (film_id, genre),
    FOREIGN KEY (film_id) REFERENCES Film
    ON DELETE CASCADE
);
CREATE TABLE Tv_gen(
    tv_id INT,
    genre VARCHAR(255) NOT NULL,
    PRIMARY KEY (tv_id, genre),
    FOREIGN KEY (tv_id) REFERENCES TvSeries
    ON DELETE CASCADE
);
```

Normalization

As we were designing the schemas, we looked at the way TV shows and films are designed and broke them down to its entities and their relationships. There was times that we felt like some attributes would conflict others within the same entity, so we just created separate tables that would allow for better search time and just makes sense in terms of design structure that tv shows and films present. Therefore, the current schemas we have did not need to be normalized. Most, if not all information is unique to their specific tv show which removes duplicate data unless absolutely necessary for relationships. There is no trivial functional dependencies within our schema; each schema is functionally dependent only on their primary key.

Queries:

To run the code please type .read final363SQL.sql into your terminal after running SQLite; all the tables and views will be created, sample data inserted and all queries should give the same results.

The queries we chose to implement for our database were ones that we found would appear commonly on the actual IMDb page. We looked at what people would commonly search when looking for a movie/TV series that they either did not know all of the information to, or common information commonly related to the movie, such as rating, runtime, episode information, and season information.This was another determining factor we considered while making these queries.

Example queries about genres:

```
--Show title, release year and genres for films that are both action and
adventure.
select faction.title, faction.release_year, faction.genre,
fadventure.genre from faction join fadventure on faction.film_id =
fadventure.film_id;
```

Result:

```
Thor: Ragnarok|2017|Action|Adventure
Avengers: Endgame|2019|Action|Adventure
Avengers: Infinity War|2018|Action|Adventure
Captain America: Civil War|2016|Action|Adventure
Spider-Man: Homecoming|2017|Action|Adventure
```

```
--Show title, release year directors and genres for films that are both
action and adventure. Includes nulls if director is not yet in the
database (due to limited amount of sample data).
select title, first_name, last_name, year, genre1, genre2 from  (
select faction.film_id as film_idj, faction.title as title,
faction.release_year as year, faction.genre as genre1, fadventure.genre
as genre2 from faction join fadventure on faction.film_id =
fadventure.film_id
) left join fdirectors on  film_idj = fdirectors.film_id order by title;
```

Result:

```
Avengers: Endgame|Anthony|Russo|2019|Action|Adventure
Avengers: Endgame|Joe|Russo|2019|Action|Adventure
Avengers: Infinity War|||2018|Action|Adventure
Captain America: Civil War|||2016|Action|Adventure
Spider-Man: Homecoming|||2017|Action|Adventure
Thor: Ragnarok|Taika|Waititi|2017|Action|Adventure
```

```
--Show titles and genres for films AND TV series that are both action and
adventure.
select tvaction.title, tvaction.start_year, tvaction.genre,
tvadventure.genre from tvaction join tvadventure on tvaction.tv_id =
tvadventure.tv_id
union
select faction.title, faction.release_year, faction.genre,
fadventure.genre from faction join fadventure on faction.film_id =
fadventure.film_id;
```

Result:

```
Avengers: Endgame|2019|Action|Adventure
Avengers: Infinity War|2018|Action|Adventure
Captain America: Civil War|2016|Action|Adventure
Spider-Man: Homecoming|2017|Action|Adventure
The Arrow|2012|Action|Adventure
The Flash|2014|Action|Adventure
Thor: Ragnarok|2017|Action|Adventure
```

We have created multiple views on the original tables that allow us to join them by genre or profession. We chose action and adventure as the example as a lot of our sample films were of both genres. The data can be easily manipulated and any combination of genres can be searched. We also included a query about film titles, directors and genres to show how multiple tables are related.

Example queries about people:

```
--People with roles both in TV and film
select distinct first_name, last_name from (
select * from people natural join (
select * from Role_in_film join  Role_in_tv on  Role_in_tv.people_id =
 Role_in_film.people_id
));
```

Result:

```
Anthony|Russo
Joe|Russo
Chris|Pratt
Amy|Poehler
```

```
--People born after 1980, display name, date of birth, country of birth
order alphabetically by surname
select last_name, first_name, dob, country_of_birth from people where
 cast(substr(dob, 1, 4) as int) >= 1980 order by last_name;
```

Result:

```
Benoist|Melissa|1988-10-04|USA
Evans|Chris|1981-06-13|USA
Gustin|Grant|1990-01-14|USA
Hemsworth|Chris|1983-08-11|Australia
Johansen|Scarlett|1984-11-22|USA
Muniz|Frankie|1985-12-05|USA
```

```
--People not born in USA
```

```
select first_name, last_name, country_of_birth from people where
country_of_birth != "USA" order by last_name;
```
Result:
```
Chris|Hemsworth|Australia
Taika|Waititi|New Zealand
James|Wan|Malaysia
```

```
-- Count how many films the person acted in.
select first_name, last_name, count(roleIn) from role_in_film natural
join people where roleIn = "Actor" group by people_id order by
count(roleIn) desc, last_name;
```
Result:
```
Robert|Downey Jr|3
Chris|Evans|3
Chris|Hemsworth|3
Scarlett|Johansen|3
Mark|Ruffalo|3
Vin|Diesel|2
Chris|Pratt|2
Taika|Waititi|2
George|Clooney|1
Amy|Poehler|1
```

Although not overly complicated those queries would be fairly popular as often users are only interested in one attribute not a combination of them. For example how many films a certain person directed/produced/acted in.

Example query about people from film and TV:
```
-- People who directed, produced, wrote and acted in a movie or TV
series. Quadruple threats.
-- Display name, titles and year; order by year released descending then
title alphabetically.
select fn, ln, title, year from (
select people_idF, fn, ln, fi , title, release_year as year from (
select fwriters.people_id as people_idW, fwriters.first_name as fn,
fwriters.last_name as ln, fwriters.film_id as fi from fwriters join
fproducers on fwriters.people_id = fproducers.people_id ) join
(select factors.people_id as people_idF, factors.first_name,
factors.last_name, factors.film_id  from factors join fdirectors on
factors.people_id = fdirectors.people_id)
on people_idF = people_idW natural join film
union
select people_idT, fn, ln, ti, title , start_year as year from (
select tvwriters.people_id as people_idW, tvwriters.first_name as fn,
tvwriters.last_name as ln, tvwriters.tv_id as ti from tvwriters join
tvproducers on tvwriters.people_id = tvproducers.people_id) join
(select tvactors.people_id as people_idT, tvactors.first_name,
tvactors.last_name, tvactors.tv_id  from tvactors join tvdirectors on
tvactors.people_id = tvdirectors.people_id)
on people_idT = people_idW natural join TvSeries
) order by year desc, title;
```

Result:

```
Taika|Waititi|Thor: Ragnarok|2017
Taika|Waititi|What We Do in the Shadows|2014
George|Clooney|The Ides of March|2011
Amy|Poehler|Parks and Recreation|2009
```

Example query about TV series:

```
-- For Tv Series show how many season and average number of episodes per
season  order by title
select title, count(*), cast(avg(num_of_episodes) as int) from Season
natural join TVseries group by tv_id order by title;
```

Result:

```
Brooklyn Nine-Nine|6|21
Community|6|18
How I Met Your Mother|9|23
```

Sample data:

We are only including sample insert statements for our entities tables and not for relationship tables as we have inserted quite a lot of sample data to best show of our design.

```
INSERT INTO Film
VALUES
(1, 'The Ides of March', 2011, 7.1, 101),
(2, 'Thor: Ragnarok', 2017, 7.9, 130),
(3, 'What We Do in the Shadows', 2014, 7.7, 86),
(4, 'Avengers: Endgame', 2019, 7.9, 181),
(5, 'Avengers: Infinity War', 2018, 8.5, 149),
(6, 'Captain America: Civil War', 2016, 7.8, 147),
(7, 'Spider-Man: Homecoming', 2017, 7.5, 133),
(8, 'The Conjuring', 2013, 7.5, 112),
(9, 'The Conjuring 2', 2016, 7.4, 134),
(10, 'Wine Country', 2019, 5.5, 103)
;
INSERT INTO TvSeries
VALUES
(1, 'Community', 22, 2009, 2015, 8.5),
(2, 'How I Met Your Mother', 22 , 2005, 2014, 8.3),
(3, 'Saturday Night Live', 90, 1975, 'Present', 8.1),
(4, 'Brooklyn Nine-Nine', 22, 2013, 'Present', 8.4),
(5, 'Everybody Hates Chris', 22, 2005, 2009, 7.4),
(6, 'Breaking Bad', 49, 2008, 2013, 9.5),
(7, 'Malcolm in the Middle', 22, 2000, 2006, 8.0),
(8, 'Glee', 44, 2009, 2015, 6.8),
(9, 'The Flash', 43, 2014, 'Present', 7.9),
(10, 'The Arrow', 42, 2012, 'Present', 7.7),
(11, 'Parks and Recreation', 22, 2009, 2015, 8.6)
;
```

```
INSERT INTO Season
VALUES
(1, 1, 25),
(1, 2, 24),
(1, 3, 22),
(1, 4, 13),
(1, 5, 13),
(1, 6, 13),
(4, 1, 22),
(4, 2, 23),
(4, 3, 23),
(4, 4, 22),
(4, 5, 22),
(4, 6, 18),
(2, 1, 22),
(2, 2, 22),
(2, 3, 20),
(2, 4, 24),
(2, 5, 24),
(2, 6, 24),
(2, 7, 24),
(2, 8, 24),
(2, 9, 24)
;
INSERT INTO Episode
VALUES
(1, 1, 1, 'Pilot', '2009-09-17'),
(1, 1, 2, 'Spanish 101', '2009-09-24'),
(1, 1, 3, 'Intorduction to Film', '2009-10-01'),
(1, 1, 4, 'Social Psychology', '2009-10-08'),
(1, 1, 5, 'Advanced Criminal Law', '2009-10-15'),
(1, 1, 6, 'Football, Feminism and You', '2009-10-22'),
(4, 1, 1, 'Pilot', '2013-09-17' ),
(4, 1, 2, 'The Tagger', '2013-09-18'),
(4, 1, 3, 'The Slump', '2013-10-01'),
(4, 1, 4, 'M.E. Time', '2013-10-08'),
(4, 1, 5, 'The Vulture', '2013-10-15'),
(4, 1, 6, 'Halloween', '2013-10-22'),
(4, 1, 7, '48 Hours', '2013-11-05')
;
INSERT INTO People
VALUES
(1, 'George', 'Clooney', '1961-05-06', 'USA'),
(2, 'Taika', 'Waititi', '1975-08-16', 'New Zealand'),
(3, 'Anthony', 'Russo', '1970-02-03', 'USA'),
(4, 'Joe', 'Russo', '1971-07-08', 'USA'),
(5, 'Robert', 'Downey Jr', '1965-04-04', 'USA'),
(6, 'Chris', 'Evans', '1981-06-13', 'USA'),
(7, 'Chris', 'Hemsworth', '1983-08-11', 'Australia'),
(8, 'Mark', 'Ruffalo', '1967-11-22', 'USA'),
(9, 'Scarlett', 'Johansen', '1984-11-22', 'USA'),
(10, 'Andy', 'Samberg', '1978-08-18', 'USA'),
(11, 'Chris', 'Rock', '1965-02-07', 'USA'),
(12, 'Bryan','Cranston', '1956-03-07', 'USA'),
(13, 'Frankie', 'Muniz', '1985-12-05', 'USA'),
(14, 'Melissa', 'Benoist', '1988-10-04', 'USA'),
(15, 'Grant', 'Gustin', '1990-01-14', 'USA'),
(16, 'James', 'Wan', '1977-02-26', 'Malaysia'),
(17, 'Vin', 'Diesel', '1967-07-18', 'USA'),
(18, 'Chris', 'Pratt', '1979-06-21', 'USA'),
(19, 'Amy', 'Poehler', '1971-09-16', 'USA')
;
```

Conclusion

Through the process that we just went through allowed us to create our version of the IMDb database. We started by observing the way the actual IMDb site was set up, along with the type of information that was being displayed. With that information gathered, it inspired some of the entities and relationships that we created. Although this simplified some of the process, we had to really analyze the way these entities would interact. In other words, how would this database be used for? With this in mind, we were able to design some problem statements that would help with design of the database. These statements reflect the necessities of the company.

Those statements were also used to design queries to collect that data that the company requests. Using the requests of the company and the knowledge of the way TV shows and Films are constructed, we were able to design our schemas properly to meet the proper form, Boyce Codd Normal Form. This form usually becomes possible thru various changes of the schemas. Schemas were checked to make sure that each attribute is functionally dependent on the primary key. This process is called normalization, but we did not have to much simplifying since a TV show can only have one episode belonging to a specific season. The creation of this database allowed us to practice building a database that include real-world situations and allow us to see how it would be applied in the real world.