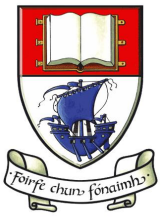


Waterford Institute of Technology

Database Design Term Project

Movie Sequel DB

Ciaran Roche(20037160)
Mathana Nair Sreedaran (20069456)

April 24, 2017

Contents

Abstract

The purpose of this paper is to design and develop a database using MySQL Workbench. The following is practical work as part of the Database Design module in BSc.(Hons) Applied Computing, at Waterford Institute of Technology. A unique project topic was selected. A proposal was submitted, a demonstration of the project was performed and project design documents and database scripts were presented.

1 Introduction

The chosen area or topic for this project is movies. The aim of the project was to create a movie database containing data essential to all users ranging from the average user looking for the description of a movie to somebody from the entertainment industry who is looking for additional information related to a movie such as the band or artiste of a song from the movie. The information included in the database covers a wide range of elements that can be of use to anybody seeking information related to a movie.

2 Project Description

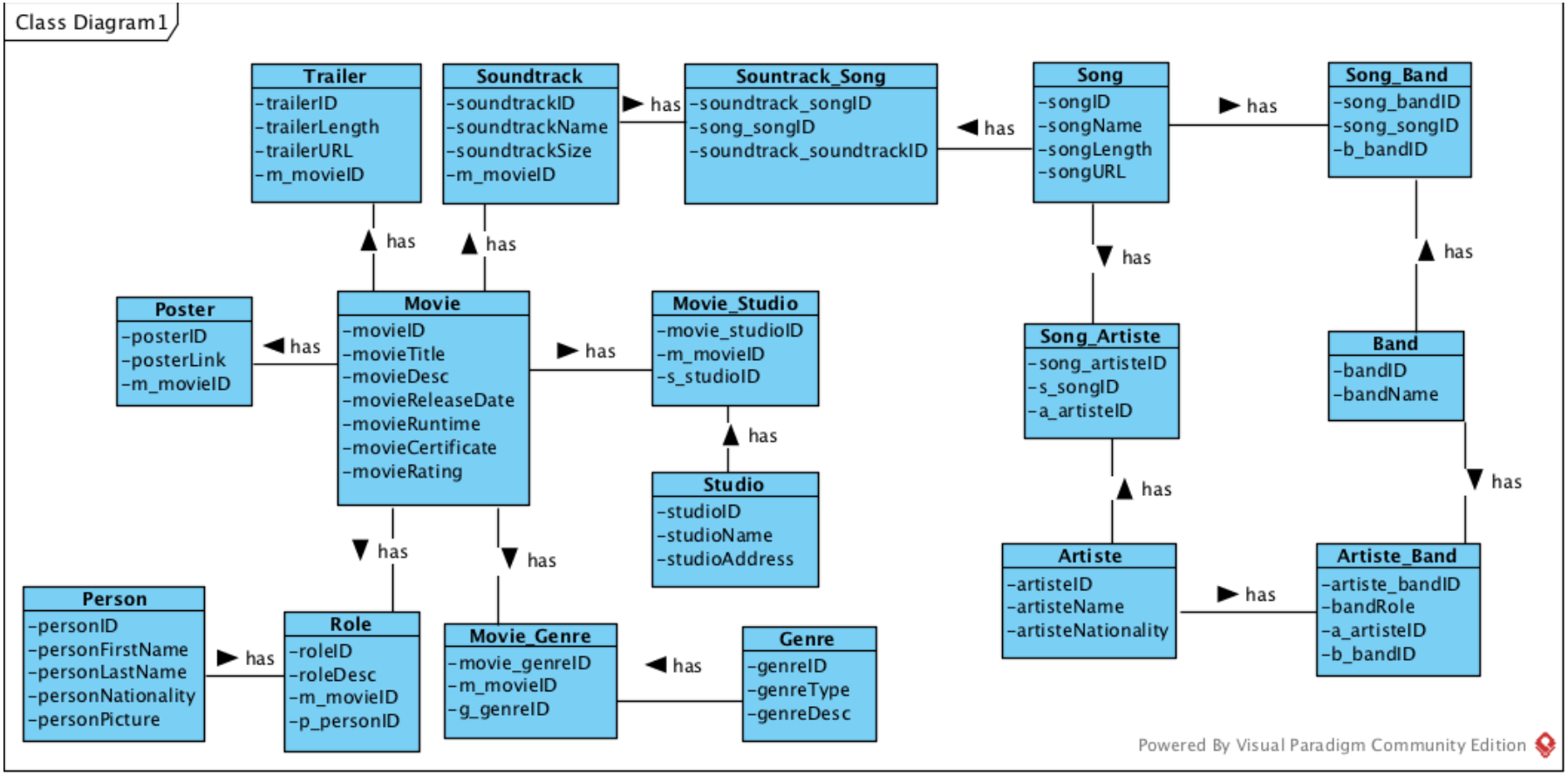
Movies are a form of entertainment to many people. Making a movie requires the collaborative effort of people with varied expertise. For example, producers are needed to plan and coordinate various aspects of a movie and the actors and actresses give life to the movie under direction of the director.

Simply put, a movie and the movie industry contain much data which should be made available to anybody who seeks it. This is facilitated with the use of a database. By centralising data in an orderly manner, users may access it efficiently.

The major functions of this database include the retrieval of information relevant to a movie such as the movie description, rating and runtime and other information which are attributed to a movie such as the soundtrack, songs of the soundtrack and artiste or band who played those songs. The people behind the movie are also included along with their respective roles such as actor, director and writer. All the tables in the database can be found in the Table Design section of this report.

3 Model

The model was designed using Visual Paradigm and underwent multiple iterations. The model is shown on the following page;


Class Diagram1
Trailer
-trailerID
-trailerLength
-trailerURL
-m_movieID
Soundtrack
-soundtrackID
-soundtrackName
-soundtrackSize
-m_movieID
Sountrack_Song
-soundtrack_songID
-song_songID
-soundtrack_soundtrackID
Song
-songID
-songName
-songLength
-songURL
Song_Band
-song_bandID
-song_songID
-b_bandID
Poster
-posterID
-posterLink
-m_movieID
Movie
-movieID
-movieTitle
-movieDesc
-movieReleaseDate
-movieRuntime
-movieCertificate
-movieRating
Movie_Studio
-movie_studioID
-m_movieID
-s_studioID
Studio
-studioID
-studioName
-studioAddress
Song_Artiste
-song_artisteID
-s_songID
-a_artisteID
Band
-bandID
-bandName
Person
-personID
-personFirstName
-personLastName
-personNationality
-personPicture
Role
-roleID
-roleDesc
-m_movieID
-p_personID
Movie_Genre
-movie_genreID
-m_movieID
-g_genreID
Genre
-genreID
-genreType
-genreDesc
Artiste
-artisteID
-artisteName
-artisteNationality
Artiste_Band
-artiste_bandID
-bandRole
-a_artisteID
-b_bandID
has
Powered By Visual Paradigm Community Edition

4 Table Design

The following are tables which resulted from the preceding model with details such as primary keys, foreign keys and integrity constraints.

movie				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
movieID	int	5	PK	not null
movieTitle	varchar	50		not null
movieDesc	varchar	150		
movieReleaseDate	date			
movieRuntime	int	3		movieRuntime >=25
movieCertificate	varchar	4		Must be N/A or G or PG or 12 or 12A or 15 or 15A or 16 or 18
movieRating	int	1		movieRating >0 && movieRating <=5
poster				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
posterID	int	5	PK	not null
posterLink	varchar	200		default 'http://www.uidownload.com/files/478/82/442/error-404-page-not-found-icon.jpg'
p_movieID	int	5	FK	not null
trailer				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
trailerID	int	5	PK	not null
trailerLength	int	2		
trailerURL	varchar	150		
t_movieID	int	5	FK	not null
movie_studio				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
movie_studioID	int	5	PK	not null
m_movieID	int	5	FK	not null
s_studioID	int	5	FK	not null
studio				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
studioID	int	5	PK	not null
studioName	varchar	50		not null
studioAddress	varchar	200		
movie_genre				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
movie_genreID	int	5	PK	not null
m_movieID	int	5	FK	not null
g_genreID	int	5	FK	not null
genre				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
genreID	int	5	PK	not null
genreType	varchar	25		not null
genreDesc	varchar	200		
role				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
roleID	int	5	PK	not null
roleDesc	varchar	25		not null
m_movieID	int	5	FK	not null
p_personID	int	5	FK	not null

person				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
personID	int	5	PK	not null
personFirstName	varchar	50		not null
personLastName	varchar	50		not null
personNationality	varchar	50		
personPicture	varchar	150		
soundtrack				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
soundtrackID	int	5	PK	not null
soundtrackName	varchar	100		not null
soundtrackSize	int	2		
m_movieID	int	5	FK	not null
soundtrack_song				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
soundtrack_songID	int	5	PK	not null
song_songID	int	5	FK	not null
soundtrack_soundtrackID	int	5	FK	not null
song				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
songID	int	5	PK	not null
songName	varchar	25		not null
songLength	int	3		
songURL	varchar	150		
song_artiste				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
song_artisteID	int	5	PK	not null
s_songID	int	5	FK	not null
a_artisteID	int	5	FK	not null
artiste				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
artisteID	int	5	PK	not null
artisteName	varchar	50		not null
artisteNationality	varchar	50		
song_band				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
song_bandID	int	5	PK	not null
song_songID	int	5	FK	not null
b_bandID	int	5	FK	not null
band				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
bandID	int	5	PK	not null
bandName	varchar	25		not null
artiste_band				
ColumnName	**DataType**	**Size**	**PK/FK**	**Constraint**
artiste_bandID	int	5	PK	not null
bandRole	varchar	50		
a_artisteID	int	5	FK	not null
b_bandID	int	5	FK	not null

5 SQL Scripts

5.1 Create Database Script

The complete copy of all scripts can be found at the following link:
https://github.com/ciaranRoche/mySQL-movie-db

Below is an example of a create table script. The complete script can be found at this link.

```
drop database sequelmovie;
create database sequelmovie;
use sequelmovie;

create table movie(
    movieID int(5) not null,
    movieTitle varchar(50) not null,
    movieDesc varchar(150),
    movieReleaseDate date,
    movieRuntime int(3) check (movieRuntime > 25),
    movieCertificate varchar(4) check (movieCertificate in
    ('N/A','PG','12','12A','15','15A','16','18')),
    movieRating int(1) check (movieRating > 0 and movieRating <= 5),
    constraint movie_pk primary key (movieID)
)engine innodb;
```

5.2 Triggers Script

Below are the scripts for triggers;

The trigger below ensures the movie runtime is above 25.

```
delimiter $$
create trigger before_movie_insert_movieRuntime before insert on movie
for each row
begin
  if new.movieRuntime <= 25 then
  signal sqlstate '42000'
  set message_text = 'Check constraint on movieRuntime in table movie
    failed. Runtime too short';
  end if;
end$$
delimiter ;
```

The trigger below ensures the movie certificate adheres to those defined by the Irish Film Classification Office.

```
delimiter $$
create trigger before_movie_insert_movieCertificate before insert on
    movie
for each row
begin
  if new.movieCertificate not in ('N/A','PG','12','12A','15','15A','16'
    ,'18') then
     signal sqlstate '41000'
     set message_text = 'Check constraint on movieCertificate in table
    movie failed. Only Irish ratings';
     end if;
end$$
delimiter ;
```

The following trigger ensures movie ratings are between 1 to 5 only.

```
delimiter $$
create trigger before_movie_insert_movieRating before insert on movie
for each row
begin
  if (new.movieRating < 1) OR (new.movieRating > 5) then
  signal sqlstate '42000'
  set message_text = 'Check constraint on movieRating in table movie
    failed. Outrageous rating my good sir/madame';
  end if;
end$$
delimiter ;
```

5.3 Insert Script

Below is an example of a movie insert script. The complete scripts can be found at this link.

```
use sequelmovie;

insert into movie values (0001, 'The Shawshank Redemption',
'Two imprisoned men bond over a number of years',
'1994-10-14', 142, '18', '4.5');
```

5.4 Queries Script

Below is an example of a few query scripts. The complete scripts can be found at this link.

```
use sequelmovie;

describe movie;

select movieTitle as Title, movieDesc
as Description, movieReleaseDate as 'Release Date', movieRuntime as
    Runtime
from movie;

update person
  set personPicture = ifnull(personPicture, 'http://www.uidownload.com/
    files/478/82/442/error-404-page-not-found-icon.jpg');

delete from movie
where movieTitle = 'Alien';

alter table movie
change movieDesc moviePlot varchar(150);
```

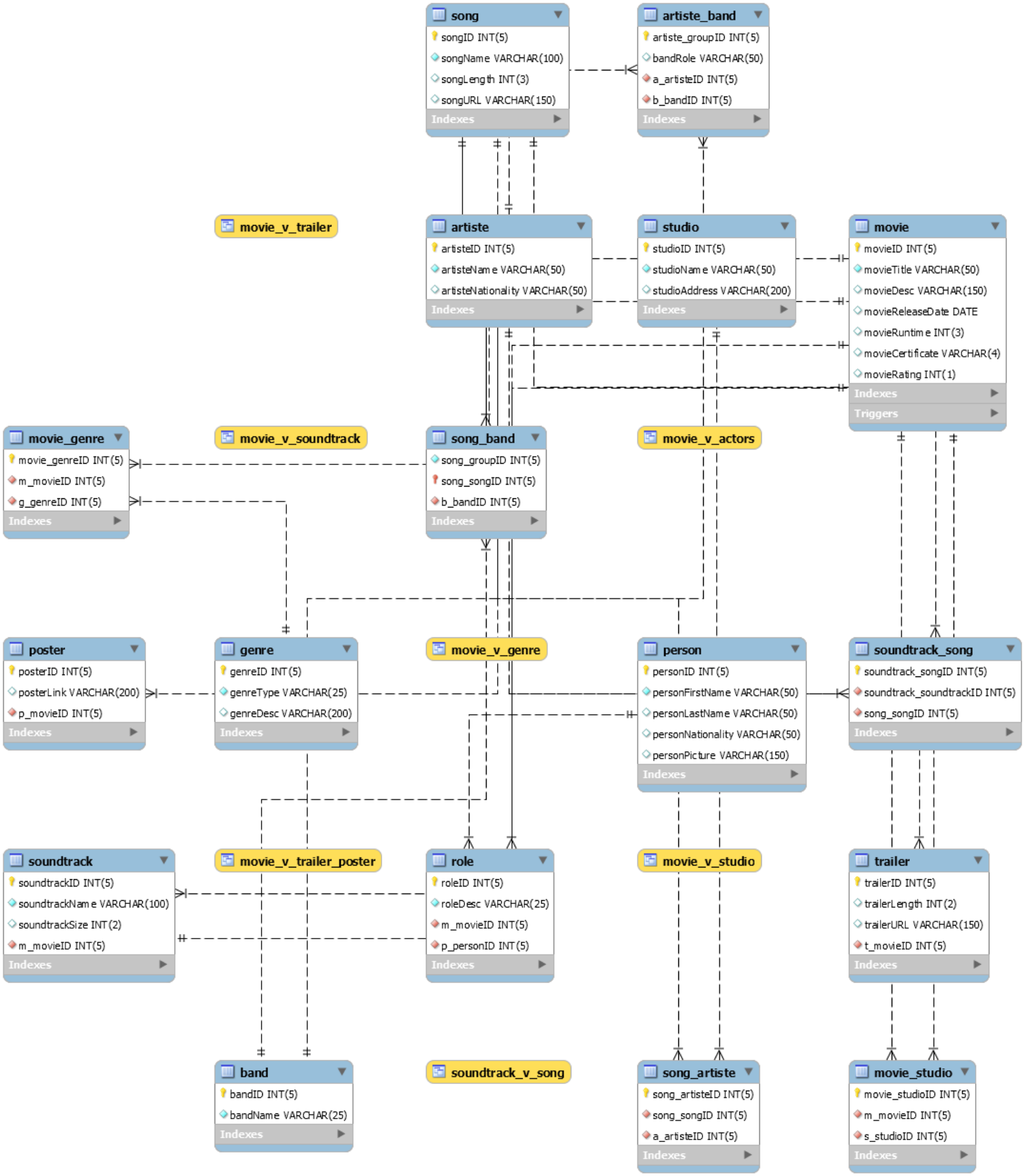
5.5 Views Script

Views were created with different users in mind. The name of the views indicate which tables it uses. Below are the scripts to create views;

```
use sequelmovie;

create view movie_v_trailer as
  select movieTitle, trailerURL
  from movie, trailer
  where movieID = t_movieID;

create view movie_v_trailer_poster as
  select movieTitle, trailerURL, posterLink
  from movie, trailer, poster
  where movieID = t_movieID and movieID = p_movieID;

create view movie_v_soundtrack as
  select movieTitle, soundtrackName
  from movie, soundtrack
  where movieID = m_movieID;

create view movie_v_genre as
  select movieTitle, genreType
  from movie, genre, movie_genre
  where movieID = m_movieID and g_genreID = genreID;

create view soundtrack_v_song as
  select soundtrackName, songName
  from soundtrack, song, soundtrack_song
  where soundtrackID = soundtrack_soundtrackID and song_songID=songID;

create view movie_v_actors as
  select movieTitle, personFirstName, personLastName, roleDesc
  from movie, person, role
  where movieID = m_movieID and p_personID = personID;

create view movie_v_studio as
  select movieTitle, studioName
  from movie, studio, movie_studio
  where movieID = m_movieID and s_studioID = studioID;
```

6 ER Diagram

The ER diagram generated by reverse engineering the database and is shown in the following page;


song
songID INT(5)
songName VARCHAR(100)
songLength INT(3)
songURL VARCHAR(150)
Indexes
artiste_band
artiste_groupID INT(5)
bandRole VARCHAR(50)
a_artisteID INT(5)
b_bandID INT(5)
Indexes
movie_v_trailer
artiste
artisteID INT(5)
artisteName VARCHAR(50)
artisteNationality VARCHAR(50)
Indexes
studio
studioID INT(5)
studioName VARCHAR(50)
studioAddress VARCHAR(200)
Indexes
movie
movieID INT(5)
movieTitle VARCHAR(50)
movieDesc VARCHAR(150)
movieReleaseDate DATE
movieRuntime INT(3)
movieCertificate VARCHAR(4)
movieRating INT(1)
Indexes
Triggers
movie_genre
movie_genreID INT(5)
m_movieID INT(5)
g_genreID INT(5)
Indexes
movie_v_soundtrack
song_band
song_groupID INT(5)
song_songID INT(5)
b_bandID INT(5)
Indexes
movie_v_actors
poster
posterID INT(5)
posterLink VARCHAR(200)
p_movieID INT(5)
Indexes
genre
genreID INT(5)
genreType VARCHAR(25)
genreDesc VARCHAR(200)
Indexes
movie_v_genre
person
personID INT(5)
personFirstName VARCHAR(50)
personLastName VARCHAR(50)
personNationality VARCHAR(50)
personPicture VARCHAR(150)
Indexes
soundtrack_song
soundtrack_songID INT(5)
soundtrack_soundtrackID INT(5)
song_songID INT(5)
Indexes
soundtrack
soundtrackID INT(5)
soundtrackName VARCHAR(100)
soundtrackSize INT(2)
m_movieID INT(5)
Indexes
movie_v_trailer_poster
role
roleID INT(5)
roleDesc VARCHAR(25)
m_movieID INT(5)
p_personID INT(5)
Indexes
movie_v_studio
trailer
trailerID INT(5)
trailerLength INT(2)
trailerURL VARCHAR(150)
t_movieID INT(5)
Indexes
band
bandID INT(5)
bandName VARCHAR(25)
Indexes
soundtrack_v_song
song_artiste
song_artisteID INT(5)
song_songID INT(5)
a_artisteID INT(5)
Indexes
movie_studio
movie_studioID INT(5)
m_movieID INT(5)
s_studioID INT(5)
Indexes

7 Conclusion

In conclusion, the aim of the project to build a movie database was achieved. Much research was done to obtain movie data to populate the database. An open API (Application Program Interface) from The Open Movie Database, which returned data from IMDb, was used. Information not supplied by the API was sourced manually. A Python script was written to automate the population of the database. However, due to time constraints, the script was not used. It can be found at this link.

All scripts were tested thoroughly to ensure they produced the intended output. Overall, a greater understanding of SQL with the use of MySQL workbench was obtained. In addition, the process of researching and building a database was learnt.