Dataset Description

1. **Title.akas**
    - titleId (string)
    - ordering (integer)
    - title (string)
    - region (string)
    - language (string)
    - types (array)
    - attributes (array)
    - isOriginalTitle (boolean)

    - This dataset mainly gives us info on the release of a movie in different regions.
    - Hence, we may have different title names in various languages in different region for the same movie.
    - The titleId which is given in **string** format identifies a given movie but may be repeated for different regions.
    - The **Ordering** attribute helps us to overcome this and uniquely identify a particular tuple.
    - Thus, we and use Ordering and titleId together as a **primary key**.
    - Other attributes like region tell us the place in the world where the corresponding title is ongoing.
    - We also have information on whether given **title name** is original.
    - As we know many movies have alternate titles. The type attribute further gives us more info of under what categories the title has been released. "dvd", "festival", "tv", "video", "working", etc

    - Hence, in **summary** if someone searches for a movie from a different region(country) or language, we can use this file to provide faster and better access to info.

2. **Title.basics**
    - tconst (string)
    - titleType (string)
    - primaryTitle (string)
    - originalTitle (string)
    - isAdult (boolean)
    - startYear (YYYY)
    - endYear (YYYY)
    - runtimeMinutes
    - genres (string array)

    - In this dataset, each movie has a unique ID.

- This files contents can be used to get info on the Genre of a particular movie
  Eg: when we search of horror movies on Netflix
- This file can also be used to sort out movies which are adult or not adult.
  Eg: The children section in Netflix would not have any adult movies.
- Title.basics also tells us the start and end date of a movie or a tvseries.

- The **TitleType** tells us the type of the title. As we many times see when we search of any title online.

  - Eg: Interstellar : Movie The last one : TvEpisode

- This file will be useful for making a table which consists of **movies** with t_const as PK.

3. **Title.crew**
   - tconst (string)
   - directors (array of nconsts)
   - writers(array of nconsts)

- This file has contains info of directors and writers.
- We can extract info of the **directors** and the **writers** who directed a particular movie

4. **Title.episode**
   - tconst (string)
   - parentTconst (string)
   - seasonNumber (integer)
   - episodeNumber (integer)

- This file contains information on tv series and their episodes
- We have tconst which is a unique identifier for each episode.
- We can also extract info of the parent episode.
- Season and episode number of corresponding to a given episode is also present.

  Example: When we click on a particular tvseries, we are able to view info of
  The seasons, the episodes in each season. Also each episode has a recap of the previous episodes associated with it.

5. **Title.principal**
   - tconst (string)
   - ordering (integer)
   - nconst (string)
   - category (string)
   - job (string)
   - characters (string)

- This a tuple in this file consists of :

  - a unique identifier of a particular movie **(tconst)**
  - a unique identifier for the cast of that particular movie **(nconst)**
  - The role or category that cast played in the given movie. (actor,director,writer,etc) **(category)**
- This dataset can be used to make various tables like actors , directors , writers, producers, etc. Hence we can then write a query to get all the actors which are associated with a particular movie.

6. **Title.ratings**
   - tconst (string)
   - averageRating
   - numVotes

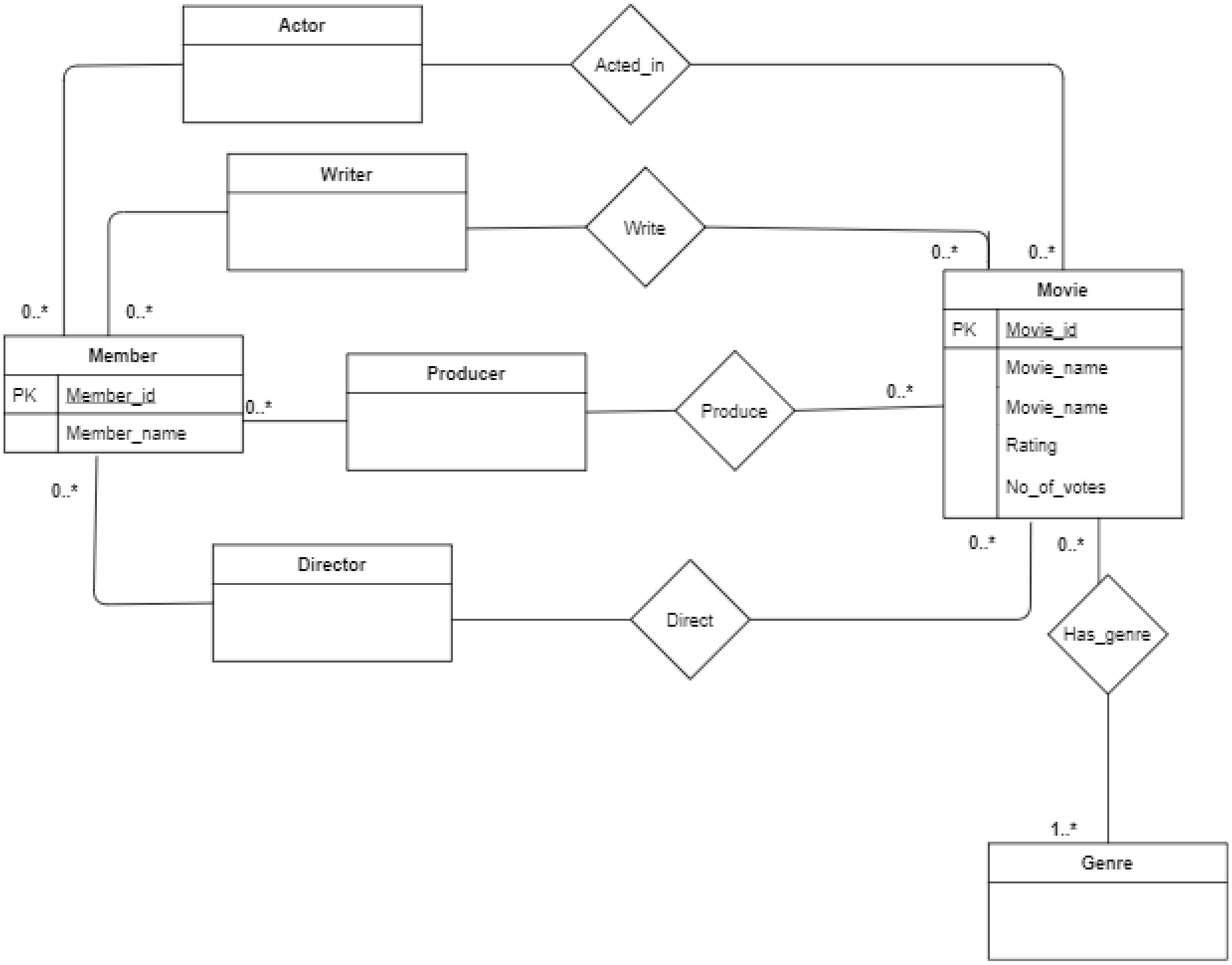
- Every movie has a rating associated with it.
- The **averageRating** gives us the mean of all ratings divided by the total number of votes given.
- Each movie can be mapped to one rating.
- Thus, we can use this datasets averageRating and numvotes attribute can add it to the movies table.
- Hence, when we search for a movie, we will be able to the rating given and the number of votes associated with it.

7. **Name.basics**
   - nconst (string)
   - primaryName (string)
   - birthyear (YYYY)
   - deathYear (YYYY)
   - primaryProfession(array of strings)
   - knownForTitles (array of tconsts)

- This file consists of a unique attribute **n_const**.
- We can extract the name , birth and death year of a person from a table made from this dataset.
- We can also extract the unique IDs of the movie that person is known for and thus the movie names the person is known for.
- We also get info of the Primary profession of every person.

Relational Model for IMDB dataset

Ans.



Relational Model and table descriptions

Table Name : Movie (**Movie_id**(PK) , movie_name, Rating, no_of_votes)

Movie_id (PK)	movie_name	Rating	no_of_votes
1	Interstellar	10.00	10000
2	Rush	10.00	5000

- Movie table has been populated from the **title.basics** data file.

- To ensure that we are not committing everytime we insert a tuple, I have used **preparedStatement.addBatch**, which will be executed only when a certain counter value is reached. This helps to save time.

- It consists of the all non-adult movies
    - **Solving Issues**:
        - I chose t_const as my primary key as it was unique. (eg: tt0000001)
        - To convert it from **String to Integer** I discarded the first 2 alphabets, so I was left with only numbers.
        - Then I used Interger.parseInt() to covert the key from String to integer format
        - Some of the movies have really big names. Hence, I have used TEXT for movie_name to accommodate large values
        - To make sure I'm adding only non-adult movies I simply skip the movie if is_adult ==”1”
        - The values of each movie visited as been stored in a Hashmap so we can get it in O(1) time.

- **Ratings(double)** and **no_of_votes(double)** have been taken from the title.ratings table. Converted to double using **Double.parseDouble()**

- We store them in a Hashmap comprising of **Movie_primary_key: Rating** so we can add to the movie table in O(1) time.

- Null values have been checked using regex and if true, we skip.

- If we don't have a ratings or No_of_votes, put -1 instead

Table Name : Member (Member_id (PK) , member_name)

Member_id (PK)	member_name
56	Matthew McConaughey
92	Chris hemsworth

- The member table consists of all the people and has been loaded from the **name.basics** dataset
- We store each Member_id in a HashMap so that we can access it in O(1) time when needed while creating the Writer, Acted_in, Producer and Director table.

Table Name : Has_genre (**moviegenre_id** , genre_name)

- moviegenre_id FK Movie (Movie_id)

moviegenre_id (FK) Movie (Movie_id)	genre_name
1	Fiction
2	Thriller
2	Short

- Has_genre has been loaded from the **title.basics** dataset.

- Instead of creating another table for called Genre(id, genre_name), Has_genre has been created which does the same function of mapping different genres to a movie and visa versa.

Table Name : Acted_in (**movieacted_id , name_id**)

- Movieacted_id FK Movie(Movie_id)
- Name_id FK member(member_id).

Movieacted_id FK Movie(Movie_id)	Name_id FK member(member_id)
1	56
2	

- Created Since each member can act in many movies and each movie can have many members, (many to many relation)
- This table maps Movie_ids to Name_ids of actors, i.e only members with category = "actor" or "actress" will be included.
- We maintain FK constraint by checking if values we are adding are in the member hash table as well as in the Movie Hashtable

Table Name : Direct (directed_id**, name_id**)

- directed_id FK Movie(Movie_id)
- Name_id FK member(member_id).

directed_id FK Movie(Movie_id)	Name_id FK member(member_id)
400	79
30	45

- Created Since each member can Direct many movies and each movie can have many members, (many to many relation)
- This table maps Movie_ids to Name_ids of Director, i.e only members with category = "director" will be included.
- We maintain FK constraint by checking if values we are adding are in the member hash table as well as in the Movie Hashtable

Table Name : Write (**written_id , name_id**)

- **written_id** FK Movie(Movie_id)
- Name_id FK member(member_id).

written_id FK Movie(Movie_id	Name_id FK member(member_id)
4	70
33	40

- Created Since each member can Write many movies and each movie can have many members, (many to many relation)
- This table maps Movie_ids to Name_ids of Writer, i.e only members with category = “Writer” will be included.
- We maintain FK constraint by checking if values we are adding are in the member hash table as well as in the Movie Hashtable

Table Name : Produce (producer_**id , name_id**)

- producer_id FK Movie(Movie_id)
- Name_id FK member(member_id).

Producer_id FK Movie(Movie_id	Name_id FK member(member_id)
6	4570
39	140

- Created Since each member can Produce many movies and each movie can have many members, (many to many relation)
- This table maps Movie_ids to Name_ids of Writer, i.e only members with category = “Producer” will be included.
- We maintain FK constraint by checking if values we are adding are in the member hash table as well as in the Movie Hashtable

Time Taken : 25mins to load the whole dataset.