Database Design for a Movie and TV Show Streaming Platform (NUTFLUX)

Sai Srujan G S
sai.srujan889@gmail.com

Table of Contents

List of Figures

1. Introduction

The database is the most integral part of the application. The project is intended to design a database system using the MySQL framework which caters to support the working of a movie and tv show streaming platform (Nutflux). The database forms the backend of the streaming platform which is the server side and the movie/tv show streaming platform forms the frontend which is the client side. The streaming platform communicates with the server to modify and retrieve the data which is present in the database tables with the help of transactions and queries.

The database of the streaming platform consists of the tables which hold data ranging from various Movies and TV Shows to information about the various users that make use of the platform along with the ratings they have provided to the various tv shows or movies. This user information would be used to make recommendations and provide relevant data to the users, which helps enhance the user experience.

The movies/tv shows data consists of a large variety of information. Every movie/tv show is mapped to a NSAN (Nutflux Standard Audio and Visual Number) that can be used for the identification of each movie/tv show and retrieve the entire information which is spread across multiple tables in the database. The data includes information such as the title, year of release, synopsis/summary of the plot, runtime of the whole movie, runtime of the episode in case of the tv show, production company and language. It also consists of the various genres a movie/tv show belongs to and there can be multiple genres to which each of them belongs. The data contains information about the various awards won by the various movies/tv shows and includes details about the type of award, the category of the award, the year award was presented and also the person who won the award for a particular role. It also includes the various nominations for an award. The popular quotes from various movies/tv shows are also included in the data.

The data also includes the various casting information of the actors/actresses including all the main and the guest roles. It includes the various roles played by actors/actresses specifying the nature of the role. The nature of the role is the various categories a role belongs to such as hero, superhero, villain, detective, spy and so on. The crew of the movie/tv show are as important as the cast. The data also consists of the various crew members and their role such as director, editor, writer, singer and so on. The data consists of the personal details of each person in the cast and the crew such as the name, nationality and the biggest hits in their career. The relationships between the various actor and actress in real life are of particular interest for many people. The data also consists of this information which specifies the type of the relationship and the year of the relationship.

As described a movie/TV show consists of a lot of data. There are many movies and TV shows that are available to the users and every movie/TV show has a huge number of cast and crew members working in it. So, the volume or the scale of the data would be very high.

The user information consists of information such as name, nationality, gender and a unique user ID that is associated with each of the user. This user ID can be used as the key to get all the information associated with that user. The data also includes the ratings given by the various users to the movies/tv shows. This data can be very huge as the number of users across an online streaming platform can be high.

The online streaming platform (Nutflux) primarily consists of two types of users, the Standard and Professional (Power) users.

The standard users are one type of users who will be interacting with the streaming application and browsing through the various movies or tv shows that the streaming platform contains. The standard users decide to watch a particular movie or tv show based on the information that is provided to them. The various details that the user would be interested would include the information of the various actors/actresses that are involved in the movie/tv show, crew members information, awards won by the movie or tv show, various roles, genres, rating for a movie/tv show and so on. As, the database consists of this various information spread across the tables, database provides a number of views with consolidated and diverse information that could be used by the streaming platform. The streaming platform can make use of the information that is provided with these views and display it to the standard users in the required format. The database also consists of some procedures that provides some recommendation based on the ratings given by the user for various movies/tv shows. The user id will be used by the stored procedure to gain the required details and provide the recommendations to the user.

The Power users of the streaming application are assumed to be having knowledge about SQL querying. The streaming platform provides a particular shell where the power users can query for detailed and specific information. The database provides various tables, views and procedures that the power users can make use of to gain the required information. The tables contain all the required data in a consistent and normalized format that would make it easier for the user to query and get the required results. The various categories of data are split up into the required tables and are assigned a unique id where required (Ex: nsan for a movie/tv show, id for a cast or crew member and so on). Using these unique ids, all the required information about a particular record can be easily retrieved. This makes it easier for the power users to query by joining the various tables and mapping the various ids to get the complete record of information.

The database plays a very important role in supporting the online streaming platform (nutflux) as it stores the entire information required by the streaming platform and hence becomes the backbone of the streaming platform. It provides fast access to the required data and a range of functionalities that can be used to provide the required information. The database is not the streaming application entirely but rather supports the functioning of the streaming application to take place with the required functionalities smoothly. The various procedures, triggers, views and tables would cater to the needs of the streaming application and enhance the user experience.

2. Database Plan: A Schematic View

The database consists of multiple tables with the information of the movies/tv shows spread across the various tables. Each of these tables are in relation to each other. The database also consists of tables related to the various users of the platform which when combined provides the required functionality of the streaming platform (Nutflux).

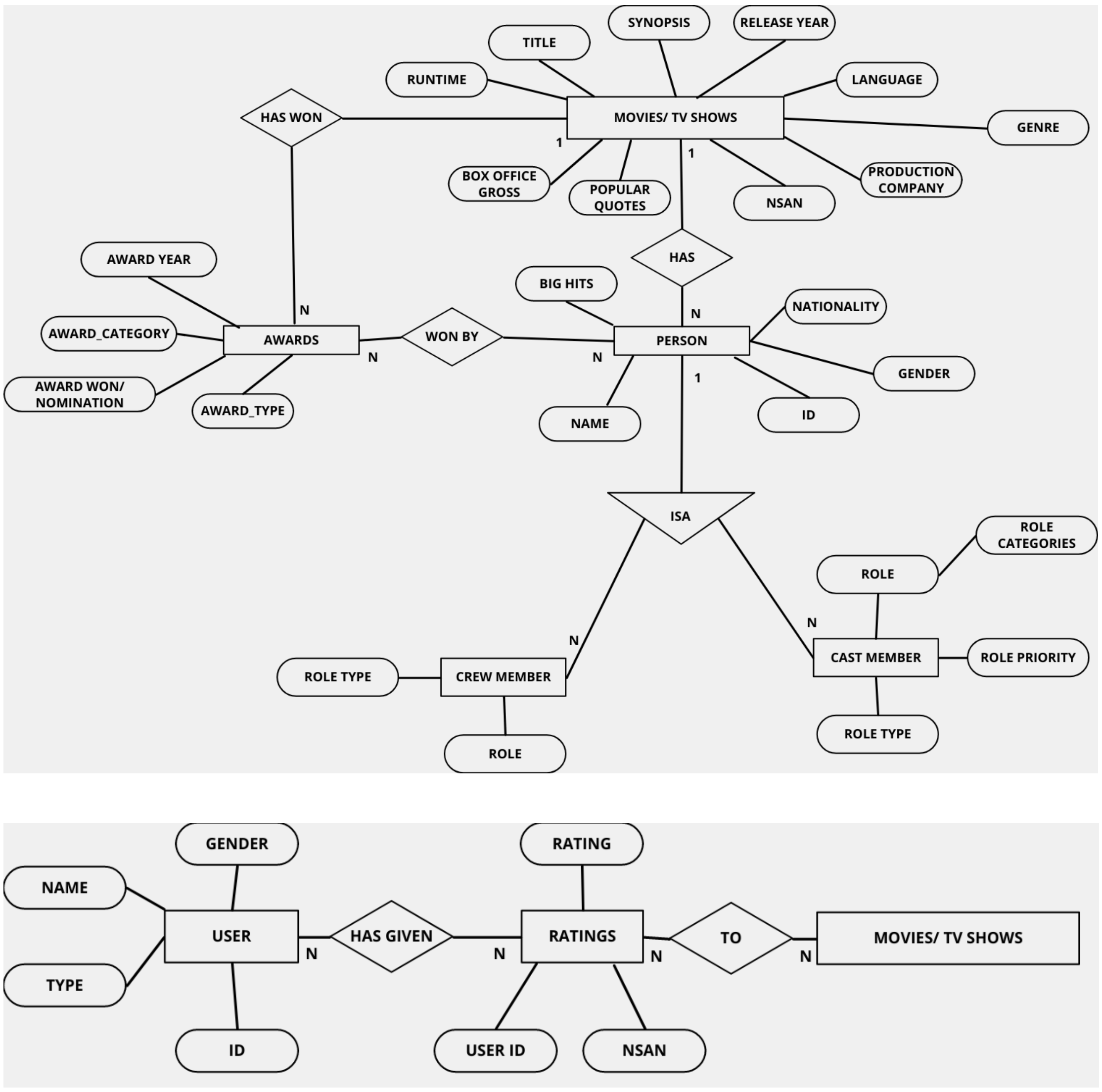


Figure 1: ER Diagram of the database (High Level Schema)

The ER diagram of the database gives a high-level picture of the type of data and relationship between the various entities. A movie/tv show can have multiple genres associated with it, so a genre and genre_map table is created. Also, there can be multiple crew and cast members working in a movie/tv show and hence cast_map and crew_map tables along with person_info is created. Similarly, awards can be given to multiple people and a movie/tv show can have multiple awards and hence the awards, awards_map, awards_info and award_person_map tables are created. There can be multiple roles and each role can be associated with many categories, so the role_info, role_category and role_map tables are created. Similary, where required the information is separated and stored in relevant tables to ensure there is no redundancy and is organized and stored for easy access.

The movies_shows table is an integral part of the database. It consists of all the movies/tv shows that are present in the database. It consists of the title, release_year and the type columns. Each record in this table is identified by a unique id (nsan). The nsan column is of type varchar and is the primary key of the movies_shows table as it uniquely identifies each movie/tv show. The nsan is assigned to each movie/tv show as there can be two movies with the same title and same release year or a movie and tv show with the same title released in the same year. The release_year column is of type year as it consists of the year details when a movie/tv show was released and type column will have the values 'Movie' or 'TV' as the database consists of movies/tv shows information.

The movies_shows_info table consists of the attributes such as bo_gross(Box Office gross), language, synopsis, runtime, production_company and the plot_type. Synopsis is of varchar type as it consists of summary of the movie/tv show, runtime is smallint type as it has information about the total runtime of a movie and in case of tv show it would be the runtime of an episode (since all episodes will have same runtime) and the value will not be more than 250 minutes or less than 20 minutes in both the cases. The plot_type field is of varchar type as it specifies whether a given movie/ tv show is an original script(default value) or if it is a remake of some other movie/tv show or if it is a sequel of a particular series or if it is based on some book. production_company consists of the main Production Company presenting the film and is of varchar type. language is also varchar type as it consists of textual data and bo_gross is of type integer as it consists of the worldwide box office gross collection. The nsan column is the primary key of the table and it maps the various movies/tv shows to the additional information that is provided in this table and hence is also defined as the foreign key.

The genre table consists of the genre_id and genre_name attributes and both are varchar type. The genre_name column contains the list of genres that a movie/ tv show can belong to and has a unique constraint to ensure that the same genre is not added twice in the table. Each of this genre is identified using a genre_id which is the primary key of the table. This ensures that when a genre is added it is consistent across the tables which makes it easy to identify and query the required results.

The genre_map table consists of the nsan and genre_id columns that are of both varchar type. The nsan and genre_id columns are together defined as the primary key for this table as a single movie/ tv show can belong to multiple genres and hence there will be multiple entries with the same nsan. The nsan column derives its values from the movies_shows table and the genre_id column derives its values from the genre table and hence both are defined as foreign key. This ensures that the mapping can be done to only the movies/tv shows and the genres that are valid and stored in the database.

The person_info table consists of the name, nationality and gender columns which are of varchar type. The name column is used to store the names of the various cast and crew members that are involved in the various movies/tv shows and the nationality and gender of that person would be stored in the other two columns. Each person has a unique id associated with them and this id is stored in the person_id column which is of varchar type and is the primary key of the table.

The person_big_hit table consists of the person_id and big_hit columns. The big_hit column is used to store the various big hits or classic films/tv shows in that person's career. The person_id and big_hit columns are together a combined primary key as there can be multiple big hits or classics for a particular person and person_id is also defined as a foreign key to ensure the data is mapped for a valid person.

The role_info table consists of role_id and the role columns. The role column is of varchar type as it stores all the various character names. Each of this character role is associated with a role_id which is the primary key of the table as there can be same character names in different movies/tv shows but the nature of the role could be different in each case. So, mapping the id makes it easier to identify various character roles and ensure the role information is consistent across various tables.

Each of the roles also belong to some categories which defines the nature of the role. The role_category table consists of the category_id and category columns. The category column consists of the list of all the various categories that a role can belong to and the category_id column which is of type varchar is the primary key of the table which is used to identify the various categories with a unique id.

The role_map table consists of the role_id and the category_id columns. The role_id and category_id columns are together defined as the primary key for this table. This is adopted as a single role can consist of multiple categories and hence the same role_id can be used against more than one category_id. The role_id column refers to the role_info table to derive its values to ensure that only a valid role is recorded. The category_id column refers to the column present in the role_category table to ensure that a valid category is being mapped and hence both are defined as foreign key.

The cast_map table consists of the columns nsan, person_id, role_id, role_type and role_priority. nsan, person_id, role_id is a combined primary key as a movie/tv show can have multiple people acting in it and one actor/actress can have multiple roles in a single movie/tv show. role_type is of type varchar and is used to store information about the type of the role (Ex: Male lead, Guest, Female lead, Support, etc.), role_priority is of type smallint as it would consists of numbers starting from 1 and it determines the importance of the role in the cast which can be used to display in that order, also if there are some guest roles that are important for the movie/ tv show it can be prioritized accordingly based on this priority value. The role_id column refers to the role_info table to derive its values to ensure that only a valid role is recorded, the nsan column refers to the column present in the movies_shows table to ensure that a valid movie/tv show is being mapped, the person_id column refers to the column present in the person_info table to ensure that a valid actor/actress is being mapped and hence all these three are defined as foreign key.

The crew_map table consists of the columns nsan, person_id, crew_role, and crew_type. nsan, person_id, crew_role is a combined primary key as a movie/tv show can have multiple people working as a part of the crew and one person can have multiple roles as a crew member in a single movie/tv show (Ex: Same person can be both writer and director for that movie/tv show). crew_role is of type varchar and is used to store information about the type of the role (Ex: Director, Editor, Singer etc.), crew_type is of type varchar as it would consists of textual data determining the type of crew_role(Ex: If the director is the main director or guest director in that particular tv show). The nsan column refers to the column present in the movies_shows table to ensure that a valid movie/tv show is being mapped and the person_id column refers to the column present in the person_info table to ensure that a valid person is being mapped and hence both are defined as foreign key.

The connections table consists of the columns actor1_id, actor2_id which is of type varchar and refers to the person_id column from the person_info table to ensure that a valid actor/actress is mapped and hence both are defined as foreign key. It also consists of the relationship column of type varchar which denotes the type of relationship between the two actors such as Marriage, Dating, Divorce, etc and also consists of the relationship_year column which is of type year specifying the year in which the relationship started. actor1_id, actor2_id, relationship is a combined primary key for this table as there can be multiple relationships existing between the same two actors (Ex: Marriage and Divorce), so there can be multiple entries with same two pairs of actors.

The awards table consists of the columns award_id and award_type which is of type varchar. The award_type consists of the list of the various types of awards (Ex: Oscar, European Film awards, etc). Each of this award is associated with a unique id which is stored in the award_id column of this table and this column is the primary key of the table.

The awards_map table consists of the columns award_id and nsan to map the various awards to the movie/tv show and hence the nsan column refers to the column present in the movies_shows table to ensure that a valid movie/tv show is being mapped and the award_id column refers to the column present in the awards table to ensure that a valid award is being mapped and hence both are defined as foreign key. The award_description column gives information about the award such as Best Movie, Best TV Series, Best Male Lead and so on. Each of these records are mapped with an awards_map_id that can be used in other tables to map additional information and is the primary key of this table.

The awards_info table consists of the award_won column, if it has the value 'Yes' then the award was won else it can be considered that the award was not won but the movie/tv show was nominated for that particular category of the award. The award_year column stores the year when that movie/tv show was nominated for that particular category of award. The awards_map_id and award_year together is the primary key for this table as a tv show can be nominated for the same award in a different year. Ex: The tv show Friends was nominated for American Comedy Awards, Best Comedy TV series every year it was telecasted.

The award_person_map consists of the awards_map_id, award_year and person_id columns. The awards_map_id, award_year columns together is defined as the foreign key to the awards_info table as these two are the combined primary key for the awards_info table, person_id column refers to the person_info table to ensure that the valid person is being mapped and is also defined as a foreign key. These three columns are combined primary key for the table as a person can be associated with many awards and there can be multiple people for the same award.

Tables like genre, role_category, role_info, person_info and awards are used to store the various details in separate table and ids are used to map in the corresponding tables to ensure that there is no inconsistency while mapping in the various tables and also any update can be done in one place. It also makes it easier to check the data available in the database for these respective categories and provide the results quickly. It also provides the flexibility to map the same information to various records by just making use of ids and not having to rewrite the whole value in case the value is big.

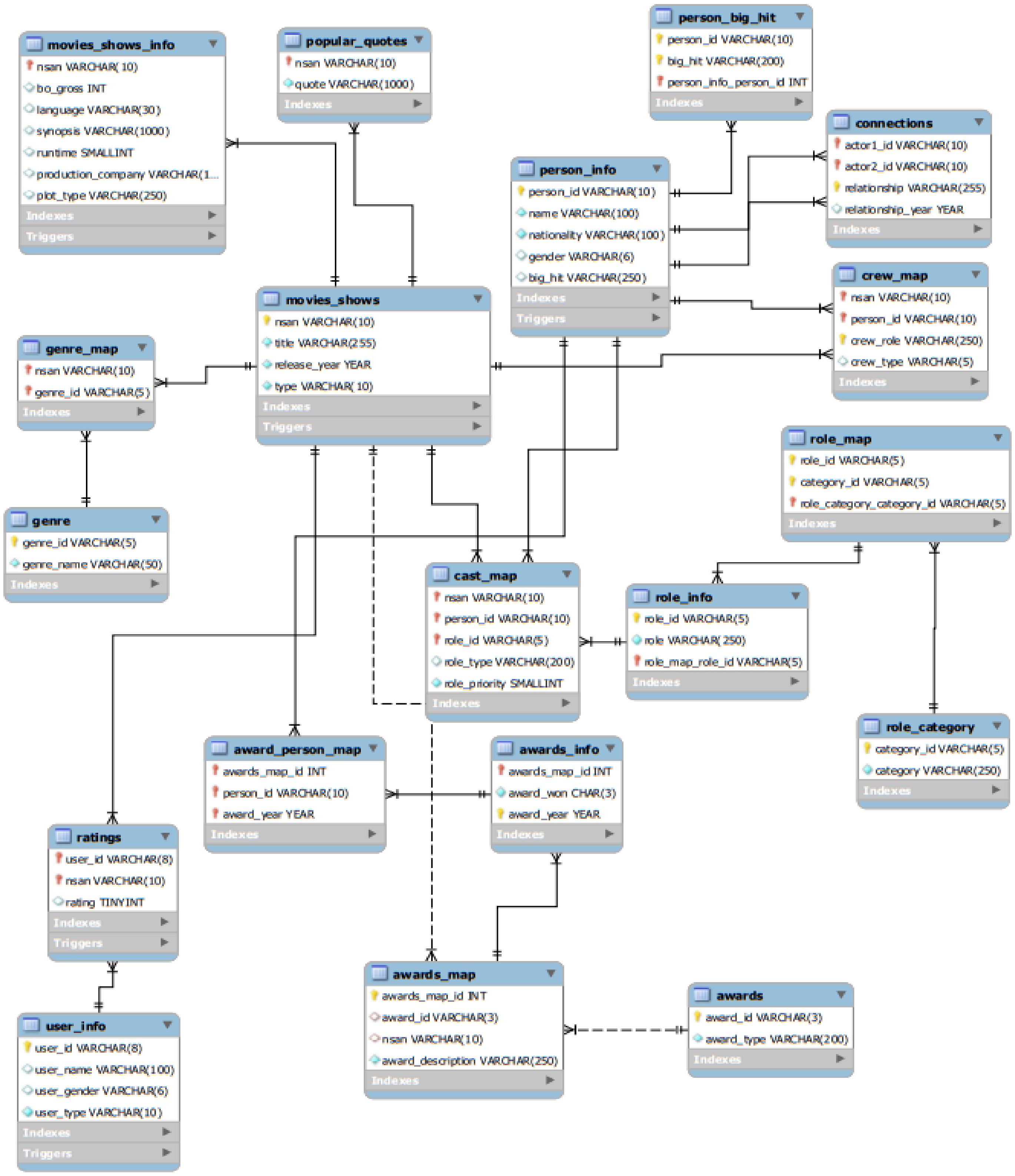


Figure 2: ER Table Diagram of the database

The user_info table consists of the user_name, user_gender and user_type columns which are of all varchar type. The user_name column is used to store the name of the user, the user_gender column is used to store the gender of the user and the user_type column specifies whether the user is a Standard or a Professional(Power) user which can be used to provide the required privileges when accessing the application. Each user is identified by their own id which is stored in the user_id column and is the primary key of the table.

The ratings table consists of the user_id, nsan and rating columns. The nsan column refers to the column present in the movies_shows table to ensure that a valid movie/tv show is being mapped and the user_id column refers to the column present in the user_info table to ensure that the user is present in the database and hence both are defined as foreign key. The rating column is of type tinyint as it consists of rating given by a user to a particular movie/tv show and the valid values would be between 1-10. The user_id and nsan columns together are defined as the primary key for the table as a single user can give ratings to multiple movies/tv show but he can only give rating to a particular movie/tv show only once.

3. Database Structure: A Normalized View

Normalization is the sequence of steps by which a relational database model is both created and improved upon. The sequence of steps involved in the normalization process is called Normal Forms. (Powell, 2006, p.73).

A table is said to be in the First normal form(1NF) if a primary key is present for that table and there are no redundant, repeating groups of data.

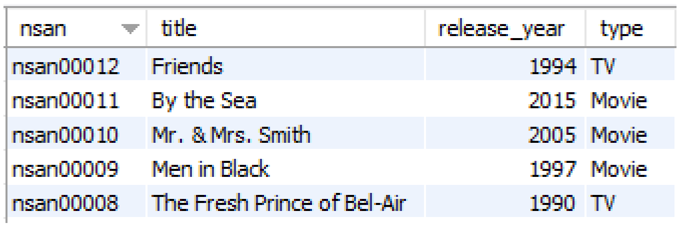
If the table is in 1NF and all the attributes are dependent on the whole primary key without any partial dependencies among the non-candidate keys, then the table is said to be in the second normal form (2NF).

If the table is in 2NF and every field is only dependent on the whole primary key, then the table is said to be in the Third Normal Form (3NF).

If the table is in 3NF and for every functional dependency, there exists a super-key without any non-candidate key dependencies or no determinants overlap then the table is said to be in Boyce Codd Normal Form. So, if a table is in Boyce Codd Normal Form, it would ensure that the table complies to first, second and third normal forms.

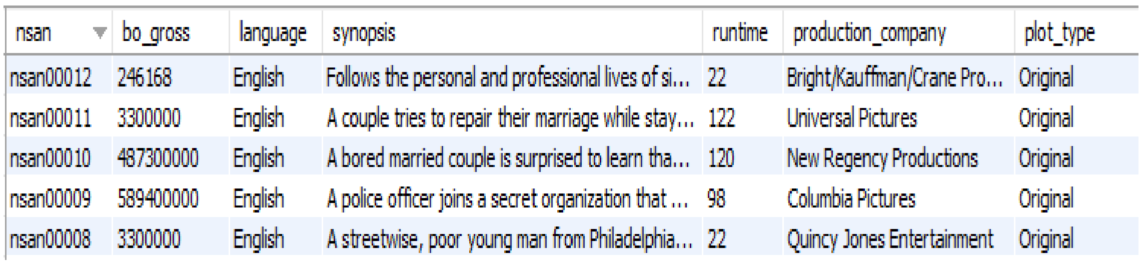
The movies_shows table consists of all the various movies/tv shows that are present in the database. Each of the movie/tv show has unique id(nsan) which is used to map the information regarding the movie/tv show across various tables. The table has a primary key which is the nsan column of this table. The title, release_year and type columns are dependent only on the nsan value of the record with no partial dependencies among the non-candidate keys and nsan is the only determinant for a record in the table as there can be movies/tv shows with same title and same year of release.

The movies_shows_info table consists of all the additional details regarding the movie/tv show such as the runtimes, summary of the story, the plot_type of the movie/tv show and so on. The nsan is the primary key for each of the record in the table and all the other columns are based on the nsan and have no dependencies among the non-candidate keys. Each column value such as the plot_type, summary, production_company, etc. are all specific to the movie/tv show and is dependent on the nsan which it is mapped to in the table and hence nsan is the only determinant in the table and it is a candidate key.



nsan	title	release_year	type
nsan00012	Friends	1994	TV
nsan00011	By the Sea	2015	Movie
nsan00010	Mr. & Mrs. Smith	2005	Movie
nsan00009	Men in Black	1997	Movie
nsan00008	The Fresh Prince of Bel-Air	1990	TV

(i)movies_shows table



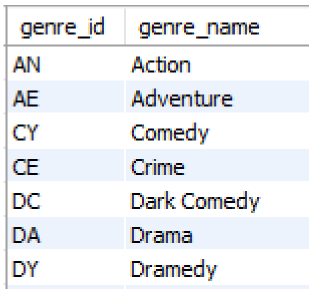
nsan	bo_gross	language	synopsis	runtime	production_company	plot_type
nsan00012	246168	English	Follows the personal and professional lives of si...	22	Bright/Kauffman/Crane Pro...	Original
nsan00011	3300000	English	A couple tries to repair their marriage while stay...	122	Universal Pictures	Original
nsan00010	487300000	English	A bored married couple is surprised to learn tha...	120	New Regency Productions	Original
nsan00009	589400000	English	A police officer joins a secret organization that ...	98	Columbia Pictures	Original
nsan00008	3300000	English	A streetwise, poor young man from Philadelphia...	22	Quincy Jones Entertainment	Original

(ii) movies_shows_info table

Figure 3: Excerpt of movies_shows and movies_shows_info table

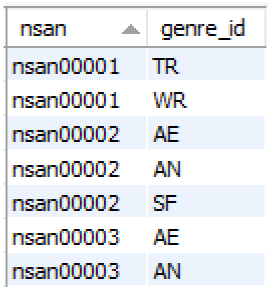
The genre table consists of all the various genres that a movie/tv show can belong to. It consists of only two columns that is the genre_id and genre_name columns where genre_id is the primary key of the table and genre_name is unique key to avoid duplicate entires. The genre_name and genre_id are candidate keys and hence there are no non-candidate key dependencies.

The genre_map table is used to map the various movies/tv shows to the genre they belong. The genre_id and nsan columns are together a combined primary key. As, there are only two columns and are together a whole primary key, there are no non-candidate keys dependencies.



genre_id	genre_name
AN	Action
AE	Adventure
CY	Comedy
CE	Crime
DC	Dark Comedy
DA	Drama
DY	Dramedy

(i) genre table

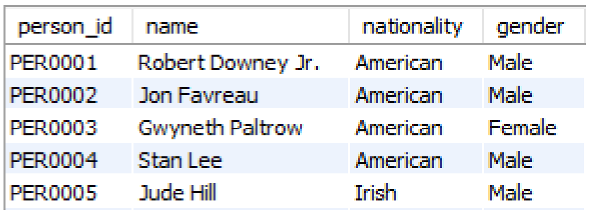


nsan	genre_id
nsan00001	TR
nsan00001	WR
nsan00002	AE
nsan00002	AN
nsan00002	SF
nsan00003	AE
nsan00003	AN

(ii)genre_map table

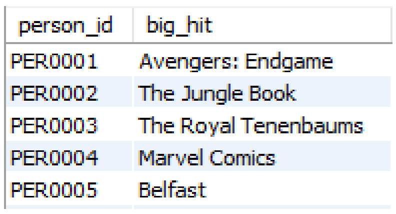
Figure 4: Excerpt of genre and genre_map table

The person_info table is used to store the details of all the people that are involved in a movie/tv show. The person_id column is the primary key of this table and this person_id is used in other tables to map the person as either a cast or crew member for the movie/tv show. The person_id is used to retrieve all the required details of a person and hence all the other columns in this table are dependent on this column. There are no partial-dependencies among the non-candidate keys and the person_id is the only determinant in the table as there can be multiple people with the same name and nationality, also the names can be gender-neutral, so we cannot identify the gender based on the name or vice-versa. The big hits are stored in another table as there can be multiple big hits in a person's career and the redundancy would be eliminated.



person_id	name	nationality	gender
PER0001	Robert Downey Jr.	American	Male
PER0002	Jon Favreau	American	Male
PER0003	Gwyneth Paltrow	American	Female
PER0004	Stan Lee	American	Male
PER0005	Jude Hill	Irish	Male

(i) person_info table



person_id	big_hit
PER0001	Avengers: Endgame
PER0002	The Jungle Book
PER0003	The Royal Tenenbaums
PER0004	Marvel Comics
PER0005	Belfast

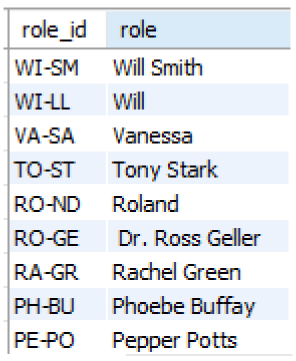
(ii)person_big_hit table

Figure 5: Excerpt of person_info and person_big_hit table

The role_info table is used to store all the various characters that have been portrayed by the various actors/actresses in different movies/tv shows. The role_id column is the primary key of the table and is used to identify the various characters in the table. So there are no determinant overlaps or non-candidate key dependencies in the table.

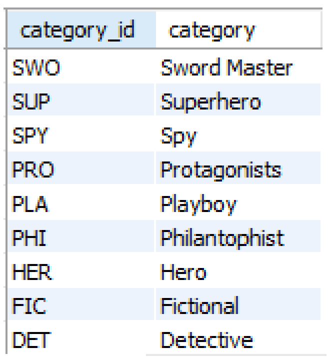
The role_category table is used to store the various categories that a role can belong to. The categories which the role belongs to defines the nature of the role in the movie/tv show. The category_id column is the primary key for the table. The category is a unique key to ensure the same category is not added twice and hence there are no non-candidate key dependencies in the table.

The role_map table is used to map the roles to the various categories it belongs. This table consists of role_id and category_id columns which are the combined primary key for this table. So, a single role_id can be entered multiple times in the table to ensure that all the various categories are mapped properly. As, there are only two columns and are together a whole primary key, there are no partial dependencies and non-candidate key dependencies in the table.



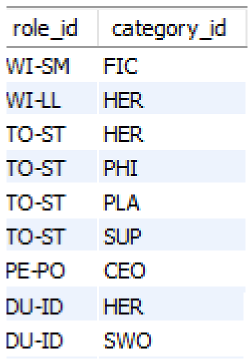
role_id	role
WI-SM	Will Smith
WI-LL	Will
VA-SA	Vanessa
TO-ST	Tony Stark
RO-ND	Roland
RO-GE	Dr. Ross Geller
RA-GR	Rachel Green
PH-BU	Phoebe Buffay
PE-PO	Pepper Potts

(i)role_info table



category_id	category
SWO	Sword Master
SUP	Superhero
SPY	Spy
PRO	Protagonists
PLA	Playboy
PHI	Philantophist
HER	Hero
FIC	Fictional
DET	Detective

(ii)role_category table



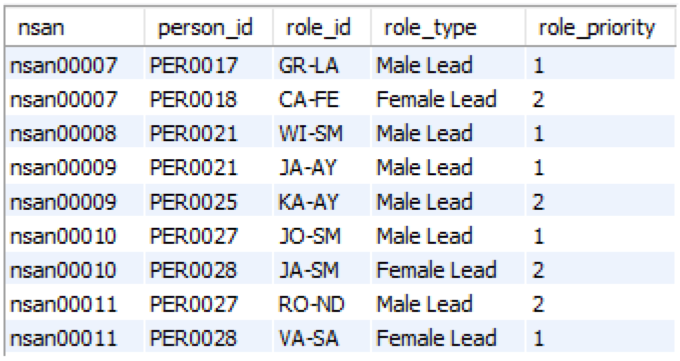
role_id	category_id
WI-SM	FIC
WI-LL	HER
TO-ST	HER
TO-ST	PHI
TO-ST	PLA
TO-ST	SUP
PE-PO	CEO
DU-ID	HER
DU-ID	SWO

(iii)role_map table

Figure 6: Excerpt of role_info, role_category and role_map tables

The cast_map table is used to associate the various actors/actresses with the movie/tv show. The nsan, person_id and role_id together is the combined primary key for this table. The role_type and role_priority values depend on the whole primary key values. Ex: Ben Afleck acting as Batman in Justice league will be of type main lead and have a priority of 1. Also, Henry Cavill acting as

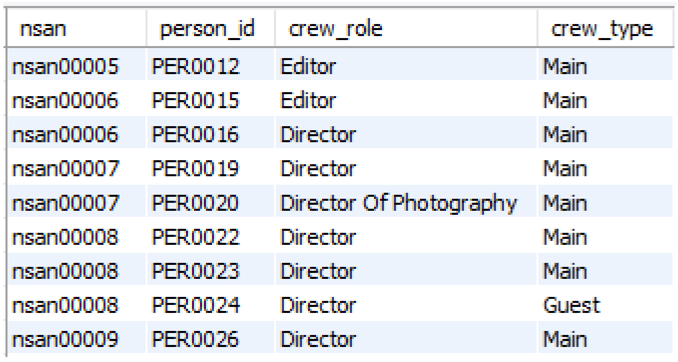
Superman in Justice League will be of type main lead and can have priority of 2. So, both values are based on the whole primary key value and knowing these two values we cannot determine the other information in this table which ensures that there are no non-candidate key dependencies and determinants overlap as the non-prime columns are dependent on the super key.



nsan	person_id	role_id	role_type	role_priority
nsan00007	PER0017	GR-LA	Male Lead	1
nsan00007	PER0018	CA-FE	Female Lead	2
nsan00008	PER0021	WI-SM	Male Lead	1
nsan00009	PER0021	JA-AY	Male Lead	1
nsan00009	PER0025	KA-AY	Male Lead	2
nsan00010	PER0027	JO-SM	Male Lead	1
nsan00010	PER0028	JA-SM	Female Lead	2
nsan00011	PER0027	RO-ND	Male Lead	2
nsan00011	PER0028	VA-SA	Female Lead	1

Figure 7: Excerpt of cast_map table

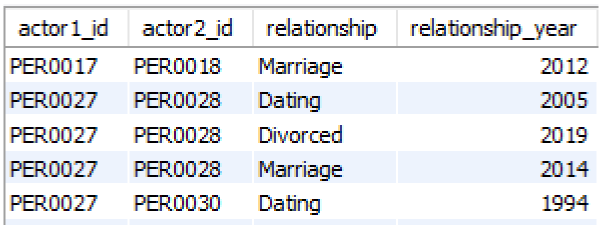
The crew_map table is used to associate the various people working as crew members in a particular movie/tv show. The nsan, person_id and crew_role combined together is the primary key for this table. Similar to cast_map table, there are no partial or non-candidate key dependencies in the table and all the non-prime columns are dependent on the super key and the record can be determined only with the super key.



nsan	person_id	crew_role	crew_type
nsan00005	PER0012	Editor	Main
nsan00006	PER0015	Editor	Main
nsan00006	PER0016	Director	Main
nsan00007	PER0019	Director	Main
nsan00007	PER0020	Director Of Photography	Main
nsan00008	PER0022	Director	Main
nsan00008	PER0023	Director	Main
nsan00008	PER0024	Director	Guest
nsan00009	PER0026	Director	Main

Figure 8: Excerpt of crew_map table

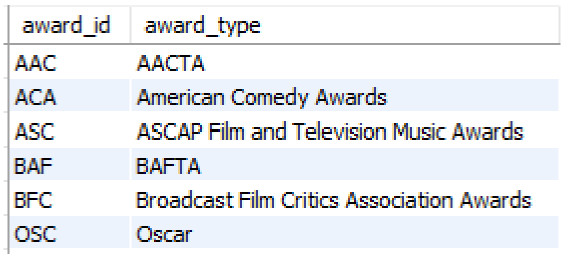
The connections table is used to store the various relationships between the actors/actresses in real life. The actor1_id, actor2_id, relationship together form the primary key for the table. The relationship year is dependent on the whole primary key as the value will be based on the type of the relationship and also the people involved in the relationship and hence there are no non-candidate key dependencies.



actor1_id	actor2_id	relationship	relationship_year
PER0017	PER0018	Marriage	2012
PER0027	PER0028	Dating	2005
PER0027	PER0028	Divorced	2019
PER0027	PER0028	Marriage	2014
PER0027	PER0030	Dating	1994

Figure 9: Excerpt of connections table

The awards table is used to store the various types of the awards. The award_id column is the primary key of the table and the various awards are mapped with an id which can be used to refer in other tables. The award_type is a unique key to ensure the same type of award is not added twice and hence there are no non-candidate key dependencies in the table.



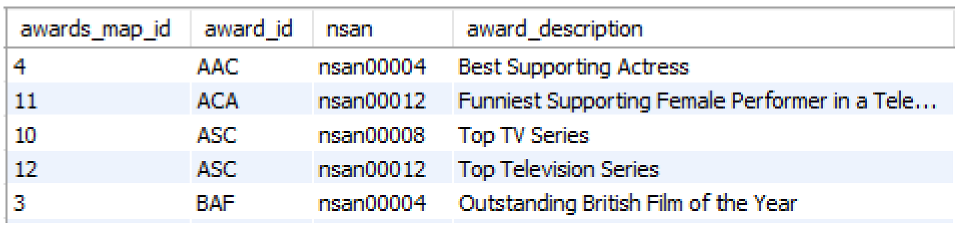
award_id	award_type
AAC	AACTA
ACA	American Comedy Awards
ASC	ASCAP Film and Television Music Awards
BAF	BAFTA
BFC	Broadcast Film Critics Association Awards
OSC	Oscar

Figure 10: Excerpt of awards table

The awards_map table is used to store the information of the various awards that a movie/tv show has won. The awards_map_id is the primary key of the table and the various movies and awards are mapped with the description of the award. The award_id, nsan, award_description columns together is a unique constraint as there can be many types of awards won by the same movie/tv show and hence there are no non candidate key dependencies in the table.

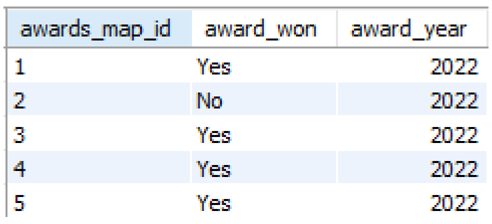
The awards_info table is used to store the information about the year of the awards nomination and also if the award was won or not. The awards_info table has awards_map_id and the award_year together as the combined primary key as the same type of award can be given to the same TV show but the year can be different (Ex: Game Of Thrones TV show nominated for Best TV series for multiple years). The columns in awards_info table is separated from awards_map table to ensure that there is no redundant data and also ensure there is no non-candidate key dependencies.

The award_person_map is used to map the information about the person that has won the award in that movie/tv show. A separate table has been maintained for this as some awards are presented to the movie/tv show as a whole (Ex: Best Film of the year, Top TV series, etc.) and some awards are specific to the individuals (Ex: Best Supporting actress, Best Director, etc.), so this table will specify the individuals to which the award was presented to and the previous table specifies the movie/tv show which won the award. This way we can map the individual to the movie/tv show and get the role from the cast_map table as we will have the nsan for which this award was won. The awards_map_id along with award_year and person_id is the primary key of the table as there can be multiple awards associated to the same person. As all the columns together is the combined primary key of the table there are no non-candidate key dependencies.

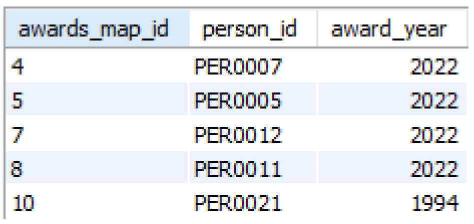


awards_map_id	award_id	nsan	award_description
4	AAC	nsan00004	Best Supporting Actress
11	ACA	nsan00012	Funniest Supporting Female Performer in a Tele...
10	ASC	nsan00008	Top TV Series
12	ASC	nsan00012	Top Television Series
3	BAF	nsan00004	Outstanding British Film of the Year

(i)awards_map table



awards_map_id	award_won	award_year
1	Yes	2022
2	No	2022
3	Yes	2022
4	Yes	2022
5	Yes	2022



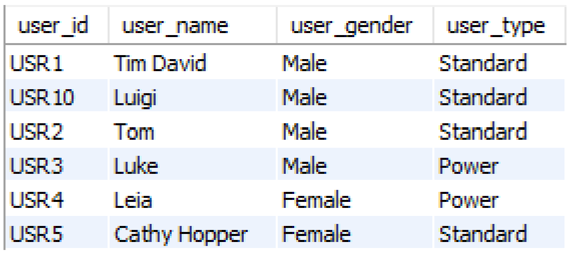
awards_map_id	person_id	award_year
4	PER0007	2022
5	PER0005	2022
7	PER0012	2022
8	PER0011	2022
10	PER0021	1994

(ii)awards_info table *(iii)award_person_map table*

Figure 11: Excerpt of awards_map, awards_info, award_person_map tables

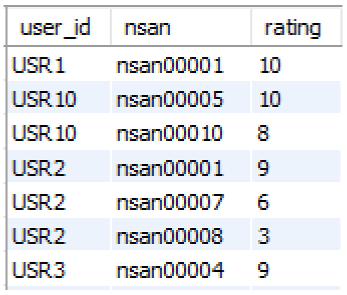
The user table is similar to the person_info table but it is used to store the information of the various users that access the streaming platform. The user_id column is the primary key of the table and this user_id can be used to identify the various persons and get their respective details. All the other columns in this table are dependent on the user_id column similar to the person_info table.

The ratings table is used to store the information about the various ratings a particular user has provided for the various movies/ tv shows. This table can be used to determine the kind of movies/tv shows a particular user likes and it can also be used to get an average rating for the movie/tv show based on the rating given by all the users. The ratings table has user_id and nsan as a combined primary key as a particular user can provide ratings to multiple movies/tv shows and different users can provide their rating to the same movie/tv show. The rating will depend on the movie/tv show and the user who is giving the rating and since these both combined is the primary key for this table, there are no partial dependencies or alternative determinant in the table.



user_id	user_name	user_gender	user_type
USR1	Tim David	Male	Standard
USR10	Luigi	Male	Standard
USR2	Tom	Male	Standard
USR3	Luke	Male	Power
USR4	Leia	Female	Power
USR5	Cathy Hopper	Female	Standard

(i)user_info table



user_id	nsan	rating
USR1	nsan00001	10
USR10	nsan00005	10
USR10	nsan00010	8
USR2	nsan00001	9
USR2	nsan00007	6
USR2	nsan00008	3
USR3	nsan00004	9

(ii)ratings table

Figure 12: Excerpt of user_info and ratings tables

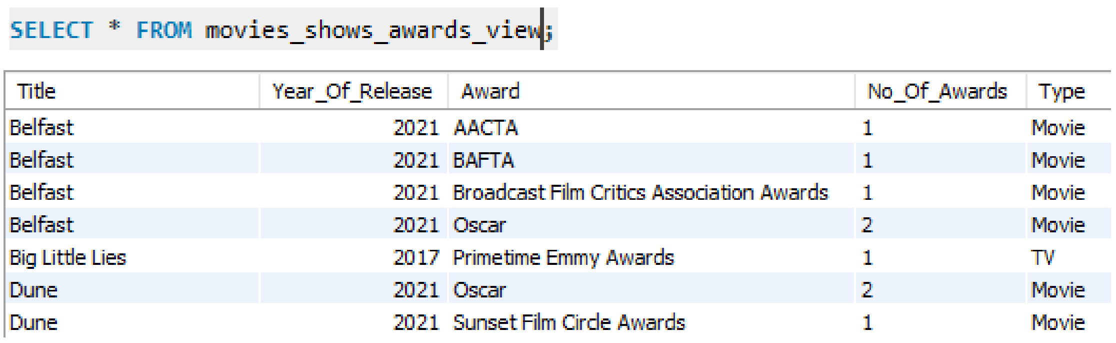
The database is in the first normal form as all the tables have atomic values and every table has its own primary key. And as specified for each of the table there are no partial dependencies amongst the various non candidate key attributes and in case if any dependency exists the values of the non-candidate attributes are dependent on the whole primary key. If the movies/tv show information was stored along with the various actors/actress information then there would be some transitive dependencies where the actor details are dependent on the person_id and the movie/tv show details are dependent on the nsan. But since all these attributes are stored separately in the various tables and are mapped using the cast_map table the transitive dependency is removed. Similarly, even in all other tables the various details where required are stored in separate tables and are mapped using the keys ensuring that the database is also in third normal form. The database is in the Boyce Codd Normal Form as it complies with the third normal form and for every functional dependency, there exists a super-key. Each table is identified uniquely using the super-key, where the super-key is a single key and in some tables a group of keys which are used to identify the records uniquely. Ex: The movie/tv show details can be accessed using the nsan defined across the database. The user details can be accessed using the user_id. Similarly, for all the various information to be retrieved the determinant is only the super key in that table without any alternative determinant.

4. Database Views

The views that are provided in the database are used to provide information about the various types of movies/tv shows that are available for the users in the streaming platform. As the standard users browse through the application and view the information that is provided to them, these views can be used by the application to provide the required information and ensure that the users are provided with interesting and relevant information to help them decide which movie/tv show they would like to watch. The views also ensure that they provide access to the standard users to the columns only contained in the view and the underlying tables data would not be exposed to all the users. So, based on the privilege of users, various data can be provided to them without providing access to the main tables of the database.

The views in the database are used to provide the required information by combining the various tables by making use of select statements with the joins, sub-queries and particular conditions. The relevant data will be provided in a tabular format that could be used to gain insights easily.

The movies_shows_awards_view consists of the Title, Year_Of_Release, Award and Type columns. This view displays all the award-winning movies/tv shows along with the count of various types of awards that these movies/tv shows have won.



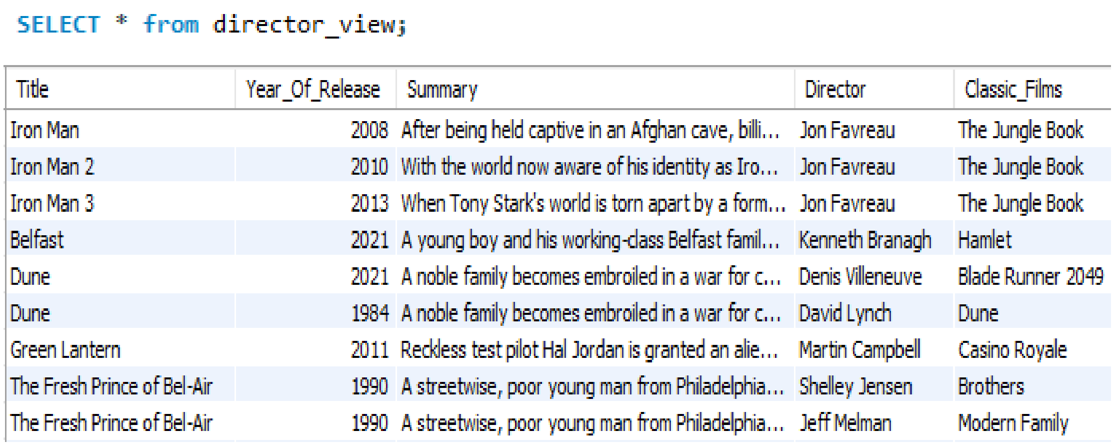
```
SELECT * FROM movies_shows_awards_view;
```

Title	Year_Of_Release	Award	No_Of_Awards	Type
Belfast	2021	AACTA	1	Movie
Belfast	2021	BAFTA	1	Movie
Belfast	2021	Broadcast Film Critics Association Awards	1	Movie
Belfast	2021	Oscar	2	Movie
Big Little Lies	2017	Primetime Emmy Awards	1	TV
Dune	2021	Oscar	2	Movie
Dune	2021	Sunset Film Circle Awards	1	Movie

Figure 13: Snapshot of movies_shows_awards_view

This view can be used by all types of users of the streaming platform and can also be used for recommendation as a lot of people decide to watch a movie/ tv show if it has won a lot of awards.

The director_view is used to provide all the movies/tv shows that are present in the database along with the year of it's release, the brief summary of what the movie or tv show is about, the director information for the movie/tv show along with the information of the biggest hit of that director. The biggest hit of the director is also mentioned as most of the users tend to watch a movie/tv show if it is directed by a director who has some previous big classic hit or if it is from the same director of a movie/tv show that is a personal favorite to a particular user. If there are multiple directors for a movie/tv show, all of the directors would be displayed.



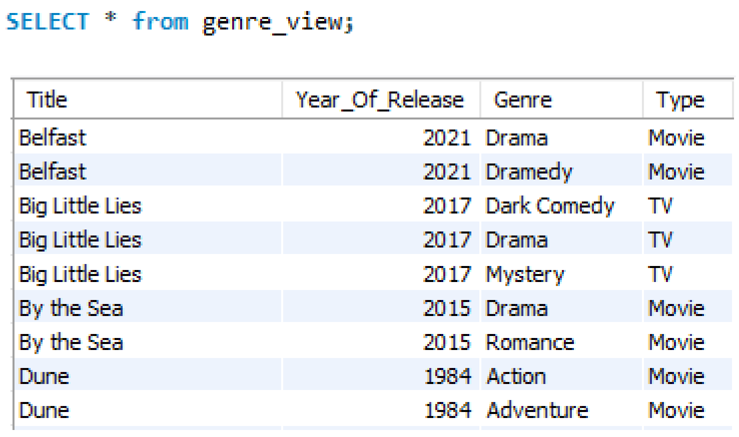
```
SELECT * from director_view;
```

Title	Year_Of_Release	Summary	Director	Classic_Films
Iron Man	2008	After being held captive in an Afghan cave, billi...	Jon Favreau	The Jungle Book
Iron Man 2	2010	With the world now aware of his identity as Iro...	Jon Favreau	The Jungle Book
Iron Man 3	2013	When Tony Stark's world is torn apart by a form...	Jon Favreau	The Jungle Book
Belfast	2021	A young boy and his working-class Belfast famil...	Kenneth Branagh	Hamlet
Dune	2021	A noble family becomes embroiled in a war for c...	Denis Villeneuve	Blade Runner 2049
Dune	1984	A noble family becomes embroiled in a war for c...	David Lynch	Dune
Green Lantern	2011	Reckless test pilot Hal Jordan is granted an alie...	Martin Campbell	Casino Royale
The Fresh Prince of Bel-Air	1990	A streetwise, poor young man from Philadelphia...	Shelley Jensen	Brothers
The Fresh Prince of Bel-Air	1990	A streetwise, poor young man from Philadelphia...	Jeff Melman	Modern Family

Figure 14: Snapshot of director_view

The above image is an excerpt of the director_view and it can be observed that a user who likes the movie Hamlet that is directed by Kenneth Branagh would like to watch Belfast as it is directed by him or if a user knows Hamlet is a big classic, then the user would prefer to watch other movies directed by that director.

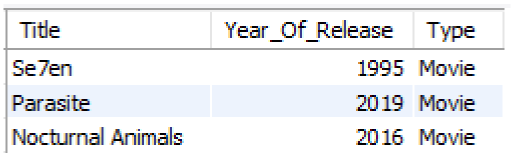
The genre_view displays all the movies/tv shows along with the genres that each of them belongs to. The genre of the movie/tv show is one of the deciding factors for most of the users to watch a movie/tv show.



```
SELECT * from genre_view;
```

Title	Year_Of_Release	Genre	Type
Belfast	2021	Drama	Movie
Belfast	2021	Dramedy	Movie
Big Little Lies	2017	Dark Comedy	TV
Big Little Lies	2017	Drama	TV
Big Little Lies	2017	Mystery	TV
By the Sea	2015	Drama	Movie
By the Sea	2015	Romance	Movie
Dune	1984	Action	Movie
Dune	1984	Adventure	Movie

(i) genre_view



Title	Year_Of_Release	Type
Se7en	1995	Movie
Parasite	2019	Movie
Nocturnal Animals	2016	Movie

(ii) psychological_thriller_view

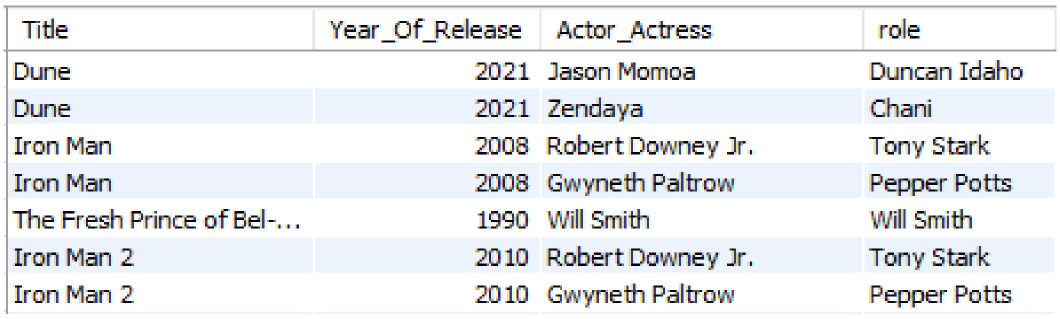
Figure 15: Snapshot of genre_view and psychological_thriller_view

The above image is an excerpt of the genre_view and it can be observed that a user who likes Mystery genre TV Show would like to watch Big Little Lies. So, this view can be used to recommend the movies/tv shows based on the genre they like.

Views with specific complex genre type also can be very useful as many of the movies now a days have combined genres which user look out for. So, the view psychological_thriller_view stores all the movies which are of Psychological Thriller genre which is a combination of 'Psychological Fiction' and 'Thriller'. Similarly, we can have multiple views with complex genres such as Romcom, Scifi-Thriller and so on, which would be of interest to many users.

The trending_view displays all the movies/tv shows that are present in the database in the order of the trend, with most trending movie/tv show on the top. The movies/tv shows is considered to be trending if a lot of users have given ratings to that movie/tv show. So, the movies/tv shows in this view will be ranked based on the number of users who have given rating for that movie/tv show. It also displays the actor/actress with the character that they portray in that movie. Most of the users decide to watch a movie/tv show based on the various actors/actress that are involved in the movie. Users tend to watch the movies/tv shows with cast that is familiar to them and also if they've liked some movie/tv show with a similar actor/actress.

```
SELECT * FROM trending_view;
```



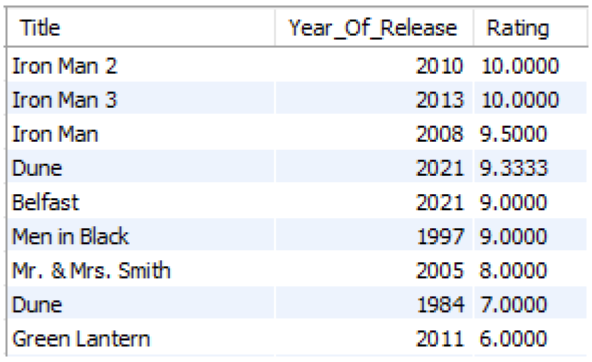
Title	Year_Of_Release	Actor_Actress	role
Dune	2021	Jason Momoa	Duncan Idaho
Dune	2021	Zendaya	Chani
Iron Man	2008	Robert Downey Jr.	Tony Stark
Iron Man	2008	Gwyneth Paltrow	Pepper Potts
The Fresh Prince of Bel-...	1990	Will Smith	Will Smith
Iron Man 2	2010	Robert Downey Jr.	Tony Stark
Iron Man 2	2010	Gwyneth Paltrow	Pepper Potts

Figure 16: Snapshot of trending_view

The above image is an excerpt of the trending_view and it can be observed that Dune is trending on top, so the user could decide to watch that movie or a user who likes the actress Zendaya would like to watch Dune or based on the role/character if Tony Stark is of interest to them then that particular movie can be picked.

The rating_view displays all the movies/tv shows along with the ratings. The rating for a particular movie/tv show is calculated by taking the average of the ratings given by all users in the database. Ratings for a movie is very important as most of the users decide to watch a movie/tv show based on the rating. Most of them decide to watch only if a rating is above 5 or 7, based on their preference. So, this view provides all the movies and tv shows with the ratings which would be useful for all types of users of the streaming application.

```
SELECT * from rating_view;
```



Title	Year_Of_Release	Rating
Iron Man 2	2010	10.0000
Iron Man 3	2013	10.0000
Iron Man	2008	9.5000
Dune	2021	9.3333
Belfast	2021	9.0000
Men in Black	1997	9.0000
Mr. & Mrs. Smith	2005	8.0000
Dune	1984	7.0000
Green Lantern	2011	6.0000

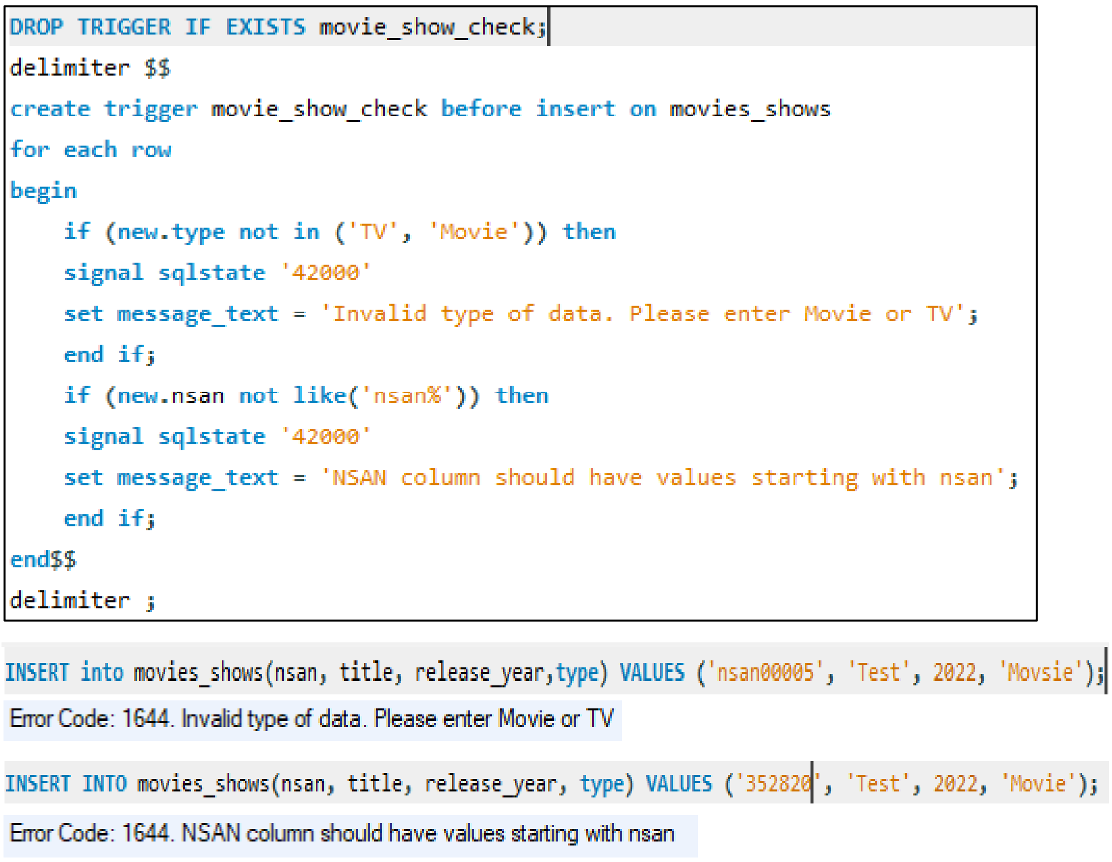
Figure 17: Snapshot of rating_view

5. Procedural Elements

The database employs triggers and stored procedures. The triggers in the database ensure that the data entered in the various tables are consistent and also ensures the integrity of the data.

A trigger is a statement, or set of statements, that is stored and associated with a particular event happening on a particular column or table (Kruckenberg and Pipes, 2005, p.444).

The movie_show_check trigger on the movies_shows table is used to ensure that in the type column a valid type is entered in the movies_shows table. As, the database consists of various information related to movies or TV shows only, the various titles that are entered should be of a movie or TV show. It also ensures that all the values in NSAN column starts with 'nsan' to ensure it is consistent. When a new row is added to the table, if any of these condition fails, the error signal is triggered and the message is displayed informing about the error.



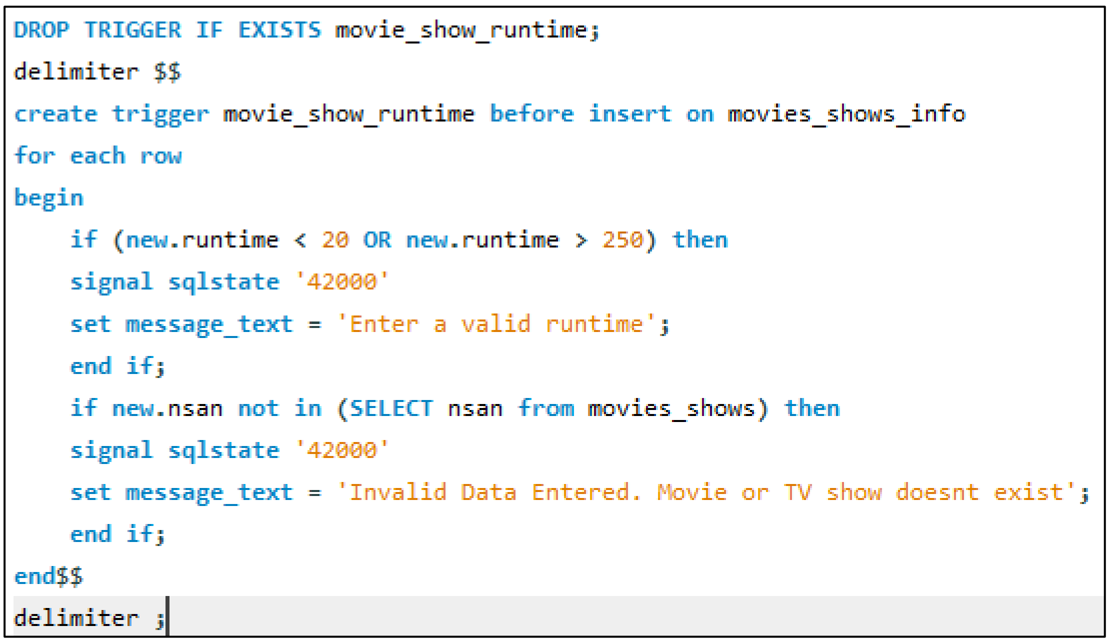
```sql
DROP TRIGGER IF EXISTS movie_show_check;
delimiter $$
create trigger movie_show_check before insert on movies_shows
for each row
begin
    if (new.type not in ('TV', 'Movie')) then
    signal sqlstate '42000'
    set message_text = 'Invalid type of data. Please enter Movie or TV';
    end if;
    if (new.nsan not like('nsan%')) then
    signal sqlstate '42000'
    set message_text = 'NSAN column should have values starting with nsan';
    end if;
end$$
delimiter ;
```

```
INSERT into movies_shows(nsan, title, release_year,type) VALUES ('nsan00005', 'Test', 2022, 'Movsie');
Error Code: 1644. Invalid type of data. Please enter Movie or TV
```

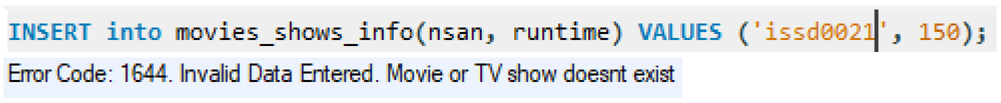
```
INSERT INTO movies_shows(nsan, title, release_year, type) VALUES ('352820', 'Test', 2022, 'Movie');
Error Code: 1644. NSAN column should have values starting with nsan
```

Figure 18: movie_show_check trigger in action

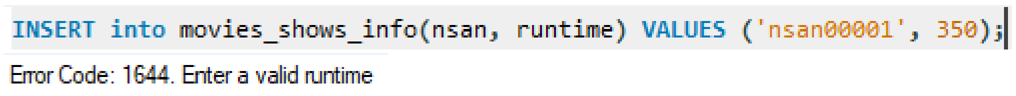
The movie_show_runtime trigger on the movies_shows_info table ensures that the runtime entered for the movies is between 20 to 250 minutes as this is the ideal runtime for all the movies. In case of TV show the runtime indicates the time for episode. Most of the TV shows have episodes runtime ranging from 20 – 120 mins and so this value will be well within 250 minutes. This trigger also checks if the nsan being entered into the table exists to ensure that a valid movie/ tv show information is mapped. If any of the two conditions fail, an error signal is triggered and the message is displayed.



```
DROP TRIGGER IF EXISTS movie_show_runtime;
delimiter $$
create trigger movie_show_runtime before insert on movies_shows_info
for each row
begin
    if (new.runtime < 20 OR new.runtime > 250) then
    signal sqlstate '42000'
    set message_text = 'Enter a valid runtime';
    end if;
    if new.nsan not in (SELECT nsan from movies_shows) then
    signal sqlstate '42000'
    set message_text = 'Invalid Data Entered. Movie or TV show doesnt exist';
    end if;
end$$
delimiter ;
```



```
INSERT into movies_shows_info(nsan, runtime) VALUES ('issd0021', 150);
Error Code: 1644. Invalid Data Entered. Movie or TV show doesnt exist
```

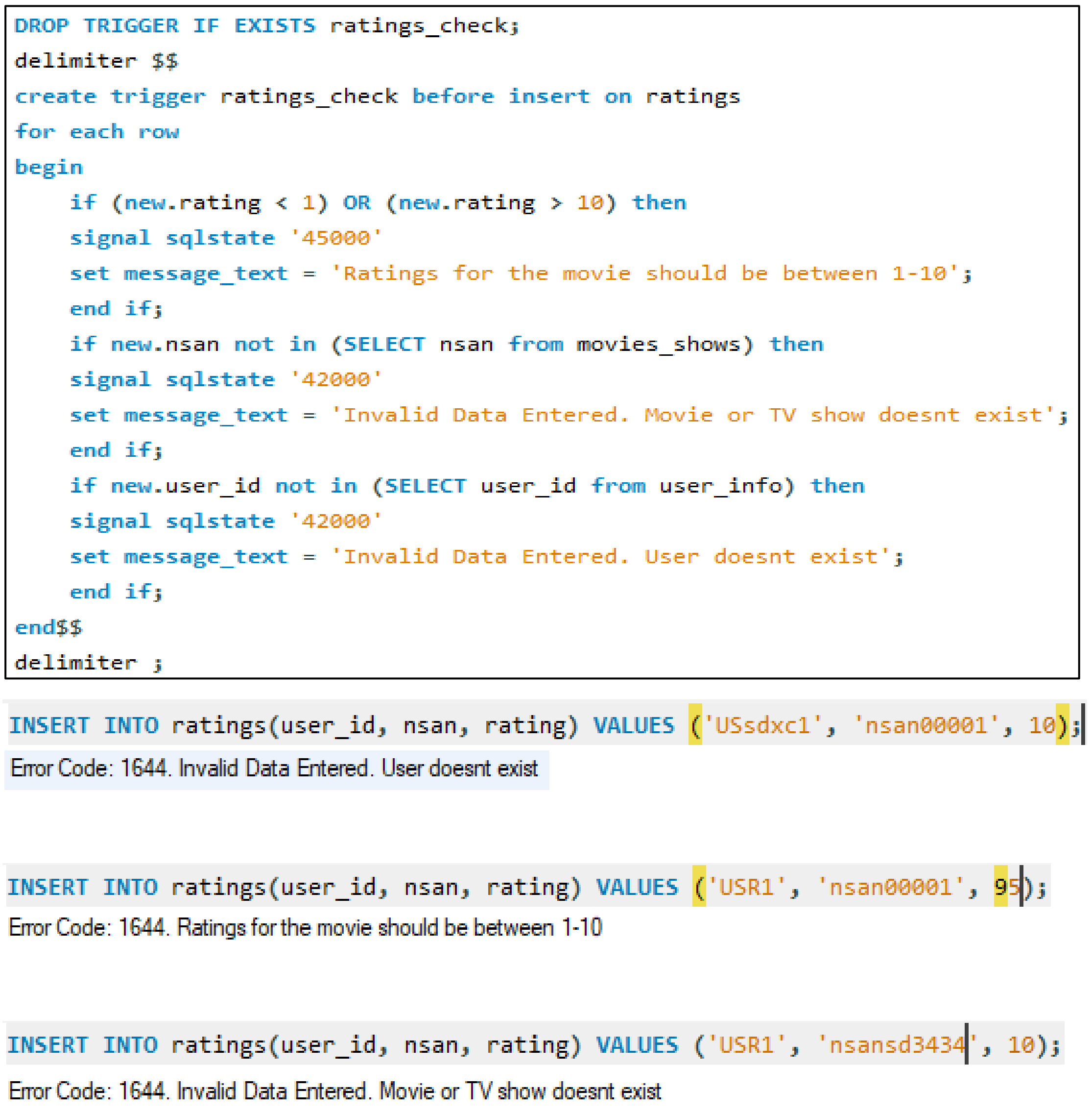


```
INSERT into movies_shows_info(nsan, runtime) VALUES ('nsan00001', 350);
Error Code: 1644. Enter a valid runtime
```

Figure 19: movie_show_runtime trigger in action

The gender_check trigger on the person_info table ensures that the right type of gender has been added to the gender column. It makes sure that the column would either contain ‘Male’ or ‘Female’ values. So, this makes sure that the data across the table is consistent.

The ratings_check trigger on the ratings table ensures that the rating given for a particular movie/tv show is between 1-10 as these are the valid values for a rating. It also checks if a particular user_id and the nsan for which the rating is being given are valid by checking if these values exist in the corresponding tables. If any of these checks fail an error signal is triggered and the message is displayed.



```
DROP TRIGGER IF EXISTS ratings_check;
delimiter $$
create trigger ratings_check before insert on ratings
for each row
begin
    if (new.rating < 1) OR (new.rating > 10) then
    signal sqlstate '45000'
    set message_text = 'Ratings for the movie should be between 1-10';
    end if;
    if new.nsan not in (SELECT nsan from movies_shows) then
    signal sqlstate '42000'
    set message_text = 'Invalid Data Entered. Movie or TV show doesnt exist';
    end if;
    if new.user_id not in (SELECT user_id from user_info) then
    signal sqlstate '42000'
    set message_text = 'Invalid Data Entered. User doesnt exist';
    end if;
end$$
delimiter ;
```

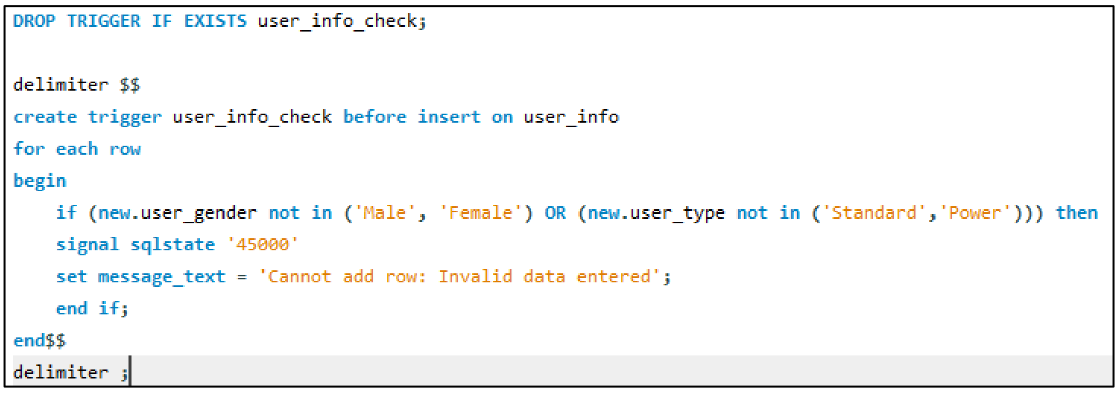
```
INSERT INTO ratings(user_id, nsan, rating) VALUES ('USsdxc1', 'nsan00001', 10);
Error Code: 1644. Invalid Data Entered. User doesnt exist
```

```
INSERT INTO ratings(user_id, nsan, rating) VALUES ('USR1', 'nsan00001', 95);
Error Code: 1644. Ratings for the movie should be between 1-10
```

```
INSERT INTO ratings(user_id, nsan, rating) VALUES ('USR1', 'nsansd3434', 10);
Error Code: 1644. Invalid Data Entered. Movie or TV show doesnt exist
```

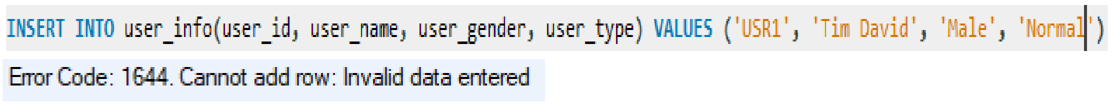
Figure 20: ratings_check trigger in action

The user_info_check trigger on the user_info table ensures that the gender entered for each user is either ‘Male’ or ‘Female’. As the streaming platform consists of two types of users, i.e. ‘Standard’ and ‘Power’ users this trigger ensures that the user_type column contains only those two values. If any other values are entered the error signal is triggered. This makes sure that the user_type column has the data consistent and when querying for a particular type of user there will not be any mismatches and all the records will be returned.



```
DROP TRIGGER IF EXISTS user_info_check;

delimiter $$
create trigger user_info_check before insert on user_info
for each row
begin
    if (new.user_gender not in ('Male', 'Female') OR (new.user_type not in ('Standard','Power'))) then
    signal sqlstate '45000'
    set message_text = 'Cannot add row: Invalid data entered';
    end if;
end$$
delimiter ;
```



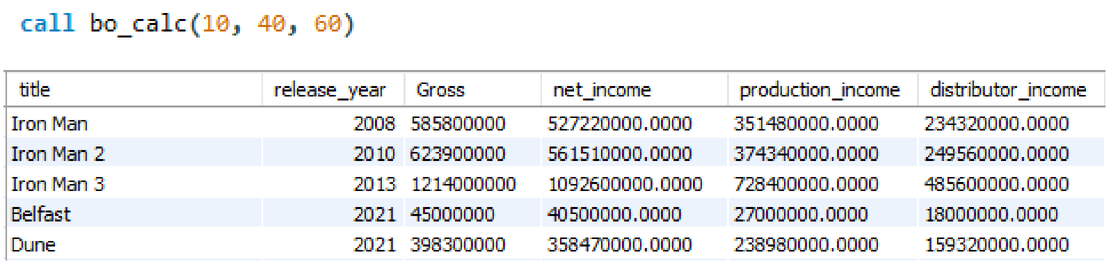
```
INSERT INTO user_info(user_id, user_name, user_gender, user_type) VALUES ('USR1', 'Tim David', 'Male', 'Normal')
Error Code: 1644. Cannot add row: Invalid data entered
```

Figure 21: user_info_check trigger in action

Stored procedures are the most common type of stored program. A stored procedure is a generic program unit that is executed on request and that can accept multiple input and output parameters (Harrison and Feuerstein, 2006, p.4).

Stored procedures can be used to provide the required information by not letting the users interact directly with the underlying tables. They provide a built-in access control that can be used to enhance the security of the database.

The stored procedure bo_calc is used to calculate the breakup of the box office gross collection. The net income is the amount that will be gained by the production company after the tax deductions. As the box office gross collection will include tax this procedure does the calculations and provides the exact amount. Also, based on the values that are passed to the procedure, net income, income gained by the production company and the income gained by the distributors for each movie/tv show is calculated. The values provided to the procedure will include tax percentage, production and distribution share percentage. So, a user can call this procedure and get to know the various incomes that is gained by the various departments of the movie/tv show. Ex: If we have 10% tax deduction, 40% as the income of production company and 60% to the distributors, then if we call the procedure with the required parameters as shown below, we get the results accordingly.



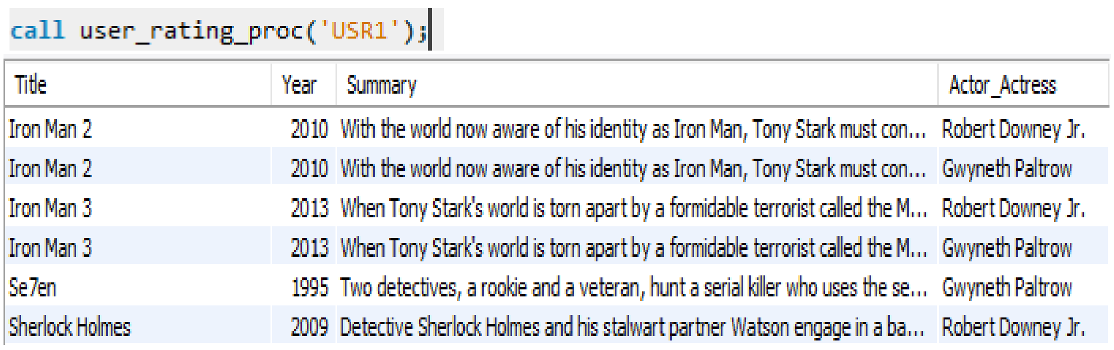
```
call bo_calc(10, 40, 60)
```

title	release_year	Gross	net_income	production_income	distributor_income
Iron Man	2008	585800000	527220000.0000	351480000.0000	234320000.0000
Iron Man 2	2010	623900000	561510000.0000	374340000.0000	249560000.0000
Iron Man 3	2013	1214000000	1092600000.0000	728400000.0000	485600000.0000
Belfast	2021	45000000	40500000.0000	27000000.0000	18000000.0000
Dune	2021	398300000	358470000.0000	238980000.0000	159320000.0000

Figure 22: Snapshot of bo_calc procedure output

The user_rating_proc in the database is used to provide the movie/tv show recommendation to the users based on the ratings they have provided for the various films or tv shows. This procedure takes the user ID as the input and checks the rating table for the movies/tv shows that the user has given rating. The movies/tv shows where the user has given a rating of more than 5 is considered and all the actors/actresses that are involved in the various movies/tv shows is gathered and then all the other movies/tv shows that include the actor/actress are displayed to the user. The movies/tv shows that are given a rating by the user is considered to be already watched by the user, so all the other movies/tv shows excluding that will be displayed to the user.

Ex: If the user has given a rating for Iron Man where Robert Downy Jr. and Gwyneth Paltrow have acted, then all the movies where Robert Downy Jr. or Gwyneth Paltrow have acted except Iron man will be displayed to the user.



```
call user_rating_proc('USR1');
```

Title	Year	Summary	Actor_Actress
Iron Man 2	2010	With the world now aware of his identity as Iron Man, Tony Stark must con...	Robert Downey Jr.
Iron Man 2	2010	With the world now aware of his identity as Iron Man, Tony Stark must con...	Gwyneth Paltrow
Iron Man 3	2013	When Tony Stark's world is torn apart by a formidable terrorist called the M...	Robert Downey Jr.
Iron Man 3	2013	When Tony Stark's world is torn apart by a formidable terrorist called the M...	Gwyneth Paltrow
Se7en	1995	Two detectives, a rookie and a veteran, hunt a serial killer who uses the se...	Gwyneth Paltrow
Sherlock Holmes	2009	Detective Sherlock Holmes and his stalwart partner Watson engage in a ba...	Robert Downey Jr.

Figure 23: Snapshot of user_rating_proc procedure output

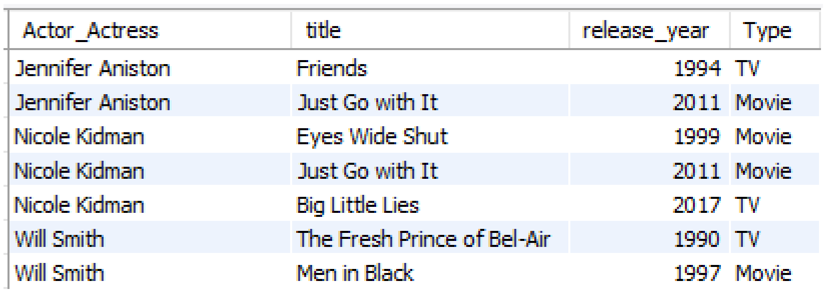
The user_proc procedure is used to provide the movies/tv shows information to the Standard users. This procedure takes the user id as the input and from the user_info table it checks whether the particular user is Standard or Power user. If the user is of type Standard, the procedure displays all the views that are present in the table. It also calls the user_rating_proc by passing the user_id and displays the movies/tv shows as a recommendation based on the rating provided by this particular user.

```
call user_proc('USR1');
```

6. Example Queries:

i) Query to find all the actors/ actresses who have worked in both movies and TV shows along with all the movies and TV shows that they have worked, in order of the year they first started working in.

```
SELECT name as Actor_Actress, title, release_year, type as Type from
movies_shows
JOIN cast_map ON movies_shows.nsan = cast_map.nsan
JOIN person_info ON person_info.person_id = cast_map.person_id
WHERE person_info.person_id in
(SELECT person_id from movies_shows
JOIN cast_map ON movies_shows.nsan = cast_map.nsan
WHERE type='TV'
AND person_info.person_id in
(SELECT person_id from movies_shows
JOIN cast_map ON movies_shows.nsan = cast_map.nsan
WHERE type='Movie'))
ORDER BY name, release_year;
```



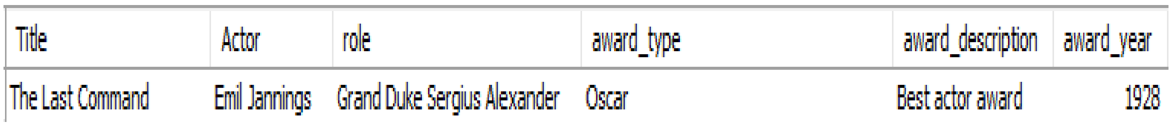
Actor_Actress	title	release_year	Type
Jennifer Aniston	Friends	1994	TV
Jennifer Aniston	Just Go with It	2011	Movie
Nicole Kidman	Eyes Wide Shut	1999	Movie
Nicole Kidman	Just Go with It	2011	Movie
Nicole Kidman	Big Little Lies	2017	TV
Will Smith	The Fresh Prince of Bel-Air	1990	TV
Will Smith	Men in Black	1997	Movie

Figure 24: Query(i) Output

This query result shows that the power users who are interested to get to know the various actors/actresses who have worked in both Movies and TV shows can query the results easily by making use of the select statement using the joins and sub-queries. The query results are also displayed in the order of the year they started working in, so with this the users can get to know if the actors/actresses started their career in TV shows and then moved towards movies or vice-versa. This query result would be useful for both the types of users of the streaming platform as it can also be used to provide recommendations to some users who have their favorites in either TV shows or movies and would like to watch a movie/tv show with the same actor/actress.

ii) Query to find the first male actor who has won an Oscar along with the movie and the role for which the award was won.

```
SELECT title as Title, name as Actor, role, award_type, award_description,
award_person_map.award_year from movies_shows
JOIN cast_map ON movies_shows.nsan = cast_map.nsan
JOIN role_info ON cast_map.role_id = role_info.role_id
JOIN person_info ON person_info.person_id = cast_map.person_id
JOIN award_person_map ON award_person_map.person_id =
person_info.person_id
JOIN awards_map ON award_person_map.awards_map_id =
awards_map.awards_map_id
AND movies_shows.nsan = awards_map.nsan
JOIN awards ON awards.award_id = awards_map.award_id
JOIN awards_info ON awards_info.awards_map_id = awards_map.awards_map_id
WHERE award_person_map.award_year IN
(SELECT MIN(award_person_map.award_year) FROM awards
JOIN awards_map ON awards_map.award_id = awards.award_id
JOIN awards_info ON awards_map.awards_map_id = awards_info.awards_map_id
JOIN award_person_map
ON awards_info.awards_map_id = award_person_map.awards_map_id
AND awards_info.award_year = award_person_map.award_year
JOIN person_info
ON award_person_map.person_id = person_info.person_id
where gender='Male' and award_type='Oscar');
```



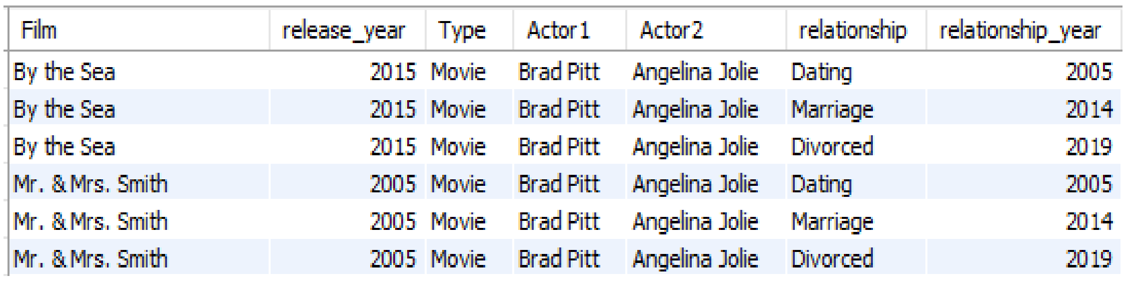
Title	Actor	role	award_type	award_description	award_year
The Last Command	Emil Jannings	Grand Duke Sergius Alexander	Oscar	Best actor award	1928

Figure 25: Query(ii) Output

This query result displays the actor who has won the award along with the information of the role actor had portrayed. This query results shows that the power users can query and find the result for some specific questions which they want to explore. The database supports querying of the required information easily with some table joins and conditions. Similarly, the power user can query for some specific actor/ actress and get the details of the awards won by them, query for some specific category of award by adding the condition to the where clause, they can also query for awards from a specific year and so on. The database supports all kinds of queries and returns the required results.

iii) Query to find the pair of actors who had been together the longest before getting divorced and have also acted together in a movie/tv show. Also, finding the movie/tv show they have worked together.

```
SELECT distinct title as Film, release_year, type as Type, p1.name as Actor1,
p2.name as Actor2, relationship, relationship_year  FROM movies_shows
JOIN cast_map mc1 ON movies_shows.nsan=mc1.nsan
JOIN person_info p1 ON mc1.person_id=p1.person_id
JOIN cast_map mc2 ON movies_shows.nsan=mc2.nsan
JOIN person_info p2 ON mc2.person_id=p2.person_id
JOIN connections
ON (p1.person_id=connections.actor1_id AND
p2.person_id=connections.actor2_id)
WHERE (mc1.person_id, mc2.person_id) IN
(SELECT c1.actor1_id,c1.actor2_id as year_diff from connections c1
JOIN connections c2
ON c1.actor1_id = c2.actor1_id AND
c1.actor2_id = c2.actor2_id
WHERE c1.relationship != c2.relationship
AND c2.relationship = 'Divorced'
AND c2.relationship_year-c1.relationship_year IN
(SELECT MAX(year_diff) FROM
(SELECT c1.actor1_id,c1.actor2_id,c2.relationship_year-c1.relationship_year as
year_diff from connections c1
JOIN connections c2
ON c1.actor1_id = c2.actor1_id AND
c1.actor2_id = c2.actor2_id
WHERE c1.relationship != c2.relationship
AND c2.relationship = 'Divorced') con))
order by title, relationship_year;
```



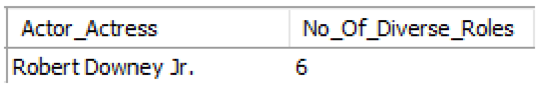
Film	release_year	Type	Actor1	Actor2	relationship	relationship_year
By the Sea	2015	Movie	Brad Pitt	Angelina Jolie	Dating	2005
By the Sea	2015	Movie	Brad Pitt	Angelina Jolie	Marriage	2014
By the Sea	2015	Movie	Brad Pitt	Angelina Jolie	Divorced	2019
Mr. & Mrs. Smith	2005	Movie	Brad Pitt	Angelina Jolie	Dating	2005
Mr. & Mrs. Smith	2005	Movie	Brad Pitt	Angelina Jolie	Marriage	2014
Mr. & Mrs. Smith	2005	Movie	Brad Pitt	Angelina Jolie	Divorced	2019

Figure 26: Query(iii) Output

This query result provides the pair of actors who had some relationships in real life and had been together the longest before getting divorced. This query illustrates that the power users can even query for some of the personal information related to the various actors and gain insights. The various relationships of actors in real life are always of interest to many of the users and database enables the users to search the required information related to that easily. The query results also include the movies information which the actors had worked in, so this shows that the personal information of the actors can be easily joined with the movies information by making use of joins and querying the required results.

iv) Query to find the most diverse actor/actress based on the number of various types of roles they have portrayed in different movies/tv shows.

```
SELECT name as Actor_Actress, COUNT(distinct category_id) as
No_Of_Diverse_Roles from person_info
JOIN cast_map
ON cast_map.person_id = person_info.person_id
JOIN role_map
ON role_map.role_id = cast_map.role_id
group by name
HAVING No_Of_Diverse_Roles IN
(SELECT MAX(t.diverse_roles) FROM
(SELECT name, COUNT(distinct category_id) as diverse_roles from person_info
JOIN cast_map
ON cast_map.person_id = person_info.person_id
JOIN role_map
ON role_map.role_id = cast_map.role_id
group by name) t);
```



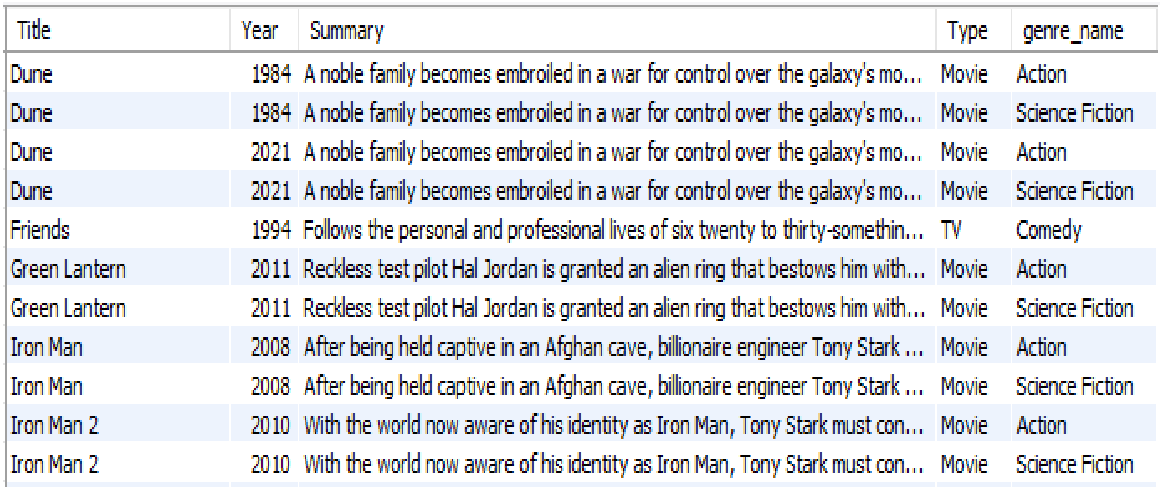
Actor_Actress	No_Of_Diverse_Roles
Robert Downey Jr.	6

Figure 27: Query(iv) Output

This query returns the actor/actress with the most number of different types of roles they have performed. The number of diverse roles is calculated by considering the various categories of the roles as the category of the role determines the nature of a specific role. Example: Tony Stark role is of the category Hero, Superhero, Playboy and Philanthropist. So, this query takes the sum of all the various distinct categories based on the roles an actor/actress have performed in and returns the actor/actress with maximum count. This query illustrates that the power users can even query for the most detailed information related to a character and get the required results.

v) Query to find the preferred movies/tv shows for a user with ID='USR6' by analysing the genres of the movies/tv shows that were rated highly by the same user previously. The movies/tv shows with a rating greater than 5 is considered as a high rating.

```
SELECT distinct title as Title, release_year as Year, synopsis as Summary, type as
Type, genre_name from movies_shows
JOIN movies_shows_info ON
movies_shows.nsan = movies_shows_info.nsan
JOIN genre_map
ON movies_shows.nsan = genre_map.nsan
JOIN genre
ON genre.genre_id = genre_map.genre_id
WHERE (genre_map.genre_id) IN
(SELECT genre_id from ratings
JOIN genre_map
ON ratings.nsan = genre_map.nsan
WHERE user_id = 'USR6'
AND rating> 5)
AND movies_shows.nsan NOT IN
(SELECT nsan from ratings where
  user_id = 'USR6')
ORDER BY title, release_year;
```



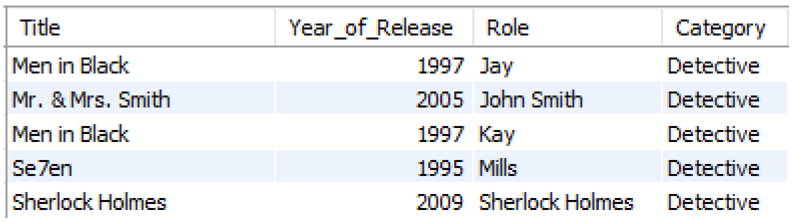
Title	Year	Summary	Type	genre_name
Dune	1984	A noble family becomes embroiled in a war for control over the galaxy's mo...	Movie	Action
Dune	1984	A noble family becomes embroiled in a war for control over the galaxy's mo...	Movie	Science Fiction
Dune	2021	A noble family becomes embroiled in a war for control over the galaxy's mo...	Movie	Action
Dune	2021	A noble family becomes embroiled in a war for control over the galaxy's mo...	Movie	Science Fiction
Friends	1994	Follows the personal and professional lives of six twenty to thirty-somethin...	TV	Comedy
Green Lantern	2011	Reckless test pilot Hal Jordan is granted an alien ring that bestows him with...	Movie	Action
Green Lantern	2011	Reckless test pilot Hal Jordan is granted an alien ring that bestows him with...	Movie	Science Fiction
Iron Man	2008	After being held captive in an Afghan cave, billionaire engineer Tony Stark ...	Movie	Action
Iron Man	2008	After being held captive in an Afghan cave, billionaire engineer Tony Stark ...	Movie	Science Fiction
Iron Man 2	2010	With the world now aware of his identity as Iron Man, Tony Stark must con...	Movie	Action
Iron Man 2	2010	With the world now aware of his identity as Iron Man, Tony Stark must con...	Movie	Science Fiction

Figure 28: Excerpt of Query(v) Output

This query result can be used to provide movies/tv shows recommendation to users by examining the genres of the movies/tv shows that were rated previously by the same user. A user might prefer to watch more movies/tv shows of the same genre.

vi) Query to find all the movies/tv shows that have roles of similar nature. If a user wants to search for all the movies/tv shows that are of type 'Detective' category.

```
SELECT title as Title, release_year as Year_of_Release, role as Role, category as
Category from movies_shows
JOIN movies_shows_info
ON movies_shows.nsan = movies_shows_info.nsan
JOIN cast_map
ON movies_shows.nsan = cast_map.nsan
JOIN role_info
ON cast_map.role_id = role_info.role_id
JOIN role_map
ON role_map.role_id = role_info.role_id
JOIN role_category
ON role_category.category_id = role_map.category_id
JOIN person_info
ON person_info.person_id = cast_map.person_id
where category = 'Detective';
```



Title	Year_of_Release	Role	Category
Men in Black	1997	Jay	Detective
Mr. & Mrs. Smith	2005	John Smith	Detective
Men in Black	1997	Kay	Detective
Se7en	1995	Mills	Detective
Sherlock Holmes	2009	Sherlock Holmes	Detective

Figure 29: Query(vi) Output

This query returns all the movies/tv shows which have roles of similar character type. So, in the above result we get all movies/tv shows where the role type is Detective. Many users decide to watch a movie/tv show based on the kind of characters they are interested in, so this query can be used to provide all the movies /tv shows of similar character which the user searches. This query also illustrates that the power users can query for the various categories of roles and get the results based on the search conditions.

7. Conclusions

The database is designed to support all types of users of the online streaming platform (Nutflux). The user data is stored in separate tables and this ensures that additional information of the various users can be added easily by referencing the user id in the various tables.

User data such as age, hobbies and preferences can be added to the database and that can be used to provide more specific recommendations to a particular user and enhance their experience of using the platform. If the country where the user resides is stored, all the movies/tv shows that are watched a lot in that specific country can be provided first to the users and also make the views more specific to the kind of information being searched in that country.

The queries that the power users search a lot for can be converted into new views as a lot of users would search for similar kind of information and those views can also be used by the Standard users.

The data related to the movies/tv shows are stored in various tables in an organized and clean format. The database is scalable and ensures that any related data can be added to the tables very easily by adding more columns in specific tables. The database allows the extension of adding new tables to store more information related to a specific movie/tv show by referencing them with the NSAN associated to them. This makes it easier for even the power users to query the required information as the tables can be joined easily with the NSAN and retrieve the details related to a specific movie/tv show.

The cast and crew members data are also decentralized, which means that more information regarding an actor/actress or a crew member can be added easily by referencing them with the unique id assigned to each one of them.

The database provides stored procedures which displays movies/tv shows for a particular user based on the ratings given by them and also performs some calculations based on the input for the box office collections. The triggers in the database ensures the data integrity and consistency, more triggers can be added to the tables to ensure the data is consistent throughout and also triggers can be added to check if custom ids are consistent as it is done for nsan values.

The database has some of the tables with ids getting generated with the auto increment feature, so for those tables custom ids can be provided to ensure that the data is more consistent.

The database is normalized and provides the flexibility for the power users to get the required information easily. It also provides a lot of views for the Standard users which makes it easier for them to decide which movie/tv show they would like to watch by looking at the interesting facts and information of that particular movie/tv show.

Acknowledgements

I would like to thank everyone who has helped me in completing this report successfully. I would like to thank and express my sincere gratitude to my professor, Tony Veale, who provided trivial suggestions, good guidelines and constant support to complete this project successfully and also was the source of knowledge which was essential in completing this project. I would also like to thank all the lab demonstrators and the teaching assistants who provided suggestions for designing the database and were always willing to help with any queries that was encountered while completing this project.

I would like to acknowledge that this project is entirely my own and every sentence in this report is written by me and me alone. Any references made in the preparation of this report is mentioned in the references section and also have been cited inline where required.

References

Powell, G. (2006). *Beginning Database Design*
Retrieved from pdfdrive:
https://www.pdfdrive.com/beginning-database-design-e19724423.html

Kruckenberg, M. and Pipes, J. (2005). *Pro MySQL*
Retrieved from pdfdrive:
https://www.pdfdrive.com/pro-mysql-e46803599.html

Harrison, G. and Feuerstein, S. (2006), *MySQL Stored Procedure Programming*
Retrieved from pdfdrive:
https://www.pdfdrive.com/mysql-stored-procedure-programming-e164802373.html

Singh, C. (2015, May). *Normalization in DBMS: 1NF, 2NF, 3NF and BCNF in Database*.
Retrieved from BeginnersBook:
https://beginnersbook.com/2015/05/normalization-in-dbms/

Johari, V. (2018, November). *Advantages & Disadvantages Of SQL Trigger*
Retrieved from techmixing:
https://www.techmixing.com/2018/11/advantages-disadvantages-of-sql-trigger.html

Weinberg, P. and Groff, J. and Oppel, A. (2010) *SQL The Complete Reference, 3rd Edition*
Retrieved from pdfdrive:
https://www.pdfdrive.com/sql-the-complete-reference-third-edition-e18766505.html

Sławińska, M. (2022, January) *What is an SQL View?*
Retrieved from learnsql:
https://learnsql.com/blog/sql-view/