DBMS PROJECT

Video on Demand and Streaming (Netflix)

Guided by Prof. P.M. Jat

<u>Group 101</u>:

Shirin Rajan	(201001007)
Ayush Jain	(201001036)
Vipul Garg	(201001049)
Shreya Shah	(201001058)

Scope and Description:

The inspiration for this project comes from the online video streaming giant, NETFLIX, which allows users to watch movies and TV shows anytime anywhere. For one low monthly price.

Our Database involves keeping records of registered users, available movies, distribution centres, Subscription Packages, Users' history, Users' followers, Users' favourite movies, Details of Production Houses and Users' Renting details.

A user can register and subscribe to at most one of the nine packages available, each package provide certain number of movies for a specific duration like Package G01 lets user watch 70 movies for one month at a price of $29. These packages are valid only for streaming service, for downloading or renting a movie user need to pay accordingly.

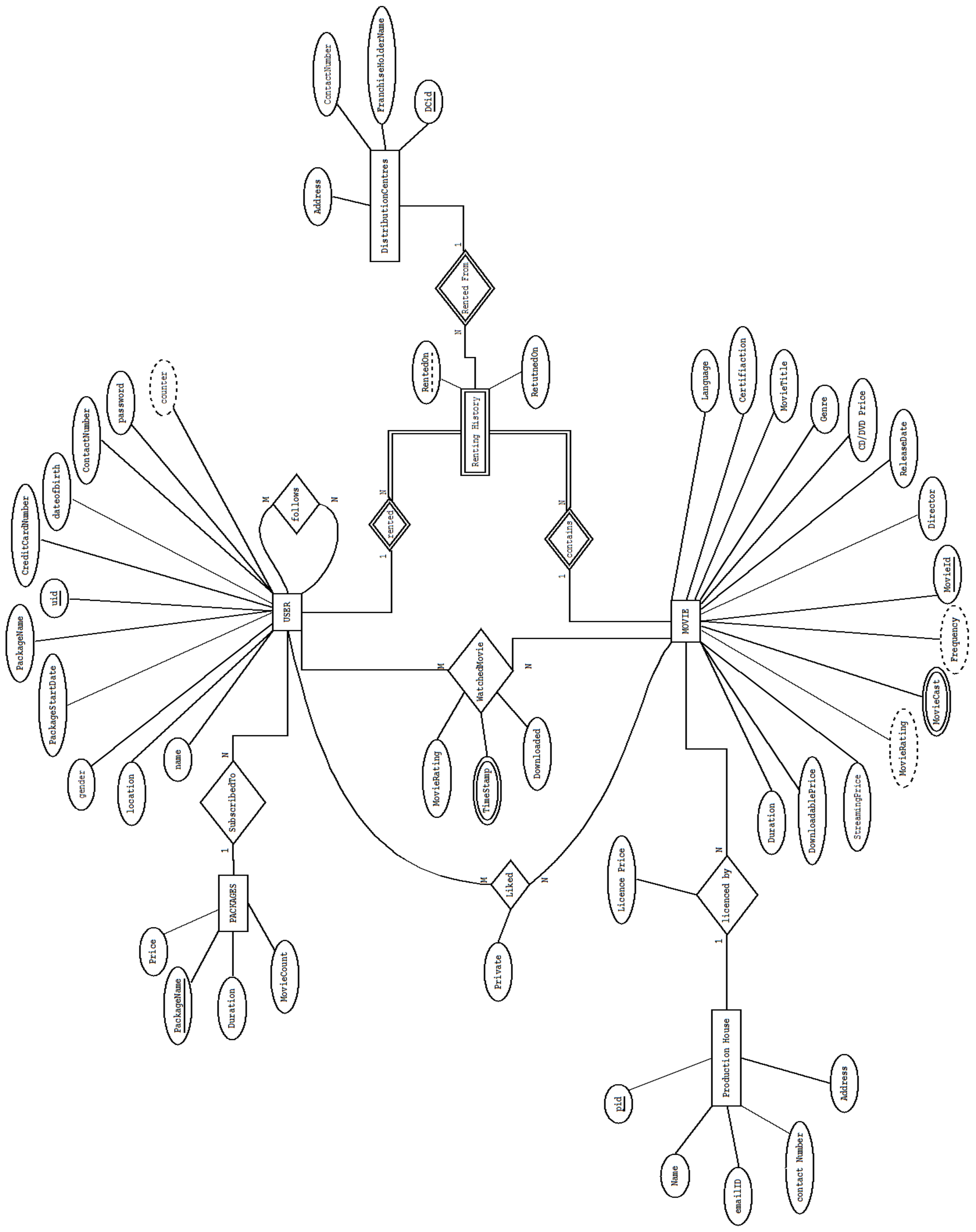
When a user rent a movie through a distribution centre, an entry is logged with details like renting date and dcid, on returning the CD/DVD the return date is updated and the user is charged accordingly if required.

As a user downloads or streams a movie, he can rate it after completing it and hence the rating for the corresponding movie is updated in the movie library. Movies are filtered according to the user's age and movie certification.

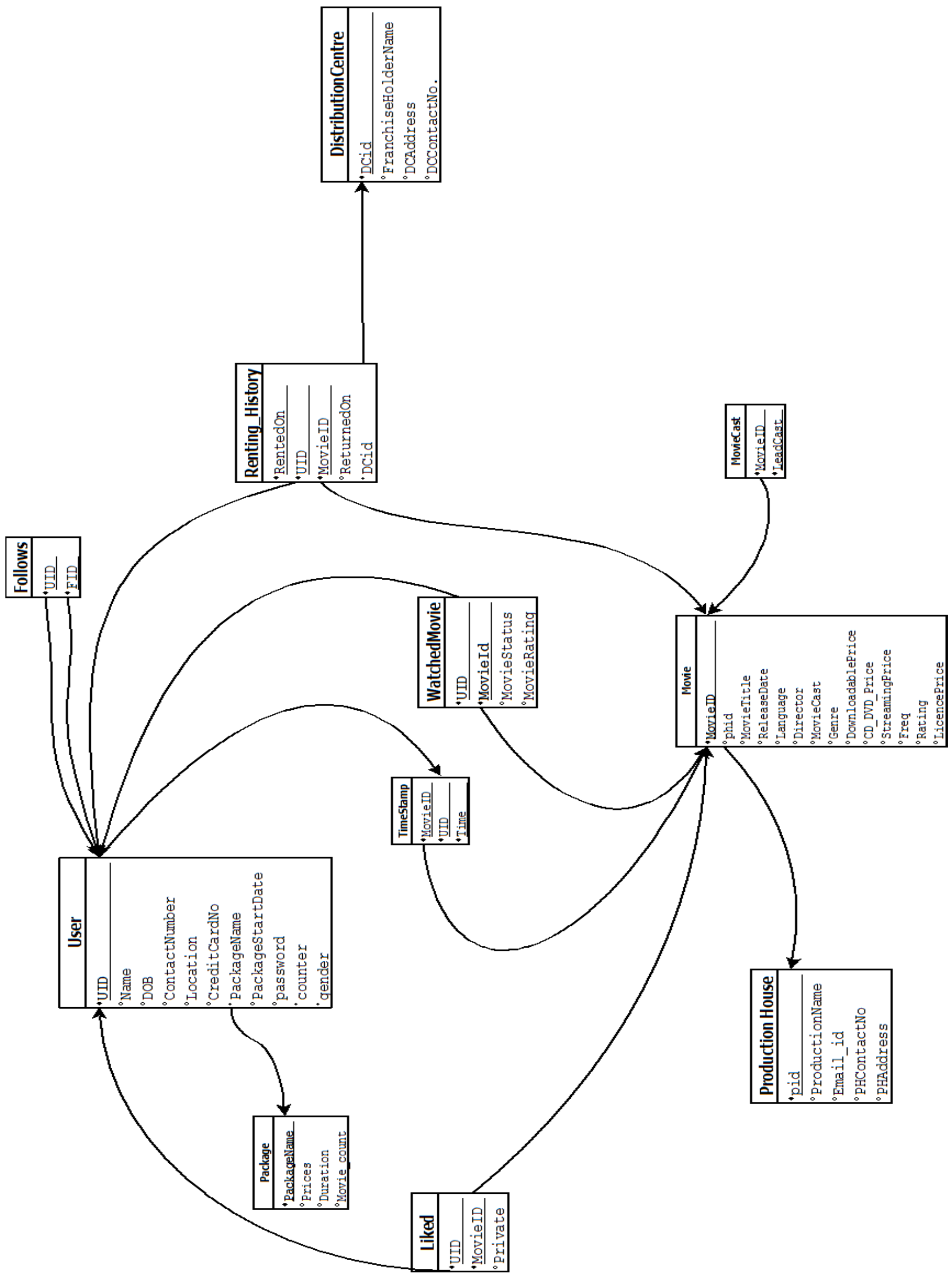
The database also keeps record of the production houses from where movies are purchased. It also stores the licence price for a particular movie.

The database keep the entries of the movies liked by users and also it allows the users to follow each other's lists.

ER Diagram


DistributionCentres
ContactNumber
FranchiseHolderName
DCid
Address
Rented From
1
N
RentedOn
RetutnedOn
Renting History
N
rented
1
N
contains
1
M
follows
N
counter
password
ContactNumber
dateofbirth
CreditCardNumber
uid
PackageName
PackageStartDate
gender
location
name
USER
M
WatchedMovie
N
MovieRating
TimeStamp
Downloaded
N
SubscribedTo
1
PACKAGES
Price
PackageName
Duration
MovieCount
M
Liked
N
Private
MOVIE
Language
Certifiaction
MovieTitle
Genre
CD/DVD Price
ReleaseDate
Director
MovieId
Frequency
MovieCast
MovieRating
StreamingPrice
DownloadablePrice
Duration
N
licenced by
1
Licence Price
Production House
pid
Name
emailID
contact Number
Address

Relational Schema:



FDs and BCNF:

1. **<u>User</u>**
   {UID} → {Name, DOB, City, ContactNumber, CreditCardNo, PackageName,PackageStartDate, Gender, Password, Counter}
2. **<u>Package</u>**
   {PackageName} → {Prices, moviecount, Duration}
3. **<u>WatchedMovie</u>**
   {UID,MovieID} → {MovieRating}
4. **<u>Movie</u>**
   {MovieID} → {MovieTitle, releaseDate, Director, Genre, LeadCast, DownlodablePrice, cd_dvdPrice, streamingPrice, Freq, Rating, certification, language, movieDuration, phid, licencePrice}
5. **<u>RentingHistory</u>**
   {UID,MovieID,RentedOn} → {rentedOn,ReturnedOn}
6. **<u>Production house</u>**
   {pid} →{ProductionName, Email_id, PHcontactNo, PHAddress}
7. **<u>Distribution Centre</u>**
   {dcid}→{FranchiseholderName, DCcontactNo, DCAddress}
8. **<u>Liked</u>**
   {UID, MovieID} → {Private}
9. **<u>Follows</u>**
   {UID, FID}
10. **<u>MovieCast</u>**
    {MovieID, LeadCast}
11. **<u>TimeStamp</u>**
    <u>{MovieID, UID, Time}</u>

The above relations are in BCNF since the FDs are of form A -> B where A is the primary key.

Hence it is in 1NF, 2NF and 3NF.

Queries and SQL Scripts:

```
/*which production house make comedy movies the most? */

select productionname

from productionhouse natural join movie

where genre='Comedy'

group by productionname

order by count(movieid) DESC

limit 1
```

```
/* Earnings due to movie downloads*/
select sum(price) as totaldwnloadedprice from
(
select r1.movieid, r1.count,(downloadableprice*r1.count)as price from
(
 select count(uid), movieid from watchedmovie

 where downloaded='TRUE'
 group by movieid
 order by count(uid)
) as r1  join movie on r1.movieid=movie.movieid

) as r2
```

```
/*Earnings due to CD/DVD renting*/
select sum(price) as totalcdprice from
(
select r1.movieid, r1.count,(cd_dvd_price*r1.count)as price from
(
 select count(uid), movieid from watchedmovie

 where downloaded='FALSE'
 group by movieid
 order by count(uid)
) as r1  join movie on r1.movieid=movie.movieid

) as r2
```

```
/*Earnings due to package subscription*/
select sum(price) from

(select package.price, packagename,uid
from users natural join package
group by packagename, uid,package.price
order by package.price) as r1
```

```
/* most favourite movie of most followed user*/

select movieid

from watchedmovie

where uid in(select fid

from follows

order by fid

limit 1)
```

```
/*which director has work with only one specific production house*/

select distinct director,productionhouse

from((select director

from movie natural join productionhouse

group by director

order by count(phid) DESC

limit 4)as r1 natural join movie natural join productionhouse)

order by distinct director
```

```
/* most famous production house region wise*/

select productionname,dcaddress

from productionhouse natural join movie natural join rentinghistory natural join distributioncentre

order by dcaddress
```

/*Which production house's movies are liked by users the most? */

```
select distinct productionname
from movie natural join productionhouse
where movieid in(select movieid
from productionhouse natural join movie natural join watchedmovie
group by movieid
order by count(uid) DESC
limit 5)
```

/* find the region in which English movies are most watched*/

```
select city,count(language),language
from users natural join watchedmovie natural join movie
where language='English'
group by city,language
order by count(language) DESC
```

/* most profitable movie */

```
select movieid,movietitle
from movie
where movieid in (
select movieid
from watchedmovie natural join movie
group by movieid,downloadableprice
order by downloadableprice*count(uid) DESC
limit 1
)
```

/*which movie is rented for maximum days from which user?*/

```
select uid,movieid,(returnedon-rentedon)
from user natural join rentinghistory natural join movie
order by (returnedon-rentedon) DESC
limit 5
```

/*list of movies watched by the whom the given user follows*/

```
select movieid
from watchedmovie
where uid in (select fid
from follows
order by fid
limit 1)
except all
select movieid from watchedmovie
where uid='dristisharma@fakeuser.com'
```

/*top 5 movies of all the time*/

```
SELECT movietitle
FROM(SELECT count(uid),movieid
FROM movie natural join watchedmovie
GROUP BY movieid
ORDER BY count(uid) DESC
limit 5)as r1 natural join movie
```

/*most famous actor/actress (region wise)   */

```
SELECT leadcast
FROM (SELECT count(movieid),leadcast
FROM moviecast
group by leadcast
order by count(movieid) DESC
limit 4)as r2
```

```
/*most profitable city*/

select city

FROM(SELECT sum(price),city

FROM users natural join package

group by city

order by sum(price) DESC

limit 3)as r4



/*most famous hindi movie*/

SELECT movietitle

FROM(SELECT count(uid),movieid

FROM movie natural join watchedmovie

GROUP BY movieid

ORDER BY count(uid) DESC

limit 5)as r1 natural join movie

where language='Hindi'


/*most famous distribution centre*/

select count(uid),dcid

from users natural join rentinghistory natural join distributioncentre

group by dcid

order by count(uid) DESC

limit 2
```

Triggers and Stored Procedures:

UPDATE_FREQUENCY()

1. Trigger to update the Frequency of a particular movie in 'movie' relation whenever a user watches a movie.

```
DECLARE
movied VARCHAR(8);
frequency numeric;
BEGIN
                movied = NEW.movieid;
        Select freq INTO frequency FROM netflix.movie where movie.movieid = movied;
                  frequency = frequency + 1;
                  UPDATE netflix.movie set freq = frequency
                  WHERE movieid = movied;
                Return NULL;
 END
```

UPDATE_RATING

2. Trigger to calculate the average rating of some movie in 'movie' relation using the individual ratings of users from 'watchedmovie' relation

```
DECLARE
        count1 numeric;
        rate numeric(3,1);
        rating1 numeric(3,1);
        movied VARCHAR(8);

        BEGIN
movied = NEW.movieid;
                select count(movieid) into count1 from netflix.watchedmovie where (watchedmovie.movieid =
movied)AND (watchedmovie.movierating is not NULL);
                select sum(movierating) into rate from netflix.watchedmovie where watchedmovie.movieid =
movied;
                rating1 = rate/count1;

                UPDATE netflix.movie set rating = rating1
                WHERE movieid = movied;

        Return NULL;

END
```

<u>Ckeck_For_timestamp()</u>

3. Trigger to check whether the package subscribed by user is expired or not, to check the age for movie viewing censorship(if the age is below 18, then 'A' certified movies are not offered to that user)

```
CREATE OR REPLACE FUNCTION netflix.timestamp_check()
  RETURNS trigger AS
$BODY$DECLARE
        current_time timestamp without time zone;
        age1 interval;
        uid1 varchar;
        movied varchar;
        dob1 date;
        pdate date;
        pk varchar;
        certificate varchar;
        count1 numeric;
Begin
        uid1 = NEW.uid;
        movied = NEW.movieid;
        select counter into count1 from netflix.users where uid = uid1;
        SELECT LOCALTIMESTAMP into current_time;
        select packagestartdate into pdate from netflix.users where uid = uid1;
        select packagename into pk from netflix.users where uid = uid1;
        select dob into dob1 from netflix.users where uid = uid1;
        select certification into certificate from netflix.movie where movieid = movied;

        if(pk = 'G01' OR pk = 'P01' OR pk = 'S01' AND (current_time - pdate) > (interval '1 month 0 days')) THEN
                return NULL;

        elsif(pk = 'G02' OR pk = 'P02' OR pk = 'S02' AND (current_time - pdate) > (interval '3 months 0 days'))
THEN
                return NULL;
        elsif(pk = 'G03' OR pk = 'P03' OR pk = 'S03' AND (current_time - pdate) > (interval '12 months 0 days'))
THEN
                return NULL;
        end if;
        if(certificate = 'A' AND (current_time - dob1) < (interval '18 years 0 month 0 days')) THEN
                return NULL;
        elsif(Exists(select * from netflix.watchedmovie where uid = uid1 AND movieid = movied)) THEN
                        INSERT INTO netflix.timestamps
                        VALUES (NEW.uid, NEW.movieid, current_time);
                        return NULL;
        else
                        INSERT INTO netflix.timestamps
                        VALUES (NEW.uid, NEW.movieid, current_time);
                        return NEW;
        end if;
END$BODY$
```

Update_MovieCount()

4. Trigger to update the 'counter' attribute in 'users' relation whenever user watches any movie and notify him/her when counter reaches 'moviecount' .

5. Trigger to notify the user when returneddate exceeds duedate in 'rentinghistory' relation of a particular rented movie.