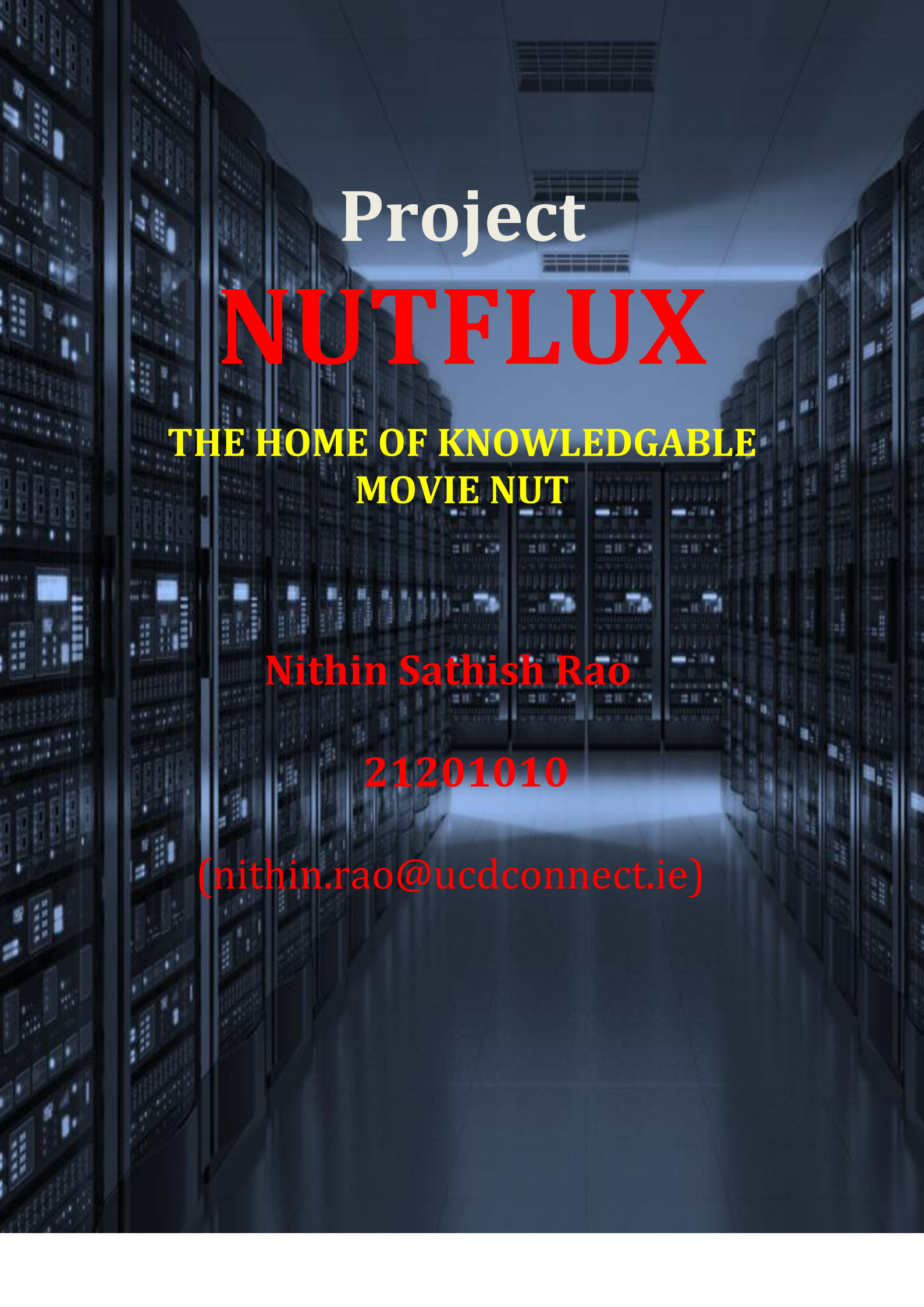

# Project

# NUTFLUX

## THE HOME OF KNOWLEDGABLE MOVIE NUT


**Nithin Sathish Rao**

**21201010**

(nithin.rao@ucdconnect.ie)

## Table of Contents

## List of Figures

# 1. Introduction

NUTFLUX is a dominant streaming platform for TV and film content which is trying to expand its potential by providing a data-based knowledge rich offering for its users. This database design is mainly focused to facilitate all these new upgrades which are being stated in the Project whitepaper. This design document just illustrates a an efficient and a consistent way of storing the data and providing them to the users in attractive manner. The main aim of this document is to report how we can structure the database with table, views and relations such that it can satisfy all the requirements put forward by the company.

This vision of this project is to satisfy both Standard and Pro users and to be robust enough to handle additionally data when deployed in real time. This design currently holds almost all kinds of characteristic information about the films in large sets of normalized tables. This design can be further expanded to handle more data by creating tables in a very easy and efficient manner. The data which we are dealing is large, but this design is capable of handling this without any huge issues.

When it comes to developing the real application such as NUTFLUX, the database design acts as foundation on which the application can be developed accordingly. So, in the larger ecosystem, this design will provide all the necessary information which the application should retrieve and store in these tables. The preliminary checks on the data provided by the users must be done at user interface level to reduce the overhead on the design. For example, the verification of user email, age and credibility of the film rating must be handled by the developer before inserting into the tables. Apart from these, it is only possible to show relevant information to users only if the data exists in the table. So it is necessary to collect all the information about movies from all genre, all languages and populate this database so that users can get an immersive experience when using NUTFLUX.

What you can expect from this design report is how easily you will be able to access the data and show relevant output to the users. There are many sample

procedures, views written in this design which shows how robust this design this. Many procedures that are defined in this take parameters such as user id, password, film id etc. These needs to be given to the procedure from the user interface. The views that are available are also can be easily accessed and displayed to users so that they can get new gist to watch these movies.

There are some very common features like recommendations, trending movies, Movie verdicts which still needs upgradation as it cannot be done completely from the database point of view. It needs some input from users to determine their likes and dislike and recommend the movies accordingly. There are triggers set up in this database that run as an event daily to determine if the subscription of the user is about to end or not. If its close to ending date, the notification needs to be sent to PRO users. But this notification cannot be handled in the design level. It needs to be done from the developer's side. At the same time, the notification for new upcoming movies or trending movies should be sent from the developers end but the right data can be provided from this design. Overall this design is just the start and many wonderful things can be built upon this.

Further details about the database design is provided in this report as we keep reading.

## 2. Database Plan: A Schematic View

In this Design, there are 32 tables which replicate all the functionalities that is needed to build the NUTFLUX application. These tables satisfy all the requirements defined in the project whitepaper and allows to add some extra features which makes NUTFLUX an attractive platform to movie nuts. But there are mainly 14 principal entities in this design which add value in terms of information to the data base. They are not dependent on any other entity in the database and are the building blocks of this database. These tables contain primary key (Simple or Composite), but this primary key is not foreign. The rest of the tables are dependent entities / derived entities whose existence depend on two or more principal entities and are connected using foreign key relationship.

## PRINCIPAL ENTITIES ATTRIBUTES AND KEY RELATIONSHIP

### 1. FILMS

This table stores the information about all the films in this database.

The attributes of this tables are unique film id, film title, year of release, runtime of the film (in minutes), summary plot of the film, famous tagline of the film, gross collection worldwide (in dollars), production rights amount (in dollars), marketing rights amount (in dollars). The film id is a primary key of this entity which is used for relationship by other tables.

### 2. ACTORS

This tables stores information about all the actors in this database. The attributes of this table are a unique actor id, real name, theatre name (some actors change their real name to get more publicity or attraction), gender and nationality. The actor id is the primary key in this entity.

### 3. ROLES

This table stores the information about all the roles played by the actor in this database. The attributes of this table are a unique role id, role name, and a famous quote specific to the role. The role id is the primary key in this table.

**FILMCAST** is the key relationship that exists between these above three table. Each entry in **FILMCAST** gives us the information about which role did an actor play in this film for how much money. The Primary Key of this table is a composite key of Film id, Actor id and Role id.

### 4. DIRECTORS

This table stores the information about all the directors in this database. The attributes of this table **are** a unique director id, Name, gender, and Nationality. The director id is the primary key of this table.

**FILM_DIRECTORS** table creates the relationship between DIRECTORS table and FILMS table. A separate table is created for this relationship

because one film may have multiple directors. The composite primary key of this table is Film id and Director id.

### 5. WRITERS

This table stores the information about all the writers in this database. The attributes of this table are a unique writer id, name, and gender. Writer id is the primary key of this table.

**FILM_WRITERS** table creates the relationship between WRITERS table and FILMS table. A separate table is created for this relationship because one film may have multiple writers. The composite primary key of this table is Film id and Writer id.

### 6. PRODUCTION

This table stores the information about all the production companies in this database. The columns in this table are a unique production company id which is the primary key and company name.

**FILM_PRODUCTION** table creates the relationship between PRODUCTION table and FILMS table. A separate table is created for this relationship because one film may have multiple production team. The composite primary key of this table is Film id and company id.

### 7. GENRE

This table stores the information about all types of genres using a unique genre id as primary key.

**FILM_GENRE** table creates the relationship between GENRE table and FILMS table. A separate table is created for this relationship because one film may have multiple genres. The composite primary key of this table is Film id and genre id.

## 8. LANGUAGES

This tables stores information about the all the languages in films have been released in this database. Each language is uniquely identified using language id which is the primary key.

**FILM_LANGUAGES** table creates the relationship between LANGUAGES table and FILMS table. A separate table is created for this relationship because one film is usually release in multiple languages. The composite primary key of this table is Film id and language id.

## 9. RELATION

This table contains all type of relationship that can exists between two actors in the film industry. Each relation is identified using a unique relation id which is the primary key.

**CONNECTIONS** table creates the relationship between ACTORS table and RELATION table. This table has actor one id and actor two id and the relationship that exists between them (Relation id). It also contains details about from when to when the relationship existed. The composite primary key of this table is Actor One id and Actor Two id. This table facilitates the situation where one actor has more than one relationship with multiple actors.

## 10. ROLE_TYPE

This table stores information about the characteristics of all the roles in this database. Each entry is uniquely identified using a role id.

**ROLE_CATEGORY** table creates the relationship between ROLES table and ROLE_TYPE table. Each role may have multiple characteristics associated with it. Hence this table satisfies this condition. The composite primary key of this table is Role id and Role type id.

## 11. AWARDING_INSTITUION

This table stores all the information about different awarding institutions. Each awarding institution is identified using a unique ID which acts as the primary key of this table.

## 12. AWARD_CATEGORY

This table contains information about the different type of awards that can be won by actor or writers or directors. Each award category is associated with a primary key ID.

**AWARD_ACTORS** table creates the relationship between ACTORS table, FILMS table, AWARDING_INSTITUTION table and AWARD_CATEGORY table. One actor may win or get nominated for multiple awards over his career for multiple movies. Hence this table is created which associates each actor to his award won/nominated for his film. The composite primary key of this table is Actor id and Film id and Award category id.
Similarly, **AWARD_WRITERS and AWARD_DIRECTORS** creates the relation between writers to their awards and directors to their awards respectively. Separate table has been created for each of them to differentiate if a person has won the award as an actor or writer or director and to hold the foreign key relation between these types.

## 13. USERS

This table contains all the information about the users in NUTFLUX database which includes both standard and Pro users. The attributes of the table are a unique user id, username, password, date of birth, type of the subscription, end date of the subscription, nationality, Email ID, a security question, answer to the security question and auto extend subscription which takes value 0/1 (0 meaning the subscription is to be extended manually by the users and 1 meaning it will be auto renewed). User id is the primary key of this table.

**USER_GENRE** table creates the relationship between GENRE table and USERS table. This stores the favorite genre of the user which will facilitate in populating the WATCHLIST table. The composite primary key of this table is User id and Genre id.

**USER_VIEWS** table also creates the relationship between FILMS table and USERS table. This stores the films watched by the user and stores the information like whether the user liked the film or not, what rating (out of 10) did the user give for the movie and what is the review comments. The composite primary key of this table is User id and Film id.

**WATCHLIST** table also creates the relationship between FILMS table and USERS table. But the only difference is it stores the data only for NUTFLUX Pro Users. This stores the Movies which Pro users might be interested to watch based on his favorite genre.

## 14. CRITICS

This table stores information about all the famous critics who are responsible to give their review of the film. These are the chosen people who give their view on the film considering all factors responsible in delivering the film. The primary key of this table is the Critics id.

**CRITICS_REVIEWS** table creates the relationship between FILMS table and CRITICS table. The rating and the comments provided by the critics for a film is stored in this table. The composite primary key of this table is films id and critics id.

**FILM_VERDICT** table stores the verdict of the film based on the revenue generated by the film. This table gets populated when the new film is inserted into the films table.

**FILM_RATINGS** table stores the rating of the film given by the viewer. This is segregated into Male viewers rating and Female Viewers rating. The average of these rating is stored as total rating in this table.

## MOTIVATION FOR THIS DESIGN

The importance of each table and the reason for its existence is explained clearly in the above design plan involving the principal entities and the relationship tables. Multiple tables like **FILMCAST, FILM_DIRECTORS, FILM_WRITERS, FILM_PRODUCTION** is created keeping in mind the fact that one movie can have multiple actors, directors, writers, and production company. Similarly, **FILM_LANGUAGES** table is created because one movie can be released in more than one language. These tables are necessary to keep the database normalized and make it easier to update or do some queries.

The major reason for the creation of multiple similar tables for awards information was to distinguish the type of people who won or got nominated for the award. We created three tables **AWARD_ACTOR, AWARD_DIRECTOR, AWARD_WRITER** mainly to satisfy the situation where multiple actors / writers / directors can win multiple awards over their illustrious career in the film industry. There are scenarios where an actor who became a director later in his career might also win some awards. This design is created keeping these in mind. Some of the tables related to users like **USER_VIEWS** are created to keep track of count of movies watched by the users. These data can be very useful for someone who wants to do some data analysis on usage of NUTFLUX application or to determine the popularity of this application in relation to other streaming platforms in the market once the application is launched. The **USER_GENRE** table is created to store some favorite type of movies which the users might like so that they can get suggestions based on this from **WATCHLIST** table. In this design, this feature is limited to genre but when building the application, it can be extended to favorite actors, writers, directors, production company. This can give a personalized experience to users. In this design we have stored detailed information about the actor and their relationship history in the **CONNECTIONS** table. These data can be valuable for PRO users who are interested in knowing the actors

history before watching the film or to those who are interested in trying to make a connection with actors real life after watching a movie.

Apart from these features we have provided reviews to movie using **CRITICS_REVIEWS** table. These are the people who are well known in analyzing the film and give accurate opinion taking various factors into consideration without any bias. These opinions will be valuable for both standard and Pro users.

Well, this design still can be improved to store more information about movies like Music produces, or singers or stunt double or team. These can be easily incorporated by creating tables similar to writers , actor and creating a relationship similar like filmcast or film_writers.

## ER DIAGRAM

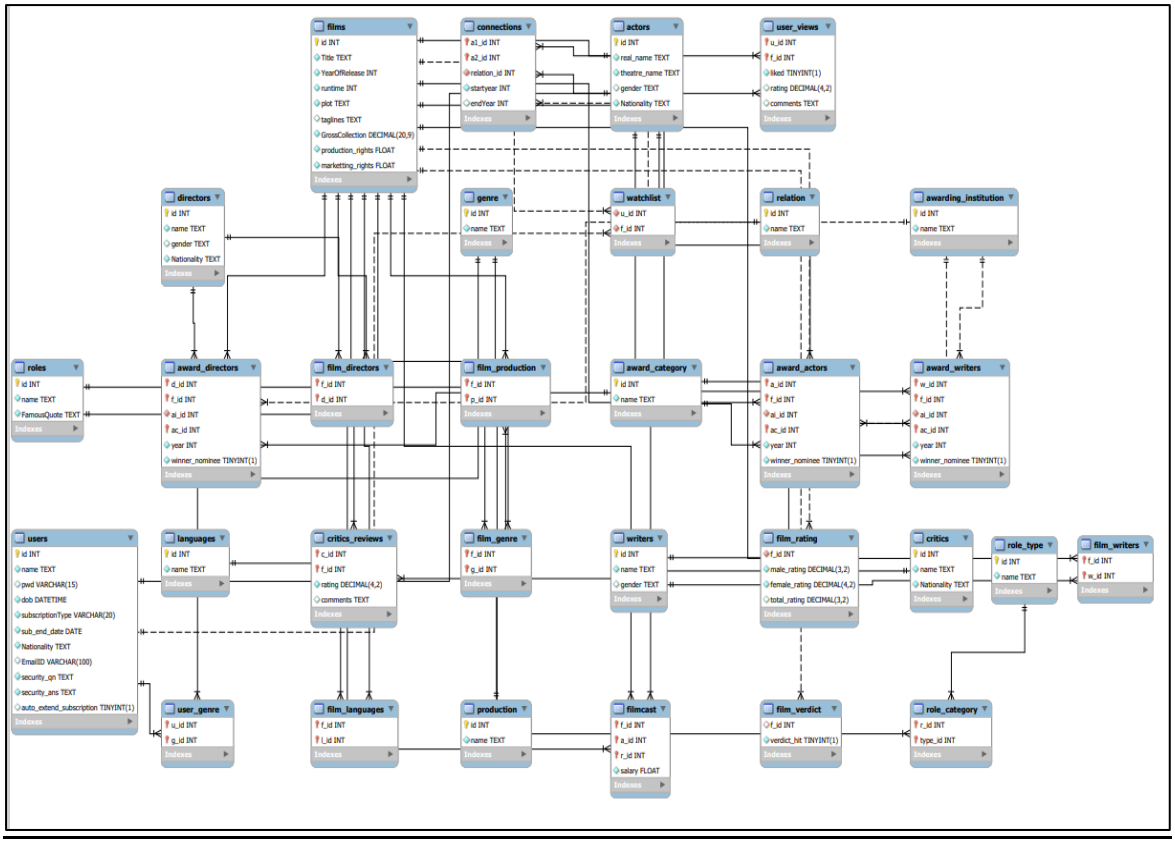


*Figure 1 Entity Relationship Diagram of NUTFLUX*

**NOTE: A high quality image Is uploaded as a pdf for clearing understanding. Along with this a MWB file is uploaded to get clearer and intuitive view of the relationship between entities.**

## 3. Database Structure: A Normalized View

### Independent entities

#### 1. USERS

This table contains the information about the users that exists in this table. User id acts as the primary key in this table. This table does not contain any multivalued attributes and each row is uniquely identified by the primary key. All the other attributes depend only on primary key. Since there is no transitively dependency, this table is in 3NF and also in BCNF as the functional dependency is from the only candidate key user ID. User id -> email ID and Email ID -> User id. But since email id is unique key, it is in BCNF.

**Sample Data**

| id | name | pwd | dob | subscriptionType | sub_end_date | Nationality | EmailID | security_qn | security_ans | auto_extend_subscription |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Nithin | chicharito#@0M | 1997-10-18 00:00:00 | Standard | 2022-10-18 | Indian | nsrao48@gmail.com | My favourite Car | Audi | 0 |
| 2 | Paul | qwert1234M@ | 1993-10-23 00:00:00 | Pro | 2022-04-24 | Italian | paulwalker98@gmail.com | My Favourite Film | Fast and Furious | 1 |
| 3 | hrithik | xecrv12@#@ | 1995-11-25 00:00:00 | Pro | 2022-04-24 | Indian | hrithikshg98@gmail.com | My Favourite food | biryani | 0 |

#### 2. DIRECTORS

This table contains all the information about the directors. Director id is the primary key in this table, and it uniquely identifies each director. There are no atomic values nor any transitive dependencies. Hence this table is in 3NF. Along with this the table is in BCNF as there is no functional dependency without the primary key director id.

**Sample Data**

| id | name | gender | Nationality |
|---|---|---|---|
| 1 | Greg McLean | Male | Austrailian |
| 2 | David Twohy | Male | American |
| 3 | Justin Lin | Male | American |
| 4 | Quentin Tarantino | Male | American |

## 3. ACTORS

This table contains all the information about the actors of the film. Every actor is uniquely identified by a primary key actor id. Again, this table is like DIRECTORS table and hence this is in BCNF since there is only one candidate key.

**Sample Data**

| id | real_name | theatre_name | gender | Nationality |
|---|---|---|---|---|
| 1 | Kevin Bacon | Kevin Bacon | Male | American |
| 2 | Radha Mitchell | Radha Mitchell | Female | Austrailian |
| 3 | Daniel Radcliffe | Daniel Radcliffe | Male | English |
| 4 | Thomas Kretschmann | Thomas Kretschmann | Male | German |
| 5 | Alex Russel | Alex Russel | Male | Austrailian |

## 4. ROLES

This table contains all the information about the roles played by the actors in the films. Each role is uniquely identified using role id and since all the other attributes non candidate key, this table is in 3NF and BCNF.

**Sample Data**

| id | name | FamousQuote |
|---|---|---|
| 1 | Peter Taylor | You take me instead you live him alone |
| 2 | Bronny Taylor | You are scaring him |
| 3 | Yossi Ghinsberg | I told my parents I'd be back in a year, but I do... |
| 4 | Karl | This is the last frontier on earth. Still alive, still ... |

## 5. PRODUCTION

This table contains information about all the production companies that are responsible for developing the films. Each item in this table is identified by a production id which is the primary key in this table. Since all the other attributes are atomic and there is no other candidate key in this table, This is in BCNF.

**Sample Data**

| id | name |
|---|---|
| 1 | Blum House production |
| 2 | Chapter One Films |
| 3 | Babber Films |
| 4 | Cutting Edge Group |

## 6. GENRE

This table stores the information about all types of genre using a unique id to identify which is the primary key. Similar to Production table, this is also in BCNF.

**Sample Data**

| id | name |
|---|---|
| 1 | Horror |
| 2 | Thriller |
| 3 | Action |

## 7. WRITERS

This table contains information about all the writers of the films. Every writer is uniquely identified using writer id which is the only primary key in this table. This table is again similar to the above table and thus its in BCNF.

**Sample Data**

| id | name | gender |
|---|---|---|
| 1 | Greg McLean | Male |
| 2 | Shayne Armstrong | Male |
| 3 | S.P. Krause | Male |

## 8. LANGUAGES

This table is used to store all the languages in which a particular movie has been released. Each language is uniquely identified using an id which is the primary key. This table again does not have any other candidate key and there is no transitive dependency. Hence it is in BCNF.

**Sample Data**

| id | name |
|---|---|
| 1 | English |
| 2 | Arabic |
| 3 | Danish |
| 4 | Spanish |

## 9. AWARDING_INSTITUTION

This table stores all the information about different awarding institutions. Each awarding institution is identified using a unique ID which is the only candidate key in this table. Hence this table is also in BCNF.

**Sample Data**

| id | name |
|---|---|
| 1 | Oscar |
| 2 | MTV |
| 3 | Choice |

## 10. AWARD_CATEGORY

This table contains information about the different type of awards that can be won by actor or writers or directors. Each award category is associated with a primary key ID which is the only candidate key in this table. Hence it is in BCNF.

**Sample Data**

| id | name |
|---|---|
| 1 | Best Dramatic Feature |
| 2 | Best Male Action Superstar |
| 3 | Best Achievement in Directing |
| 4 | Best Original Screenplay |

## 11. RELATION

This table contains all type of relationship that exists between two actors in the film industry. Each relation is identified using a unique id which forms the primary key of the table. Since there are no other candidate keys in these table, this table is in BCNF.

**Sample Data**

| id | name |
|---|---|
| 1 | married |
| 2 | Divorced |
| 3 | Dating |

## 12. ROLE_TYPE

This table contains information about all the types of roles played by the actors. Each row is uniquely identified by ID which acts as the only primary key. Hence this table is in BCNF.

**Sample Data**

| id | name |
|---|---|
| 1 | doctor |
| 2 | superhero |
| 3 | magician |
| 4 | hero |

## 13. FILMS

This table contains all the information about the films. Every film is uniquely identified using film id. There are no multivalued attributes in this table. This table is In BCNF because there is no other candidate key apart from film id. Film title cannot be considered as a candidate key as there many movies with duplicate names. (Example Movie: The godfather). Since it satisfies all the conditions, it is in BCNF.

**Sample Data**

| id | Title | YearOfRelease | runtime | plot | taglines | GrossCollection | production_rights | marketting_rights |
|---|---|---|---|---|---|---|---|---|
| 1 | The Darkness | 2016 | 92 | A family unknowingly awakens an ancient super… | Evil comes home | 10898293.000000000 | 8000000 | 3000000 |
| 2 | Jungle | 2017 | 145 | In the pursuit of self-discovery and authentic e… | Nature Has Only One Law - Survival. | 1906640.000000000 | 500000 | 400000 |
| 3 | Pitch Black | 2000 | 109 | A commercial transport ship and its crew are ma… | Fight Evil With Evil | 53187659.000000000 | 25000000 | 1000000 |
| 4 | Fast and Furious 6 | 2013 | 130 | Hobbs has Dominic and Brian reassemble their c… | All roads lead to this | 160000000.000000000 | 18000000 | 10000000 |

## 14. CRITICS

This table stores information about all the famous critics who are responsible to give their review of the film. Each critic is uniquely identified using critics ID and there is no other candidate key in this table to uniquely identify the row. Hence this table is in BCNF

**Sample Data**

| id | name | Nationality |
|---|---|---|
| 1 | Richard Cross | English |
| 2 | Mathew Huntley | Austrailian |
| 3 | Harry Wilson | American |

## Dependent entities

### 15. FILMCAST

This table is responsible for mapping all the actors with the role they played in the film and the salary they earned for this role. In this table, the film id, actor id and role id together form the primary. Hence the only functional dependency is with the primary key and salary of the actor. We cannot consider role as the candidate key, as one role may be played by many actors in different films. (Example: James Bond). Since there is no other candidate key, this table is in BCNF. W

**Sample Data**

| f_id | a_id | r_id | salary |
|---|---|---|---|
| 1 | 1 | 1 | 1000000 |
| 1 | 2 | 2 | 500000 |
| 2 | 3 | 3 | 780000 |
| 2 | 4 | 4 | 550000 |

### 16. FILM_LANGUAGES

This table stores the connection between films table and languages table. Each row depicts the languages in which this film has been released. The primary key is this table is film id and language id. Since there are no other attributes to which there exists a functional dependency, this table is in BCNF.

**Sample Data**

| f_id | l_id |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 1 |
| 3 | 2 |

## 17. FILM_PRODUCTION

This table stores the information about the production team of the film. The primary key in this table is film id and production id. Since there are no other attributes to which there exists a functional dependency, this table is in BCNF.

**Sample Data**

| f_id | p_id |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 2 | 3 |
| 2 | 4 |

## 18. FILM_WRITERS

This table stores the information about the writers of the film. The primary key in this table is film id and writer id. Since there are no other attributes to which there exists a functional dependency, this table is in BCNF.

**Sample Data**

| f_id | w_id |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |
| 2 | 4 |

## 19. FILM_DIRECTORS

This table stores the information about the directors of the film. The primary key in this table is film id and director id. Since there are no other attributes to which there exists a functional dependency, this table is in BCNF.

**Sample Data**

| f_id | d_id |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |

## 20. FILM_GENRE

This table stores the information about the film genre. The primary key in this table is film id and genre id. Since there are no other attributes to which there exists a functional dependency, this table is in BCNF.

**Sample Data**

| f_id | g_id |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 2 | 3 |
| 2 | 4 |

## 21. ROLE_CATEGORY

This table describes the type of role played by each actor. The primary key in this table is role id and type id. Since there are no other attributes to which there exists a functional dependency, this table is in BCNF.

**Sample Data**

| r_id | type_id |
|---|---|
| 18 | 4 |
| 18 | 10 |
| 18 | 13 |
| 19 | 10 |
| 19 | 13 |

## 22. CONNECTIONS

This table stores the relationship information between costars. In this table each relation is uniquely identified by actor one id and actor two id. All the functional dependencies originate from this primary key. Since there are no other candidate keys in this table, This table is in BCNF.

**Sample Data**

| a1_id | a2_id | relation_id | startyear | endYear |
|---|---|---|---|---|
| 11 | 15 | 2 | 2014 | 2019 |
| 11 | 16 | 2 | 2000 | 2005 |
| 17 | 18 | 1 | 2012 | 2022 |

### AWARD_ACTORS

This table stores the details about all the actors who have won or got nominated for prestigious awards for acting in the movie. The primary key in this table is combination of actor id, film id and award id. In this table we cannot find any other candidate key and all functional dependency is coming from the primary key, this table is in BCNF.

**Sample Data**

| a_id | f_id | ai_id | ac_id | year | winner_nominee |
|---|---|---|---|---|---|
| 7 | 4 | 5 | 2 | 2014 | 1 |
| 11 | 5 | 1 | 5 | 2020 | 1 |
| 14 | 6 | 1 | 5 | 2013 | 1 |
| 11 | 7 | 4 | 7 | 2006 | 1 |
| 15 | 7 | 3 | 6 | 2006 | 1 |

### 23. AWARD_WRITERS

This table exactly replicates the functionality of *AWARD_ACTORS* table. The only difference is here the writers' awards are taken into consideration and there is no role The primary key in this table is combination of writer id, film id and award id. Again in this table we cannot find any other candidate key and all functional dependency is coming from the primary key, this table is in BCNF.

**SAMPLE DATA**

| w_id | f_id | ai_id | ac_id | year | winner_nominee |
|---|---|---|---|---|---|
| 10 | 5 | 1 | 4 | 2020 | 0 |
| 10 | 6 | 1 | 4 | 2013 | 1 |
| 10 | 10 | 1 | 4 | 2010 | 0 |

### *24. AWARD_DIRECTORS*

This table is again very similar AWARD_WRITERS with only one primary key which is (director id, film id, award id). Since there are no functional dependency without the primary key, This table is in BCNF.

**Sample Data**

| d_id | f_id | ai_id | ac_id | year | winner_nominee |
|---|---|---|---|---|---|
| 1 | 2 | 5 | 1 | 2017 | 1 |
| 4 | 5 | 1 | 3 | 2020 | 0 |
| 4 | 10 | 1 | 3 | 2010 | 0 |

## 25. WATCHLIST

This is a table specifically for storing the Movies which the NUTFLUX Pro users might be interested in watching based on their favorite genre. This table is not populated manually in this design but rather it gets populated using a procedure *PRO_USERS_WATCHLIST* based on the genre chosen as favorite by pro users.

**Sample Data**

| u_id | f_id |
|---|---|
| 2 | 5 |
| 2 | 4 |
| 3 | 2 |

## 26. USER_VIEWS

This table stores all the films that have been watched by both standard and Pro users. Every row is uniquely identified using the primary key (user id, film id). Since all the other attributes are non-candidate key, This table is also in BCNF.

**Sample Data**

| u_id | f_id | liked | rating | comments |
|---|---|---|---|---|
| 1 | 2 | 1 | 7.90 | Too confusing, But one time watch |
| 1 | 5 | 1 | 9.00 | Leanardo Dicaprio is the best actor and should ... |
| 1 | 8 | 1 | 8.90 | Awww.. I love them both together. Must watch... |
| 1 | 11 | 1 | 7.90 | Too confusing, But one time watch |
| 2 | 1 | 1 | 8.90 | Great screen Play, Great story telling |

## 27. USER_GENRE

This table is mainly created for NUTFLUX Pro users. The Pro users are given a special functionality of choose their favorite genre. Each row in

this table is uniquely identified using the primary key (user id, genre id). Since there are no other attributes apart from the primary key, this table is in BCNF.

**Sample Data**

| u_id | g_id |
|---|---|
| 2 | 7 |
| 2 | 2 |
| 2 | 5 |

## 28. USER_REVIEWS

This tables stores the reviews provided by the users to any film. Every review is uniquely identified by (user id, film id) primary key. Since the other attributes are non-candidate and all the functional dependency is initiated from the primary key, this table is in BCNF.

**Sample Data**

| u_id | f_id | rating | comments |
|---|---|---|---|
| 1 | 5 | 9.00 | Leanardo Dicaprio is the best actor and should definitely win an oscar. |
| 1 | 8 | 8.90 | Awww.. I love them both together. Must watch for Brad Pitt Fans |
| 1 | 2 | 7.90 | Too confusing, But one time watch |
| 2 | 1 | 8.90 | Great screen Play, Great story telling |

## 29. CRITICS_REVIEWS

This table stores reviews given by the Critics for all the films in the database. Every review is uniquely identified by (critics id, film id) primary key. Since the other attributes are non-candidate and all the functional dependency is initiated from the primary key, this table is in BCNF.

**Sample Data**

| c_id | f_id | rating | comments |
|---|---|---|---|
| 1 | 1 | 6.60 | It's the sort of movie in which websites concerni... |
| 1 | 4 | 8.20 | They managed to change things up while still sti... |
| 1 | 7 | 7.50 | It may be unfair to say, but its easily admitted t... |
| 1 | 8 | 3.50 | Worst supernatural movie till date having no ba... |

### 30. FILM_RATING

This table stores the rating provided by men and women for each movie and stores the average these two ratings as total rating. This table has only one primary key which is the film id and hence it is in BCNF.

**Sample Data**

| f_id | male_rating | female_rating | total_rating |
|---|---|---|---|
| 1 | 4.40 | 4.60 | 4.50 |
| 2 | 6.60 | 6.80 | 6.70 |
| 3 | 7.00 | 7.20 | 7.10 |
| 4 | 7.10 | 6.90 | 7.00 |

### 31. FILM_VERDICT

This table is used to store verdict of the film whether it is a hit or flop using. This table is again like the above table with only film id as the primary key. Since there are no other candidate keys, BCNF normalization is satisfied.

**Sample Data**

| f_id | verdict_hit |
|---|---|
| 1 | 0 |
| 2 | 1 |
| 3 | 1 |

## 4. Database Views

In SQL, a VIEW is a virtual table that stores data from one or more tables. It does not contain any data and is not physically present in the database. There are four views defined in this database.

### 1. COMBINED_FILM_RATINGS

In this database design, there are three types of ratings provided to every movie in the database. One of the ratings is similar IMDB Rating which is split into male and female rating categories. Second is user ratings. And the

last one is critics rating. Multiple users and critics give their opinion about the movie in the form of ratings which might be positive or negative. This view will store the average of all the user rating as a separate attribute and will store average of all the critics rating for a film as a sperate attribute. So, this view will have film id, film title, IMDB rating, average user rating, average critics rating as its attributes.

QUERY

```
create view combined_film_ratings as
select f.id as filmid, f.title as films, fr.total_rating, avg(uv.rating) as avg_user_rating, avg(cr.rating) as avg_critics_rating
from films f join film_rating fr on f.id = fr.f_id
join user_views uv on uv.f_id = f.id
join critics_reviews cr on cr.f_id = f.id
group  by f.id;
```

Targeted Audience

NUTFLUX Pro users will be the main audience for this view. Pro users might not be just interested about the rating provided by the IMDB as it can be skewed. They might find it fairer to choose the movies which are highly rated by the critics as they are independent bodies who provide ratings considering all factors. So, this view will give a very clear understanding about the rating provided by everyone for the Pro Users.

Example Query

Show me details of all the hit movies in which Critics Rating is greater than the IMDB rating.

```
select cfr.films, cfr.imdb, cfr.avg_user_rating, cfr.avg_critics_rating, f.yearofrelease, fv.verdict_hit
from combined_film_ratings cfr
join films f on f.id = cfr.filmid
join film_verdict fv on cfr.filmid = fv.f_id
where cfr.avg_critics_rating > cfr.imdb and fv.verdict_hit = 1
```

Output

| films | imdb | avg_user_rating | avg_critics_rating | yearofrelease | verdict_hit |
|---|---|---|---|---|---|
| Fast and Furious 6 | 7.00 | 10.000000 | 7.800000 | 2013 | 1 |
| Once Upon A Time in Hollywood | 7.60 | 9.450000 | 8.480000 | 2019 | 1 |
| Mr And Mrs Smith | 6.50 | 8.900000 | 6.620000 | 2005 | 1 |

## 2. AWARDED_ACTORS_RATING

This view is responsible for storing the details of actors who have won one or more awards during their career and add a relationship to get the IMDB rating of the film for which they won the award. This view will try to relate the actor's ability to film direction. For example, an actor might have acted wonderfully but the movie was very bad that it got a very bad rating from IMDB. This view will display the actor information and awards he has won along with the film rating. The attributes of this view are actor id, actor name, film id, film title, role name, IMDB rating, awarding institution and award name.

Query

```
create view awarded_actors_rating as
select a.id as actor_id, a.theatre_name as actor_Name, f.id as film_id, f.title as title,r.name as role, fr.total_rating, ac.name as award, ai.name as award_institution
from films f
join filmcast fc on f.id = fc.f_id
join actors a on fc.a_id = a.id
join roles r on fc.r_id = r.id
join film_rating fr on fr.f_id = f.id
join award_actors aa on aa.a_id = a.id and aa.f_id = f.id
join award_category ac on aa.ac_id = ac.id
join awarding_institution ai on aa.ai_id = ai.id
group by a.id, f.id;
```

Targeted Audience

Both standard users and pro users can be provided with this View. This view is just an information to users which is different from normal tables. Here it is left to interpretation of the users. They may feel the actors is good, but the movie is bad. So, they can get this information from this view. Pro users can be able to write more complex queries like based on the requirement.

Example Query

Show me the actors who have won Choice awards for the film with IMDB rating below 5.0?

```
select *
from awarded_actors_rating
where total_rating <5.0 and award_institution= "Choice";
```

Output

| actor_id | actor_Name | film_id | title | role | total_rating | award | award_institution |
|---|---|---|---|---|---|---|---|
| 15 | Angelina Jolie | 7 | Mr And Mrs Smith | Jane Smith | 4.70 | Choice Movie Actress: Action Adventure/Thriller | Choice |

## 3. VERSATILITY_SCORE

This view provides a versatility score for each actor in this database. The versatility score is generated based on type of film genre an actor has acted in. So, if an actor has acted a film which is of genre romantic and thriller, then the actor's versatility score is 2. Bu this view handles the duplicates and makes sure it gives the score to each actor based on unique genre. The attributes in this view are actor id and versatility score. This view is ordered based on score to show actor with the highest score at the top.

QUERY

```
create view versatility_score as
select  actorID, actorName, count(*) as versatility_score
from (select distinct a.id as actorID, a.theatre_name as actorName, g.name as genre
      from actors a join filmcast fc on a.id = fc.a_id
      join film_genre fg on fg.f_id = fc.f_id
      join genre g on g.id = fg.g_id
group by a.id , g.name) as actor_genre
group by actor_genre.actorID
order by versatility_score desc;
```

Targeted Audience

This view will be suitable for Pro Users as they will be interested in knowing which actor has highest versatility score or some other complex queries. This will provide the users to know more about the actor and explore which all types of movies this actor has acted in.

Example Query

Sort the pair of socially connected pair of actors who have the highest versatility score?

```
select vs1.actorName as actor1, vs2.actorName as actor2, sum(vs1.versatility_score + vs2.versatility_score) as total_score
from versatility_score vs1 join connectionS c on vs1.actorID = c.a1_id
join versatility_score vs2 on vs2.actorID = c.a2_id
where vs1.actorID <> vs2.actorID
group by vs1.actorID,vs2.actorID
order by total_score desc;
```

Output

| actor1 | actor2 | total_score |
|---|---|---|
| Brad Pitt | Angelina Jolie | 11 |
| Ryan Reynolds | Blake Lively | 6 |
| Emily Blunt | John Krasinski | 6 |

## 4. FILMS_AWARDS_NOMINATIONS

This view stores the information about the total awards that are won by actors, writers, and directors of a particular movie. This view will have attributes film id, total awards won by actors, total awards won by directors, total awards won by writers. This information will be useful in determining how good the movie was based on awards won.

QUERY

```
create view films_awards_nomination as
select f.id,ifnull(taa,0) as actor_awards,ifnull(tda,0) as director_awards,ifnull(twa,0) as writer_awards
from films f left join
(select f.id as filmid, f.title as film, ifnull(count(*),0) as taa
from films f
join filmcast fc on f.id= fc.f_id
join award_actors aa on aa.f_id = f.id and aa.a_id = fc.a_id
group by f.id) actors_awards on actors_awards.filmid = f.id
left join
(select f.id as filmid, f.title as film, ifnull(count(*),0) as tda
from films f
join film_directors fd on f.id= fd.f_id
join award_directors ad on ad.d_id = fd.d_id and ad.f_id= f.id
group by f.id) director_awards on actors_awards.filmid = director_awards.filmid
join
(select f.id as filmid, f.title as film, ifnull(count(*),0) as twa
from films f
join film_writers fw on f.id= fw.f_id
left join award_writers aw on aw.w_id = fw.w_id and aw.f_id= f.id
group by f.id) writer_awards on writer_awards.filmid = f.id
group by f.id;
```

Targeted Audience

The information from this view can be shown to both standard users and Pro users. These statistics will be an attractive feature for the audience to choose the movie to watch. For NUTFLUX Pro users, they can write complex query to meet specific needs using this view.

Example Query

Show me the movies in which actors have won at least 1 award and overall, the movie has won at least 2 awards.

Query

```
select f.id, f.title,fan.actor_awards,actor_awards+director_awards+ writer_awards as total_awards
from films_awards_nomination fan join films f on f.id = fan.id
where fan.actor_awards >=1 and actor_awards+director_awards+ writer_awards >2 ;
```

Output

| id | title | actor_awards | total_awards |
|---|---|---|---|
| 4 | Fast and Furious 6 | 1 | 3 |
| 5 | Once Upon A Time in Hollywood | 1 | 3 |
| 7 | Mr And Mrs Smith | 2 | 3 |
| 10 | Inglourious Basterds | 1 | 3 |
| 11 | The Dark Knight | 1 | 3 |

## 5. Procedural Elements

This design consists of procedural extras such as triggers, procedures, and events. Each of them is responsible to depict a specific functionality in real world application of NUTFLUX.

## Triggers

Triggers is a stored program which is automatically invoked on occurrence of a specific event like insert, update or delete. They are responsible to check the integrity of data present in the database. They just provide extended validations of data. Simple validation of data has been done using check constraints in this database. I have defined four triggers in this database design.

### 1. VALIDATE_PASSWORD_STRENGTH

Every user of NUTFLUX will have an account associated with them which requires high level security to prevent leaking of information. The password of the account will be one of the important factors that can help in preventing these unwanted leaks. So, in this design, trigger validates if a password meets the needs of the current password policy in NUTFLUX or not.

The current policy states that the password should be minimum 9 characters long and should have at least 1 or more numeric integer and special characters like '@, %, *, !' . This can be made more complicated based on the needs when implemented in real time.

Query Snippet

```
DELIMITER //
CREATE TRIGGER  validate_password_strength
BEFORE INSERT ON users FOR EACH ROW
Begin
    IF  New.pwd  regexp '.*[0-9]+.*' = 0  or char_length(New.pwd)<=(8) or new.pwd regexp '.*[@|%|*|!]+.*' = 0 THEN
        SIGNAL SQLSTATE VALUE '10002'
        SET MESSAGE_TEXT = "Pwd size should be greater than 8 and should have numbers and should have 1 or more special characters";
    end if;
end //
```

As you can see in the above query, The event is before insert for this trigger. So, whenever a new user creates an account and enters the information, If the password does not meet the policy standards, then it will send out an error with the above MESSAGE_TEXT. This is done using regex pattern matching in MySQL. This verification can be even handled

at the front end, but in case if it is not handled, it will still not allow the user to create the account at the backend by throwing this error.

### 2. USER_AGE_VERIFICATION_INSERT

Whenever a user tries to create an account, He needs to add his date of birth. This is to verify if the user is not a minor or if it's a fake user. So, this trigger helps in validating this by calculating the age from the date of birth and verifies accordingly. If the age is unrealistic, it will throw an error stating the same. In this case, for minor users, the trigger will throw an error. But when building the application, it will be wise to create a KIDS SECTION (Which will not have A-Rated Movies) instead of not allowing them to create an account.

Query Snippet

```
DELIMITER //
CREATE TRIGGER  user_age_verification_insert
BEFORE INSERT ON users FOR EACH ROW
Begin
    declare age int;
    set age =  DATE_FORMAT(FROM_DAYS(DATEDIFF(now(),New.dob)), '%Y')+0 ;
    IF Age <18 THEN
        SIGNAL SQLSTATE VALUE '20000'
        SET MESSAGE_TEXT = "[table:users] -Age is below 18";
    else if Age >150 then
        SIGNAL SQLSTATE VALUE '20000'
        SET MESSAGE_TEXT = "[table:users] -Age is Unrealistic";
    end if;
    end if;
end //
```

As you can see in the above query, The event is before insert for this trigger. So, whenever a new user adds his date of birth, If the age is less than 18, or greater than 150, then the trigger will not allow the user to create the account.

### 3. FILM_VERDICT_INSERT

This trigger is responsible to define the verdict of the film (Hit/Flop) based on the Revenue generated by the film. A film is considered as hit if the total cost of making the film and marketing the film (production rights +marketing rights) is less than the Gross collection of the movie. So whenever a new film is inserted, This trigger gets activated and adds the verdict of the movie into the FILM_VERDICT table.

Query Snippet

```
DELIMITER //
CREATE TRIGGER  film_verdict_insert
    after INSERT
    ON films
    FOR EACH ROW
Begin
    IF NEW.GrossCollection < (NEW.production_rights + New.marketting_rights) THEN
        insert into film_verdict values (NEW.id, false);
    else
        insert into film_verdict values (NEW.id, true);
    END IF;
end //
```

As you can see in the above query, The event is after insert for this trigger. So, whenever a new film gets added to the database, after inserting to the films table, this trigger gets activated and populates the FILM_VERDICT table with right data.

### 4. IMDB_RATING_VERIFICATION_INSERT

This trigger is responsible to validate the ratings before inserting into the FILM_RATING table. Each new entry into this table consists of average rating for the film provided by males and average ratings by female. It also consists of a total average. So it is necessary for the total ratings to be

the average of both males and females and this verification is done by this trigger.

Query Snippet

```
DELIMITER //
CREATE TRIGGER  imdb_rating_verification_insert
    BEFORE INSERT
    ON film_rating
    FOR EACH ROW
Begin
    IF NEW.total_rating <> (NEW.male_rating + New.Female_rating) /2 THEN
        SIGNAL SQLSTATE VALUE '10580'
            SET MESSAGE_TEXT = "[table:film_rating] -Total Rating Does Not match";

    END IF;
end //
```

As you can see in the above query, The triggers validates if the average rating is equal, and if it's not it throws an error stating 'Total Rating Does Not Match".

## Procedures

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

1. **PASSWORD_RECOVERY_USER**

   This is a procedure which helps in allowing the users to change their password if they have forgotten the old password. This is a basic functionality which will be provided to all the users of NUTFLUX. Every user will be asked to choose a security question while signing up to NUTFLUX. So, if a user forgets the password, then they change it if they answer the security question correctly. So, this entire process will be done by this procedure.

Query Snippet

```
delimiter //
create procedure password_recovery_user(in username  text, in emailid text, in answer text,in new_pwd text)

begin
    declare question text ;
    if (exists (select *  from users where (name = username or EmailID=emailid) and security_ans = answer)) then
       update users set pwd = new_pwd where (name = username or EmailID=emailid);
    end if ;
end //
```

In the above Query, after validating the user response to the security question, it will validate and if it matches and then updates the users table with the new password.

### 2. PRO_USERS_WATCHLIST

This is a procedure mainly for NUTLUX PRO Users. Here the Users get the privilege to maintain their own watchlist. In our database design, there is *watchlist* table which stores the films which the user might be interested in watching based on the genres that they have chosen as their favorite in *user_genre* table. The main purpose of this table is to provide premium service for the Pro users. The specialty of this procedure is that the users need not take the hurdle of searching the movies and adding it to watchlist manually. This procedure takes the *from_year* as an input which basically ask the procedure to return all the movies that are released after this year based on user's favorite genre. We can extend this procedure to satisfy other conditions like people can have their favorite actors/directors /writers or production company and based on this, if any new movies are launched, they can be populated into watchlist table.

Query Snippet

```
delimiter //
create procedure pro_users_watchlist(in user_id  int, in from_year int)
begin
    begin
     DECLARE finished INTEGER DEFAULT 0;
     DECLARE filmid int;
    if (exists (select * from users where id = user_id and subscriptionType = "Pro" and sub_end_date > current_date())) then
      begin
        DECLARE wishlist_film cursor for
        select distinct f.id
        from films f join film_genre fg on f.id = fg.f_id
        where fg.g_id in (select ug.g_id  from user_genre ug where ug.u_id = user_id) and f.id not in (select uv.f_id from user_views uv where uv.u_id = user_id) and f.YearOfRelease > from_year
        order by f.YearOfRelease desc;

        DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;

        open wishlist_film;
    getFilm : LOOP
            fetch wishlist_film into filmid;
            IF finished = 1 THEN LEAVE getFilm;
            END IF;

            -- insert into watchlist table if the film does not already exists in the table
            if filmid not in (select f_id from watchlist where u_id= user_id) then
            insert into watchlist values (user_id, filmid);
            end if;
    END LOOP getFilm;
    CLOSE wishlist_film;
       end;
     end if;
    end;
end //
```

## 3. SORT_MOVIE_RATINGS

This is a procedure mainly for NUTLUX PRO Users. The Pro users can sort the movies that are released based on either imdb rating or critics rating. Here the procedure takes three parameters: start_year, end_year, and sort_on. So, all the films that were released in this period will be sorted based on condition (imdb rating / critic rating).

Query Snippet

```
delimiter //
create procedure sort_movie_ratings(in start_year  int, in end_year int, in sort_on text)

begin
    if sort_on = "imdb_rating" then
        select f.id, f.title, f.yearofrelease, cfr.imdb
        from films f
        join combined_film_ratings cfr
        on f.id = cfr.filmid
        where  f.YearOfRelease >start_year and f.YearOfRelease<end_year
        order by cfr.IMDB desc;
    elseif sort_on = "critics_rating" then
        select f.id, f.title, f.yearofrelease, cfr.avg_critics_rating
        from films f
        join combined_film_ratings cfr
        on f.id = cfr.filmid
        where f.YearOfRelease >start_year and f.YearOfRelease<end_year
        order by  cfr.avg_critics_rating desc;
    else
        SIGNAL SQLSTATE VALUE '10581'
        SET MESSAGE_TEXT = "Incorrect Sorting type : Choose either imdb_rating / critics_rating";
    end if;
end //
delimiter ;
```

## 4. EXTEND_SUBSCRIPTION

This procedure is responsible for extending the subscription of the pro users. In this database design, all pro users have a data field called as *auto_extend_subscription* which is a Boolean value. So, if this is set as True, then when the Pro users subscription ends, This procedure will take the responsibility of renewing it. In real time, this procedure can be manually triggered by the Pro user as well by doing the payment before the expiring date.

Query Snippet

```
delimiter //
create procedure extend_subscription()

begin
	DECLARE finished INTEGER DEFAULT 0;
	DECLARE userId int;
	DECLARE pro_user_id cursor for
	    select id  from users where subscriptionType = "Pro" and sub_end_date = current_date() and auto_extend_subscription = true;
	DECLARE CONTINUE HANDLER
	FOR NOT FOUND SET finished = 1;

	open pro_user_id;
	getID : LOOP
		fetch pro_user_id into userId;
		IF finished = 1 THEN
		LEAVE getID;
		END IF;
		update users set sub_end_date = date_add(sub_end_date, interval 29 day) where id = userId;
	END LOOP getID;
	CLOSE pro_user_id;
end //
```

This procedure is being called up every day using an event called *Subscription_Extension.* So basically, on a daily basis, every pro users subscription_end_date is compared with the current day and if it matches, then it adds 28 days to the current subscription_end_date. Hence renewing the subscription.

Query Snippet

```
CREATE EVENT if not exists Subscription_Extension
  ON SCHEDULE EVERY 1 day
STARTS CURRENT_TIMESTAMP

DO
    call extend_subscription;
```

## 6. Example Queries: Your Database In Action

### Query 1

Give Movie recommendations to users based previously liked films having the same actors.

This Query can be used to display the recommendations for both Standard users as well as Pro users. Whenever a user has watched a film, It will be recorded in USER_VIEWS and if the user has liked a particular movie then it will be stored accordingly in the above table. This information can be very useful in determining what are the kind of movies the user may like and hence can be used for providing recommendation. In this query we have taken the actors who were in the liked film, but this can be extended to director, writers, or production company etc. But the base query format will be the same.

Query Snippet

```
select f.title, f.yearofrelease, f.plot, f.taglines, f.grosscollection from films f join filmcast fc on f.id = fc.f_id
where fc.a_id in
    (select a_id
    from user_views uv
    join filmcast fc on uv.f_id = fc.f_id
    where u_id = 1 and uv.liked = 1)
    and
    f.id not in
    (select f_id
     from user_views
     where u_id = 1);
```

In the above query we chose the user with id number 1 and we made sure that this user has previously watched these movie using the USER_VIEWS table.

Output

| title | yearofrelease | plot | taglines | grosscollection |
|---|---|---|---|---|
| Django Unchained | 2012 | With the help of a German bounty-hunter, a fre... | Life, liberty and the pursuit of vengeance. | 426074373.000000000 |
| Mr And Mrs Smith | 2005 | A bored married couple is surprised to learn tha... | Assasins Game of Life | 487287646.000000000 |
| Inglourious Basterds | 2009 | In Nazi-occupied France during World War II, a ... | If You Need Heroes, Send In The Basterds | 70000000.000000000 |

### Query 2

Give insight about the movie ("Once Upon A time in Hollywood") with

information about the previous hit of director.

This will be very useful for standard users as they can get a better insight about the movie and can choose the movie accordingly. This can again be extended to previous hit of actors, writers, or production team. This information is valuable for new users who do not have any idea about films.

Query Snippet

```
select distinct title as previous_hit, d.name as director, f.YearOfRelease as year
from film_verdict fv
join films f
on fv.f_id = f.id
join film_directors fd
on fv.f_id = fd.f_id
join directors d
on fd.d_id = d.id
where fd.d_id in
                (select d_id
                 from films join film_directors on id = f_id
                 where title ="Once Upon A time in Hollywood")
                 and fv.verdict_hit = 1 and f.title <> "Once Upon A time in Hollywood";
```

Output

| previous_hit | director | year |
|---|---|---|
| Django Unchained | Quentin Tarantino | 2012 |
| Inglourious Basterds | Quentin Tarantino | 2009 |

**Query 3 (Based on the procedures that are explained before)**

Create a watchlist for NUTFLUX Pro user (User id = 2) which involves films released after 2000.

This query is specially for pro users who want to generate a watchlist based on their favorite genre. The working of the procedure is explain in the previous heading.

Query Snippet

```
call pro_users_watchlist(2,2000);
select *
from watchlist w join films f on w.f_id  = f.id
where u_id = 2;
```

Output

| u_id | f_id | id | Title | YearOfRelease | runtime | plot | taglines | GrossCollection | production_rights | marketting_rights |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 2 | Jungle | 2017 | 145 | In the pursuit of self-discovery and authentic e... | Nature Has Only One Law - Survival. | 1906640.000000000 | 500000 | 400000 |
| 2 | 8 | 8 | Green Lantern | 2011 | 114 | Reckless test pilot Hal Jordan is granted an alie... | One of us... becomes one of them. | 219851172.000000000 | 280000000 | 200000000 |
| 2 | 11 | 11 | The Dark Knight | 2008 | 152 | When the menace known as the Joker wreaks h... | Why So serious | 185000000.000000000 | 25000000 | 20000000 |

## Query 4(Based on the procedures that are explained before)

Sort the Movies based on average critics rating that were released between 2000 and 2020.

This query is specially for pro users if they want to choose the movie based on critics rating instead of normal IMDB rating, then this procedure will be useful to fulfill this requirement.

Query Snippet

```
call sort_movie_ratings(2000,2020,"critics_rating");
```

Output

| id | title | yearofrelease | avg_critics_rating |
|---|---|---|---|
| 5 | Once Upon A Time in Hollywood | 2019 | 8.480000 |
| 4 | Fast and Furious 6 | 2013 | 7.800000 |
| 9 | Quiet Place | 2018 | 6.700000 |
| 7 | Mr And Mrs Smith | 2005 | 6.620000 |

## Query 5

Show the movies in which the villain has won an Oscar for the role played in the movie.

This Query is targeted for Pro users who are very much picky before watching the movies. Query which are of this type can be implemented easily using this database design. This is a sample query which proves that this design can handle any query related to awards and movie cast.  This query identifies the role type which is 'villian' and checks if the actor who played this role has won an Oscar or

not and then displays the information about the film and role played by the actor.

Query Snippet

```
select  a.theatre_name as actor, r.name as role, f.title as film, ai.name as awarding_institution,ac.name as award, aa.year as year
from films f join award_actors aa on aa.f_id = f.id
join award_category ac on aa.ac_id = ac.id
join actors a on aa.a_id = a.id
join filmcast fc on aa.a_id = fc.a_id and fc.f_id = aa.f_id
join roles r on r.id = fc.r_id
join awarding_institution ai on aa.ai_id = ai.id
join role_category rc on r.id = rc.r_id
join role_type rt on rc.type_id = rt.id
where ai.name = "Oscar" and rt.name = "villian";
```

Output

| actor | role | film | awarding_institution | award | year |
|---|---|---|---|---|---|
| Heath Ledger | Joker | The Dark Knight | Oscar | Best Performance by an Actor in a Supporting Role | 2009 |
| Christoph Waltz | Col. Hans landa | Inglourious Basterds | Oscar | Best Performance by an Actor in a Supporting Role | 2010 |

Query 6

What is the name of the actors to win two Oscars for the same role?

This query is specifically for Pro users who are interested in doing some research regarding the award-winning actors. These complex queries got simplified because of the availability of the view *awarded_actors_rating*. Any more complex query related to Awards and cast can be easily queried using this database design.

Query Snippet

```
select a.theatre_name as actor, r.name as role, f.title as film, f.yearofrelease
from films f join filmcast fc on f.id = fc.f_id
join actors a on a.id = fc.a_id
join roles r on r.id = fc.r_id where r.name in
(select role
from awarded_actors_rating aar
where award_institution = "Oscar"
group by aar.role
having count(*) =2);
```

Output

| actor | role | film | yearofrelease |
|---|---|---|---|
| Marlon Brando | Don Vito Corleone | The Godfather 1972 | 1972 |
| Robert De Niro | Don Vito Corleone | The Godfather 1975 | 1975 |

## 7. Conclusions

Overall, this report has provided a detailed explanation about the design and how it can be used to satisfy the needs of the standard and Pro users. Even though this table consists of large number of tables to satisfy various kinds of data like directors, actors, writers, awards etc., this has just touched the tip of the iceberg. Data related to films is a huge pool of data and it needs to be added accordingly to this design. We need to create tables to satisfy these information and query them accordingly. But this design can act as base for future improvements.

In this design, lost of functionalities of standard and pro users has be explained with proper queries and their outputs. But this is just to show that design is robust to handle these kinds of complex queries. This design provides some views which are meaningful and add quality information for the users. There will be lot more complex queries coming into picture once users start using the application. We need to build upon this design carefully by keeping it normalized and preventing storage of unwanted duplicate data.

There are some important functionalities that might be missing in this design like sending notifications to the users regarding new movies, or giving the list of trending movies, These are not fulfilled because MySQL does not provide a functionality send notification and send output text. This notification assignment needs to be done by developer from the user interface. This design can provide

the right data that is needed for the users.

Overall there is always room for improvement in terms of database designing but these will be coming into picture when this application goes into development.

## Acknowledgements

I wish to acknowledge the guidance that was provided by my module coordinator Prof. Tony Veale who supported me through the SQL Programming sessions and provided invaluable feedback at every point. I would also like to thank my module assistants who helped me throughout the practical sessions and suggested ways to improve my knowledge on the subject. In addition to this, I gained invaluable insights from the IMDb interfaces and datasets which helped me in contemplating various aspects of this project through its vast library content.

## References

1. https://www.imdb.com/ - For populating the tables in this design with the right data
2. https://opentextbc.ca/dbdesign01/ - Adrienne Watt - Database Design (2nd Edition) - October 24, 2014
3. https://tutorialspoint.dev/language/sql/ - Quick refresher on Query syntax in MySQL.
4. https://www.pdfdrive.com/sql-books.html - Effective SQL by John L. Viescas , Douglas J. Steele, Ben G. Clothier – 2017
5. Bright Space material regarding SQL from Tony Veale