Netflix Database Design

Scenario and Requirements:

Netflix is a leading subscription based streaming platform which offers a library of films and television series through distribution deals as well as its own productions. It is especially well known amongst all on-demand video service platforms because of its recommendation system which allows users to help find a movie or TV show to watch. The motivation behind the project is to understand the way Netflix is able to manage the user base of ~ 221 million and stream thousands of movies in a click's time.

For aiding Netflix in its mission to provide stellar service to its customers there is a need to store the data efficiently in a relational database; also, ensuring the optimal and fast retrieval of the data on demand. The proposed database aspires to replicate a fundamental version of the original concept. The database is designed to keep a track of main entities involved like user, payment information, subscription and plan details, content available in the platform.

To ensure exceptional user experience, Netflix provides personalized content and recommends Movies, TV Shows based on the type of contents liked and watched by its users. Additionally, Netflix is the best OTT that successfully stores the session history and lets you resume the media from multiple devices.

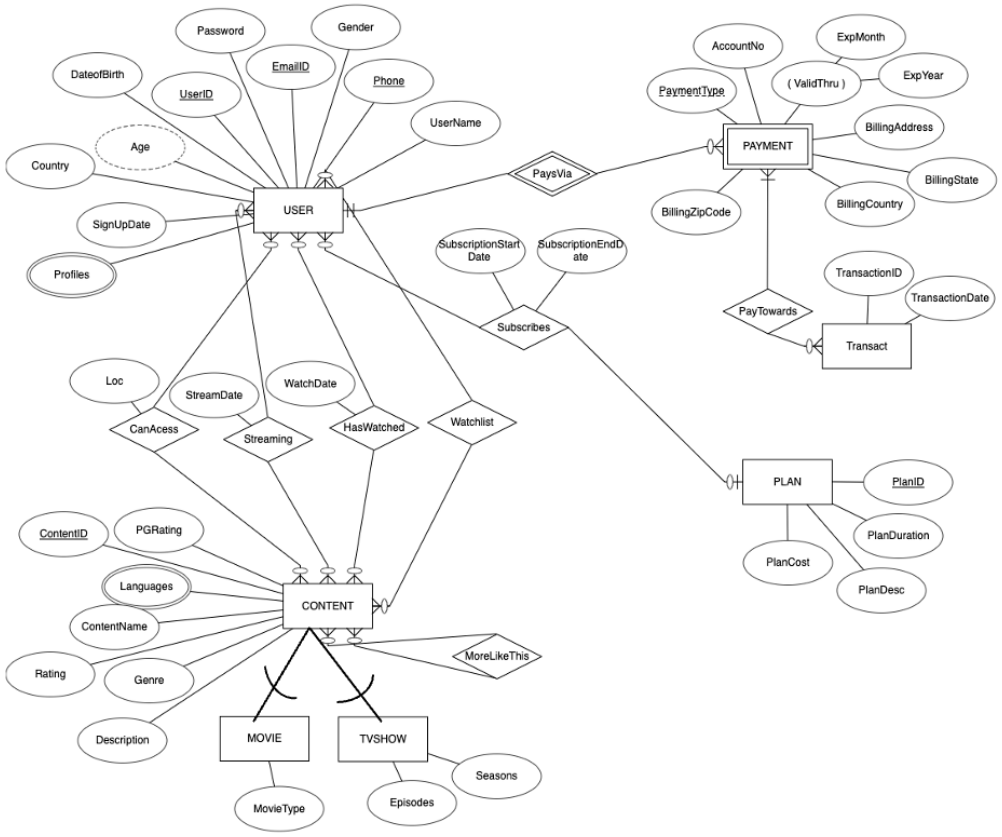
Database Requirements:

The Netflix database will contain following information:

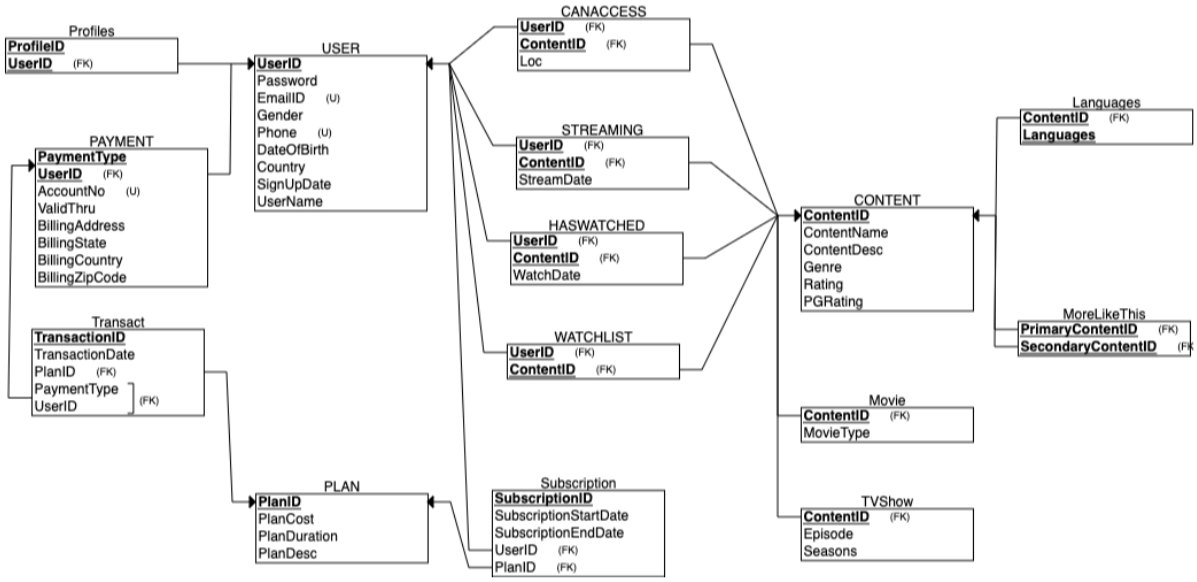
- It will keep track of Users, Payment information, Transaction details, subscription plan, and media content.
- For each user, we will keep track of unique User ID, name, unique Email ID, Password, Gender, Phone (unique), date of birth, country, sign up date. Each user can have multiple profiles.
- For each content, we will store its unique content ID, content name, description, genre, rating, PG rating, and recommendation. Each content may be available in multiple languages.
- Each content will have subclasses – Movie and TV Show. Along with the shared attributes, we will capture movie type for each movie, and season and episode details for each tv show.
- For each plan, we will store a unique plan id, plan cost, plan duration, and plan description.
- For each payment method, we will save payment type, account no., validthru, billing address, billing state, billing country, and zip code.
- For each transaction we will store a unique transaction id, and transaction date.

- A user can have no or multiple payments' information saved. A payment method is unique to a user.
- A user can subscribe to no or one plan. A plan can be subscribed by no or many users.
- A payment by a user can be attributed to 0 or many transactions. A transaction can belong to only one payment method.
- A user can have access to no or many media-contents. A media content is accessible to no or multiple users.
- A user can stream no or many media-contents. A media content can be streamed by no or many users.
- A user had watched no or many media-contents. A media content could have been watched by no or many users.
- A user can add no or many contents in the watchlist. A content can be added by no or many users.
- A new media content is recommended based on the previous media watched by a user.

Enhanced Entity Relationship Diagram (EERD):



Relational Schema:



Normalization:

All the tables in our database are in the third normal form. Since our database contains millions of users, thousands of movies, and TV shows, and billions of transactions from these users over multiple years, we found it inefficient to store any table in denormalized form.

Database Creation and Population:

We referred the following data sources to create data manually and demonstrated how media streaming data is stored in big OTT companies. Additionally, this database provides a use case to analyze and leverage insights from customer watching behavior to retain existing customers and grow new customer base.

https://www.kaggle.com/datasets/shivamb/netflix-shows
https://www.kaggle.com/datasets/harshitshankhdhar/netflix-and-amazon-prime-tv-series-
dataset
https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data

https://www.kaggle.com/datasets/prasertk/netflix-subscription-price-in-different-countries
https://data.world/chasewillden/netflix-shows
https://www.kaggle.com/datasets/karvalo/indian-card-payment-data-set

- All IDs and codes are kept fixed length for a particular table in our database. We used *char* data type for zipcode and all primary/foreign keys.
- We have gender and PGRating as *enum* datatype to define limited number of categories in the respective fields.
- We used *date* data type for relevant fields to capture the timestamp of an event occurring.
- *Float* and *int* data type are used for calculative fields such as cost, rating, and plan duration.
- Rest of our data is in string format (varchar).

Except the validthru(expiry date) field in payment table and rating field in the content table, all other fields are kept not null. Validthru field is null for payment methods which do not have an expiry date such as payments saved as echeck or savings account. Also, there will be cases where a movie has no rating either because it has not been watched by anyone or has not been rated by any user.
All the tables in our database are created with update and delete integrity constraints to ensure that update and deletion of any primary key does not affect the data with foreign keys in another table.

SQL Statements:

Query 1:

This query would help us to find inactive users for marketing team to build custom strategies.

```
384
385 •  select userid, emailid, phone,
386    case when monthdiff between 1 and 3 then 'Category 1'
387    when monthdiff between 4 and 6 then 'Category 2'
388    when monthdiff between 7 and 12 then 'Category 3'
389    else 'Inactive' end as reminder_category
390    from
391    (select u.userid, u.emailid, u.phone, TIMESTAMPDIFF(month, subscriptionenddate, CURDATE()) as monthdiff,
392    row_number() over (partition by u.userid order by s.subscriptionenddate desc) as rownum
393    from user u
394    join subscription s
395    on u.userid = s.userid) x
396    where rownum = 1 and monthdiff >= 1;
397
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

userid	emailid	phone	reminder_category
C127893444	claus@gmail.com	9834145635	Inactive
O997893455	viper@reddit.com	7835234466	Category 1

Query 2:

Recommend movies/TV shows based on the content watched by the user.

```
398
399 •  Select distinct h.userid as user, c1.contentname as watched, c2.contentname as recommended
400    from morelikethis m join haswatched h
401    on m.primarycontentid = h.contentid
402    join content c1
403    on c1.contentid = m.primarycontentid
404    join content c2
405    on c2.contentid = m.secondarycontentid
406    where c2.contentid not in (select h0.contentid from haswatched h0 where h0.userid=h.userid);
407
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user	watched	recommended
A838990311	Jab Tak Hai Jaan	Titanic
A838990311	Money Heist	Inventing Anna
A838990311	Money Heist	Aranyak
A838990311	13 Going On 30	Titanic
B657123422	Money Heist	Inventing Anna
O997893455	Conversations with a Killer: The Ted Bundy Tapes	Parasite

Query 3:

This query would serve as a backend API to notify users of content in their watchlist.

```
408
409 •  select distinct w.userid as user, c.contentname as NotifyToWatch
410    from watchlist w left join streaming s
411    on w.userid = s.userid
412    join content c on w.contentid = c.contentid
413    where s.userid is null;
414
415
416
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

user	NotifyToWatch
B657893433	Reply 1988
B657893433	Frozen

Query 4:

This query would help users to filter content based on their choice.

```
418
419 ●  Select c.contentname, count(t.seasons) as CountSeasons
420    from tvshow t join content c
421    on t.contentid = c.contentid
422    group by t.contentid
423    having count(t.seasons) > 2;
424
425
426
```

contentname	CountSeasons
How I Met Your Mother	6
Money Heist	4