Project

NUTFLUX

THE HOME OF KNOWLEDGABLE MOVIE NUT

Pawan Prashanth Kamath
21201009
(pawan.kamath@ucdconnect.ie)

Table of Contents

List of Figures

1. Introduction

NUTFLUX is a platform where the users get all the details from movie details to cast and crews involved in the making of a movie. This should be user friendly where users can make use of this with ease and get more information out of it.

This is basically a movie database where every bit about the movie is stored for later use by any user and based on user operations the platform should understand and learn the user likes and dislikes. This is also provided with different show types like **TV shows, movies, and series.** The dashboard will also contain the **upcoming** movies that the user is awaiting based on his/her previous liked movies.

There are 3 main sections that are concentrated more on:

1. Show details
2. Award details
3. User acceptance and critics review

So, for the **Standard users** it will have details about all types of shows that are released and upcoming. This will be purely based on users likes of previously watched movies. The dashboard view will have the entire cast and crew information along with movie details from duration to gross collection till date, It's popularity, critics reviews, user reviews etc.

They will also have access to the casts award details with movie for which they were awarded. This can further made to show more details on actors/directors previous hits/flops/disaster movies if needed(designed keeping all these in mind).

For the **pro users,** they can get more complex details like pairs of actors audience likes, director comfortable actors etc. This will further help the power users to get more insights on the popular movies and casts. In addition to this power users will be given access to critics details, They can be given an opportunity to join the crew that promotes/roasts the celebrity. Along with this the power users can get the access to upcoming films details way ahead of standard users, so they will be more likely to know the pre-movie release promotions agendas. This is the vision of this

platform.

The database is designed in such a way that it is more flexible if on any future improvements on this design too be made can be done with minimal changes.

2. Database Plan: A Schematic View

A principal entity is the entity that will act as a parent in a relationship. This entity holds the main key in a database, which will be utilized by dependent entities to form relationships.

This design has two kinds of tables:

1. Base table

Base table itself is a stand alone table independent of any other table. This is a data table having all the details of an entity

2. Derived table

Derived table is a dependent table which is dependent on the base table. This table is mainly created to handle multiple values for a single entity. For example: 2 directors directing a movie etc.

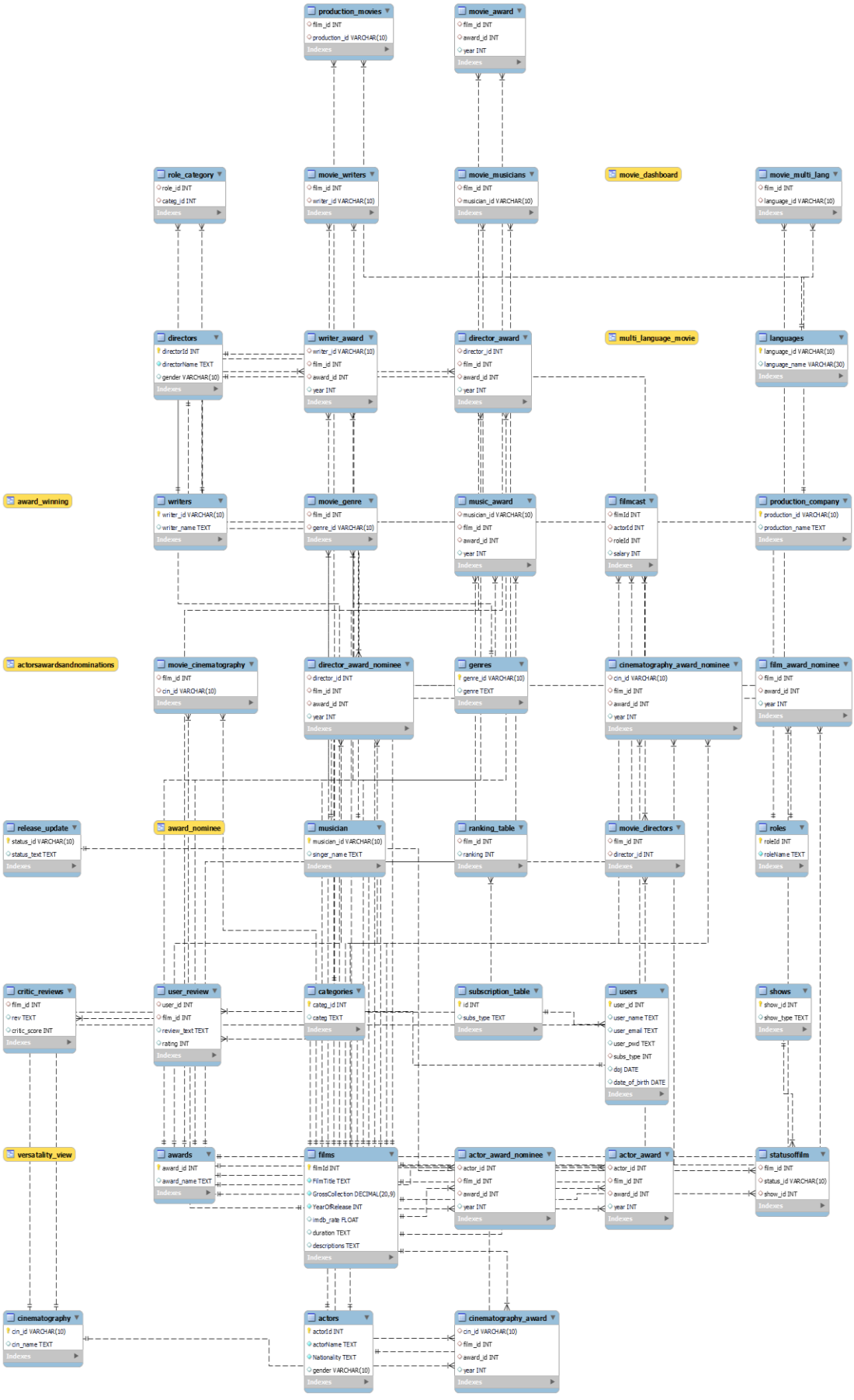
There are total 39 tables out of which:

16 are base tables and 23 are derived tables.

High level database design:

ER diagram is used to represent the high level design of a schema. This is a Entity-Relationship model which describes all the entities in the schema and how they are associate with each other.

The below figure depicts the ER diagram for the database design of NUTFLUX:


production_movies
film_id INT
production_id VARCHAR(10)
movie_award
film_id INT
award_id INT
year INT
role_category
role_id INT
categ_id INT
movie_writers
film_id INT
writer_id VARCHAR(10)
movie_musicians
film_id INT
musician_id VARCHAR(10)
movie_dashboard
movie_multi_lang
film_id INT
language_id VARCHAR(10)
directors
directorId INT
directorName TEXT
gender VARCHAR(10)
writer_award
writer_id VARCHAR(10)
film_id INT
award_id INT
year INT
director_award
director_id INT
film_id INT
award_id INT
year INT
multi_language_movie
languages
language_id VARCHAR(10)
language_name VARCHAR(30)
award_winning
writers
writer_id VARCHAR(10)
writer_name TEXT
movie_genre
film_id INT
genre_id VARCHAR(10)
music_award
musician_id VARCHAR(10)
film_id INT
award_id INT
year INT
filmcast
filmId INT
actorId INT
roleId INT
salary INT
production_company
production_id VARCHAR(10)
production_name TEXT
actorsawardsandnominations
movie_cinematography
film_id INT
cin_id VARCHAR(10)
director_award_nominee
director_id INT
film_id INT
award_id INT
year INT
genres
genre_id VARCHAR(10)
genre TEXT
cinematography_award_nominee
cin_id VARCHAR(10)
film_id INT
award_id INT
year INT
film_award_nominee
film_id INT
award_id INT
year INT
release_update
status_id VARCHAR(10)
status_text TEXT
award_nominee
musician
musician_id VARCHAR(10)
singer_name TEXT
ranking_table
film_id INT
ranking INT
movie_directors
film_id INT
director_id INT
roles
roleId INT
roleName TEXT
critic_reviews
film_id INT
rev TEXT
critic_score INT
user_review
user_id INT
film_id INT
review_text TEXT
rating INT
categories
categ_id INT
categ TEXT
subscription_table
id INT
subs_type TEXT
users
user_id INT
user_name TEXT
user_email TEXT
user_pwd TEXT
subs_type INT
doj DATE
date_of_birth DATE
shows
show_id INT
show_type TEXT
versatality_view
awards
award_id INT
award_name TEXT
films
filmId INT
FilmTitle TEXT
GrossCollection DECIMAL(20,9)
YearOfRelease INT
imdb_rate FLOAT
duration TEXT
descriptions TEXT
actor_award_nominee
actor_id INT
film_id INT
award_id INT
year INT
actor_award
actor_id INT
film_id INT
award_id INT
year INT
statusoffilm
film_id INT
status_id VARCHAR(10)
show_id INT
cinematography
cin_id VARCHAR(10)
cin_name TEXT
actors
actorId INT
actorName TEXT
Nationality TEXT
gender VARCHAR(10)
cinematography_award
cin_id VARCHAR(10)
film_id INT
award_id INT
year INT

1. **FILMS:**

   This table contains the details about films which includes film title, year of release, gross collection in dollars, description about the film, duration of film, and IMDB rating. The check constraint is written to validate IMDB rating to be a valid integer greater than zero. **filmId** is the **PRIMARY KEY.** Films has **check constraint** to validate IMDB rate. The movie **with same name** is handled by the descriptions given to the film along with the year of release and duration.

2. **USERS:**

   This table provides all the user information in the NUTFLUX database, including both standard and Pro users. Every user has a distinct user id, username, password, date of joining, date of birth, subscription type, and email address. We're also putting a check constraint on the password to make sure it's strong. **user_id** is the **PRIMARY KEY . This has a check constraint to validate password.**

3. **DIRECTORS:**

   This table contains the information about the director like director name and gender. Each director has a unique identification number. **directorId** is the PRIMARY KEY .

MOVIE_DIRECTOR references filmId from **FILMS** and directorId from **DIRECTCTORS.**

4. **LANGUAGES:**

   The language table contains the list of languages in which the movies are released. This is intended to help the user to check if the movie is released in the languages, he is aware of. This table consists of unique ID for each language and the name of the language. **Language_id** is the PRIMARY KEY.

   MOVIE_MULTI_LANG references language_id from **LANGUAGES** and filmId from **FILMS.**

5. **ACTORS:**

This table contains actor details like actor name, nationality, and gender. Each actor has a unique identification number. **actorId** is the PRIMARY KEY.

FILMCAST references actor_id from **ACTORS** and filmId from **FILMS.**

6. **ROLES:**

This table contains various roles played by actors in a movie. It consists of role ID and role name played by a actor. **role_id** is the PRIMARY KEY. **role_category** references role_id from **roles.**

7. **CATEGORIES:**

This table contains the various categories associated with a role like hero, heroine, villain, superhero etc. **categ_id** is the PRIMARY KEY. **role_category** references categ_id from **categories.**

8. **WRITERS:**

This table has details about the writers who write stories for a movie. This table has a unique ID and the name of the writer. **Writer_id** is the PRIMARY KEY. **MOVIE_WRITERS** references filmId from **films** and writer_id from **writers**.

9. **SUBSCRIPTION_TABLE:**

This table has the data of options of subscription given to the user. The data stored are **standard** and **pro** subscriptions. **Id** is the PRIMARY KEY. **User** references subs_type from **subscription_table.**

10. **CINEMATOGRAPHY:**

This table consists of details of cinematographers. Each cinematographers have a unique ID and their name is stored in the table. **Cin_id** is the PRIMARY KEY. **MOVIE_CINEMATOGRAPHY** references cin_id from **CINEMATOGRAPHY.**

11. **AWARDS:**

This table has a list of awards across different categories and levels. By levels it means to include industry awards, national awards and international awards. **Award_id** is the PRIMARY KEY. All **award** and **nominations** tables reference **awards** and **films.**

12. **RELEASE_UPDATE:**

This table is associated with the release update of the movie or TV show. It has either **released** or **upcoming** as status.

13. **SHOWS:**

This table has information about if the video is a **movie** or **TV show** or **Series.**

14. **PRODUCTION_COMPANY:**

This table gives the list of production companies that produce films. **Production_id** is the PRIMARY KEY. **PRODUCTION_MOVIES** references production_id from **PRODUCTION_COMPANY.**

15. **MUSICIAN:**

This table gives the list of musicians who have produced albums or songs in a movie. **Musician_id** is the PRIMARY KEY. **MOVIE_MUSICIANS** references musician_id from **MUSICIAN.**

16. **GENRES:**

This table has the list of genres the movies are usually based on like comedy, romance, crime, thriller and so on. **Genre_id** is the PRIMARY KEY. **MOVIE_GENRE** references genre_id from **GENRE.**

MOTIVATION:

- For every data table, there is an **FILMS -associated** table, the reason being to consider the situation like one movie having multiple directors, writers, producers etc.
- Similarly for awards tables, such as **winning** and **nominations** for each cast.
- It has a separate **language** table, to support movies being released in multiple languages.
- There is a **shows** table to inform user if it is a movie or a tv show and **release_update** table to inform users about the release dates.
- Tables like **category** is made to maintain Normalizations.
- **Check constraints** are written wherever vaidations are necessary.

3. Database Structure: A Normalized View

In my nutflux database there are total 39 tables which consists of **base tables**, tables that stores main entities and **derived tables** which uses the references from the base table. For instance, base table stores the details of directors where as movie_directors, a movie can have multiple directors. This table will reference directors table for a single film.

The tables and it's details are described below:

DIRECTORS:

This table contains details about the director, such as his or her name and gender. Each director is assigned a unique number .

```
create table if not exists directors(
  directorId integer  PRIMARY KEY,
  directorName text NOT NULL,
  gender varchar(10)
);
```

Sample output:

directorId	directorName	gender
1	Frank Darabont	male
2	Prashanth Neel	male
3	Sydney Newman	male
4	Joss Whedon	male
5	W.S. Van Dyke	male

Here in this table, We have a primary key and every attribute is atomic. So, this obeys **1NF.**It has no non-prime characteristic in a relation that is functionally dependent on any candidate key's proper subset. So, it's in **2NF**. For non-prime characteristics, there is no transitive dependency so, it's in **3NF.** For a range of functional dependency for this table. The left hand side is always a key. Since only directorId identifies the unique dependency. Also, any non prime attribute combined with a key will be unique. So, this is in **BCNF.**

LANGUAGES:

The language table lists the languages in which the films are available. This is meant to assist the user in determining whether the film is available in the

languages that he is familiar with. This table contains a unique ID for each language as well as the language's name.

```
create table if not exists languages(language_id varchar(10) primary key,
                                     language_name varchar(30));
```

Sample output:

language_id	language_name
lang1	English
lang2	Irish
lang3	Japaneese
lang4	Korean
lang5	Kannada
lang6	Hindi
lang7	Spanish

This tale has one to one relationship since there is one prime attribute and one non prime attribute. This follows all the Normal forms.

FILMS:

The film title, year of release, gross collection in dollars, description of the film, duration of the film, and IMDB rating are all listed in this table.

```
CREATE TABLE if not exists films(
  filmId integer PRIMARY KEY,
  FilmTitle text NOT NULL,
  GrossCollection decimal(20,9) NOT NULL,
  YearOfRelease integer NOT NULL,
  imdb_rate float,
  check(imdb_rate>=0),
  duration text,
  descriptions text
);
```

Sample output:

filmId	FilmTitle	GrossCollection	YearOfRelease	imdb_rate	duration	descriptions
11	The Shawshank Redemption	28884504.000000000	1992	9.3	2h 22m	Two imprisoned men bond over a number of ye...
12	KGF 2	7000000000.000000000	2022	9.6	2h 48m	In the blood-soaked Kolar Gold Fields, Rocky's n...
13	KGF 1	288754198.000000000	2018	8.5	2h 36m	In the 1970s, a gangster goes undercover as a ...
14	The avengers	150056732.000000000	1969	8.3	1961-1969	A quirky spy show of the adventures of eccentr...
15	The avengers	623357910.000000000	2012	8	2h 23m	Earth's mightiest heroes must come together an...
16	Avengers: Age of Ultron	642317670.000000000	2015	7.3	2h 21m	When Tony Stark and Bruce Banner try to jump...
17	Manhattan Melodrama	165234852.000000000	1934	7.1	1h 33m	The friendship between two orphans endures e...
18	The thin Man	165234852.000000000	1934	7.9	1h 32m	Former detective Nick Charles and his wealthy ...

The check constraint is written to validate IMDB rating to be a valid integer greater than zero.

We have a primary key in this table, and each attribute is atomic. As a result, this complies with **1NF**. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in **2NF**. There is no transitive dependency for non-prime qualities, hence it is in **3NF**. This table has a variety of functional dependencies. The left hand side is always a key. Only filmId identifies the one-of-a-kind dependency. Additionally, any non-prime quality coupled with a key creates a unique combination. As a result, this is under **BCNF**.

MOVIE_MULTI_LANG:

This table was established to ensure that if a film is released in numerous languages, each of them should be treated as the same film in a separate language. As a result, if the user has any language preferences, the user will be presented with the movie in their favorite language from among those available

```
create table movie_multi_lang( film_id int,language_id varchar(10),
                               foreign key(language_id) references languages(language_id),
                               foreign key(film_id) references films(filmid));
```

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in 3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

MOVIE_DIRECTORS:

This table offers information on directors and the films they have directed. This table also addresses the situation where a film has numerous directors. This is a derived table made up of films and directors from the base table.

```
create table movie_directors(film_id int,
                             director_id int,
                             foreign key(film_id) references films(filmId),
                             foreign key(director_id) references directors(directorId));
```

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in 3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

ACTORS:

This table contains information about the actors, such as their name, nationality, and gender. Each actor is assigned a number that is unique to them .

```
CREATE TABLE if not exists actors(
  actorId integer PRIMARY KEY,
  actorName text NOT NULL,
  Nationality text NOT NULL,
  gender varchar(10)
);
```

Sample output:

actorId	actorName	Nationality	gender
21	Tim Robbins	USA	male
22	Freeman	USA	male
23	Bob Guntan	USA	male
24	Yash	India	male
25	Sanjay Dutt	India	male
26	Srinidhi Shetty	India	female
27	Ramachandra Raju	India	male
28	Patrick Macnee	UK	male

We have a primary key in this table, and each attribute is atomic. As a result, this complies with **1NF**. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in **2NF**. There is no transitive dependency for non-prime qualities, hence it is in

3NF. This table has a variety of functional dependencies. The left hand side is always a key. Only actorId identifies the one-of-a-kind dependency. Additionally, any non-prime quality coupled with a key creates a unique combination. As a result, this is under **BCNF**.

ROLES:

The roles played by actors in a film are included in this table. It comprises of an actor's role ID and role name.

```
CREATE TABLE if not exists roles (
   roleId integer PRIMARY KEY,
   roleName text NOT NULL
);
```

Sample output:

roleId	roleName
31	Andy Dufresne
32	Ellis Boyd Red Redding
33	Warden Norton
34	Rocky Bhai
35	Adheera
36	Reena
37	Garuda
38	John steed

This tale has one to one relationship since there is one prime attribute and one non prime attribute. This follows all the Normal forms.

CATEGORIES:

This table contains the various categories associated with a role like hero, heroine, villain, superhero etc.

```
create table categories(categ_id int primary key,categ text);
```

Sample Output:

categ_id	categ
1	hero
2	heroine
3	anti hero
4	villian
5	love interest
6	superhero
7	spy

This tale has one to one relationship since there is one prime attribute and one non prime attribute. This follows all the Normal forms.

ROLE_CATEGORY:

This table corresponds to the role's category. As a result, this table is derived from roles and classifications. There is a role ID and a category ID in this table.

```
create table role_category(role_id int, categ_id int,
                           foreign key(role_id) references roles(roleId),
                           foreign key(categ_id) references categories(categ_id));
```

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in 3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

WRITERS:

The writers who write the stories for a movie are listed in this table. This table has a unique ID as well as the author's name.

```
create table if not exists writers(writer_id varchar(10) primary key,writer_name text);
```

Sample Output:

writer_id	writer_name
writer1	Stephen King
writer10	Stan Lee
writer11	Jack Kirby
writer12	Oliver H.P. Garrett
writer13	Arthur Caesar
writer14	Albert Hackett
writer15	Jo Swerling
writer2	Frank Darabont

This tale has one to one relationship since there is one prime attribute and one non prime attribute. This follows all the Normal forms.

MOVIE_WRITER:

This table connects the writers to the films that depict their stories. This table deals with the circumstance where there are numerous authors on a single film.

```
create table if not exists movie_writers(film_id int ,writer_id varchar(10),
                                    foreign key(film_id) references films(filmId),
                                    foreign key(writer_id) references writers(writer_id));
```

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in 3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

FILMCAST:

This table contains the names of the actors who appeared in the film, as well as the roles they performed and their pay for the film. Before inserting, this table uses a check constraint to validate an actor's income .

```
CREATE TABLE if not exists FILMCAST (
  filmId integer,
  actorId integer,
  roleId integer,
  salary integer,
  check(salary >0),
  FOREIGN KEY(filmId)
    REFERENCES films(filmId)  ,
  FOREIGN KEY(actorId)
    REFERENCES actors(actorId)  ,
  FOREIGN KEY(roleId)
    REFERENCES roles(roleId)
);
```

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in 3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

RANKING_TABLE:

This table consists of the ranks associated with the film.

```
create table if not exists ranking_table ( film_id int, ranking int unique key,check(ranking>0),
                                           foreign key(film_id) references films(filmId));
```

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in 3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

SUBSCRIPTION_TABLE:

This table has the data of options of subscription given to the user. The data stored are **standard** and **pro** subscriptions.

```
create table if not exists subscription_table( id int primary key, subs_type text);
```

Sample output:

	id	subs_type
▶	41	standard
	42	Pro

This tale has one to one relationship since there is one prime attribute and one non prime attribute. This follows all the Normal forms.

USERS:

This table contains all of the NUTFLUX database's user information, including both standard and Pro users. Each user has a unique user id, username, password, joining date, birth date, subscription type, and email address. We've also added a check restriction to the password to ensure that it's secure.

```
create table if not exists users( user_id int primary key, user_name text, user_email text,
                        user_pwd text,check(char_length(user_pwd)<=12),
                        subs_type int,doj Date,date_of_birth Date,
                        foreign key(subs_type) references subscription_table(id));
```

Sample Output:

user_id	user_name	user_email	user_pwd	subs_type	doj	date_of_birth
51	Pawan	pawan98ppk@gmail.com	@21March1998	41	2018-03-25	1998-03-21
52	Dominic	dominic@gmail.com	@22March1997	42	2019-09-12	1996-03-22
53	Linas	linas@gmail.com	@23March1997	42	2019-09-12	1995-03-23
54	Carly	carly@gmail.com	@24March1997	42	2022-04-24	1994-03-24

We have a primary key in this table, and each attribute is atomic. As a result, this complies with **1NF**. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in **2NF**. There is no transitive dependency for non-prime qualities, hence it is in **3NF**. This table has a variety of functional dependencies. The left hand side is always a key. Only user_id identifies the one-of-a-kind dependency. Additionally, any non-prime quality coupled with a key creates a unique combination. As a result, this is under **BCNF**.

CINEMATOGRAPHY:

This table contains information about cinematographers. Each cinematographer is assigned a unique ID, and their names are recorded in the table..

```
create table cinematography(cin_id varchar(10) primary key,cin_name text);
```

Sample output:

cin_id	cin_name
cin1	Roger Deakins
cin2	Bhuvan Gowda
cin3	Sydney Newman
cin4	Joss Whedon
cin5	Seamus McGarvey
cin6	Ben Davis
cin7	James Wong Howe
cin8	William H. Daniels

This tale has one to one relationship since there is one prime attribute and one non prime attribute. This follows all the Normal forms.

MOVIE_ CINEMATOGRAPHY:

This table connects cinematographers to the films on which they worked. This table deals with the circumstance where a single film has many cinematographers..

```
create table movie_cinematography(film_id int,cin_id varchar(10),
                                  foreign key(film_id) references films(filmId),
                                  foreign key(cin_id) references cinematography(cin_id));
```

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in 3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

AWARDS:

This table includes a list of awards in various categories and levels. Industry honors, national awards, and international accolades are all included in the tiers.

```
create table if not exists awards(award_id int primary key, award_name text);
```

award_id	award_name
61	Japan Academy Prize for Outstanding Foreign Language Film
62	American Society of Cinematographers Award for Outstanding A...
63	Humanitas Prize for Best Film
64	USC Scripter Award
65	Directors Guild of America Award for Outstanding Directing
66	Writers Guild of America Award for Best Adapted Screenplay
67	Saturn Award for Best Action or Adventure Film
68	Saturn Award for Best Writing

This tale has one to one relationship since there is one prime attribute and one non prime attribute. This follows all the Normal forms.

FILM_AWARD_NOMINEE:

This table has the list of nominations of films and the awards for which it is nominated.

```
create table if not exists film_award_nominee(film_id int ,award_id int,year int,
       foreign key(award_id) references awards(award_id), foreign key(film_id) references films(filmId));
```

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in

3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

MOVIE_AWARD:

This table has the list of films and the awards won by that film. It has the ID associated with the film and the award.

```
create table if not exists movie_award(film_id int,award_id int,year int,
        foreign key(film_id) references films(filmId), foreign key(award_id) references awards(award_id));
```

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in 3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

RELEASE_UPDATE:

This table is associated with the release update of the movie or TV show. It has either **released** or **upcoming** as status.

```
create table if not exists release_update(status_id varchar(10) primary key,status_text text);
```

Sample output:

	status_id	status_text
▶	s1	Released
	s2	Upcoming

This tale has one to one relationship since there is one prime attribute and one non prime attribute. This follows all the Normal forms.

SHOWS:

This table has information about if the video is a **movie** or **TV show** or **Series.**

```
create table if not exists shows(show_id int primary key,show_type text);
```

Sample output:

	show_id	show_type
▶	1000	Movie
	1001	TV show
	1002	Series

This tale has one to one relationship since there is one prime attribute and one non prime attribute. This follows all the Normal forms.

StatusOfFilm:

This table is a derived table which informs about the film in relation to its **release update** and **show type.**

```
create table if not exists statusOfFilm( film_id int, status_id varchar(10),show_id int,
                                        foreign key(film_id) references films(filmId),
                                        foreign key(status_id) references release_update(status_id),
                                        foreign key(show_id) references shows(show_id));
```

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in 3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

DIRECTOR_AWARD, ACTOR_AWARD, WRITER_AWARD, CINEMATOGRAPHY_AWARD , MUSIC_AWARD:

These 3 tables are associated with the awards won by the director, actor, writer, cinematography, and musician respectively.

```
create table if not exists director_award(director_id int, film_id int,award_id int,year int,
                                foreign key(director_id) references directors(directorId),
                                foreign key(film_id) references films(filmId),
                                foreign key(award_id) references awards(award_id));

create table if not exists writer_award(writer_id varchar(10), film_id int,award_id int,year int,
                                foreign key(writer_id) references writers(writer_id),
                                foreign key(film_id) references films(filmId),
                                foreign key(award_id) references awards(award_id));

create table if not exists actor_award(actor_id int, film_id int,award_id int,year int,
                                foreign key(actor_id) references actors(actorId),
                                foreign key(film_id) references films(filmId),
                                foreign key(award_id) references awards(award_id));
```

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in 3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

DIRECTOR_AWARD_NOMINEE, ACTOR_AWARD_NOMINEE, CINEMATOGRAPHY_AWARD_NOMINEE :

These tables describe the award nominations for directors, cinematographers, and actors respectively.

```
create table if not exists director_award_nominee(director_id int, film_id int,award_id int,year int,
                                        foreign key(director_id) references directors(directorId),
                                        foreign key(film_id) references films(filmId),
                                        foreign key(award_id) references awards(award_id));
create table if not exists actor_award_nominee(actor_id int, film_id int,award_id int,year int,
                                        foreign key(actor_id) references actors(actorId),
                                        foreign key(film_id) references films(filmId),
                                        foreign key(award_id) references awards(award_id));
```

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in 3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

PRODUCTION_COMPANY:

This table gives the list of production companies that produce films.

```
create table if not exists production_company(production_id varchar(10) primary key, production_name text);
```

Sample Output:

production_id	production_name
prod1	Castle Rock Company
prod2	Hombale films
prod3	Albert Fennell
prod4	Brian Clemens
prod5	Victoria Alonso
prod6	David O. Selznick

This tale has one to one relationship since there is one prime attribute and one non prime attribute. This follows all the Normal forms.

PRODUCTION_MOVIES:

This table gives the list of films produced by the production company. Multiple production companies can produce a film situation is handled using this table.

```
create table production_movies( film_id int ,production_id varchar(10),
                                foreign key(film_id) references films(filmId),
                                foreign key(production_id) references production_company(production_id));
```

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in 3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

MUSICIAN:

This table gives the list of musicians who have produced albums or songs in a movie.

```
create table if not exists Musician(musician_id varchar(10) primary key, singer_name text);
```

Sample Output:

musician_id	singer_name
mus1	Thomas NewMan
mus2	Ravi Basrur
mus3	Vijay Prakash
mus4	A.W. Lumkin
mus5	William Axt
mus6	Wayne Allen

This tale has one to one relationship since there is one prime attribute and one non prime attribute. This follows all the Normal forms.

MOVIE_MUSICIAN:

This table gives the list of musicians and the films they have sung or composed in. This gives film details and musician details as well.

```
create table movie_musicians(film_id int,musician_id varchar(10),
                             foreign key(film_id) references films(filmId),
                             foreign key(musician_id) references Musician(musician_id));
```

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in 3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

GENRES:

This table has the list of genres the movies are usually based on like comedy, romance, crime, thriller and so on.

```
create table if not exists genres(genre_id varchar(10) primary key, genre text);
```

Sample Output:

	genre_id	genre
▶	gen1	Drama
	gen2	Action
	gen3	Crime
	gen4	Thriller
	gen5	Sci-Fi
	gen6	Adventure
	gen7	Romance
	gen8	Comedy

This tale has one to one relationship since there is one prime attribute and one non prime attribute. This follows all the Normal forms.

MOVIE_GENRES:

This table has the list of movies and it's genres. A movie can have multiple genres eg: comedy thriller. These cases are handled by this table.

```
create table if not exists movie_genre(film_id int,genre_id varchar(10),
                                       foreign key(film_id) references films(filmId),
                                       foreign key(genre_id) references genres(genre_id));
```

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in 3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

CRITIC_REVIEWS:

This table has a critic review provided on a film. It has film Id and the review provided.

```
create table if not exists critic_reviews(film_id int, rev text,critic_score int,
                                          foreign key(film_id) references films(filmId));
```

Sample Output:

	film_id	rev	critic_score
▶	11	It's the no-bull performances that hold back the flood of banalities....	90
	11	Gripping...compelling.	90
	11	Central to the film's success is a riveting, unfussy performance fro...	90
	11	At times poignant, joyful, and terrifying, Shawshank Redemption i...	89
	11	Whitmore's Brooks is a brilliantly-realized character, and the scene...	88
	11	Some of The Shawshank Redemption" comes across as outrageous...	75
	11	Shouldering a laconic-good-guy, neo- Gary Cooper role, Robbins n...	67
	11	Speaking of jail, Shawshank-the-movie seems to last about half a li...	40

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in 3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

USER_REVIEW:

This table has a review provided by a specific user to specific film. This table has user_id, film_id and review_comments.

```
create table if not exists user_review(user_id int,film_id int,review_text text,rating int,
                                       foreign key(film_id) references films(filmId),
                                       foreign key(user_id) references users(user_id));
```

Sample Output:

	user_id	film_id	review_text	rating
▶	51	11	Amazing!! must watch	9
	51	12	Superb movie	10
	52	13	crazy direction and acting...	9
	51	14	fun overloaded	8
	52	15	Sucks,no logic	1
	51	16	ROFL	1
	51	17	Bang bang!	9
	51	18	cute couple, awwwww...	9

We have a primary key in this table, and each attribute is atomic. As a result, this complies with 1NF. In a relation that is functionally dependent on any candidate key's appropriate subset, it has no non-prime characteristics. As a result, it's in 2NF. There is no transitive dependency for non-prime qualities, hence it is in

3NF. In this case both attributes are keys, So these will always identify a unique row. As a result, this is under BCNF.

4. Database Views

A view is a virtual table created from a SQL statement's result set. A view is like a table in that it has rows and columns. Fields from one or more real tables in the database are used in views.

I have created 4 views for my database each representing a information about the movie or the cast associated with the movie.

1. **Movie_Dashboard:**

   This view represents the film details along with the cast information. This view is intended for both standard and power users. When user logs into NUTFLUX, user will be seeing the dashboard with all the released movies, upcoming movies, TV shows released as well as upcoming.

```
create view movie_dashboard as
select  films.filmId,films.FilmTitle ,group_concat(distinct a.actorName) as actors,
    group_concat(distinct d.directorName) as directors,group_concat(distinct c.cin_name) as cinematographers,
    group_concat(distinct w.writer_name) as writers,group_concat(distinct pc.production_name) as producers,
    group_concat(distinct Musician.singer_name) as Musicians
    from films
    join movie_directors as md on films.filmId = md.film_id
    join directors d on d.directorId = md.director_id
    join filmcast as m on films.filmId = m.filmId
    join actors a on a.actorId = m.actorId
    join movie_cinematography as mc on mc.film_id = films.filmId
    join cinematography c on c.cin_id = mc.cin_id
    join production_movies as pm on films.filmId = pm.film_id
    join production_company pc on pc.production_id = pm.production_id
    join movie_musicians as mm on films.filmId = mm.film_id
    join Musician on Musician.musician_id = mm.musician_id
    join movie_writers mw on mw.film_id = films.filmId
    join writers w on w.writer_id=mw.writer_id
    group by films.FilmTitle;
```

2. **VERSATILITY_VIEW:**

   This view gives us the information about the actor and actors versatility

factor (actor taking on various role categories). The view is sorted in the descending order of versatility factor as the users can get the most versatile actors and their role history.

```
create view versatality_view as
select actorName,count(*) as factor
from (
select actorName,role_categ
from actors a
join filmcast fc on fc.actorId=a.actorId
join role_category rc on rc.role_id = fc.roleId
group by actorName,role_categ) as vc
group by actorName
order by factor desc;
```

3. ACTOR AWARDS AND NOMINATIONS:

This view has actors career awards and nominations. This view gives actorName, awards won by actor and awards actor was nominated for. This view can be further modified to get more details like “for which movie he won the award?”, “for what role?” etc.

```
create view award_winning as (select ac.actorName,group_concat(a.award_name) as winning
   from actors ac
   join actor_award ma on ma.actor_id = ac.actorId
   join awards a on a.award_id=ma.award_id
   group by ac.actorName);


create view award_nominee as (select ac.actorName,group_concat(a.award_name separator " ,") as nominee
   from actors ac
   join actor_award_nominee ma on ma.actor_id = ac.actorId
   join awards a on a.award_id=ma.award_id
   group by ac.actorName);

create view ActorsAwardsAndNominations as
select A.actorname,winning,nominee from (select * from award_winning) as A left join
   (select * from award_nominee ) as B ON A.actorName=B.actorName;
```

4. **MULTI_LANGUAGE_MOVIES:**

This view is in intention of the user for being flexible with the language of their favorite movie. If the user is prompted about his/her favorite movie in the **language of his priority (a feature that can be added on requirement)** then user can be prompted with this view.

```
create view multi_language_movie as
select filmTitle, group_concat(language_name) in_languages from films f
join movie_multi_lang mml on mml.film_id=f.filmId
join languages l on l.language_id = mml.language_id
group by filmtitle
having count(*)>2;
```

5. Procedural Elements

Triggers, procedures, and events are among the procedural extras in this design. Each of them is in charge of depicting a specific functionality in a NUTFLUX real-world application.

Triggers

Triggers are stored programs that are automatically run when a specified event occurs, such as an insert, update, or delete. They are in charge of ensuring that the data in the database is accurate. They merely perform additional data validations. In this database, check constraints were used to do simple data validation. In this database design, I've defined three triggers.

1. VALIDATE_USER_EMAIL

Every NUTFLUX users are associated with an user email on creating their account. The registered email should be a valid email to keep the users posted. The email is important for account validation and for the password related operations.

Query Snippet

```
CREATE TRIGGER validate_user_email BEFORE INSERT ON users
for each row
BEGIN
    declare msg varchar(128);
    if NEW.user_email NOT LIKE '%_@%_.__%' THEN
        set msg = concat('Invalid email: ', cast(new.user_id as char));
        signal sqlstate '45000' set message_text = msg;
    end if;
END;
```

Figure 1:email validation trigger

The trigger is triggered before new data is inserted into the table, as shown in the preceding example. So, everytime a new user opens an account and enters information, if the email does not fulfill the trigger's requirements, it will send an error message with the statement MSG. In MySQL, regex pattern matching is used to accomplish this.

2. USER_EMAIL_ALREADY_EXISTS:

Whenever a user tries to create an account, When the user enters his/her email id then it must be validated if the email id is already registered. If it is already registered the user must be prompted with the message that "Email already exists". This is the whole intention of this trigger.

Query Snippet

```
CREATE TRIGGER user_email_already_exists
BEFORE INSERT
ON users
FOR EACH ROW
Begin
IF NEW.user_email in (select users.user_email from users) THEN
set msg = concat('Email Already exists. Please Login using Credential:');
signal sqlstate '45000' set message_text = msg;

END IF;
end
```

As you can see in the above query, The event is invoked before the insert

of new data. It validates to check the new email entered is already present or not. If it doesn't exist, then new data is inserted into database else it is intended to prompt the user.

Procedures

Procedures are group of SQL statements that are stored together in a database which can reused.

1. PASSWORD_UPDATE

This is a method for allowing users to change their passwords. This is a fundamental feature that will be made available to all NUTFLUX users. So, if a user wants to change his password at any moment, he or she can do so using this method. As a result, this method will complete the entire process.

Query Snippet

```
create procedure sp_password_update(in username text,in old_pwd text, in new_pwd text)
begin
    if exists(select * from users where user_email=username and user_pwd=old_pwd) then
        update users u1 set u1.user_pwd = new_pwd  where u1.user_email = username and u1.user_pwd=old_pwd;
    else
        SIGNAL SQLSTATE VALUE '10581'
        SET MESSAGE_TEXT = "Wrong username and password";
    end if;
end
```

In the above Query, after validating the user email and his old password, the user will be given access to update his password.

2. EMAIL UPDATE

This is a method for allowing users to change their user email. This is a fundamental feature that will be made available to all NUTFLUX users. So, if a user wants to change his email on which he will be posted about the movies and tv shows, he or she can do so using this method. As a result, this method will complete the entire process.

Query Snippet

```
create procedure sp_email_update(in username text,in old_pwd text, in new_email text)
begin
    if exists(select * from users where user_email=username and user_pwd=old_pwd) then
        update users u1 set u1.user_email = new_email  where u1.user_email = username and u1.user_pwd=old_pwd;
    else
        SIGNAL SQLSTATE VALUE '10581'
        SET MESSAGE_TEXT = "Wrong username and password";
    end if;
end
```

3. FILTER_MOVIES

This procedure filters the movies based on the feature that user wants like IMDB ratings, critics reviews and release date. This includes features like the user can wish to select the movies within a window period. If the user fails to select among these options then the user will be promoted to choose among the options given.

Query Snippet

```
create procedure filter_movies(in start_year int, in end_year int, in sort_on text)
begin
if sort_on = "imdb_rating" then
select filmtitle,descriptions,imdb_rate,YearOfRelease,duration
from films f
where f.YearOfRelease >start_year and f.YearOfRelease<end_year
order by f.imdb_rate desc;
elseif sort_on = "critics_rating" then
select filmtitle,descriptions,imdb_rate,YearOfRelease,duration,critic_score
from films f
join critic_reviews cr
on f.filmId = cr.film_id
where f.YearOfRelease >start_year and f.YearOfRelease<end_year
group by f.filmId
order by avg(cr.critic_score) desc;
elseif sort_on = "release date" then
select filmtitle,descriptions,imdb_rate,YearOfRelease,language_name,duration
from films f
join languages l on l.language_id=f.language_id
where f.YearOfRelease >start_year and f.YearOfRelease<end_year
group by f.filmId
order by f.YearOfRelease;
else
SIGNAL SQLSTATE VALUE '10581'
SET MESSAGE_TEXT = "Incorrect Sorting type : Choose either imdb_rating / critics_rating / release date";
end if;
end //
```

The above procedure uses a if-else block to describe various options given to the users and the results are displayed accordingly as option selected by the user.

Events

MySQL Events are scheduled tasks that run on a regular basis. As a result, MySQL events are also referred to as planned events. MySQL Events are named objects that have one or more SQL statements in them. They're saved in the database and run at predetermined intervals.
I have two events scheduled for my database:

1. event scheduled for new pro user to get notified 5 days before the subscription ends.

```
-- subscription end notification
drop event if exists subscription_end_notify;
CREATE EVENT subscription_end_notify
    ON SCHEDULE EVERY  25 day
    COMMENT ''
    DO
      SIGNAL SQLSTATE VALUE '10581'
      SET MESSAGE_TEXT = "subscription about to end";
```

2. Event scheduled every day, if a new movie is inserted and user liked the previous movie by the same director/actor then user should be recommended.

```
CREATE EVENT recommendation_event
    ON SCHEDULE
      EVERY 1 day
    COMMENT ''
    DO
      call sp_userRecommendations('pawan98ppk@gmail.com');
```

The above event is specific to user, so when the user creates an account in NUTFLUX. The event gets activated. If the user don't have any liked movies previously liked movies, then dashboard is displayed.

6. Example Queries: Your Database In Action

1. Query -1: Pair of actors audience love to see together:

This query is based on pair of actors who have made the movies together most of the times are likely to be working together because the audience want to see them together on the screen. This can then be further extended to list all the movies that this pair of actors worked together and recommend to the user who loved one of their film based on their review and ratings provided in the user_review table.

```
select f1.filmTitle, a1.actorName,a2.actorName,count(*) as films_together
from films f1
join filmcast fc1 on fc1.filmId=f1.filmId
join actors a1 on a1.actorId = fc1.actorId
join films f2 on f1.filmId=f2.filmId
join filmcast fc2 on fc2.filmId=f2.filmId
join actors a2 on a2.actorId = fc2.actorId
where a1.actorId<>a2.actorId
group by a1.actorName,a2.actorName
having count(*)>2 limit 1;
```

Sample output:

	filmTitle	actorName	actorName	films_together
▶	KGF 2	Yash	Srinidhi Shetty	3

2. Query 2 – Pair of actors director is comfortable working with:

This query is designed on basis of pair of actors who have worked under same director but different films multiple times. Directors get comfortable with the actors or director-actor combination goes well in both screen play as well as for the story line which make people love them.

```
select DISTINCT D.DirectorName,a1.ActorName as Actor1,a2.ActorName as Actor2
from movie_directors md
JOIN DIRECTORS D ON md.director_Id=D.directorId
JOIN filmcast fc1 ON md.film_Id = fc1.filmId
JOIN actors a2 ON fc1.actorId=a2.actorId
JOIN filmcast fc ON fc.filmId = md.film_Id
JOIN actors a1 ON fc.actorId=a1.actorId
WHERE a1.ActorName <> a2.ActorName
AND EXISTS (SELECT * FROM movie_directors f1
    WHERE md.director_Id = f1.director_Id
    AND md.film_Id <> f1.film_Id)
GROUP BY md.film_Id;
```

Sample output:

	DirectorName	Actor1	Actor2
▶	Jack Conway	James McAvoy	Michael Fassbender
	Joss Whedon	Chris Evans	Robert Downey Jr.
	Prashanth Neel	Michael Fassbender	Ian McKellen
	Prashanth Neel	Sanjay Dutt	Yash
	Prashanth Neel	Yash	Srinidhi Shetty

3. **Query 3- Kevin Bacon degree 1** – which can be further nested for more degree. Bacon number is a factor which denotes the level at which pair of actors have acted together directly or indirectly. According to the game, an actor's Bacon number is the number of degrees of separation he or she has from Bacon. The greater the distance between the actor and Kevin Bacon, the higher the Bacon number.

```
select count(distinct actorId) as Bacon_number from filmcast where filmid in (
    select filmid from filmcast where actorid in (
        select distinct actorid from filmcast where filmid in (
            select filmid from filmcast join actors on filmcast.actorid=actors.actorid where actorName='Kevin Bacon'))
and actorid not in
    (select distinct actorid from filmcast where filmid in (
        select filmid from filmcast join actors on filmcast.actorid=actors.actorid where actorName='Kevin Bacon'));
```

Sample output:

	Bacon_number
▶	1

4. **Query 4 – Hits percentage for a director in the industry:**

The hits percentage of a director shows how successful the director is. Based on this the users can easily shortlist the best directors whose movies can be watched. This is again based on the budget spent and gross collection made as well as user likes for the movie.

```
select hits.directorName,hits, (hits/total_movies)*100 as hit_percentage from
(select directorName, count(*) as hits
from directors d
join movie_directors md on directorId=director_id
join films f on f.filmId = md.film_id
join user_review ur on ur.film_id=f.filmId
where ur.rating > 5 and GrossCollection > @budget
group by directorName
order by hits desc) as hits
join
(select directorName,count(*) as total_movies from directors d
join movie_directors md on md.director_id=d.directorId
join statusOfFilm s on s.film_id=md.film_id
join release_update r on r.status_id = s.status_id
where r.status_text="Released"
group by directorName
order by total_movies desc)as tot_movies on hits.directorName = tot_movies.directorName;
```

Sample output:

	directorName	hits	hit_percentage
▶	Prashanth Neel	3	100.0000
	W.S. Van Dyke	2	100.0000
	Jack Conway	1	50.0000
	Jocky	1	100.0000
	Sydney Newman	1	100.0000
	George Cukor	1	100.0000

7. Conclusions

NUTFLUX is currently a platform which has movies, TV shows and series. This attracts adults, but we still have a room for improvements for age groups under 18.

So, in the future the vision for improvements that are in the check list are:

- This platform is currently designed for adults (i.e., age over 18) using the check constraint in the users table. This can be expanded more to deal with kids by **adding a kids section** in the shows. This

can be made very easily with out any major changes to the script by just adding a kids section in the **shows** table.

- For the pro users, the area of improvements thought of are actors career growth. This can be determined by amount of movies a actor has done in a year over the years. This determines the actors career growth. This can be planned on large scale data. Since the data is limited in the current database, it is not worthy to include this.
- For both standard and pro users, I have planned to perform an analysis showing how with age the user likes to the genre of the movie varies. So, when a new user of any age signs up to NUTFLUX we can give the user with better recommendations to start with.
- Apart from the querying, The platform can be made more aware of spam reviews if the reviews come of the same user name multiple times.
- Since, this has TV shows in it, we can also check on how users respond to tv shows and add more of such TV shows which can help in attracting more subscribers.

Acknowledgements

All the books and materials I have referred for these projects are attached in the reference section. Apart from this all the queries and analysis listed are inferred by me alone. I have not quoted anything from any books or websites directly apart from movie details and descriptions.

References

https://www.geeksforgeeks.org/sql-tutorial/

https://ieeexplore.ieee.org/document/9101036 - for SQL concepts like events and triggers

https://www.imdb.com/?ref_=nv_home – for movie details

https://brightspace.ucd.ie/d2l/le/content/154267/Home - Bright space material for most ideas.